Article can be accessed online at http://www.publishingindia.com

An Optimisation Approach for Construction of a Distributed Minimum Spanning Tree (DMST) Using MPI

Md. Akkas Ali*

Abstract

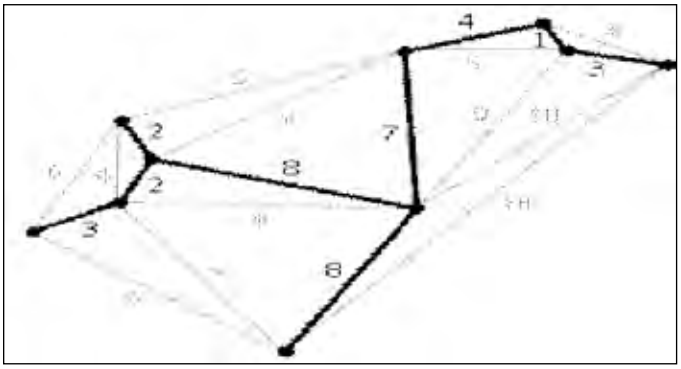
The present paper determines Distributed Minimum Spanning Tree (DMST) of very large graphs. It is very time consuming to calculate in a single machine. So the researcher has used parallel programming. One of the DMST algorithms that support parallel computing is Boruvka's algorithm. The researcher has used this algorithm. To avail the parallelism, we have used the MPI architecture.

Keywords: Distributed Minimum Spanning Tree (DMST), Message Passing Interface (MPI), Parallelism, Boruvka's Algorithm

Introduction

The distributed minimum spanning tree (DMST) problem involves the construction of a minimum spanning tree by a distributed algorithm, as shown in Fig. 1, in a network where nodes communicate by message passing. It is radically different from the classical sequential problem, although the most basic approach resembles Boruvka's algorithm. One important application of this problem is to find a tree that can be used for broadcasting. In particular, if the cost for a message to pass through an edge in a graph is significant, a MST can minimize the total cost for a source process to communicate with all the other processes in the network. The message-passing model is one of the most commonly used models in distributed computing. In this model, each process is modeled as a node of a graph. The communication channel between two processes is an edge of the graph (books.google.com.bd). Two commonly used algorithms for the classical minimum spanning tree problem are Prim's algorithm and Kruskal's algorithm (en.wikipedia.org). However, it is difficult to apply these two algorithms in the distributed message-passing model.

Fig. 1: An Example of Distributed Minimum Spanning Tree (DMST)



The main challenges are:

- Both Prim's algorithm and Kruskal's algorithm require processing one node or vertex at a time, making it difficult to make them run in parallel. (For example, Kruskal's algorithm processes edges in turn, deciding whether to include the edge in the MST based on whether it would form a cycle with all previously chosen edges.)
- Both Prim's algorithm and Kruskal's algorithm require processes to know the state of the whole graph, which is very difficult to discover in the message-passing model.

Due to these difficulties, new techniques were needed

* Lecturer, Department of CSE, Pabna University of Science and Technology, Pabna, Bangladesh.
Email: akkas.buet@gmail.com

for distributed MST algorithms in the message-passing model. Some bear similarities to Boruvka's algorithm for the classical MST problem.

Message Passing Interface (MPI)

Message passing in computer science is a form of communication used in parallel computing, object-oriented programming, and interprocess communication. In this model, processes or objects can send and receive messages (comprising zero or more bytes, complex data structures, or even segments of code) to other processes. By waiting for messages, processes can also synchronise. Message Passing Interface (MPI) is a standardised and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in Fortran 77 or C programming language. Several well-tested and efficient implementations of MPI are free and in the public domain. These fostered the development of a parallel software industry, and there encouraged development of portable and scalable large-scale parallel applications. MPI, the message passing interface standard, defines an API (application programming interface) for message passing in parallel programs. MPI defines both point-to point communication routines, such as sending and receiving between pairs of processes, and collective communication routines that involve a group of processes that need to perform some operations together. Examples include broadcasting data from a root process to other processes and finding the global minimum or maximum of data values on all processes (called a reduction operation). Collective communication routines provide the user with a simple interface for commonly required operations. They also enable an implementation to optimize these operations for the particular machine architecture. As a result, collective communication is widely and frequently used in many applications, and the performance of collective communication routines is often critical to the performance of the overall application.

