Parallel External Memory Algorithm for computing minimum spanning forest of a graph

Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Dan Singh Pradhan
(204101020)

under the guidance of

Prof G.Sajith

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, ASSAM

CERTIFICATE

This is to certify that the work contained in this thesis entitled “Parallel External Memory Algorithm for Computing Minimum Spanning Forest of a graph" in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.

Supervisor: **Prof. G. Sajith**

Department of Computer Science & Engineering,

Indian Institute of Technology Guwahati, Assam.

Abstract

Our work is focused on designing Parallel external Memory algorithm for computing Minimum Spanning forest of a graph.Minimum spanning forest is a group of minimum spanning tree for each connected component present in the graph.We will look at why parallel external memory algorithm is considered,what is parallel external memory model,previous parallel and external memory algorithms for solving this problem.We compare the algorithms and design a parallel external memory algorithm for computing Minimum Spanning Forest of a graph.

Contents

Chapter 1

Introduction

1.1 Problem Definition

Given a weighted undirected graph G that is disconnected we have to find out the minimum spanning tree for each connected component of the graph and at last, we would take the union of each of those minimum spanning trees to generate Minimum Spanning Forest of the graph.

1.2 Why parallel external memory algorithm is used for this problem

Here to solve the given problem Parallel external memory algorithm was used instead of a Parallel algorithm because most of the parallel algorithms have an assumption that the data on which the algorithm will be working resides in the main memory of the system. So if the size of the problem is greater than that of the RAM then there would be an extra cost (I/O operation cost)associated which is involved while bringing the data from the disk to the RAM and also storing back the data into the disk after the computation is done on the data. Therefore most of the time is exhausted while completing these I/O operations rather than performing the computations involved in the algorithms. To reduce these I/O cost external memory algorithms were introduced. These algorithms are able to access the disk efficiently and they reduce the I/O cost involved. Parallel external memory algorithms are an extension to these external memory algorithms by introducing parallelism in them. And as some graphs can have a huge number of vertices and edges which may not fit in the main memory so that's why for solving this type of problem we are using a parallel external memory algorithm.

1.3 Parallel External Memory Model

These algorithms would run on a parallel external memory model which is based on a bulk synchronous parallel model. The bulk synchronous parallel model consisted of n virtual

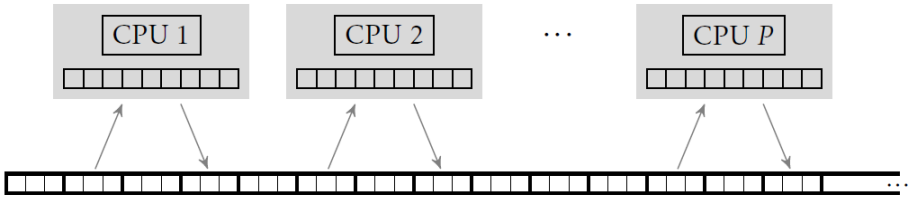
machines with each having its individual cache and all the cache combined size is limited by the main memory of the system. These virtual machines would execute in a series of supersteps consisting of communication superstep and computation superstep. In the communication superstep each of the virtual machines would be doing the work assigned to them initially. In the communication superstep, there will be communication done between different virtual machines in which they will be sending data to other machines or otherwise they will receive data from other machines. In the PEM model these virtual machines(from BSP) would be connected by a shared external memory(Main memory)and both the external memory and individual caches will be divided into blocks of size B . So if a virtual machine has to do computation on some data then the data needs to be present inside its own cache otherwise the data need to be brought from the shared external memory by bringing one block at a time. For communication in PEM model between two different virtual machines, there is no direct communication involved like which is present in the BSP model but it is done through reading or writing in the shared external memory. So if any virtual processor wants to communicate they can generate a message where its size is limited by the cache size and the message will be stored in a buffer inside the shared external memory, So when the destination processor becomes free it can read the message from the buffer present. There are three different variations of the PEM model with each having restrictions on what operations are allowed to be done concurrently on the same block of shared external memory by multiple virtual machines.

1. Exclusive Read/Exclusive Write: In this model, there is no simultaneous access by multiple processors allowed.

2. Concurrent Read/Concurrent Write: Simultaneous Read and write by multiple processors to the same block of main memory are allowed in this model.

3. Concurrent Read/Exclusive Write: In this model Simultaneous reads are allowed but not simultaneous writes.

For solving the given problem we will be considering CREW PEM model. Here in this model the number of block transferred in parallel between the main memory and cache determines the complexity of model,this complexity measure is known as I/O complexity.Given below is a diagram of parallel external memory with M=9 and B=3(here B is the number of word that can fit inside a cache block and M is the number of word that can fit inside a cache)



1.4 Foundation algorithm for Parallel external memory model

1. Scanning($Scan_p(N)$):The most fundamental task of the model is Scanning.Here we apply a function f to each record in the input which will perform some computation and give a correct result in constant time.The I/O complexity involved while scanning n input record is O(N/PB).

2. Sorting($Sort_p(N)$):For sorting N record the I/O complexity involved in CREW PEM is O($(N/PB) * log_d(N/PB)$) by using $O(p * /logB)$ processors.
   Note:- $p* = O(N/(B^2))$

3. Prefix Sum:If a given array is positioned in contigious shared memory,then Prefix sum of that array will be solved by CREW PEM with I/O complexity of O(scan_p(N)+logP).

