Minimum Spanning Tree in DAS-5

Kerem Bildiren & Eduard Ruiz

July 26, 2024

1 Introduction

In this assignment, we will be implementing a parallel Minimum Spanning Tree (MST) algorithm. The algorithm will be based on Boruvka's algorithm by using Message Passing Interface (MPI) for interprocess communication. After completing the implementation, we will be doing some benchmark tests.

The spanning tree can be defined as a sub-graph of an undirected graph, which is a tree containing all the vertices from that graph. If there is a missing vertex, we can not call this a spanning tree. The MST is a spanning tree where the total weight of the edges is kept to a minimum.

The MPI is a standardized method that allows communicating numerous processors running concurrent programs across distributed memory. It includes libraries for Fortran, C, and C++.

2 Boruvka's algorithm

Boruvka's algorithm allows measuring Minimum Spanning Trees easily. The idea behind Boruvka's algorithm is to choose the edge with the lowest weight from each vertex (node), then contract each connecting component into a vertex. Firstly, our algorithm will initialize all nodes as spanning trees of one node. While there is more than one component, it will find the closest weight edge that connects this component to a random component and it will add this to MST if not already added. Then it will return MST.

The developed program considers all the edges derived from that spanning tree (considering many nodes can be compressed inside) and find the one with minimum connection. Once the algorithm found the minimum connection, it joins both spanning trees, removing one spanning tree per iteration and keeping only the edges that connect to other spanning trees, having a model which requires $log(n)$ steps. The connection edge between the two spanning trees is stored as part of the MST.

By this method we will be analyzing each spanning tree individually in each iteration until all of them have no more edges connecting to other spanning trees. Once this process is finished, we sum all the connection weights to count the minimum spanning tree connection. Notice that negative values are possible and the resulting weight can be negative.

3 MPI communication

MPI is a stardardized method that allows communicating between different processors. This system allows parallelizing programs by running the same process into different computers. In most cases there is one processor that acts like the master and the rest are slaves, which run the program and send the information to the master and act depending on what the master dictates.

This method results extremely useful for supercomputers like DAS-5 which include multiple processors and allow running a program parallel in up to 20 different computers.

OpenMPI is the library which allows running the processes in parallel, initializing the MPI communication and exchanging data between processors. It also includes the ability to send structs which can include arrays of multiple values.

Algorithm 1 Boruvka's algorithm

Require: N (nodes), E (edges), ST (spanning trees), M_{st} (memory addresses ST)
Ensure: ST (resulting spanning trees)

```
stop ← 0
s ← False
while s ≠ True do
    if (len ST == 1) or (stop == len ST) then
        break
    end if
    st = M_st[0]
    if len st → nodes = 0 then
        continue                                  ▷ Go to the next iteration
        stop+ = 1
    end if
    Find minimum edge from st
    st_a ← Find adding spanning tree
    Join both spanning trees (st and st_a)
    Remove st_a from M_st
    Move st to the back of M_st
end while
ST = ST[M_st]                                     ▷ Index the remaining spanning trees
```

4 Implementation of a parallel program

The generated program is able to perform a MST algorithm into different processors and exchange data between them in order to find the total weight of the resulting spanning trees.

The resulting algorithm has been developed using an object oriented programming approach which include three different classes:

- Node: defines each node in the spanning tree, contains the vertex value (an integer) and a vector of pointers to all the edges that derive from that node (all edges are stored in a vector to save information).
- Edge: all the edges from the spanning tree are stored without duplicates by reading only the upper matrix (since it is symmetric). Each edge has information of the two nodes that connect the edge (references to two nodes objects) and the weight of the connection.
- Spanning tree: it starts with one node per spanning tree and expands by encompassing the other spanning trees. It contains a vector of the nodes included, the edges that connect with the rest of spanning trees and the connection edges, the ones that allowed it to join with other spanning trees.

In order to divide the procedure into different spanning trees is necessary to do data partitioning. Nodes are divided into processors including the same amount of nodes per each processor (independent of the number of edges, which can variate and delay one processor). Each processor includes all the nodes but only spanning trees referring to the nodes for that certain processor, and the same to the edges. By this way, we avoid requiring to read the whole matrix for each processor.

