A New Parallel Algorithm for Minimum Spanning Tree(MST)

Alok Ranjan Tripathy
Department of Computer Science & Engineering
College of Engineering Bhubaneswar
Bhubaneswar, India

B. N. B. Ray
Department of Computer Science & Application
Utkal University, Vani Vihar
Bhubaneswar, India

***Abstract*— This paper proposes a new algorithm for finding minimum spanning tree (MST) of a weighted graph which is intrinsically parallel, unlike Kruskal's algorithm. As far as the running time of the algorithm is concerned, it is faster than both Kruskal's and Boruvka's algorithms. However the Prim's algorithm is bit faster than the proposed algorithm. The parallel implementation of the proposed algorithm in shared memory is also better than Boruvka's parallel algorithm, but comparable with Prim's parallel algorithm.**

Keywords- Minimum Spanning Tree; Parallel Algorithm;

I. INTRODUCTION

The minimum spanning tree problem (MST) for a weighted graph is defined as follows:

Given a directed connected graph $G = (V, E, w)$. V : Set of vertices, E : Set of edges and $w: V \times V \to R$ weight function. The minimum spanning tree problem is to find a spanning tree with minimum total weight. The MST problem has applications in combinatorial problems with practical applications in VLSI layout, wireless communication, medical imaging [8], chemical warfare [2], graph steiner tree problem and many other graph theoretical applications [6, 7, 11]. There exist serial and parallel algorithms for MST. The earliest serial algorithm for finding MST was due to Boruvka [8]. This algorithm has parallel structure which can be parallelized easily. Other two frequently used MST are Kruskal algorithm and Prim's algorithm [4]. They are difficult to parallelize. Because of intrinsic parallel nature many parallel formulation of Boruvka's are available in literature.

These include Chung *et al.* [10] and Chong *et al.* [3]. Recently a hybrid approach of Prim's and Boruvka was used by Bader *et al.* in [5]. In [1,5] many parallel variants of Prim's algorithm are discussed.

The Parallel MST due to R. Setia *et al.* [9] was implemented using threads and posix signal. They used cut property of the graph to grow the tree by multiple threads and finally trees were merged when they collide. The parallel formulation of Boruvka uses a pointer jumping algorithm [10] for forming connected components which is bit complex. Bader et al. [5] implemented parallel Prim's algorithm in shared memory using multiple cores. They used flexible adjacency list data structure which is simpler than pointer jumping algorithm of Boruvka's algorithm discusses in [10]. From this study it appears that there is a gap between the runtime of Boruvka's and Prim's algorithm. Boruvka's algorithm has more parallel structure than Prim's algorithm at the cost of slower runtimes.

However for large scale graphs MST problem parallel implementations is an obvious choice. Thus there is a need to develop parallel MST algorithms having intrinsic parallel structure like Boruvka's algorithm and runtimes comparable with Prim's algorithm. To this end a new MST is proposed, which is faster than both Kruskal's and Boruvka's algorithm but bit slower than Prim's algorithm. In short our contributions are as follows:

- Developed a new sequential algorithm for solving MST problem of a graph.
- Parallelized the new algorithm in shared memory architecture.
- Compared the Running Time (RT) of new sequential algorithm with sequential Prim's and Kruskal's and Boruvka's algorithms for both sparse and dense graphs. The runtime of the proposed algorithm is better than both Krushkal's and Boruvka's algorithms but higher than Prim's algorithm. However proposed algorithm has more parallel structure than Prim's algorithm and is amenable for achieving high degree of parallelism for large scale weighted graphs.
- Studied the runtime of new sequential parallel algorithm.

The rest of the paper is organized as follows:

Section II reviews the necessary back ground. Section III presents our proposed sequential and parallel MST algorithms. Section IV discusses the parallel formulation of the new algorithm. Finally conclusions are offered in Section V.

II. BACKGROUND

This section present some commonly used sequential MST algorithms for weighted graphs.