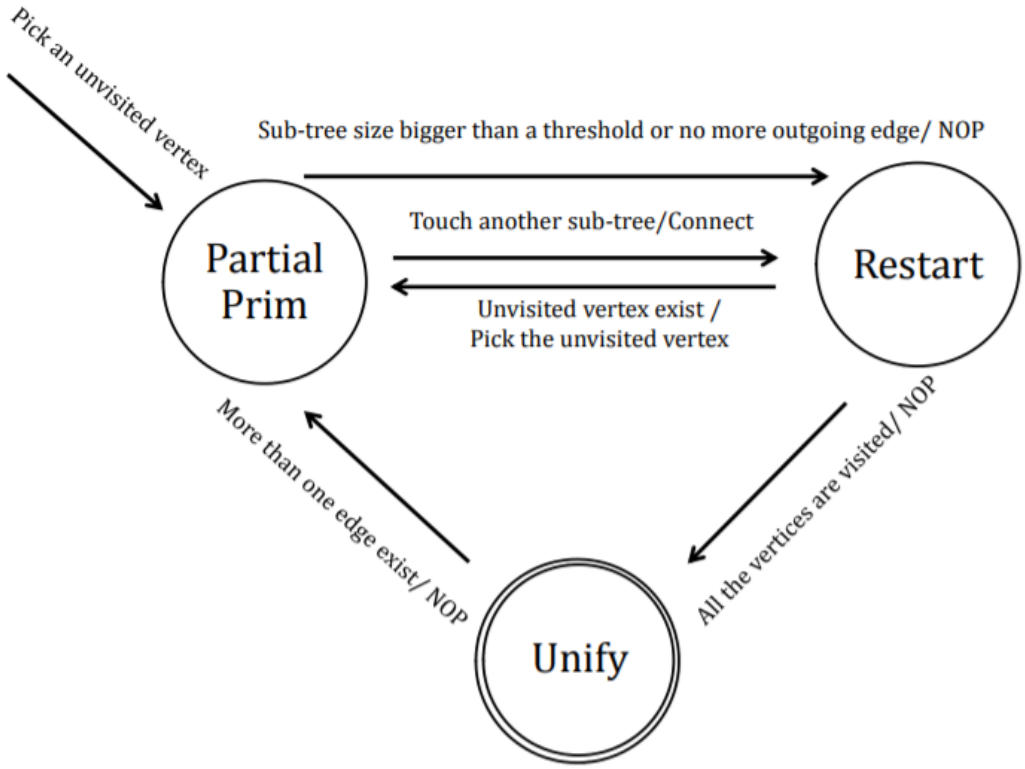
Chapter 2

Review of Prior Works

To solve the given problem different sequential algorithms are there for single processor such as Prims, Kruskal, and Boruvka's algorithm. Also, different parallel MSF algorithms were introduced to solve the problem. These algorithms would follow an approach in which different processor would be forming their own subtree after picking a vertex from the graph and if a conflict occurs such as if one processor while expanding its own subtree discover a vertex belonging to subtree created by another processor. Then both of these processors would have to merge their subtrees. As the merging of the two subtrees would generate huge communication costs. So we'll discuss one of the parallel algorithm called the PMA algorithm for solving the given problem while reducing the communication and synchronization cost which was present in other parallel algorithms.

2.1 PMA algorithm

In the PMA algorithm [2], each of the p processors will be generating their own subtree using the partial prim algorithm in parallel by picking any unvisited vertex from the graph. After all the subtrees are formed then we do unification of all the subtrees produced using partial prim algorithm by each of the P processors done in parallel and removing any self-loop edges or cycle present in the graph. The process is repeated till all the edges are visited and the Minimum spanning forest will be the final output at the end after union of all the subtrees. There may arise conflicts like when there is more than one processor trying to pick the same vertex initially while executing partial prim algorithm. To avoid this all the vertices are divided into P groups(same as no of virtual processors), so that each processor will only pick vertices in its own group.Given below is the state transition diagram of PMA algorithm:-



Algorithm 1 PMA Algorithm

```
Input:G(V, E) : An undirected weighted graph; P : Number of processors;
Output: The MSF of graph G
while |E| > 0 do
    Initialize the successor arrays ;
    Perform Partial Prim on P processors;
    Compact each connected component into a vertex;
end while
```

2.2 Partial Prim

Its execution is nearly similar to that of a single processor Prim's algorithm except that partial prim will stop executing due to certain conditions like once a vertex has been discovered during subtree formation that is part of another process subtree and then both of the processors will stop growing their subtree or if the current running subtree reaches a size γ given as input. The input to the algorithm will be a successor array that is being shared by all the processors and there will be a size γ to define till what size the subtree will grow after Partial prim will have to stop executing. Initially, the Partial prim algorithm will pick an unvisited vertex to grow a subtree from that vertex, and then the vertex is added to the list of processed vertex which were discovered while growing the subtree. For

choosing the next vertex, the minimum weighted outgoing edge found using the list of the processed vertex is selected and from that edge, the next vertex to be processed is selected using the endpoint of the edge. Using successor array we will know whether a vertex has been visited or not. While building the subtree we found a vertex (v) that is already visited then using the successor array we will find out its root vertex i.e root of the component to which v belongs. Then that root vertex will be attached to the current running subtree thus merging the two subtrees and then partial prim algorithm will stop. Here the updation of values in the successor array is done atomically using test set instructions so that no conflict may occur. The Prim's algorithm will stop executing once there are no unvisited outgoing edge present from the list of processed node or once the growing subtree reaches the size γ which given as input initially.

