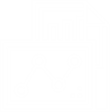
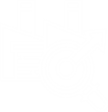
Parallel and Distributed Computing

A picture containing text, envelope

Description automatically generated

­

2021

Minimum spanning trees parallel formulations (Prims algorithm)

## Instructor:

**Sir:** Muhammad Husnain

Team

Awais Zahid ( 18F-0269 )

M. Lal Ahsan ( 18F-0436 )

Abu-Bakar ( 18F-0407 )

MPI/OpenMP/CUDA

Semester Project

Minimum spanning trees parallel formulations (Prims algorithm)

**How Prim's algorithm works**

It falls under a class of algorithms called [greedy algorithms](https://www.programiz.com/dsa/greedy-algorithm) that find the local optimum in the hopes of finding a global optimum.

We start from one vertex and keep adding edges with the lowest weight until we reach our goal.

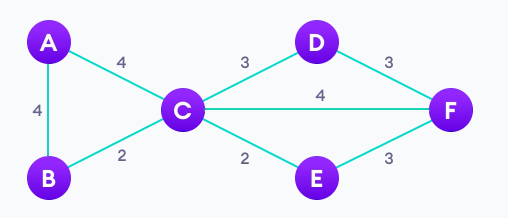
The steps for implementing Prim's algorithm are as follows:

1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
3. Keep repeating step 2 until we get a minimum spanning tree

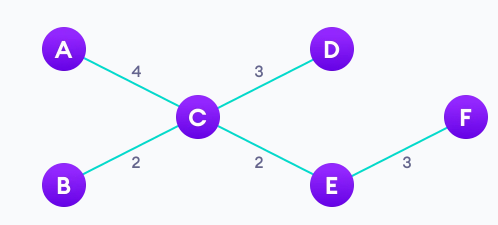
It requires O(n2) time.

Visual Example:

Given Graph



After applying MSP tree on it we get



**What we have to do?**

Algorithms target message passing parallel machine with distributed memory. Primary characteristic of this architecture is that the cost of inter-process communication is high in comparison to cost of computation.

**1) Our goal was to develop algorithms which minimize communication, and to measure the impact of communication on the performance of algorithms.**

**2) Our primary interest were graphs which have significantly larger number of vertices than processors involved in computation.**

**3) Since graphs of this size cannot fit into the memory of a single process, we use a partitioning scheme to divide the input graph among processes.**

First algorithm is a parallelization of Prim’s serial algorithm. Each process is assigned a subset of vertices and in each step of computation, every process finds a candidate minimum-weight edge connecting one of its vertices to MST. The root process collects those candidates and selects one with minimum weight which it adds to MST and broadcasts result to other processes. This step is repeated until every vertex is in MST.

Bulk of the research into parallel MST algorithms has targeted shared memory computers like PRAM, i.e. computers where entire graph can fit into memory. Our algorithms target distributed memory computers and use partitioning scheme to divide the input graph evenly among processors. Because no process contains info about partition of other processes, we designed our algorithms to use predictable communication patterns, and not depend on the properties of input graph

**Parallelization of Prim’s algorithm using MPI**

Thus, parallelization can be achieved in the following way:

1. Partition the input set V into p subsets, such that each subset contains n/p consecutive vertices and their edges, and assign each process a different subset.

Each process also contains part of array d for vertices in its partition.

Let Vi be the subset assigned to process pi, and di part of array d which pi maintains.

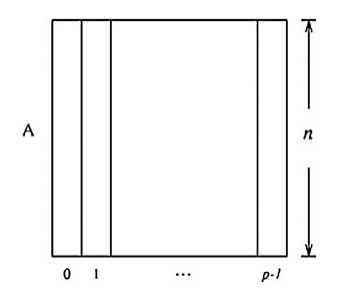


Figure 1.1

Partitioning of adjacency matrix is illustrated in Fig. 1. 1.

2. Every process pi finds minimum-weight edge ei (candidate) connecting MST with a vertex in Vi.

3. Every process pi sends its ei edge to the root process using all-to-one reduction.

4. From the received edges, the root process selects one with a minimum weight (called global minimum-weight edge emin), adds it to MST and broadcasts it to all other processes.

5. Processes mark vertices connected by emin as belonging to MST and update their part of array d. 6. Repeat steps 2–5 until every vertex is in MST.

Time and Complexity of parallel

Finding a minimum-weight edge and updating of di during each iteration costs O(n/p).

Each step also adds a communication cost of all-to-one reduction and all-to-one broadcast. These operations complete in **O(log p).** Combined, cost of one iteration is **O(n/p + log p).** Since there are n iterations,

**total parallel time this algorithm runs in is:**

**Tp = O(n^2/P) + O ( n log p)**

----------------Code file is in the main folder

Results:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Number of processors: 2

Number of vertices: 128

Time of execution: 0.003699

Total Weight: 184

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Number of processors: 4

Number of vertices: 128

Time of execution: 0.019089

Total Weight: 184

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Number of processors: 8

Number of vertices: 128

Time of execution: 0.034099

Total Weight: 184

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

The total weight might be different because the matrix is random

The execution time for two prcessors is less than excecution time for 4 or 8 processors

the execution time for 1 processor is :

Number of processors: 1

Number of vertices: 128

Time of execution: 0.005692

Total Weight: 184

So the speedup for 2 processors is : 0.005692 / 0.003699 = 1.5

So the speedup for 4 processors is : 0.005692 / 0.019089 = 0.3

So the speedup for 8 processors is : 0.005692 / 0.034099 = 0.166

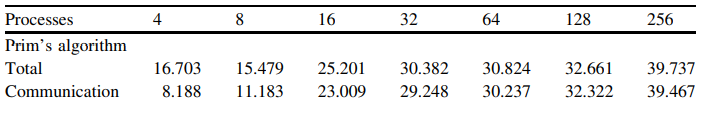
Impact of Communication Overhead

Cost of communication is much greater than the cost of computation, so it is important to analyse the time spent in communication routines.

During tests we measured the time spent waiting for the completion of the communication operations.

In case of Prim’s algorithm, we measured the time that the root process spends waiting for the completion of MPI\_Reduce and MPI\_Bcast operations.

Communication versus computation time (in seconds)



Analysis

when comparing communication time with a total computation time it can be noted that the Prim’s algorithm spends most of time in communication operations, and by increasing number of processes almost all the running time of the algorithm is spent on communication operations. A bottleneck in Prim’s algorithm is the cost of MPI\_Reduce and MPI\_Bcast communication operations. These operations require communication between all processes, and are much more expensive than local computation within each process, because all processes must wait until the operation is completed, or until the data are transmitted over the network. This prevents Prim’s algorithm from achieving substantial speedup of running time with increasing number of processes. Therefore, this algorithm is most efficient on the fewest number of processes that the partitioned input graph can fit.