Parallel Dijkstra's Algorithm

Po-Kai Wang
Department of Computer Science
National Chiao Tung University
HsinChu City 300, Taiwan
adkevin3307@gmail.com

Aman Sinha
Electrical Engineering and Computer Science
National Chiao Tung University
HsinChu City 300, Taiwan
amansinha.sw@gmail.com

Cheng-Tes Ho
Department of Computer Science
National Chiao Tung University
HsinChu City 300, Taiwan
djjjimmyyy@gmail.com

1 INTRODUCTION / MOTIVATION

The shortest path problem is finding the shortest path between nodes in a graph, which may represent the problem like road networks or Internet connection. The problem is important when the cost of connections is really expensive.

Finding the shortest path is a well-known problem in the algorithm area, and it already has many solutions for both single-source path and all-pairs shortest path. Although most of the solutions for the shortest path problem have already reduced time complexity, most of them will struggle in a large graph.

The algorithm we choose is Dijkstra's algorithm. The algorithm is a popular algorithm which can deal with single-source shortest path issue, so we are interested in speed up the algorithm. Moreover, we hope we can widely apply the parallel method to other similar algorithms.

2 STATEMENT OF PROBLEM

The graph in the shortest path problem can see as an Internet connection graph, we assume the graph is a directed and positive weight graph in our case. With the graph in our case, we can find a single-source shortest path from source to destination by Dijkstra's algorithm.

Because the series Dijkstra's algorithm has $O(n^2)$ time complexity, which may be too slow when dealing with a big graph, we want to use the parallel methods to improve the algorithm.

3 PROPOSED APPROACHES

First, we divide vertices into equal length vertex sets and assign them to threads. Each thread identifies the local closest vertex in its vertex set to the source vertex. Second, store this value in shared memory, then perform a parallel algorithm to select the globally nearest vertex, that should be barriers at each iteration. Finally, each process updates the value of vertices in its vertex set. The block diagram is shown at Figure 1.

4 LANGUAGE SELECTION

- MPI: Ability to have distributed processing over a cluster, instead of a single node
- OpenCL: Faster vector primitives available for some explicit parallelism; GPU acceleration
- OpenMP: Sometimes better than OpenCL for reduction operations

5 RELATED WORK

Since Dijkstra's algorithm is a complex graph algorithm that has low computation to memory access ratio and requires a lot of synchronizations, the following concerns need to be answered for solution on parallel hardware:

- Optimize memory accesses
- Minimize synchronizations
- Exploit parallelism in the algorithm

Following are some of the related works for parallelization of Dijkstra's algorithm:

- Graph partitioning into neighboring sub-graphs: kd-tree is utilized in [1] in order to generate arc labels that mark, for each arc, possible target nodes of a shortest paths that start with this arc. Dijkstra's algorithm can then be restricted to arcs whose label mark the target node of the current search, because a sub-path of a shortest path is also a shortest path. Partitioning algorithms from computational geometry are utilized in combination with arc labels. But, the graph partitioning is a NP-Hard problem and hence may be impractical for large graphs.
- Delta-stepping algorithm: [2] divides original Dijkstra's algorithm into a number of phases that can execute in parallel. This algorithm is an approximate bucket implementation of Dijkstra's original algorithm. For random graphs with uniformly distributed edge weights, this algorithm runs in sub-linear time with linear average case work. [3] shows 30x improvements over a high-end Shared memory platform, also displaying good vertical scalability.
- GPU-based implementation: It has been observed in [4] that GPU-based solution is worse performing than the sequential implementation for graphs with more than a million vertices. Moreover, the device memory is not enough to handle large graphs.
- Intel Many Core Integrated (MIC) architecture: [5] implements a asynchronous version on the shared memory architecture, with aim to have cache optimizations. The work achieved the highest performance yet for Dijkstra's algorithm on Intel Xeon Phi processor by minimizing irregular memory access patterns.

6 STATEMENT OF EXPECTED RESULTS

We hope the parallel Dijkstra's algorithm can faster than the serial Dijkstra algorithm with at least two times efficiency. And we also want to find out the limit improvement of our parallelism strategies.