A. *Boruvka's Algorithm* [10]

This algorithm also known as Sollin's algorithm which constructs a minimum spanning tree iteratively using following steps:

Algorithm 1

Step 1 (Choose lightest): Each vertex selects the edge with the lightest weight incident on it. Each of the connected components thus created has one cycle of size two between two vertices that each selects the same edge. Of this pain, the one with the smaller number is designated as the root of the component and the cycle is removed. The component is then a tree.

Step 2 (Find root): Each vertex identifies the root of the tree to which it belongs.

Step 3 (Rename vertices): In the edge lists, each vertex is renamed with the name of the root of the component to which it belongs.

Step 4 (Merge edge lists): Edge lists, which belong to the same component, are merged into the edge list of the root. In other words, each connected component shrinks into a single vertex.

Step 5 (Clean up): Now the edge lists may have self loops and multiple edges. All self loops are removed. Multiple edges are removed such that only lightest edge remains between a pair of vertices.

In Boruvka's algorithm the ith iteration is an input to (i+1)st iteration, unless it has just one vertex in which case it halts. The output spanning tree is the union of the set of edges selected in Step 1, taken over all iterations. The components formed in all iteration can be implemented using depth first search in $O(V+E)$ time. As the number of vertices at the (i+1)st iteration is at most half of the number of vertices at the ith iteration. The number of iterations of Boruvka's algorithm is atmost $\log_2 n$. Thus the total running time of Boruvka's algorithm is $O(E\log V)+O(E+V)=O(E\log V)$. Its parallel implementation in shared memory with p processor is given by $O\left(\frac{E\lg V}{p}\right)$ [1].

B. *Prim's Algorithm* [4]

Prim's algorithm grows minimum spanning tree of a weighted graph $G=(V,E,w)$ from an arbitrary vertex 'r' as root. Following notations are used in Prim's algorithm.

For $u\in V$, with $key[u]$: key associated with u.

$\pi[u]$: Key associated with parent of u.

Q: Priority Queue keyed with key value $key[u]$, $\forall\ u\in V$.

$adj[u]$: Adjacent list of u.

$w(u,v)$: Weight of the edge $(u,v)\in E$.

The steps of the Prim's algorithm are as follows.

Algorithm 2

MST-PRIM (G,w,r)

1. For each $u\in V[G]$
2. do $key[u]\leftarrow\infty$
3. $\pi[u]\leftarrow NIL$
4. $key[r]\leftarrow 0$
5. $Q\leftarrow V[G]$
6. while $Q\neq\phi$
7. do $u\leftarrow$ EXTRACT-MIN(Q)
8. for each $v\in Adj[u]$
9. do if $v\in Q$ and $w(u,v)<key[v]$
10. then $\pi[v]\leftarrow u$
11. $key[v]\leftarrow w(u,v)$

The running time of the Prim's algorithm is $O(V\log_2 V+E\log_2 V)$ and $O(E+V\log_2 V)$ using Binary and Fibonacci heaps respectively. Recently Bader *et al*. [1] gave a parallel Prim's algorithm in shared memory with p processors, whose running time is $O\left(\frac{E\log V}{p}\right)$. Though this parallel runtime is asymptotically same with Boruvka's parallel runtime, but experimentally using a new data structure call flexible adjacent list (FAL), they found that the performance of Prim's using FAL was better than Boruvka,s parallel algorithm, where as parallel Boruvka discussed in [1] uses adjacency list representation of the graph.

C. *Kruskal's Algorithm* [4]

It finds a safe edge to add to the growing forest by finding, of all the edges that connect any two trees in the forest, an edge (u,v) of least weight. It uses a disjoint-set data structure to maintain several disjoint sets of elements. Each set contains the vertices in a tree of the current forest. The operation FIND-SET(u) returns a representative element from the set that contains u. Thus, we can determine whether two vertices u and v belong to the same tree by testing whether FIND-SET(u) equals FIND-SET(v). The combining of trees is accomplished by the UNION procedure.

Algorithm 3

MST-KRUSKAL (G,w)

1. $A\leftarrow\phi$

2. for each vertex $v \in V[G]$
3. do MAKE-SET (v)
4. Sort the edges of E into nondecreasing order by weight w
5. for each edge $(u,v) \in E$, taken in nondecreasing order by weight
6. do if FIND-SET$(u) \neq$ FIND-SET(v)
7. then $A \leftarrow A \cup \{(u,v)\}$
8. UNION (u,v)
9. return A

The running time of Kruskal's algorithm is $O(E \log_2 E)$.

