October 1990

UILU-ENG-90-2250
ACT-114

Applied Computation Theory

AN O(LOG N) TIME COMMON CRCW PRAM ALGORITHM FOR MINIMUM SPANNING TREE

Michael M. Wu

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-90-2250 (ACT #114)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A
7a. NAME OF MONITORING ORGANIZATION Office of Naval Research	
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85-K-0570	
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
An O(log n) Time Common CRCW PRAM Algorithm for Minimum Spanning Tree

12. PERSONAL AUTHOR(S)
Wu, Michael M.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ______ TO ______	October, 1990	10

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Parallel algorithms, minimum spanning tree, CRCW PRAM

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We present an algorithm for finding the minimum spanning tree of a graph with n vertices and m edges on a Common CRCW PRAM using $m+n^{1+2\epsilon}$ processors and $O(m+n^{1+\epsilon})$ space in O(log n) time, where ϵ is a constant such that $0 < \epsilon \leq 1/2$.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

An O(log n) Time Common CRCW PRAM Algorithm for Minimum Spanning Tree

Michael M. Wu

Department of Electrical and Computer Engineering
and
Beckman Institute
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801, U.S.A.

17 October 1989

Abstract

We present an algorithm for finding the minimum spanning tree of a graph with n vertices and m edges on a Common CRCW PRAM using $2m+n^{1+2\varepsilon}$ processors and $O(m+n^{1+\varepsilon})$ space in $O(\log n)$ time, where ε is a constant such that $0 < \varepsilon \leq 1/2$.

Keywords: parallel algorithms, minimum spanning tree, CRCW PRAM

Supported by the Office of Naval Research under Contract N00014-85-K-0570.

1. INTRODUCTION

Let $G = (V,E)$ be a connected graph with a set of vertices V and a set of edges E in which each edge e has a weight $w(e)$. Without loss of generality, assume that the edge weights are distinct. Hence the minimum spanning tree (MST) of G is unique. Let $n = |V|$ and $m = |E|$. We present an algorithm for finding the MST of G on a Common CRCW PRAM using $2m + n^{1+2\varepsilon}$ processors and $O(m+n^{1+\varepsilon})$ space in $O(\log n)$ time, where ε is a constant such that $0 < \varepsilon \leq 1/2$.

The MST problem is an important problem of combinatorial optimization. Some practical applications of MST's include the design of computer, communication, and transportation networks. Graham and Hell [11] gave an extensive history of the MST problem.

Yao [13] and Cheriton and Tarjan [4] designed sequential MST algorithms that ran in time $O(m \log\log n)$. Fredman and Tarjan [9] gave an improved algorithm which ran in time $O(m\beta(m,n))$, where $\beta(m,n) = \min \{i \mid \log^{(i)} n \leq \frac{m}{n}\}$. If $m \geq n$, then $\beta(m,n) \leq \log^* n$. Gallager *et al.* [10] presented a distributed MST algorithm which used at most $5n\log n + 2m$ messages and $O(n \log n)$ time. Awerbuch [1] presented an optimal distributed MST algorithm that required $O(m+n \log n)$ messages and $O(n)$ time.

There have also been several parallel MST algorithms. Chin *et al.* [5] presented an efficient algorithm for the CREW PRAM which ran in time $O(\log^2 n)$ using $O\left(\frac{n^2}{\log^2 n}\right)$ processors. Hirschberg [12] gave an algorithm for the Common CRCW PRAM which ran in time $O(\log n)$ using n^3 processors. Awerbuch and Shiloach [2] designed an algorithm for the Priority CRCW PRAM which ran in time $O(\log n)$ using $m+n$ processors and $O(m+n)$ space.

In this paper, we employ some of the results of Fich *et al.* [7] and modify the algorithm of [2] to obtain a Common CRCW PRAM MST algorithm. A straightforward modification yields an algorithm that runs in time $O(\log n)$ using $mn + n^{1+2\varepsilon}$ processors. We then reduce the number of processors to

$2m + n^{1+2\varepsilon}$. The amount of space used by our algorithm is $O(m+n^{1+\epsilon})$. Our algorithm has the same running time as the algorithm of [12] and uses fewer processors. For mildly dense graphs, where $m = \Omega(n^{1+2\varepsilon})$, our algorithm has the same performance as the algorithm of [2] and uses a weaker CRCW PRAM model. Boppana [3] and Fich, *et al.* [8] established that the time separation between the Priority PRAM and the Common PRAM each with p processors is $\Theta\left(\frac{\log p}{\log \log p}\right)$.

