Parallel and Sequential Implementation of Borůvka's Algorithm with OpenMP

Abstract

The problem of finding suitable Minimum Spanning Trees (MSTs) of undirected graphs has been popular for quite some time in the field of graph theory, and many algorithms have come into being in order to solve this problem efficiently. The problem arises in fields such as that of transportation, logistics, communications, image processing , wireless networks, cluster analysis etc.. One of the algorithms that can be used to solve this problem is Borůvka's algorithm. It is one of the oldest greedy algorithms on record and was published by Otakar Borůvka, a Czech mathematician who was best known for his discoveries in graph theory. The algorithm was first modeled in order to design an electricity distribution network but can now be used to find the MST of undirected graphs. By implementing both parallel and sequential forms of the algorithm, using multi-thread programming with OpenMP, we compare the results.

1. Introduction

To find the minimum spanning tree of a graph is one of the most well-known *combinatorial optimization problems*. Methods to solve such a problem are usually simple but have led to the growth of ideas in modern combinatorics, playing a key role in scientific and computer algorithms as seen in many works [1][2]. The problem itself is as follows: Given a weighted graph, which is a graph that has certain weights or costs for each of its edges or having different length edges, find the tree that connects all nodes such that the sum of costs of all paths or the sum of all the edge lengths is minimum. To apply this problem to modern life is to take the nodes as cities or the edges as the communication network connections and in doing so the edge weights can be distances or construction costs. The important thing in this is that the tree selected should not form a cycle.

In [3], Ellis Horowitz explains one of the reasons why the MST problem is so popular amongst mathematicians even till date is because it *accepts efficient solutions*. As seen in [4] even graphs with thousands of vertices can have a solution. Enumerated in [5], [6] and other literary works by distinguished scholars, the applications of such algorithms can be found in constructing transportation networks, pipe flow networks, telephone communication networks etc.. It offers a method of solution to other real world problems in a less direct manner. These include image processing, classification problems, speech recognition, network reliability and so on. Looking at it in a more theoretical perspective, the MST algorithms usually take a greedy method route in finding the tree. This means that at each node it finds the next best edge to add to the set of edges of the tree.

2. Related Works

In 2012, a Simultaneous Localization and Map Building (SLAM) technique was proposed by Songmin Jia, Xiaolin Yu and Xiuzhi Li [7]. This technique was used for navigation by mobile robots inside buildings and houses and was based on OpenMP programming and parallelization. The technique used Particle Filter (PF) to find an efficient route by building solid frameworks but in doing so, the programming needed an optimization as the PF provided complexities and required higher computation power. Using OpenMP's multi-thread programming, they proposed a parallelized algorithm to reduce the overall computation time of PF and optimize the execution of SLAM. From the results of the experiment, they could conclude that this was successful in reducing the PF-SLAM execution time while also prioritizing the accuracy of the navigation frameworks created by SLAM.

Rolf Rabenseifner and Gerhard Wellein, in 2003, considered the usage of hybrid MPI+OpenMP programming to construct parallel applications[8]. The main idea behind this was for shared memory nodes to combine both distributed memory parallelization on the node interconnect as well as the memory parallelization of each node in the system. Tests and simulations were run to check the latency and bandwidth issues with such programming models. In the end they concluded that this hybrid architecture could only be used if the SMP model was enhanced by optimization features on the total node traffic. If the communication was to get bottle-necked at the master thread on top of which long messages were used, this hybrid system would fail and using either MPI or OpenMP programming is more viable.

Wenfei Fan , Jingbo Xu , Yinghui Wu present a parallel system known as parallel GRAPh Engine (GRAPE) [9] which was used for computing graphs. It is essentially an algorithm that takes sequential graph algorithms as input, then parallelizes and optimizes it. It promises accuracy under most conditions as long as the sequential algorithm is correct. As long as the sequential algorithms are correct, their GRAPE parallelization guarantees to terminate with correct answers under a monotonic condition. The system does the parallelization using a combination of partial evaluation and incremental computation while using graph optimizations such as indexing, partitioning and compression.

3. Implementation

Many MST algorithms use the light edge property which says that in an undirected graph G, the edge with the minimum weight that has been connected by a cut will be used in the MST. Borůvka's algorithm is a well-known algorithm for finding minimum spanning trees by using the light edge property and following the greedy method. This way, the MST can be computed by parallel programming in such a way that for each vertex V, the algorithm can find the minimum weight simultaneously. Using this logic, Borůvka's algorithm works as follows:

For each edge that is present:

1. From all the outgoing edges of each vertex, the edge with least weight cost is found, forming a connected component.
2. Create a directive each time an edge is chosen and select a super vertex for all the vertex in the now new tree.
3. Complete the step by adding the edge to the tree officially and remove all the other redundant edges.