Algorithm 2 Partial Prim

Input: : G(V, E): An undirected weighted graph; successor: The successor array, shared among all processors; γ: Maximum size that a sub-tree can grow ($\gamma \leq 2$)

Output:A part of the MSF of G

```
while there is an unvisited vertex s do
  Atomic successor[s] = s if it is not set;
  if successor[s] was set by another processor then
    Continue;
    Q = {s};
    while |Q| < γ do
      Find the lightest edge e =(u, v) such that u ∈ Q and v ∈ Q;
      if no more edge e then
        Break;
      end if
      Include in the MST
      if successor[v]not set then
        Atomic successor[v] = s;
        Q = Q ⋃ v;
      else
        successor[s] = successor[v];
        Break;
      end if
    end while
  end if
end while
```

2.3 Conclusion

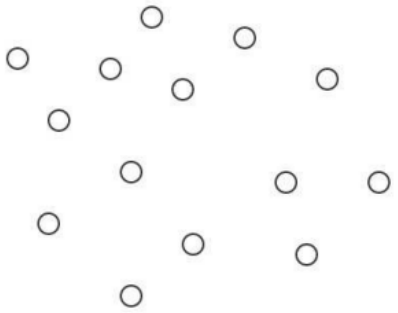
At the end we say that PMA algorithm is better than the previous parallel MSF algorithm as there was no synchronization work or huge communication cost involved while handling

the conflict that were occurring such as while merging the subtrees in previous parallel MSF algorithm.The total complexity of partial prim is O($|E|$) and as in each iteration the number of vertex reduces by half.So the total complexity of PMA algorithm is found to be O($Log(|V|)|E|$).

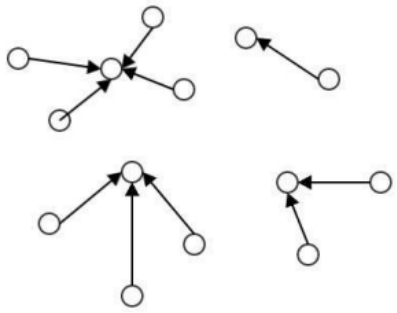
2.4 Phase 2

2.4.1 Boruvka's Algorithm

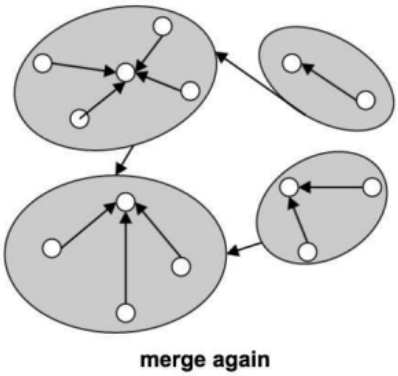
In Boruka's Algorithm initially each of the vertex will be its own component.After that any two components will be merged to form a single component if there is a minimum weighted edge is present between them.The minimum weighted edge is added to the MST solution.Each of these component is represented by a single root vertex so if two components are merged then root vertex of the combined component will be root vertex of the larger component.These steps are repeated till only one vertex remains.As in each iteration while merging the components the vertices are reduced by 2,so the number of iteration will be log(V).The work done in each iteration is to find out the minimum incident edge i.e E.So the total complexity of the algorithm is (E(log(V))).
Given below is a diagram showing how Boruvka's Algorithm works



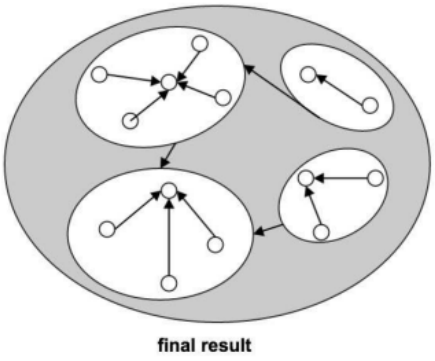
(a) Original Graph



(b) 1st merge



(c) 2nd merge



(d) final result

Algorithm 3 Boruvka's Algorithm

Initialize a forest T to be a set of one-vertex trees, one for each vertex of the graph.
while T has more than one component **do**
  for For each component C of T **do**
    Begin with an empty set of edges S
  end for
  for each vertex v in C **do**
    Find the cheapest edge from v to a vertex outside of C, and add it to S
  end for
end while
T is the minimum spanning tree of G

2.4.2 Parallel Implementation of Boruvka's Algorithm

The basic idea of the algorithm[6] is : Each vertex has a pointer P, initially pointing to itself. We use the pointers to join the vertices in trees, then join these trees if they belong to the same component, and "shorten" the trees by pointer jumping to make them stars. The algorithm uses $|V| + |E|$ processors, one for each vertex i and one for each undirected edge $[i, j]$ of the graph.Below are the steps of Parallel version of Boruvka's Algorithm:-