Boruvka's Algorithm

The version of Boruvka's algorithm (www.cs.princeton.edu) that supports parallel implementation is as follows:

Step1 (choose lightest): The edge list of each vertex is searched to find the minimum weight edge from that vertex.

Step2 (find root): Each vertex finds the root of the tree to which it belongs using the well known pointer jumping algorithm. The input to the algorithm is the set of root vertices, and the input is the set of non-root vertices.

Simple-Pointer-Jumping-Algorithm ( )

Repeat until every vertex in points to a vertex in

for each vertex that does not point to

A vertex in do

Perform a pointer jump on

Step3 (rename vertices): Each processor finds the new name of all vertices listed in its edge lists.

Step4 (merge): The edges of all vertices in a component are sent to the processor that has the edge list of the root. The edge lists are then merged by that processor.

Step5 (cleanup): Each processor executes the sequential algorithm on its own edge lists.

Related Works

The literature on DMST starts by defining algorithms for constructing minimum spanning trees (MST). The first algorithm for finding a DMST was developed in Boruvka and Jistem (1926). Its purpose was an efficient electrical coverage of Bohemia. There are now two algorithms commonly used:

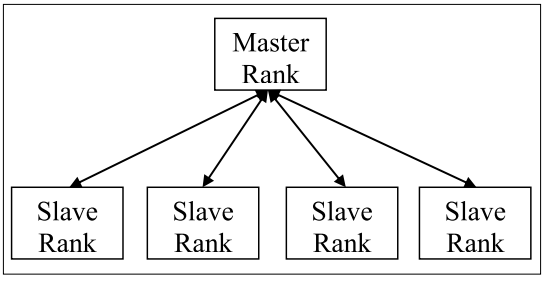
Kruskal's algorithm was developed in Kruskal (1956) and Prim's algorithm was developed in Prim (1957). All two are greedy algorithms that run in polynomial time. But constructing DMST is only a part of the problem. Another important issue is how to allocate the cost associated with the DMST among the agents. Several authors have introduced rules in MST through some algorithms for constructing DMST. The idea is to propose rules to divide the cost among the agents in a fair way1. Bird (1976), Dutta and Kar (2004) [8], and Berganti˜nos and Vidal-Puga (2007) [9] introduce three rules based on Prim's algorithm. Feltkamp, Tijs, and Muto (1994) introduce a rule based on Kruskal's algorithm. The rules introduced by Berganti˜nos and Vidal-Puga (2007) and Feltkamp

et al. (1994) coincide. A proof of this statement can be found, for instance, in Berganti˜nos and Lorenzo-Freire (2008). We call this rule the folk solution, which can be obtained in other ways. Let us mention some of them. A simple mcstp is a MST where the cost of each arc is either 0 or 1. Norde, Moretti, and Tijs (2004) prove that each mcstp can be obtained as a linear combination of simple MST where all the coefficients are non-negative. Thus, we can generate a solution from the set of simple mcstp to the set of all mcstp by using the linear combination. Branzei, Moretti, Norde, and Tijs (2004) and Berganti˜nos and Vidal-Puga (2009) prove that the folk solution can be obtained in this way. Bogomolnaia and Moulin (2008) also apply this approach to DMST for generating several solutions. For a historical survey of the classical MST algorithms and their variants (Berganti˜nos & Vidal-Puga, 2009; Bogomolnaia & Moulin, 2008), Knuth and Moret and Shapiro present empirical assessments of sequential MST algorithms. The work of Barr *et al*. is the only empirical study of parallel implementation of MST algorithms which we found in the literature that is related to ours. For further discussion of related work on parallel models and graph algorithms, the reader is referred to the full version of this paper (Berganti˜nos, & Lorenzo-Freire, 2008).