The procedure of data partitioning is done dividing the number of rows (equal to the number of nodes) by the number of processors minus one (since one of them will act as slave). In the case of DAS-5 we will be working with 19 slaves and one master. The following equation defines the number of nodes and, equivalently, initial spanning trees for each processor:

$$n_{nodes} = \frac{n_{rows}}{n_{proc} - 1} \tag{1}$$

The last processor can include a different number of nodes since the division is not always an integer and it has to take until the last row. The procedure is the showed in Figure 1. First we define the first and last rows which are going to be evaluated and we read the matrix to store those edges. Then we

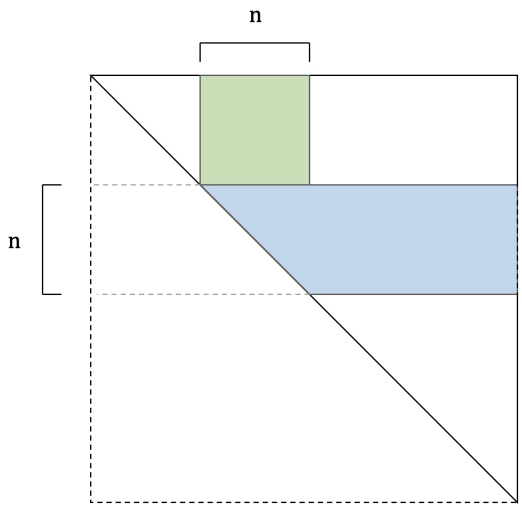

Figure 1: Data partitioning procedure. We store the edges of interest for a processor n delimited by the Equation 1.

do the same for the columns with the condition of not being in one row from inside the range. The resulting procedure reduces the computation requirements of our model and the memory required, essential when working with big sparse matrices with thousands of rows and columns.

The next procedure consists in developing the Boruvka's algorithm. The procedure consists in iterating until we get end with only one spanning tree or all spanning trees have zero remaining edges connecting to other spanning trees. To work with spanning trees and avoid replicating data, a vector containing the memory address for each of them will be generated and operated.

First of all it takes the first spanning tree from the list of memory location (will be called ST1 from here to continued) and finds the minimum edge that connect to ST1. This is done by sorting ascending and taking the first value from the list.

The connection edge is added to ST1 connection edges property list and we find the memory location of the spanning tree that is joining our connection weight. Is necessary to iterate over all spanning trees and look for the node that connects until it find it. We will name this joined spanning tree ST2.

Once we know ST2, a join method is called. This one takes ST2 and copies the nodes into the ST1 vector of nodes. Then removes the edges from ST1 that connect to one of its nodes (notice that ST2 nodes are already included). To do that, previously needs to look if both nodes from the edge are included in the spanning tree nodes. If there is one that not, it will be kept.

The same is needed to be done with the edges from ST2, copying only the ones that connect with other spanning trees (not ST1 or ST2). Finally we copy the connection edges from ST2 into ST1.

In order to parallelize, each spanning tree contains a node which refers to the edge that connect to spanning trees that are not added in that processor (that is why nodes for all columns are created), since refer to another processor. When joining ST1 with ST2, this outer connection is compared and kept the lowest one. It can be possible that one of them do not have an outer connection, in this case we keep the one that have it.

Finally, we remove ST2 from the vector of memory addresses and move the address edited to the back of the vector, to work with the next spanning tree. By this method we ensure that once a spanning tree has edges connecting to others, one spanning tree will be removed in each iteration.

When we work with parallel programming, this algorithm is executed in all the processors. Once they end their `while` loop, they send the resulting data into the master processor, which will be responsible of storing the data and joining the spanning trees generated from all processors. This is done in the same way as the Boruvka's algorithm in each processor.

The connection to other spanning trees from other processors are stored as the edges and the connection edges are saved into a vector. The algorithm works by looking for the edges, which will include a maximum of one per spanning tree and join spanning trees.

The resulting process writes the duration, the spanning trees connections and the total weight of the MST in a `.txt` file.

5 Results

Two different directories are included. One of them works only in one computer and performs MST without any error. The second directory contains the MPI communication and parallelized program, which have some problems when reading data. Results are going to be tested in the one computer program, since the other one is not able of reading data correctly and, perform the task correctly. However, by fixing that bug, the program is potentially usable.