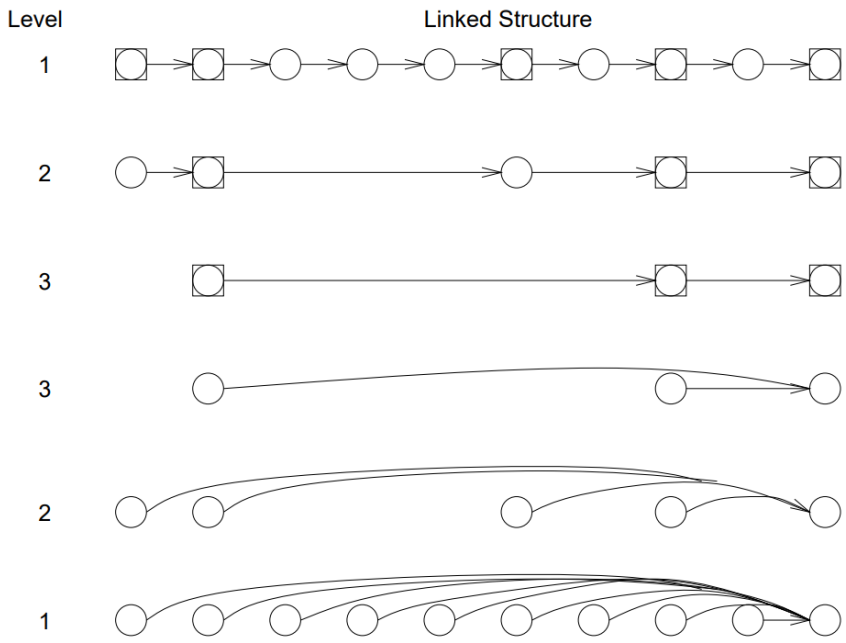
1. Given each vertex $v \in G$ find out the minimum weighted edge incident to it so that those edges will be considered for the MST.After this a forest will be formed in which each of the component will be a pseudo tree .The pseudo-tree will be having exactly a single cycle because two vertex in the same tree can choose the same edge due to it being minimum weighted edge of that tree.

2. Next we have to find out the root of each of those pseudo-trees which can be find out by using Modified Pointer Jumping algorithm(Super-vertex Algorithm).Each component is processed as follows.So given a component we have to choose randomly a number of super-vertex and then apply the pointer jumping algorithm on the remaining vertex.So after a few rounds of pointer jumping each of these non supervertex will be pointing to a supervertex and only supervertex will be remaining.After that we will recursively apply the same procedure on the same supervertices in which we will again randomly choose a new set of supervertices among them.The procedure will repeat till the supervertices points to the root and then each of the vertex will jump by one step to point to the root vertex of a component.

3. After the minimum spanning forest is formed we have to label each vertex according to the component they belong to.So to each of the processor will be assigned a vertex so as to find out its component and accordingly assign label to it.

4. Merge edge list of all the vertex belonging to same component with the root vertex of that component.

5. At last each of the processor will have to remove any duplicated edge found in its edge list.



Given above is the diagram showing how the second step is performed. The first three rows show the linked structure at the start of the three recursive calls. Vertices in squares are chosen to be supervertices at each of these iterations. The last three rows show the vertices that point to the root at the end of each recursive call, starting from the last (third) level of recursion back to the first.

Algorithm 4 Modified Pointer jumping algorithm

```
if |s| > 2 then
  for each vertex x∈ S do
    with probability 1/2, make x a supervertex
    let SV be the set of supervertices, plus the root
    execute Simple-Pointer-Jumping-Algorithm(S-SV,SV)
  end for
  for each vertex x in SV do
    perform one pointer jump on x
    comment: at this point the supervertices form a rooted tree
  end for
  recursively apply the algorithm to
  for each vertex x in S-SV do
    perform a pointer jump on x
  end for
end if
```

2.4.3 Time Complexity of Parallel Implementation of Boruvka's Implementation

The total number of iterations involved for performing the step 2 i.e Modified Pointer jumping algorithm would be $logn$.And the number of synchronisation step involved in each iteration is $loglogn$.So the total complexity for the second step is lognloglogn.The amount of work done for rest of the steps will be linear.

2.4.4 Prerequisites for External Memory Algorithm

The external memory algorithms runs on a model which consist of a single processor connected with an External Memory.
Some of the parameters involved in the model are defined below:-
N=V+E,Where V is the number of vertices and E is the number of edges in a graph.
M = number of vertices/edges that can fit into internal memory
B = number of vertices/edges per disk
The performance metric of an algorithm is the number of I/Os it performs.The number of I/O required to read N contiguous item from disk is Scan(N)=O(N/B) and the number of I/Os required to sort N item is Sort(N)=$N/B\ log_{M/B}N/B$ I/Os.

2.4.5 I/O efficient algorithm for Minimum Spanning Tree

The general idea of the algorithms for solving MST problem for large scale graphs is to reduce the number of vertices from v to E/B using $log(VB/E)$ Boruvka 's Phases. In each phase minimum weighted edge incident to each vertex is found out. After that edge contraction is done in which each of the edges of component as part of forming the MST is reduced to a single super vertex. The supervertex size is the total number of vertex it contains. It will also store the edge list of that component. At the end of each phases duplicate or multiple edges are removed. In this algorithm each of these Boruvka's Phases will be executed in a number of stages. After the vertices are reduced to E/B then it can be fitted into the main memory so we can solve it using any sequential MST algorithm like Prims or Kruskal algorithm.One of the reason that Prim's algorithm cannot work in an external memory model is because to implement Prim's algorithm we require a priority queue to maintain the priority of vertex and it is not possible to fit the priority queue in the main memory.Obtaining current priority of a vertex would require I/O operation so that is why Boruvka's algorithm is preferred over other algorithm for solving Minimum spanning forest problem in external memory model.

2.4.6 External Memory Boruvka Algorithm by Arge

The external memory Boruvka's algorithm proposed by Arge [5] divides the $log|V|B/|E|$ phases into $loglog|V|B/|E|$ Super phases.
So we'll execute these Super phases by implementing algorithm 2 and in which each of the super phase will be executing $\sqrt{S_i}$ times algorithm 1.Algorithm 1 work on a set of edges $e_i \in E$.
Note:- i=Number of the Current phase and $S_i = 2^{3/2^i}$.