Keeping this in mind, we can say that there are two ways this algorithm can be implemented:

1. Star Contraction: It uses 50/50 probability at each vertex by tossing a coin and for each bit of 0 or 1, to decide which vertex is a star and which is a satellite.
2. Edge Contraction: This is the method that was just explained including the usage of the greedy method and incorporating the light edge property.

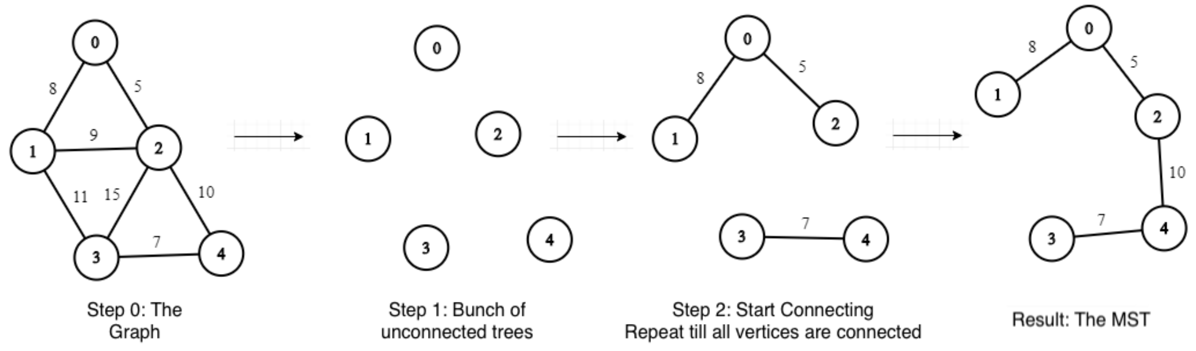
Step 0: The Graph
Step 1: Bunch of unconnected trees
Step 2: Start Connecting Repeat till all vertices are connected
Result: The MST

Figure 1:

3.1. Code

There are two parts of the implementation code that need to be enumerated. It utilizes two data structures such as the Find Union data structure and the Graph Creation function. The Find Union function was used to hold the values of the parent of the tree and the rank of each vertex in the tree. The Graph Creation function is used to hold the list of edges of the graph. The code written includes the code to create randomized graphs to find the MST of the algorithm. The speciality of this implementation is that the edges for each graph are stored twice. This is because, if it were to be stored once, it would be stored in such a way that it would be representing a directed graph. So to show the bidirectional property of an undirected edge, each edge in the list was stored two times. Each edge is stored in the list as (u,v), for an edge directed from u to v. The list is sorted before execution of the algorithm.

This, of course, is not the most efficient storage structure to use for a graph but the idea of an adjacency matrix was even less efficient so we had to take the one with higher probability of success. Using the compare and swap synchronization principle [10] to find the parent and rank of the tree nodes but the rank attribute was only needed for the sequential implementation.

3.2. Sequential Implementation

As the idea suggests, the algorithm goes through each vertex of the graph sequentially and finds the edge with the least weight cost for each. Once doing so, the edge is contracted into the tree and the redundant edges are removed. Each of the edges are split between processors for this computation. The same has been done in the parallel computation. This is to optimize the execution of the algorithm. The minimum edges are stored in separate arrays for each vertex and then compared and added during the contraction. In the end the comparison of edges stored is done to check for any cycles in the MST. If such a cycle exists, one edge that is forming the cycle is removed from the list. This is repeated recursively until a single component remains (connected graph) or if the number of components remaining doesn't change after an iteration at a vertex. This is to show that no more edges can be removed as the minimum edges have been computed and contracted already.

3.3. Parallel Implementation

The idea was to take the sequential execution of the algorithm and add loops to parallely compute the minimum edges for vertices while using synchronization techniques. The program calls the function for calculating MST and returns an ordered array list of all the edges that are present in the MST.

The various functions that collaborate for this process are those used for adding the edges to the list, calculating the nodes left in the graph (essentially removing the redundant nodes and edges) and of course for calculating the minimum edge at the node. The previously mentioned edge list is shared amongst a predetermined number of processors and each processor is assigned a node. Now each processor executes the program to find the minimum edge of their assigned node and store it as an array, as opposed to using hashmaps, since all of this is done in constant time.

Table 1: Comparative analysis of the serial and parallel implementation of the Boruvka's Algorithm

Sequential Implementation	**Parallel Implementation**
Each node is visited serially to find minimum weight edges for them	Nodes are assigned to multiple processors to compute minimum edge
Runs on a single processor	Runs on multiple processors
Has constant execution time	Slower than sequential implementation only when there are less threads used for computation

These calculated edges are merged to form a global array of minimum edges using a compare and swap function. Having the original edge list sorted leads to this global array of edges representing a contiguous data structure as well, which happens to be our minimum spanning tree in this case. Another function that is used to add the values of these edges to get the total weight is executed using a parallel Disjoint Set Union (DSU) algorithm.