III. Proposed Minimum Spanning Tree (MST) Algorithm.

In this section we present a new sequential algorithm for finding the minimum spanning tree of a connected graph $G = (V, E, w)$. Then we discuss its runtime with the runtimes of Kruskal's, Prim's and Boruvka's algorithms

Algorithm 4

1. The proposed algorithm takes the input graph G and keeps on applying Boruvka's algorithm's Step 1 through Step 3 recursively till the number of super vertices reduces to a threshold value $n_0 \geq 2$.
2. For $n_0 \geq 2$, let $G' = (V', E')$ be the multi graph resulting from Step 1, Where $(|V'| = n_0)$. Then apply sequential Prim's algorithm to G' to get MST T' of G. Note that the vertices of T' are super vertices of G.
3. In order to obtain the MST of G, just uncluster the vertices of T' until their sizes reduces to one.

Since in the proposed algorithm Step 1 and Step 2 are called recursively and they are independent steps of Brouvaka's algorithm, so they can be easily parallelized in multiprocessor system whereas the parallelization of Step 3 is of no interest as it is executed only once on limited number of super vertices of multi graph in the life time of the algorithm. Thus unlike classical Prim's and Kruskal's algorithms, our sequential algorithm has more parallel structure and amenable for the development of parallel algorithms on various interconnection networks.

A. Running Time

We know for graph $G = (V, E, w)$, the Boruvka,s algorithm in each step reduces the number of vertices to half. After Step 1 the number of vertices of the resulting graph is $V/2$, and finding light edges across V - vertices inspects the adjacent list of V.

The cost incurred at this step $= \sum_{v \in V} \deg(v) = O(2E) = O(E)$.

In Step 2 the number of vertices reduces to $V/2^2$, and the cost for finding light edges across $V/2$ vertices is $O(E)$. If we stops at k-th step, then the number of vertices at this step is at most $V/2^k = n_0$ and the cost for finding light edges is also $O(E)$.

Now $V/2^k = n_0$

$\Rightarrow 2^k = V/n_0$

$\Rightarrow k = \log_2(V/n_0)$

In the above algorithm one can obtain super vertices for G invoking connected component algorithm i.e. depth first search, in $O(V+E)$ time. As $O(E \log(V/n_0))$ dominate $O(V+E)$, thus the running time for obtaining multigraph $G' = (V', E')$ of step 2 is $O(E \log(V/n_0)) + O(V+E)$

$= O(E \log_2(V/n_0))$ (1.1)

Then running Prim's algorithm on G' to get MST T' using Fibonacci heap at final phase of Step 2 takes $O(E' + V' \log V')$ time. (1.2)

Adding (1.1) and (1.2), the total running time of the proposed algorithm is

$O(E \log_2(V/n_0) + E' + V' \log V')$ (1.3)

Theorem 1

If RT(Krusksl's), RT(Boruvka's), RT(Prim's) and RT(Proposed) be runtimes of Kruskal's, Boruvka's, Prim's and Proposed algorithms. Then

i) For dense graph
RT(Prim's) ≤ RT(Proposed) ≤ RT(Boruvka's) ≤ RT(Krusksl's).

ii) For sparse graph
RT(Krusksl's) ≤ RT(Boruvka's) ≤ RT(Proposed) ≤ RT(Prim's).

Proof

For graph $G = (V, E, w)$

RT(Prim's) = $O(E + V \log_2 V)$

RT(Krusksl's) = $O(E \log_2 E)$

RT(Boruvka's) = $O(E \log V)$ and

RT(Proposed) = $O(E \log_2(V/n_0) + E' + V' \log V')$.