Methodology and Implementation

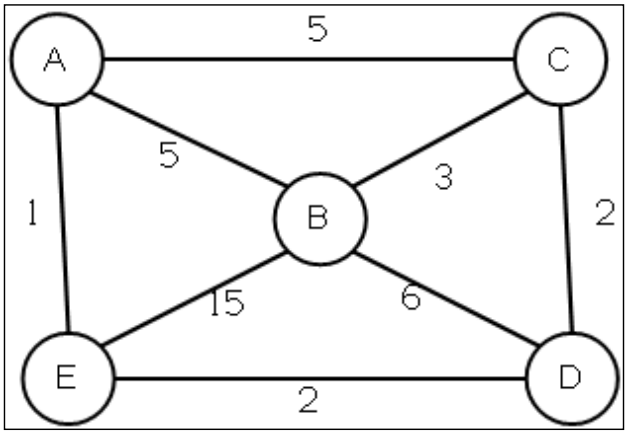
We have used master-slave architecture to implement the Boruvka's algorithm using MPI. We know that if an MPI job is run on several machines/processors; all the machines get a rank number. The rank numbers are 0 to N-1 if there are N machines running in parallel. In our implementation, the rank 0 machine will always work as the master. It will communicate with the other machines, distribute jobs among them and merge the results of their works. Let us have 5 machines/ processors at our disposal and we have run the program. Then the machines will work as shown in Fig. 2.

Fig. 2: Distribution of Machines/Processes



Let, in a particular run, we are trying to find the MST of the graph as shown in Fig. 3.

Fig. 3: Initial graph



At first, the master will create a forest of all the nodes by putting each node in a separate tree. So at first,

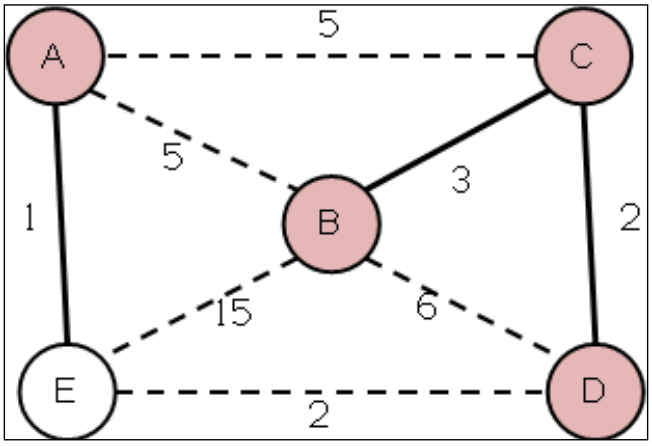
set of trees = {{A}, {B}, {C}, {D}, {E}}.

Now, the master node will look for a slave processor which is free and assign a tree to it. For this particular implementation there are 4 slave nodes and 5 trees. So in the first step only 4 trees will be assigned like below.

(Slave 1: {A}), (Slave 2: {B}), (Slave 3: {C}), (Slave 4: {D})

Each of the slaves will calculate the minimal edge which connects a node from its tree to a node in a separate tree. They will return their results to the master node. Master node will merge the trees to develop a forest with lesser trees as shown in Fig. 4.

Fig. 4: After Merging the Results of First Step

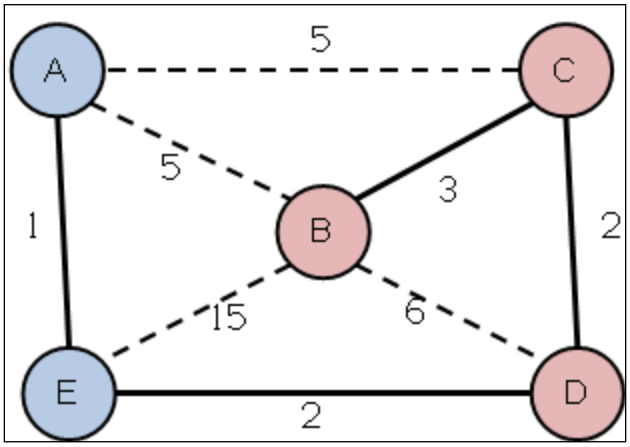


So, after the first step, there will be two individual trees in the forest, {{A, E}, {B, C, D}}. The edges AE (1), BC (3), and CD (2) are added to the result already. Now, the master will again distribute these two trees to two separate machines depending on their availability.