5.1 Small test example

In order to generate the model and check that was performing correctly, the following spanning tree was considered taken from GeeksForGeeks and displayed in Figure 2.

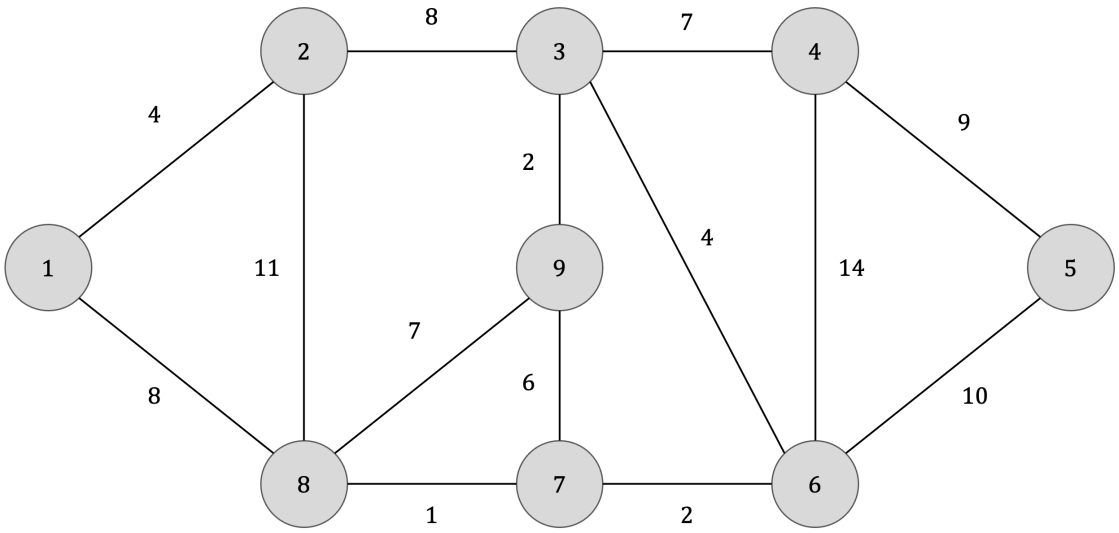

Figure 2: Spanning tree example taken from GeeksForGeeks

The model was executed and the output of the program was as following:

```
Connection between 4--5 of weight 9.00
Connection between 3--4 of weight 7.00
Connection between 3--9 of weight 2.00
Connection between 3--6 of weight 4.00
Connection between 1--2 of weight 4.00
Connection between 1--8 of weight 8.00
Connection between 7--8 of weight 1.00
Connection between 6--7 of weight 2.00
```

This returns an total weight of 37.0 that we can connect the nodes and check that it is the same as expected. Figure 3 show the output result.