Without loss of generality let us assume $n_0 = V/2$

$\Rightarrow V' = V - n_0 = V/2$

Thus RT(Proposed) = $O(E + E' + (V/2) \log_2(V/2))$.

i) For dense graph $E \approx V^2$

Thus RT(Prim's) = $O(V^2 + V \log_2 V)$

RT(Krusksl's) = $O(2V^2 \log_2 V)$

RT(Boruvka's) = $O(V^2 \log V)$

RT(Proposed) = $O(V^2 \log_2 (V/n_0) + V^{'2} + V^{'} \log V^{'})$

Clearly RT(Prim's) ≤ RT(Boruvka's) ≤ RT(Krusksl's) (1.4)

Note that the second part of inequality becomes equality is asymptotic sense if we take finding light edge of Boruvka incurred cost $O(E) = 2E = 2V^2$. In the proposed algorithm up to n_0 - super vertices Prim's algorithm is used.

Thus RT(Boruvka's) = RT(Boruvka's till number of super vertices equals n_0) + RT(Boruvka's on $V - n_0$ vertices) ≥ RT(Boruvka till number of super vertices equals n_0) + RT(Prim's on $V - n_0$ super vertices) ($\because$ RT(Prim's) ≤ RT(Boruvka's))

≥ RT(Prim's on n_0 vertices) + RT(Prim's on $V - n_0$ vertices) = RT(Prim's)

$\therefore$ RT(Boruvka's) ≥ RT(Proposed) ≥ RT(Prim's) (1.5)

From (1.4) and (1.5) we have

RT(Prim's) ≤ RT(Proposed) ≤ RT(Boruvka's) ≤ RT(Krusksl's)

Hence prove.

ii) For sparse graph $E \approx V$

Thus RT(Prim's) = $O(V + V \log_2 V)$

RT(Krusksl's) = $O(V \log_2 V)$

RT(Boruvka's) = $O(V \log V)$ and

RT(Proposed) = $O(V \log_2 (V/n_0) + V^{'} + V^{'} \log V^{'})$.

Clearly RT(Krusksl's) ≤ RT(Boruvka's) ≤ RT(Prim's).

Note that for the first part of the inequality is strict if we take finding light edges cost in each step $= 2E = O(E) = O(2V)$ instead of simply $O(V)$.

Again by the same arguments as in the Proof of (i), we have

RT(Proposed) = RT(Boruvka's on n_0 vertices)+ RT(Prim's on $V - n_0$ vertices)

≤ RT(Prim's on n_0 vertices)+ RT(Prim's on $V - n_0$ vertices)

= RT(Prim's on V vertices) (1.6)

Again RT(Proposed) = RT(Boruvka's on n_0 vertices)+ RT(Prim's on $V - n_0$ vertices)

≥ RT(Boruvka's on n_0 vertices)+RT(Boruvka's on $V - n_0$ vertices)

$\Rightarrow$ RT(Proposed) ≥ RT(Boruvka's on V vertices) (1.7)

From (1.6) and (1.7)

RT(Krusksl's) ≤ RT(Boruvka's) ≤ RT(Proposed) ≤ RT(Prim's).

Thus the Proof of (ii) follows.

Thus the proposed algorithm is a compromise between Kruskal's and Prim's algorithm.

For the following graph (Figure 1), the working of the algorithm is explained. The numeric values across the edges represent weights.

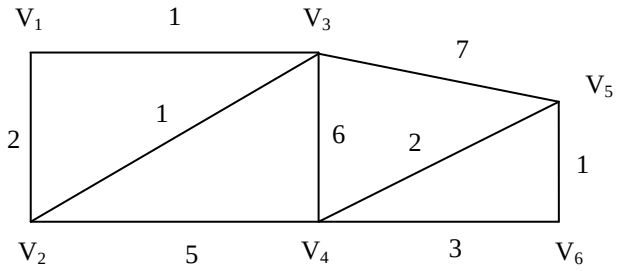

Figure 1. Weighted Graph

Step 1: At this step the light edges of vertices are selected. The selected edges are labeled with arrows. An edge with double arrows indicates this edge is chosen by both end vertices. Broken lines are the edge not selected in Step 1. Refer following Figure 2.