4. Results

For E number of edges, V number of vertices and P number of processors, a time complexity of O(E log V) and O(E $\log^2$V/P) was established for the sequential and parallel implementations respectively. To truly find the comparison between the two implementations, we had to execute the code on larger graphs having 1 million nodes and 2.5 million edges. Then, we set the thread counts as 4, 8, 12, 16, 20, 24 and 32 for each. Plotting a graph for both iterations shows that the parallel implementation is quite slow compared to the serial implementation. But the moment 32 threads were used, the parallel implementation performed exceptionally better than the serial algorithm.

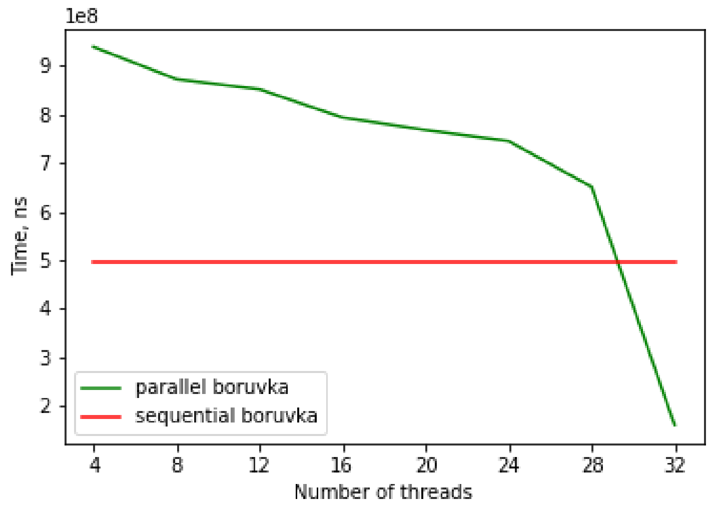

Figure 2:

From these results, it can be said that there is a huge amount of overhead caused by the parallelization of the algorithm but this can be combated with an increase in the number of threads used for execution. But care must be taken in not allowing too many threads to be created, as that too can cause a lot of overhead. This is because as more edges are added to the MST more threads are using processor resources which leads to unnecessary work being done.

5. Conclusion

A fully connected tree is a spanning tree in which two disjoint subset of vertices must be connected. Boruvka's algorithm is a greedy algorithm similar to Prim’s and Kruskal’s algorithms with a time complexity of O(ElogV). Parallelizing such an algorithm helps in making randomization algorithms quicker and more efficient. This work provides a comparison between the sequential and parallel implementations of the Boruvka's algorithm and also shows that the latter implementation leads to an improved performance as the number of threads are increased beyond 32. Although parallelising Boruvka's algorithm improves performance for random graphs and spanning trees, sparse graphs have not seen much improvement through parallelization.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. The Design and Analysis of Computer Algorithms. Reading, Mass., Addison-Wesley, 1974.
[2] E. M. Reingold, J. Nievergelt, and N. Deo. Combinatorial Algorithms: Theory and Practice. New York, Prentice-Hall, 1977.
[3] E. Horowitz and S. Sahni. Fundamentals of Computer Algorithms. Potomac, Md., Computer Science Press, 1978.
[4] J. L. Bentley and J. H. Friedman. Fast algorithm for constructing minimal spanning trees in coordinate spaces. Tech. Rept. STAN-CS-75-529. Stanford University, January 1976.
[5] W. Chou and A. Kershenbaum. A unified algorithm for designing multidrop teleprocessing networks. Data networks-Analysis and design. Proc. DATACOMM73, IEEE 3d Data Communications Symp., pp. 148-156.
[6] H. Loberman and A. Weinberger. Formal procedures for connecting terminals with a minimum total wire length. J. ACM 4, 4 (October 1957), 428-437.
[7]Songmin Jia, Xiaolin Yin, and Xiuzhi Li, "Mobile Robot Parallel PF-SLAM Based on OpenMP", IEEE, International Conference on Robotics and Biomimetics, December 2012.
[8] R. Rabenseifner, G. Wellein: Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures. International Journal of High Performance Computing Applications 17(1), 49–62 (2003).
[9] Wenfei Fan, Wenyuan Yu, Jingbo Xu, Jingren Zhou, Xiaojian Luo, Qiang Yin, Ping Lu, Yang Cao, and Ruiqi Xu. 2018. Parallelizing Sequential Graph Computations. ACM Trans. Database Syst. 43, 4, Article 18 (December 2018), 39 pages.
[10] S. Prakash, Yann Hang Lee and T. Johnson, "A nonblocking algorithm for shared queues using compare-and-swap," in IEEE Transactions on Computers, vol. 43, no. 5, pp. 548-559, May 1994, doi: 10.1109/12.280802.
[11] R. L. Graham and P. Hell, "On the History of the Minimum Spanning Tree Problem," in Annals of the History of Computing, vol. 7, no. 1, pp. 43-57, Jan.-March 1985, doi: 10.1109/MAHC.1985.10011.