In Section 2, we describe the model of computation. In Section 3, we review the MST algorithm of [2]. In Section 4, we present the results of [7] that apply to our algorithm. In Section 5, we describe the modification of the algorithm of [2] to obtain our Common CRCW PRAM algorithm.

2. MODEL OF COMPUTATION

A CRCW PRAM consists of a set of processors and a shared memory. Each step consists of three phases. In the first phase, each processor may read one shared memory cell. In the second phase, each processor may perform local computations. In the third phase, each processor may write into one shared memory cell. Any number of processors may simultaneously read from a memory cell. If more than one processor simultaneously writes into the same memory cell, then the value that is written depends on the model.

Two CRCW models are the Priority model and the Common model. In the Priority model, each processor is assigned a unique priority. If more than one processor tries to write into the same cell, then the processor with the highest priority is the one that succeeds. In the Common model, if more than one processor tries to write into the same cell, then all the processors must write the same value.

3. THE MST ALGORITHM OF AWERBUCH AND SHILOACH

The algorithm of Awerbuch and Shiloach [2] uses a Priority CRCW PRAM. The priority of each processor is determined by its index. The smaller the index, then the higher its priority. The algorithm assigns processors to edges such that the smaller the weight of an edge, the higher the priority of the corresponding processor. The assignment can be made by sorting the edges by weight and then assigning processors in order. This can be done in O(log n) time using the parallel merge sort algorithm of Cole [6]. Let $p(i,j)$ be the processor assigned to edge (i,j).

A *rooted tree* is a tree whose edges are directed toward the *root.* A *star* is a rooted tree with height 1. Assume the vertices of G are numbered from 1 to n. The number of a vertex is its *id.* In the algorithm, there are variables associated with each vertex i. We will use the name of a vertex to refer to a variable associated with that vertex. The processors that operate on these variables, however, correspond to edges.

Each vertex i has a *parent* $P(i)$, which is either another vertex or itself. If a vertex is a root, then its parent is itself. The parent-child relation defines a directed graph called the *parent's graph,* PG. PG has the same vertices as G. Define $GP(i) = P(P(i))$, and call $GP(i)$ the *grandparent* of i.

The algorithm maintains a set T of undirected edges which always forms a forest of the MST. The algorithm adds edges to T using the property that for any subset of vertices, the edge of least weight leaving the set must belong to the MST. T grows until it becomes the MST.

The algorithm maintains the invariant that after each iteration, for each directed tree in PG, there is a subtree in T spanning the same set of vertices. The algorithm finds edges of the MST by trying to hook stars to other trees in PG. Processors that correspond to edges leaving a star try to hook the star to a tree. Edges that correspond to successful processors are added to T. After the stars are hooked, the algorithm reduces the height of each tree with a shortcut operation, where each vertex takes its grandparent to be its new parent.

$T(e)$ is a boolean variable attached to each edge e. $T(e)$ is initially 0. During the algorithm, if edge e is added to the T, then $T(e)$ is set to 1. WINNER(i) contains the name of the edge corresponding to the writing processor. After the initialization, the algorithm iterates three steps until all the vertices are in the same star. The algorithm is executed in parallel by each edge processor

$p(i,j)$.

Priority CRCW PRAM Algorithm

```
Initialization:
    $T(e) := 0$ for all $e \in E$
    $P(i) := i$ for $i = 1, \cdots, n$

repeat

Step 1: (Star hooking)
    If $i$ belongs to a star and $P(i) \neq P(j)$ then
        $P(P(i)) := P(j)$ and WINNER($P(i)$) := $(i,j)$
    If WINNER($P(i)$) = $(i,j)$ then $T(i,j) := 1$

Step 2: (Cycle breaking)
    If $i < P(i)$ and $i = GP(i)$ then $P(i) := i$

Step 3: (Shortcut operation)
    $P(i) := GP(i)$

until every vertex $i$ belongs to the same star
```

Step 1 performs the hooking operation. Processors that correspond to edges leaving a star try to hook the star to another tree. A star is hooked to a tree by assigning the root of the star a parent that is a vertex of the tree to which the star is being hooked. If more than one processor tries to hook the star, then the processor with the highest priority succeeds. WINNER(i) contains the name of the edge e corresponding to the writing processor. Since edge e belongs to the MST, the algorithm sets $T(e)$:= 1. After Step 1, every star is hooked to some tree.

Step 2 eliminates any cycles that may have been formed in the parent's graph. A cycle of length two forms when an edge's endpoints belong to two different stars and the edge is the edge of least weight leaving both stars. To break a cycle, the algorithm changes the parent pointer of the vertex with the smaller id to point to itself.