Algorithm 5 Algorithm 1

while $|v| \leq |E|$ **do**
for each vertex v$\in$ g **do**
State Find the cheapest edge from v
end for
for each Component c **do**
Find out the root vertex r
end for
for each edge $E \in g$ **do**
Replace (u, v) by $(r(u), r(v))$, where $(r(u), r(v))$ is the edge connecting vertex v to the root vertex r of the component to which it belongs
end for
end while

For finding out the cheapest edge adjacent to a vertex it can be done by sorting the edges in term of edge weight and then scanning the edges,so it would take $Sort|E|$ I/O cost.For selecting a root vertex r for each pseudo-tree can be done by checking if there is cycle present.Then break one edge of that cycle mark the vertex involved in it as root.It can be done by sorting the collected edges by weight and checking if there is a duplicate edge present which can be done by O(sort(V)) I/Os.Replacing the edges of G with the unique representatives is done using a few sorting and scanning operations as described before. Here the entire edge list is sorted, and thus O(sort(E)) I/Os are needed. Total: O(E/B + sort(V) + sort(E)) = O(sort(E)) I/Os.The algorithm 1 would run a total of $log_2|V|B/|E|$ iterations because in each iteration the number of vertex gets reduced by atleast half.So from these observations the total complexity of Algorithm 1 is O($log_2|V|B/|E|$Sort(E)) I/Os.

Algorithm 6 Algorithm 2

Phase i

1: **for** each available node v **do**
2: Find out d adjacent minimum weighted edges to v,Where $d = \sqrt{S_i}$ and $S_i = 2^{3/2^i}$
3: **end for**
4: Graph G_i is induced with the selected edges where $G_i = (V_i, E_i)$
5: **for** i =1 to log d **do**
6: Apply Algorithm 1 on G_i
7: **end for**
8: **for** each edge $E \in G_i$ **do**
9: Replace (u, v) by $(r(u), r(v))$, where $(r(u), r(v))$ is the edge connecting vertex v to the root vertex r of the component to which it belongs.
10: **end for**
11: Remove isolated nodes, parallel edges and self loops

Since we want to reduce more number of edges per phase that is why in algorithm 2 we will take $\sqrt{S_i}$ adjacent edges for each available vertex ,so that per phase i the edges reduced is going to be $\sqrt{S_i}$.

For finding out d adjacent edges would take O($Sort|E|$) I/Os.Similarly replacing each edge (u, v) by $(r(u), r(v))$ would also take O($Sort|E|$) I/Os.Since one phase of Algorithm 1 takes (Sort(E_i)) I/Os then $\sqrt{S_i}$ phases of Algorithm 1 will take O($Sort|V|$) I/Os.So the total complexity for 1 phase for Algorithm 2 will be $O(Sort|E|)$ I/OS.The total number of phases required to reduce the number of vertex to E/B will be $loglog|V|B/|E|$ phases,so the total complexity of the above algorithm will $Sort|E|loglog|V|B/|E|$ I/Os.

2.4.7 Comparison between external memory Boruvka and Parallel Boruvka Algorithm

Since Parallel Boruvka algorithm is a parallel implementation of sequential boruvka algorithm where each of the steps in sequential boruvka's algorithm is parallelised for faster implementation.And in external-memory algorithm the problem size is too big so therefore we are reducing the number of vertices to E/B so that its size can fit into the main memory so that we can apply sequential MST algorithm on the reduced graph.The number of iteration in Parallel boruvka's and Sequential boruvka algorithm will be exactly same but in external memory since the graph is too big so the number of rounds involved is divided into a number of phases and each of these phases will be divided into a number of super-phases.In parallel Boruvka's algorithm we are reducing the number of vertices by half per iterations whereas in External memory boruvka's algorithm for efficient implementation we will be reducing $\sqrt{S_i}$ edges per phase.

Chapter 3

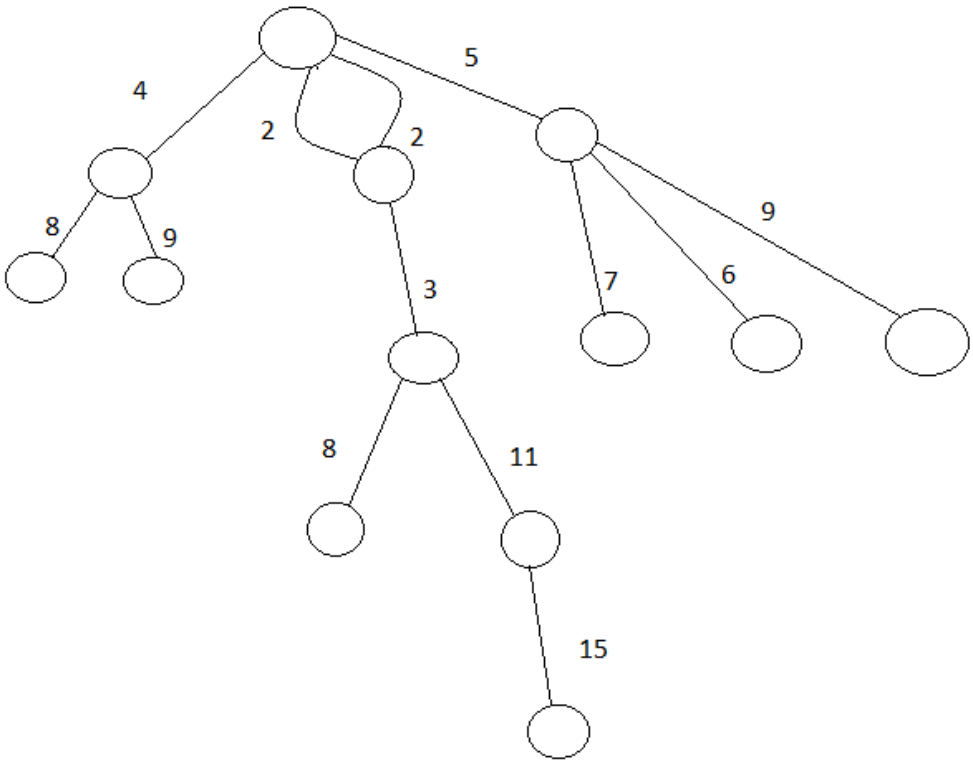
Parallel External Memory Algorithm for Computing Minimum spanning forest of a graph