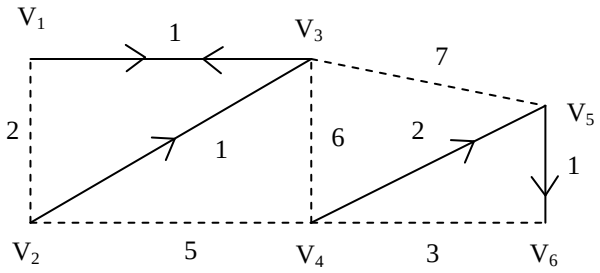

Figure 2. Light edge of Graph

After Step 1 we have two connected components C_1, C_2 and the following multi-graph. In the Figure 3 the connected components C_1 and C_2 are relabeled by super vertices V_1^s and V_2^s. We will stop applying further iterations of Boruvka's algorithm assuming the number of super vertices $n_0 = 2$.

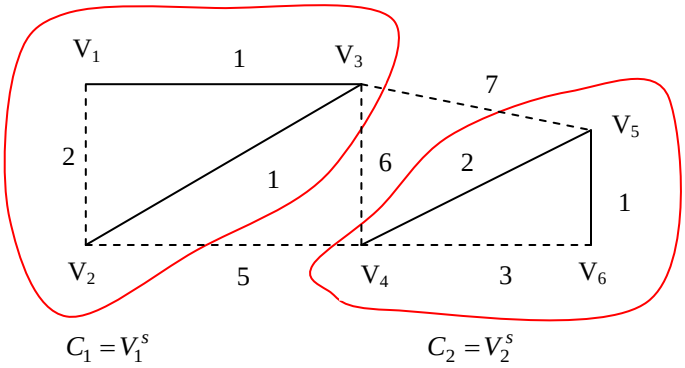

Figure 3. Multi graph with two super vertices V_1^s and V_2^s.

Step 2: Now we will apply Prim's algorithm to find the MST of the above multi graph. In the graph we have three parallel edges $\langle V_3, V_5 \rangle$: cost = 7, $\langle V_3, V_4 \rangle$: cost = 6 and $\langle V_2, V_4 \rangle$: cost = 5 running from V_1^s to V_2^s.

Choosing minimum edge $\langle V_2, V_4 \rangle$: cost = 5, connecting V_1^s to V_2^s. We have the following spanning tree T' of super vertices V_1^s and V_2^s.

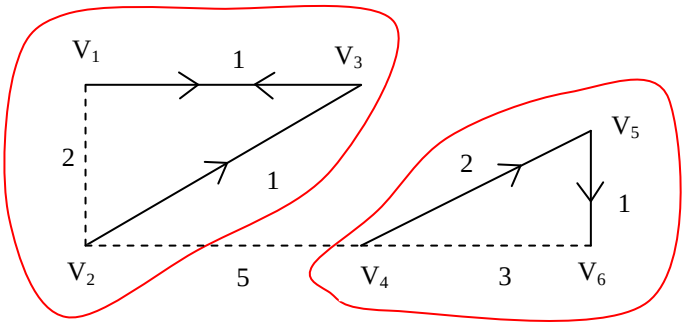

Figure 4. 2-Clustered vertices with a single edge

Step 3: After unclustering the super vertices the required MST is as follows.

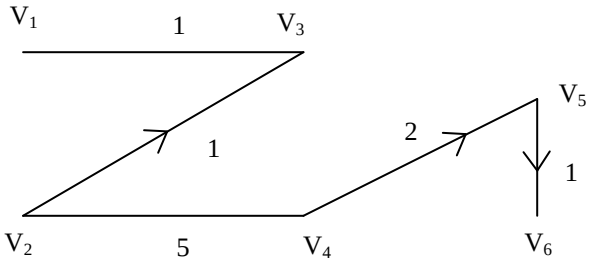

Figure 5. Final Minimum Spanning Tree

IV. PARALLEL FORMULATION OF THE NEW ALGORITHM

In this section we present a parallel algorithm for the proposed minimum spanning tree in shared memory using FAL data structure discussed in [1]. Here we have used Boruvka's and Prim's algorithm using FAL data structure. As we are using major portion of MST generation using Boruvka's algorithm, in our parallel formulation processors run Step 1 and Step 2 of Algorithm 4 (Section 3) on different set of vertices simultaneously.

