15-418/618, Parallel Computer Architecture and Programming
Final Project: Parallel Minimum Spanning Tree Algorithms

Xuren Zhou (xurenz), Wenting Ye(wye2)

October 31, 2019

1 URL

GitHub Page.

2 Summary

We will parallelize sequential algorithms for minimum spanning tree, including Kruskal's algorithm and Borůvka's algorithm, evaluate the speedup performance, and profile our implementations.

3 Background

In a connected, edge-weighted undirected graph, a *minimum spanning tree* (MST) is a subset of edges that connects all nodes together with the smallest total weights. It has been well-studied for nearly a century and can be solved in polynomial time with different greedy algorithms. For example, Prim's algorithm [1] uses the *cut-property* of MST, and constructs the MST by adding the smallest edge that connects the current nodes set and the remaining nodes set. However, Prim's algorithm is hard to parallelize in nature because each step depends on the current sub-graph built on previous steps.

In this project, we are going to focus on Kruskal's algorithm [2] and Borůvka's algorithm [3]. Kruskal's algorithm utilizes the *cycles-property* and build the MST by adding the smallest edge that does not create a cycle. The edge sorting takes $O(|E|\log|E|)$ operations and building the MST takes $O(|E|\alpha(|V|))$ operations, where α is the inverse Ackermann function. In practice, $\alpha(|V|)$ grows extremely slow and hence the overall complexity is $O(|E|\log|E|)$. Borůvka's algorithm starts from making each node as an individual component, and then keeps finding and contracting the smallest edge for each component that connects to any other component. The overall complexity for Borůvka's algorithm is $O(|E|\log|V|)$. In this project, we will focus on these two algorithms. We will implement the serial algorithms first, and then parallelize it in shared memory space.

4 Challenges

Kruskal's algorithm involves two steps: firstly, sort all edges by its weights, and then keep adding edges if no cycle is produced. Although there is a parallel sorting algorithm for the first part, the second part is inherently sequential and can not be properly parallelized. This is because whether the new edge produce cycle depends on all previous added edges.

Borůvka's algorithm has two main steps. The first is to find the adjacent edge with the smallest weight for each connected component, and the second is to contract the selected edges. The first part can be parallelized by components (vertices). However, race condition happens when edge contraction happens on the same vertex. How to improve efficiency while keeping the correctness is a challenging problem.

5 Resources

We will implement all algorithms from scratch. There are several research papers about parallel MST [4–6]. There is also a good summary in Wikipedia [7]. As far as we know, there is no code available online. We are going to run our initial experiment on GHC machines and CloudLab[1] if available.

6 Goals and Deliverables

6.1 Plan to achieve

- Implement two sequential algorithms, Kruskal's algorithm and Borůvka's algorithms, as the baseline of our performance benchmark.
- Parallelize these two sequential algorithms in shared-memory model using OpenMP:
  - Krushal's algorithm with parallel sorting and Filter-Kruskal [5],
  - Borůvka's algorithm with edge contraction [4].
- Benchmark the speedup performance of our parallel implementations under different types and sizes of input graphs. We plan to consider two kinds of graphs: dense graphs and sparse graphs.
- Explore the speedup performance of different parallel components, such as the parallel sorting in Krushal's algorithm and the parallel finding adjacent edge with the smallest weight of each vertex in Borůvka's algorithm.

6.2 Hope to achieve

- Explore the speedup performance on input graphs with power-law distribution.
- Implement a faster shared-memory MST algorithm proposed by Bader *et al.* [6].
- Explore the possibility to implement parallel Krushal's and Borůvka's algorithms in message-passing model using MPI.

6.3 Poster deliverables

We will deliver the speedup graphs of our parallel implementations under different numbers of processors and different types and sizes of input graphs. We will also deliver speedup graphs, or tables, of different parallel components. We think a reasonable speedup with respect to the number of processors will demonstrate that demonstrate we did a good job. If we cannot get a good speedup, we will try to profile our implementations to show that the parallel overhead is inevitable under our testing platform and provide potential specs of the testing platform to improve the speedup of our implementations.

7 Platform Choice

We plan to use OpenMP as our parallel framework and choose GHC machines to test our implementation. After well-tested on GHC machines, we will try to run our benchmark experiments on CloudLab if available. We use the shared-memory model so we can not take advantage of the distributed cluster, but there are more powerful machines on CloudLab so it is still a good platform for us to do tests.

[1] https://www.cloudlab.us/

8 Schedule

The initial schedule in provided in Table 1.

Week	Due	Task	Assigned to	Status
1	11.3	Project Proposal	Both	Completed
1	11.3	Implement graph data structure and graph data generator	TBD	
2	11.10	Implement sequential Kruskal's algorithm	TBD	
2	11.10	Implement parallel Kruskal's algorithm with parallel sorting	TBD	
3	11.17	Implement parallel Kruskal's algorithm with Filter-Kruskal	TBD	
3	11.17	Benchmark and profile parallel Kruskals' algorithms	TBD	
3	11.17	Project checkpoint report	TBD	
4	11.24	Implement sequential Borůvka's algorithm	TBD	
5	12.1	Implement sequential Borůvka's algorithm with edge contraction	TBD	
5	12.1	Implement parallel Borůvka's algorithm	TBD	
6	12.8	Benchmark and profile parallel Borůvka's algorithm	TBD	
6	12.8	Poster and final report	TBD	

Table 1: Initial schedule.

References

[1] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[2] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[3] J. Nešetřil, E. Milková, and H. Nešetřilová, "Otakar borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history," *Discrete mathematics*, vol. 233, no. 1-3, pp. 3–36, 2001.

[4] S. Chung and A. Condon, "Parallel implementation of bouvka's minimum spanning tree algorithm," in *Proceedings of International Conference on Parallel Processing*, pp. 302–308, IEEE, 1996.

[5] V. Osipov, P. Sanders, and J. Singler, "The filter-kruskal minimum spanning tree algorithm," in *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pp. 52–61, Society for Industrial and Applied Mathematics, 2009.

[6] D. A. Bader and G. Cong, "Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs," *Journal of Parallel and Distributed Computing*, vol. 66, no. 11, pp. 1366–1378, 2006.

[7] Wikipedia, "Parallel algorithms for minimum spanning trees — Wikipedia, the free encyclopedia," 2019. [Online; accessed 30-October-2019].