3.1 Algorithm

The main objective of the algorithm is to reduce the number of vertices V to M which is the size of the internal memory so that it can be solved easily by a sequential MSF Algorithm. So we are executing the algorithm in a number of rounds in which the number of vertex in each round get reduced by V/M.In each of the round we will create a forest induced by choosing minimum weighted vertex incident to each vertex. After that we will choose a root vertex for each component of the forest and then we will compress each vertex of that component into a single vertex which will contain all the information(ex:-individual adjacency list for each of the vertex in that component) of that component. The super vertex found in current round will be further used as input for the next round and so in it will repeat until a single vertex remains. Below are the steps of the algorithm:- Let the final Minimum Spanning forest be represented as F.Initialize as F=ϕ.Given below are the steps of the algorithm:-

1. .Remove any isolated vertex(without no adjacent edge) if present and add it to F.

2. For each vertex compute the minimum adjacent edge for it. After that let the forest induced be H'.
   Explanation for step 2:- Lightest incident edges can be collected in O(E/B) I/O s in a simple scan of the edge-list representation of G. Before that we have to sort the incident edge for each vertex which can be done by O(Sort(V)) I/O. Below diagram is one of the component of H'. Each of the component form a tree in which there will be a single cycle of size 2.The cycle will be formed using the root vertex as it contain the minimum weighted edge. The edge weights will be increasing while going from the root to the leaf of the tree.
   Given below is a digram showing one of the component of the forest H'



3. For finding out roots for each of the component check if there is cycle present. Then break one edge of that cycle mark the vertex involved in it as root.
   Explanation of step 3:- Detection of cycle can be done using single O(Sortp(N)) and O(Scanp(N)).Sort the collected edges by weight and check if there is a duplicate edge present. Then remove one of the edge to break the cycle.

4. Find Euler tour for graph H' using each of the root vertex find in the previous step.
   Explanation of step 4:- To compute Euler tour U replace each of the undirected edge with two directed edge(Incoming edge and outgoing edge).Then lexicographically sort the directed edges to order the edges and scan the edges assigning appropriate successor to each incoming edge. After that Euler tour will be formed .For each root node r of H', delete from U the first edge with r as the source. The result will be collection of disjoint linked list with each root vertex r in end of each of the linked list.

5. .Apply list ranking on each of these disjoint linked list. Before that make the rank of each vertex r which has no successor edge as r and other as 0.After this Star graph will be formed in which each of the vertex in a component will be pointing to root vertex r for that component.
   Explanation of Step 5:-Given below is the algorithm showing how the list ranking is going to performed on a list to form a star pointer.

   Initially, we assign each node rank of 1. This requires a single scanp(N) = O(N/pB)

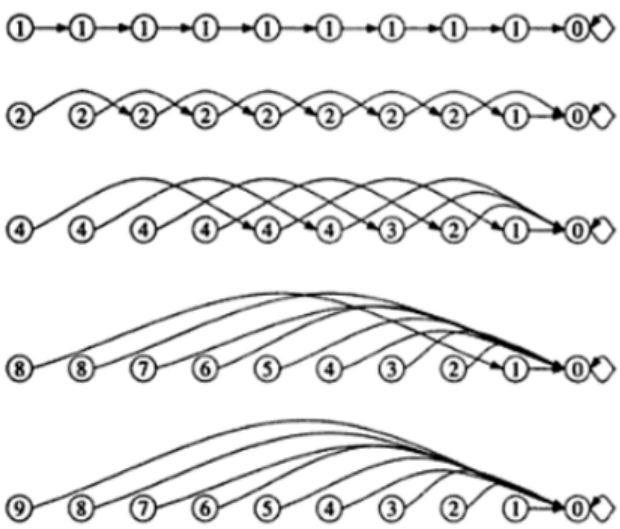
Algorithm 7 List Ranking

INPUT:L

```
for all P_i for each node i do
  if i− >next==NULL then
    i.d=0
  else
    i.d = 1
  end if
  while i− >next != null do
    i.d = i.d + i− > next.d
    i− > next = i− > next− > next
  end while
end for
```

I/Os.

Given below is the diagram showing how the star graph is formed.



6. Each connected component of H' forms a super vertex. Its root shall be its representative. Thus, we have a star graph for each super vertex. The size of a super vertex is the number of vertices it contains. To solve this in a PEM algorithm would require O(Sortp(N)).