We say a tree is growing when there exists a light weight edge that connects a tree to a vertex not yet in another tree, and mature otherwise. When all of the vertices (dedicated for a processor) have been incorporated into mature subtrees, we contract each subtree into a super vertex (Step 2 of Algorithm 4) and call Step 1 and Step 2 recursively on resulting multi-graph until a multi-graph of small enough size is obtained. When the multi-graph size is small enough, one of the processors executes sequential Prim's algorithm (Step 3 of Algorithm 4) to find MST of the finest multi-graph. Then the super vertices are unclustered to find the MST of the original graph. The description of our parallel formulation is given in Algorithm 5.

The graph shown in figure 6 and the table I depicts the running time of Kruskal's, Boruvka's and proposed algorithms for various graphs order. In the graph, vertices are taken along x-axis and the corresponding runtimes for these three algorithms are taken along y-axis.

TABLE I. COMPARISION OF RUNNING TIME

V	Kruskal's	Brouvka's	Proposed
2^2	64	32	40
2^3	384	192	152
2^4	2048	1024	600
2^5	10240	5120	2560

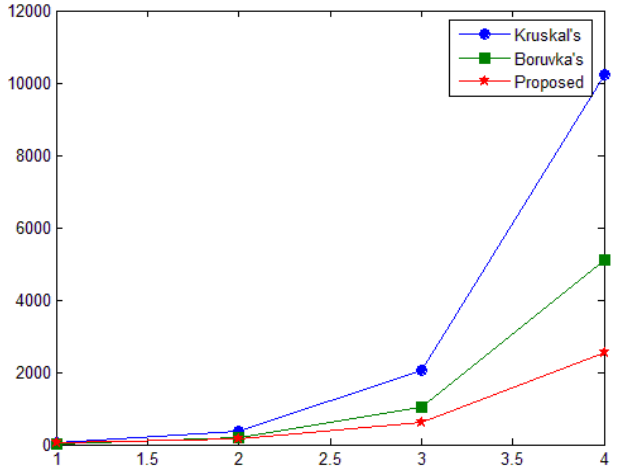

Figure 6. Running Times of Kruskal's, Boruvka's and Proposed

From the graph it is clear that the proposed algorithm runtime reduces as oppose to both Boruvka's and Kruskal's algorithm as the number of vertices of the graph increases.

Algorithm 5

Input: Graph $G = (V, E, w)$ denoted by adjacency list A with $n = |V|$, n_0 be the base problem size of the multi-graph to be solved by Prim's sequential algorithm (Step 3).
Output: MST for G

1. while $(n > n_0)$
2. initialize the color and visited arrays
3. for $v \leftarrow \left\lceil i\frac{n}{p} \right\rceil$ to $\left\lceil (i+1)\frac{n}{p} - 1 \right\rceil$

4. do color[v] = 0 visited [v] = 0
5. Run Algorithm- Boruvka's [2]
6. for $v \leftarrow \left\lceil i\frac{n}{p} \right\rceil$ to $\left\lceil (i+1)\frac{n}{p} - 1 \right\rceil$
7. if visited[v] = 0 then find the lightest incident edge e to v, and label e to be in the MST.
8. With found MST edges, run connected component algorithm on the induced graph, and shrink each component into a super-vertex.
9. Set $n \leftarrow$ the number of remaining super vertices.
10. $m \leftarrow$ the number of edges among n - super vertices
11. //end while
12. If $(n < n_0)$
13. Solve the remaining multi-graph with n - super vertices and m - edges by sequential Prim's algorithm using one of the three processors.