Figure 1: Block Diagram

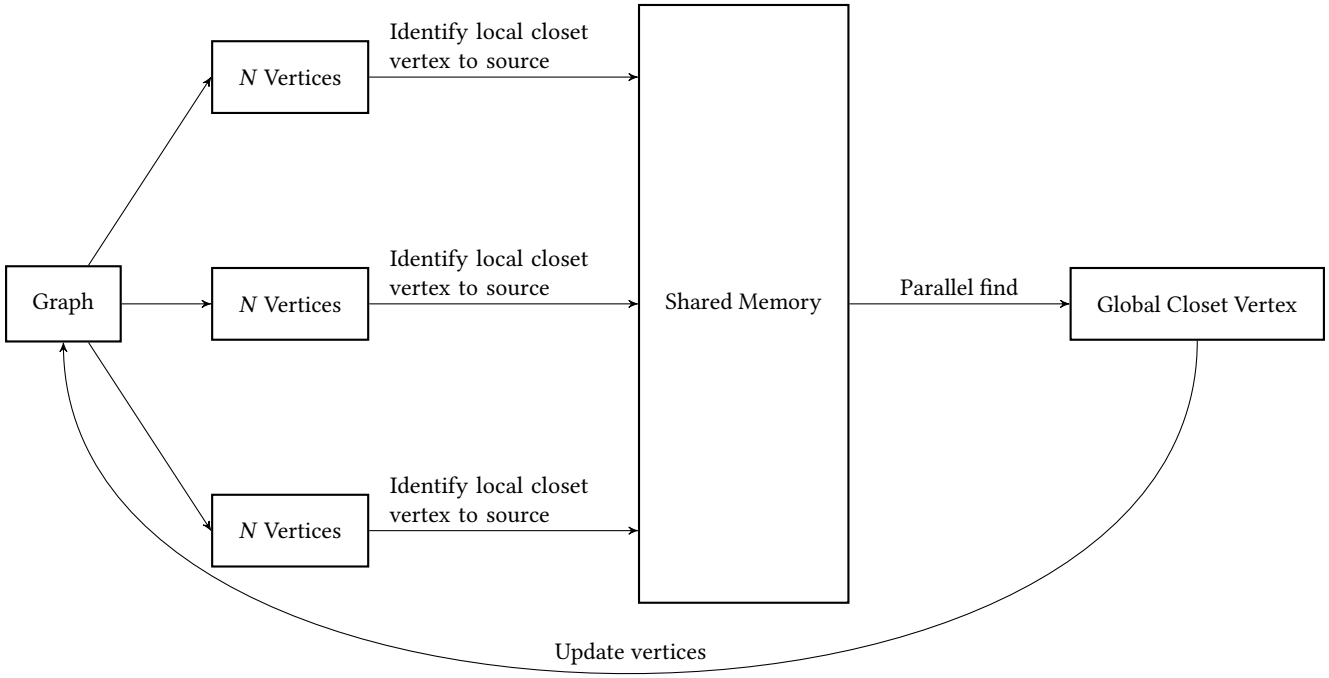

Table 1: Time Schedule

Week	Date	Description
1	11/08 - 11/14	• Begin Project • Graph Representation • Graph Partitioning • Input and Output Format • Parallelism Strategies • Design Experiment
2	11/15 - 11/21	• Implement Series Dijkstra's Algorithm • Generate Input and Output Data
3	11/22 - 11/28	• Decide Parallelism Strategies • Survey Related Work • Implement Parallel Dijkstra's Algorithm
4	11/29 - 12/05	• Finish Code • Check Series and Parallel Version Correct
5	12/06 - 12/12	• Make Optimization • Check Experiment Appropriate • Begin Experiment • Analysis Experiment Result
6	12/13 - 12/19	• Write Report and Slides • Prepare Presentation

7 TIMETABLE

The timetable is shown at Table 1.

REFERENCES

[1] Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning graphs to speed up Dijkstra's algorithm. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 189–202. Springer, Heidelberg (2005). https://doi.org/10.1007/11427186_18

[2] U. Meyer and P. Sanders. -stepping: a parallelizable shortest path algorithm.J. Algs.,49(1):114–152, 2003.

[3] Madduri, K., Bader, D.A., Berry, J.W., Crobak, J.R.: Parallel shortest path algorithms for solving large-scale instances. In: Dimacs Implementation Challenge - The Shortest Path Problem, vol. 74, pp. 249–290 (2011)

[4] Harish, P., Narayanan, P.J.: Accelerating Large graph algorithms on the GPU using CUDA. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 197–208. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77220-0_21

[5] Zhang, W., Zhang, L. and Chen, Y., 2018, November. Asynchronous parallel Dijkstra's algorithm on intel xeon phi processor. In International Conference on Algorithms and Architectures for Parallel Processing (pp. 337-357). Springer, Cham.