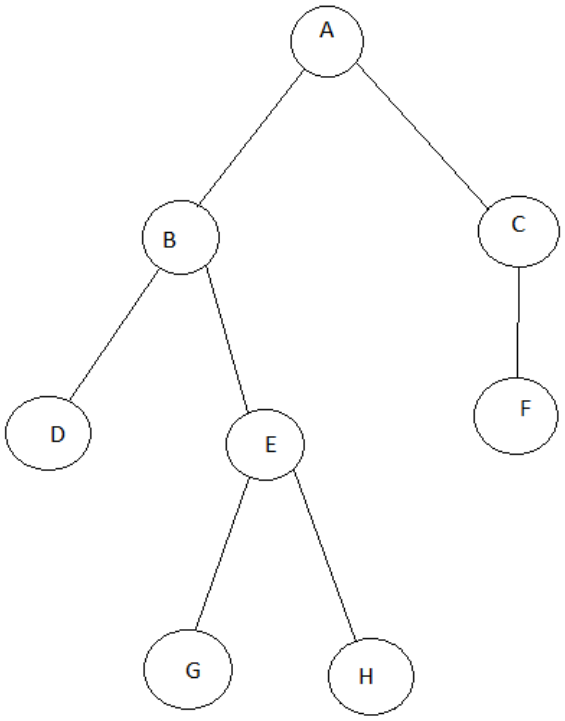
7. Remove any internal edge or multiple edge present in the edge list of each of the super vertex.
   Explanation:-A single O(1)PEM scan on the edge list remove any self loop or multiple edges present in the graph

8. For each super vertex copy all of its edges from edge list into F. The super vertex formed in this round will be used as input for the next round.
   Explanation:-O(1) PEM Scan is required to copy into F.

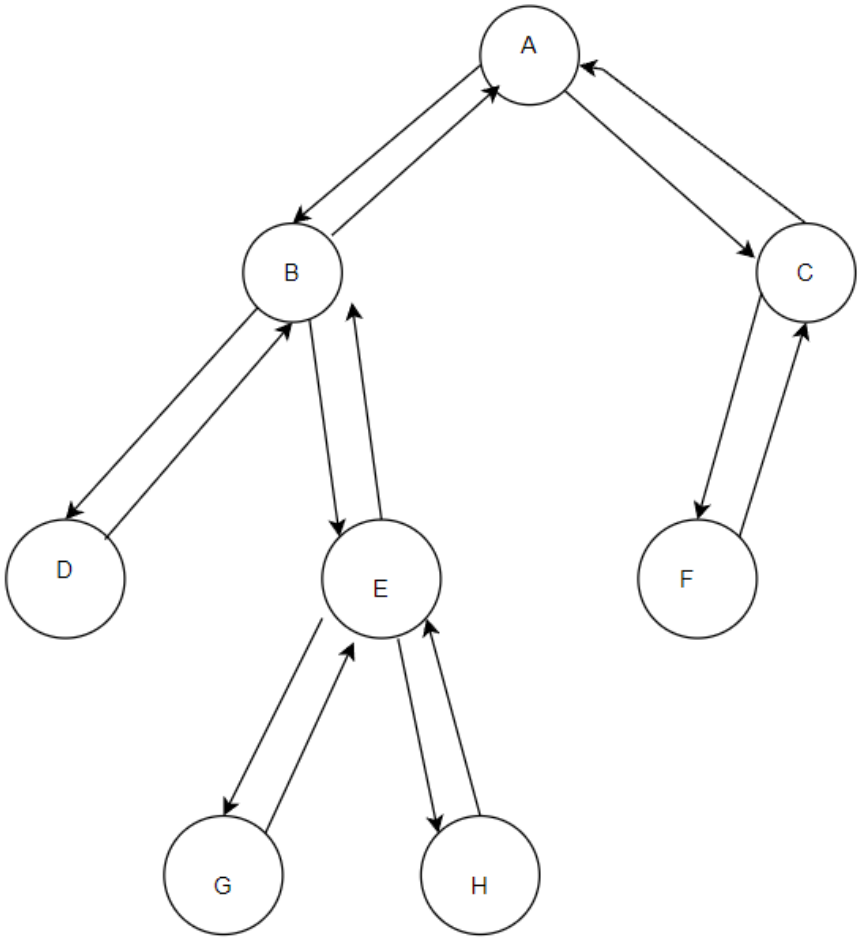
9. Repeat the above steps till only one vertex remains

3.2 Example of How the Star graph is formed from a rooted tree

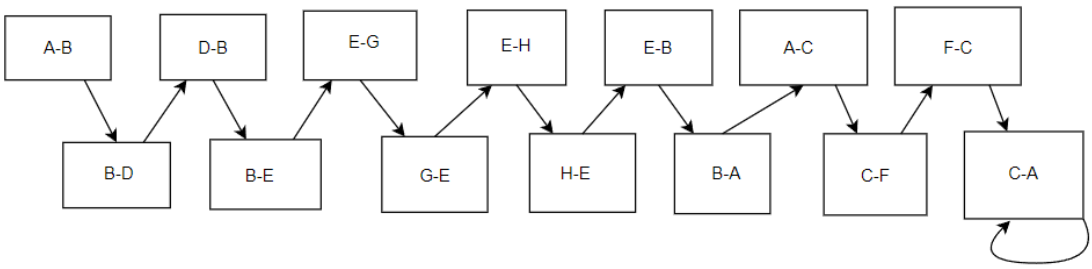
1. Given below is a rooted tree formed at A.



