Parallel Minimum Spanning Tree Data Structures and Algorithms

Connor Clayton and Jacqui Fashimpaur

Boruvka's Algorithm

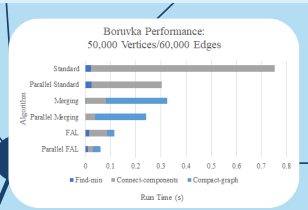
This is a common algorithm for finding the minimum spanning tree. Boruvka's algorithm initially considers each vertex to be its own component, and then proceeds in three stages until there is only one component remaining:

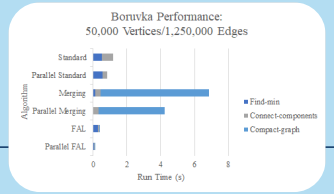
1. Find-min: for each component, find the minimum weight edge leaving that component. Add edges to the MST.
2. Connect-components: using the edges found in find-min, identify which components are now connected and group them together.
3. Compact-graph: condense each connected component into a single supervertex.

Once only one component remains, all vertices have been connected and the MST has been found.

Approach to Parallelization

The basic parallelizable unit in Boruvka's is the connected component. We organized our algorithms into distinct stages which capitalize on independent components. We also implemented Flexible Adjacency Lists to remove large sequential portions of explicit merging.



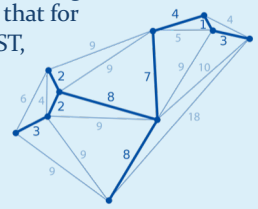


Results

Runtime for the standard implementation increased modestly with graph density, while that of the merging implementations increased dramatically. The FAL implementation out-performed all others due to large speedups in compact-graph. Overall speedup was moderate within implementations due to section dependencies and load-balancing challenges.

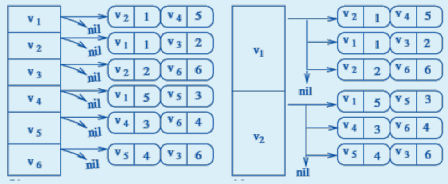
Minimum Spanning Trees

A Minimum Spanning Tree (MST) of a weighted, undirected, connected graph is an acyclic subset of the edges such that all vertices are connected, and the total edge weight is the minimum possible. There are many algorithms for finding an MST, but all rely on the property that for any subset of the MST, the lowest-weight edge intersecting this component will be in the final MST.



Flexible Adjacency Lists

This data structure stores multiple adjacency lists at each vertex as a linked list. To merge two vertices, one simply needs to point one vertex's flexible adjacency list to the other. This is useful for Boruvka's algorithm in that it shifts work away from compact-graph, which is usually the most costly step.



Reference

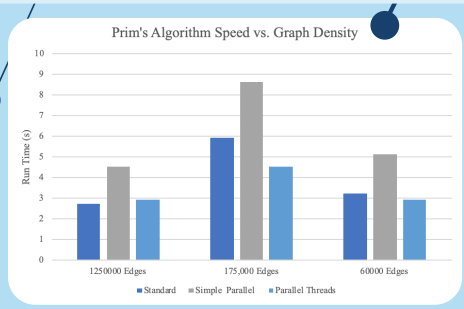
Bader, David A., and Guojing Cong. "Fast Shared-Memory Algorithms for Computing the Minimum Spanning Forest of Sparse Graphs." Journal of Parallel and Distributed Computing, vol. 66, no. 11, 2006, pp. 1366–1378., doi:10.1016/j.jpdc.2006.06.001.

Prim's Algorithm

This is a common algorithm for finding the minimum spanning tree. Prim's algorithm builds up the MST starting with a single vertex. At each iteration, it adds the lowest weight edge in the neighborhood of the current tree until all vertices are included. As written, this algorithm is inherently sequential, but we explored a variation of Prim's algorithm that allows us to add parallelism.

Approach to Parallelization

Our thread-based parallel algorithm involves multiple threads, each starting in a different place in the graph and running their own version of Prim's algorithm to build a subtree. When two subtrees run into each other, they merge and one of the threads starts over somewhere else. While the subtrees are being built, this is quite efficient, but the process of checking for collisions and merging trees is costly and complicated. We faced a lot of challenges involving race conditions and locks.



Results

Our thread-based approach was a modest improvement over our sequential algorithm on most graphs, but communication and lock contention offset many of the benefits of parallelization. Performance was improved when we limited how many times threads could re-spawn, and placed starting points so as to minimize the number of merges we would need.