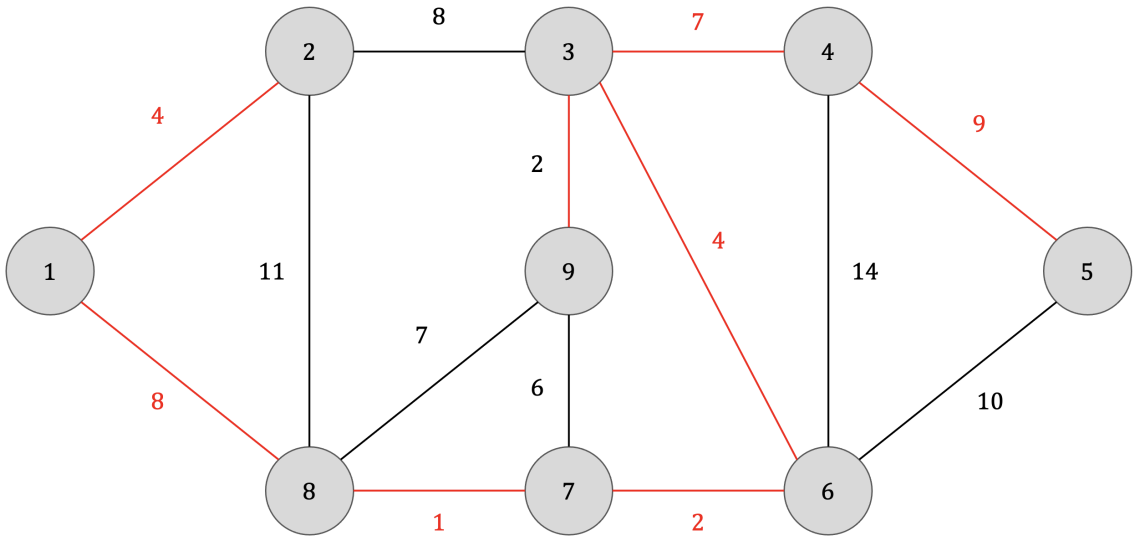

Figure 3: Resulting spanning tree after MST Boruvka's algorithm. Total weight of 37.

5.2 Bigger matrices

Once the algorithm was working for a small matrix, the algorithm was improved to be used into bigger matrices. First of all it was tested with `nopoly` which contains 10,774 rows and columns including 40,808 non-zero value. The result could be obtained in 3.60295s. The total weight of the model was of -21,544.

The model was tested for `mouse_gene` algorithm, which included 45,101 rows and columns with 14,506,196 non-zero values. However, no solution could be obtained since after three minute the model was not even capable of reading the whole matrix. This demonstrates that DAS-5 is necessary to run MST with such big matrices.

6 Problems and future work

The actual program is unfinished and cannot perform the whole MST task in parallel. Some progress need to be make in order to obtain a parallelizable MST algorithm in DAS-5. This tasks are set as future work:

- Receiving arrays properly from MPI: MPI allows sending structs by previously generating your own datatype with `MPI_Type_create_struct()`. This method expects receiving a struct template, the block length and type for each attribute to generate a new datatype. However, it also expects an array of displacements which defines the memory address length of the attribute. When implementing the send-receive communication, individual values could be correctly read but not arrays, which due to the inability to read the memory address correctly, high values were read instead, which do not correspond to the value sent. This bug could not be fixed because of the restricted time to perform the assignment.

- Sending multiple spanning trees from the same processor: once each processor finishes with the Boruvka's algorithm, it needs to send the resulting spanning trees into the master processor (rank 0). This processor will expect a certain amount of data received, which cannot be fixed previous to this study. In order to solve it, `Isend()` and `Irecv()` could be used which are non-blocking. This method would include tags which, once we received all the spanning trees for that processor, will send a confirming message. The master processor will not perform the Boruvka's algorithm itself until he not receive all the spanning trees from all the processors.

- Incorporate into DAS-5: the resulting program needs to be run into DAS-5 which allows parallelizing in 20 processors (19 will act as slaves and one as the master). As seen in the results,

bigger matrices cannot be aproached by a unique processor. During development, the program was only tested with two processors, the maximum allowed by MacBook Pro 2019. This was not a real parallelization since there was a unique slave which performed the whole task and the master which output the results. However, the MPI communication is present and the program was adapted to work into multiple processors, so it is only necessary to test it in DAS-5 computer and fix the possible bugs that can appear.

The main limitation of this project was the time available for performing it. During its development, many problems were faced which needed to be fixed. The most significant one was an error after downloading the MPI compiler. Since another version of the compiler was stored in *Anaconda*, the downloaded compiler was not called when trying to compile the program (instead the *Anaconda* one was used), getting a problem which took more than two entire days to be fixed. Few results were available in the internet facing a similar problem and any solution available for this specific situation.

7 Conclusions

Minimum Spanning Tree algorithm is not a trivial problem, and more when we are trying to work with giant matrices (containing more than 14 million non-zero values) and we want to parallelize the computation. However, considering the time limitation and the previous knowledge of the software working with (no experience with pointers and memory location in C++ as well as for MPI communication), the resulting program represents a close representation of the aimed solution.

The Boruvka's algorithm could be implemented. The program was also adapted to work in parallel in different processors through MPI communication. Information was send from slaves to a master processor, responsible of giving the output. Data itself could not be correctly read due to a problem of arrays memory location. Also the program could not be run in DAS-5 computer.

Nevertheless, the project was nearly finished and future work is required to obtain a complete implementation of a parallelizable MST algorithm in DAS-5.