A. *Parallel Runtime*

Let P be the number of processors used by Algorithm 5. After Step 12 of the Algorithm 5. Let P be the number of processors used by Algorithm 5. After Step 10, let $G' = (V', E')$ be the multi-graph with $|V'| = n_0$ super vertices and $|E'| = m$ edges obtained from $G = (V, E)$. As Boruvka's Flexible Adjacent List (FAL) [1] greatly reduces execution time of experiments over Bor-AL and Bor-EL. Using Flexible Adjacent List data structure (FAL), each processor, finds connected components of G at Step 8 in time Tcc given by

$$Tcc = O\left(\frac{(v - v')\lg(v - v')}{p}\right).$$

Boruvka's algorithm used by Step 2 runs in $O\left(\frac{E \lg V}{p}\right)$ times. Further Step 13 takes time

$$O(1) = \begin{cases} E' + V' \lg V' & \text{(Using Fibonacci Heap)} \\ E' \lg V' + V' \lg V' & \text{(Using Binary Heap)} \end{cases}$$

Thus the total complexity of the parallel algorithm PRT (P) is given by

$$PRT(P) = O(1)\begin{cases} \frac{E \lg(V) + V \lg(V)}{p} + E' + V' \lg V' \\ \frac{E \lg(V) + V \lg(V)}{p} + E' \lg V' + V' \lg V' \end{cases}$$

$$= O(1)\begin{cases} \left(\frac{E+V}{p} \lg(V)\right) + E' + V' \lg V' \text{ (Fibonacci Heap)} \\ \left(\frac{E+V}{p} \lg(V)\right) + (E' + V') \lg V' \text{ (Binary Heap)} \end{cases}$$

$$= O(1)\begin{cases} \left(\frac{E \lg V}{p}\right) + E' + V' \lg V' \quad \text{(Using Fibonacci Heap)} \\ \left(\frac{E \lg V}{p}\right) + (E' + V') \lg V' \quad \text{(Using Binary Heap)} \end{cases}$$

$$= O(1)\left(\frac{E \lg V}{p}\right)$$

V. CONCLUSIONS

This paper discusses a new algorithm for finding minimum spanning tree of a weighted graph which is intrinsically parallel, unlike Kruskal's algorithm. As far as the running time of the algorithm is concerned, it is faster than both Kruskal's and Boruvka's algorithms. However the Prim's algorithm is bit faster than ours algorithm. The parallel implementation of the proposed algorithm in shared memory is also better than Boruvka's parallel algorithm, but comparable with Prim's parallel algorithm due to Bader *et al.*

REFERENCES

[1] Bader, D.A.; Guojing Cong, "Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs," *Journal of Parallel and Distributed Computing*, vol.66, no.11, pp.1366-1378, Nov 2006

[2] Chen, C.; Morris, S., "Visualizing evolving networks: minimum spanning trees versus pathfinder networks," *Information Visualization, 2003. INFOVIS 2003. IEEE Symposium on*, vol., no., pp.67,74, 21-21 Oct. 2003

[3] Chong, Ka Wong; Han, Yijie; Igarashi, Yoshihide; Lam, Tak Wah, "Improving the Efficiency of Parallel Minimum Spanning Tree Algorithms", *Discrete Applied Mathematics-DAM*, vol.126, no.1, pp.33-54, 2003

[4] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L., "Introduction to Algorithms", *MIT Press, Cambridge*, MA. 2001

[5] Karypis, G.; Grama, A.; Gupta, A.; Kumar, V., "Introduction to Parallel Computing", *Addison Wesley*, second edition, 2003

[6] Miller, G.L.; Ramachandran, V., "Efficient parallel ear decomposition with applications", *Manuscript, UC Berkeley, MSRI*, January 1986.

[7] Moret, Bernard M.E.; Shapiro, Henry D., "An empirical assessment of algorithms for constructing a minimal spanning tree", *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 15, pp. 99–117, 1994

[8] Otakar Borůvka. O jistem problem minimalnim (About a certain minimal problem). Prace mor. Přírodověd. Spol. v Brně (Acta Societ. Scient. Natur. Moravicae), 3: pp.37–58, 1926

[9] Setia, R.; Nedunchezhian, A.; Balachandran, S., "A New Parallel Algorithm for Minimum Spanning Tree Problem", *International Conference on High Performance Computing (HiPC)*, Dec 16-19, 2009

[10] Sun Chung; Condon, A., "Parallel implementation of Bouvka's minimum spanning tree algorithm," *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International* , vol., no., pp.302,308, 15-19 Apr 1996

[11] Tarjan, R.E.; Vishkin, U., "An efficient parallel biconnectivity algorithm", *SIAM Journal on Computing*, vol.14, pp.862–874, 1985