(Slave 1: {A, E}), (Slave 2: {B, C, D})

Now, after Slave 1 returns the result, which will be DE (2), the master will again merge the results and the graph will look like as shown in Fig. 5.

Fig. 5: The MST After Two Iterations



Slave 2 will also return the same result, but as the two trees are already merged after the returned result of Slave 1, this second result will be ignored. As N-1 edges are already found, so the MST is generated. So, now the resulting edge set is:

{AE (1), BC (3), CD (2), and DE (2)}. At this point, the master will send a special message containing only -1 to all the slaves signaling them to stop executing. The master then prints the result and quits too.

Experimental Results

Input graph:

{{INF, 5, 5, INF, 1},

{5, INF, 3, 6, 15},

{5, 3, INF, 2, INF},

{INF, 6, 2, INF, 2},

{1, 15, INF, 2, INF}};

Output generated by our program:

edge 0: source: A destination: E cost: 1

edge 1: source: E destination: D cost: 2

edge 2: source: D destination: C cost: 2

edge 3: source: C destination: B cost: 3

Conclusion

We conclude that our model gives rise to efficient parallel DMST algorithms. The presented algorithms use merging of local MSTs; therefore they could not have been designed under a model that supports only fine-grained parallel computing. Furthermore, the parallel implementations have the advantage that they can cope with large input data.

References

Berganti˜nos, G., & Lorenzo, L. (2008). Non cooperative cost spanning tree problems with budget restrictions. *Naval Research Logistics*, 55(8), 747-757.

Berganti˜nos, G., & Lorenzo-Freire, S. (2008). Optimistic weighted Shapley rules in minimum cost spanning tree problems. *European Journal of Operational Research*, 185(1), 289-298.

Berganti˜nos, G., & Vidal-Puga, J. (2007). A fair rule in minimum cost spanning tree problems. *Journal of Economic Theory*, 137(1), 326-352.

Berganti˜nos, G., & Vidal-Puga, J. (2009). Additivity in minimum cost spanning tree problems. *Journal of Mathematical Economics*, 45(1), 38-42.

Bird, C. G. (1976). On cost allocation for a spanning tree. *A game theoretic approach Networks*, 6, 335-350.

Bogomolnaia, A., Holzman, R., & Moulin, H. (2008). *Sharing the cost of a capacity network. Mimeo Rice University*. Retrieved from http://www.ruf.rice.edu/˜econ/faculty/moulin.html.

Bogomolnaia, A., & Moulin, H. (2008). *Sharing the cost of a minimal cost spanning tree: Beyond the folk solution*. Mimeo. Rice University. Retrieved from http://www.ruf.rice.edu/˜econ/faculty/moulin.html.

Boruvka, O., & Jistem, J. (1926). About a certain minimal problem. *Praca Moravske Prirodovedecke Spolecnosti*, 3, 37-58.

Branzei, R., Moretti, S., Norde, H., & Tijs, S. (2004). The P-value for cost sharing in minimum cost spanning tree situations. *Theory and Decision*, 56(1), 47-61.

Dutta, B., & Kar, A. (2004). Cost monotonicity, consistency and minimum cost spanning tree games. *Games and Economic Behavior*, 48 (2), 223-248.

Feltkamp, V., Tijs, S., & Muto, S. (1994). *On the irreducible core and the equal remaining obligation rule of minimum cost extension problems*. Mimeo (1994a), Tilburg University. Retrieved from http://arno.uvt.nl/show.cgi?fid=3002

Kruskal, L. (1956). *On the shortest spanning subtree of a graph and the traveling salesman problem*. Proceedings of the American Mathematical Society 7, 48-50.

Norde, H., Moretti, S., & Tijs, S. (2004). Minimum cost spanning tree games and population monotonic allocation schemes. *European Journal of Operational Research*, 154(1), 84-97.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell Systems Technology Journal*, 36(6), 1389-1401.

Web References

http://books.google.com.bd/books

http://en.wikipedia.org/wiki/Distributed_minimum_spanning_tree

http://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures/04DemoBoruvka.pdf

Barr, R. S. (1989). Minimal spanning trees: An empirical investigation of parallel algorithms. *Parallel Computing*, 12(1), 45-52.