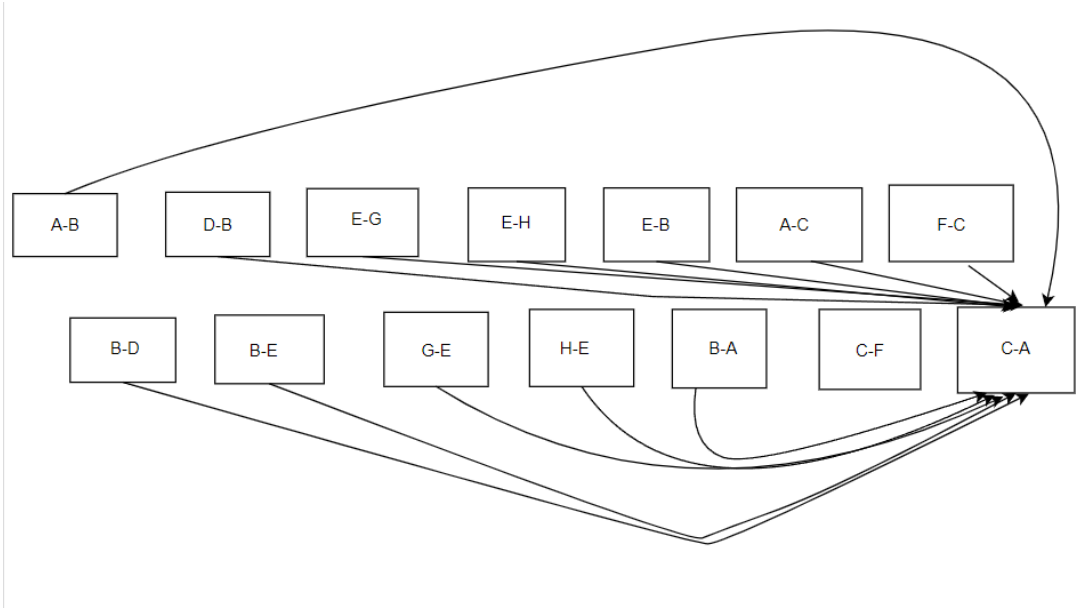
2. Given below is an euler tour formed at that tree after replacing each of the undirected with two directed edge i.e Incoming and outgoing edge.



3. Build a linked list out of directed tree edges formed from the Euler Tour.



4. Form a star graph in which each of the node in the linked points to the end node of the linked list.



3.3 Analysis of Time Complexity

For computing the Euler tour [4] of the graph,it would require ordering of edges such that in order to find the appropriate successor edge for it.For ordering of edges it would require lexicographical sorting which would take O(sort$_p$(E)) and scanning of appropriate successor would take O(scan$_p$(E).So total complexity of that operation would be O(sort$_p$(E))+O(scan$_p$(E))=O(sort$_p$(E)).And List ranking [1] of List of length of size N by PEM algorithm takes I/O complexity of O(sort$_p$(v).Rest of the steps can be done using O(1) PEM scan and sort .So for each iteration it would take O(sort$_p$(V+E)) I/O complexity. As each iteration reduces number of vertices by atleast half of the number of vertices.So the recurrence relation formed is T(G(V,E))=T(G(V/2,E))+O(sort$_p$(E+V)).Using the above recurrence relation,the final complexity of the algorithm would be O(sort$_p$(V)+Log(V) sort$_p$(E)).

3.4 Space Complexity

We are assuming that every graph problem instance has size of atmost O(N/B^2logB) [4] which can be solved by multiple processor collectively at any instance.So therefore we are taking the number of processor as O(N/B^2logB).Here N is the size of the input and B is the number of words that can fit inside a proccessor cache block.

Chapter 4

Conclusion

We have studied what is parallel external memory model and designed a parallel external memory algorithm for computing Minimum Spanning forest of a disconnected weighted graph. We found that the problem can be solved in I/O complexity of O(sort_p(V)+LogV sort_p(E)) using O(N/B^2LogB) number of processors.As Parallel external memory algorithm are parallel extension of sequential external memory algorithm so it provide a speedup of O(P) over external memory algorithms.Minimum Spanning forest has various applications like network design for telephone networks,image processing and in clustering algorithm.We will look at one of the use case i.e network design for telephone networks.

4.1 Use Case

For example if a telephone company wants to create a network in a locality by laying out telephone wire in that neighbourhood.Suppose that locality is represented as a graph where vertices are representing the houses and weighted edges for cost of laying out wire between any two house.If we want to create network which connect all the houses with minimum cost then we need to compute minimum spanning tree of that graph.If communication between different locality is required then we need to compute the minimum spanning forest.

Chapter 5

References

1. Lieber and Tobias.On Optimal Algorithms for List Ranking in the Parallel External, Memory Model with Applications to Treewidth and other Elementary Graph Problems.,2014.

2. J. J´aJ´a. An Introduction to Parallel Algorithms. Addison-Wesley, Reading, Mass., 1992.

3. Sadegh Nobari,Than Tung Cao,Stephane Bressane.Scalable Minimum Spanning Forest Computation.,2012.

4. Parallel External Memory Graph Algorithms.Lars Arge,Michael T. Goodrich,Nodari Sitchinava.,2010.

5. Arge, L., Brodal, G.S., Toma, L.: On external-memory MST, SSSP, and muti-way planar graph separation. J. Algorithms 53, 186–206 (2004)

6. Sun Chung ,Anne Condo,Parallel Implementation of Boruvka's Minimum Spanning Tree Algorithm.1996.

7. An I/O Efficient Algorithm for Minimum Spanning Trees,Alka Bhushan,Gopalan Sajith.

8. Fundamental Parallel Algorithms for Private-Cache Chip Multiprocessors,Lars Arge,Michael T. Goodrich,Nodari Sitchinava,Micheal Nelson,2008.