Step 3 performs the shortcut operation. For each vertex i, the algorithm sets the grandparent of i to be the new parent of i. Note that if more than one processor updates $P(i)$, then the processors perform a common write operation. The height of each tree that is not a star decreases a factor of at least

3/2.

A vertex determines whether it belongs to a star by using Procedure Star_Check. At the termination of Star_Check, if $ST(i)$ is **true** (**false**), then i belongs (does not belong) to a star.

Procedure Star_Check

```
$ST(i)$ := true
If $P(i) \neq GP(i)$ then $ST(i)$ := false and $ST(GP(i))$ := false
$ST(i) := ST(P(i))$
```

Awerbuch and Shiloach [2] established the correctness of their algorithm. We briefly justify the running time. Consider each iteration of the three steps. Steps 1 and 2 ensure that every star is hooked to some tree to yield a new tree with height greater than one. Since Step 3 reduces the height of every tree with height greater than one by a factor of at least 3/2, the sum of the heights of all the trees present at the start of the iteration is reduced by a factor of at least 3/2. Thus O(log n) iterations yield a single star. Since each iteration takes O(1) time, the algorithm runs in time O(log n).

4. r-COLOR MINIMIZATION PROBLEM

We obtain a Common CRCW PRAM MST algorithm by modifying the implementation of Step 1 of the algorithm of Awerbuch and Shiloach. Only Step 1 uses a priority write operation. In our algorithm, we avoid the priority write by determining the processor of highest priority wanting to write to each memory cell and having only those processors write. It can be seen that the values written in the memory cells are the same as those that would have been written in the Priority CRCW PRAM model.

To determine the processor of highest priority writing to each cell, we solve a special case of the r-color minimization problem described in Fich *et al.* [7].

r-Color Minimization Problem

Before: Each processor p_i, $i = 1, \cdots, p$, has a color x_i, $0 \leq x_i \leq r$, known only to itself. x_i represents the cell p_i wants to write, if any, and 0 otherwise.

After: Each processor p_i knows the value a_i, where $a_i = 1$ if and only if p_i is the processor of lowest index writing to the cell represented by x_i.

For our algorithm, we consider the case where $r = 1$. Fich *et al.* showed that on a Common CRCW PRAM with k memory cells the 1-color minimization problem can be solved in $O\left(\frac{\log p}{\log (k+1)}\right)$ steps. In our discussion, we present a simplified variation of their method and show how the problem can be solved in $O\left(\frac{\log p}{\log k}\right)$ steps.

Let $M_1, \cdots, M_k$ be the k memory cells. Assume without loss of generality that $k \le p^{\varepsilon}$, where ε is a consant such that $0 < \varepsilon \le 1/2$. If $k > p^{1/2}$, then only the first $p^{1/2}$ cells are needed to achieve O(1) steps.

The algorithm iterates the following steps. Processor p_i, $i = 1, \cdots, k$, writes 0 into M_i. The processors are then divided into k groups of nearly equal size, where each group is a set of consecutively numbered processors. The first $p \bmod k$ groups contain $\left\lceil \frac{p}{k} \right\rceil$ processors, and the remaining groups contain $\left\lfloor \frac{p}{k} \right\rfloor$ processors. A processor p_i in the jth group, $1 \le j \le k$, writes 1 into M_j if and only if $x_i = 1$.

The *winner* is the processor of smallest index with $x_i = 1$. Thus the winner is in the group corresponding to the M_j of smallest index containing 1. The algorithm determines the winning group by using the subroutine Leftmost One.

Leftmost One

Before: Cells M_i, $i = 1, \cdots, k$, each contain 0 or 1.

After: M_i contains 1 if and only if all M_j for $j < i$ were initially 0, and M_i was initially 1.

The Leftmost One algorithm compares all pairs of cells M_i and M_j, $1 \le i,j \le k$. If $j < i$ and M_i and M_j both contain 1, then the algorithm writes 0 into M_i. The algorithm requires $k^2 \le p$ processors. After applying the Leftmost One subroutine, processors in group j read M_j. A group determines it is the winning group if its processors read a 1.

All processors that are not in the winning group set $a_i := 0$ and stop. The processors in the winning group then repeat the 1-color minimization algorithm. This process repeats until the winning group contains only one processor, the winner.

Each iteration of the 1-color minimization algorithm reduces the number of processors that may be the winner by a factor of k. Thus the winner is determined in at most $\lceil \log_k p \rceil$ iterations. Since each iteration takes O(1) steps, the winner is determined in $O\left(\frac{\log p}{\log k}\right)$ steps.

5. COMMON CRCW PRAM MST ALGORITHM

Our Common CRCW PRAM MST algorithm is the same as the Priority CRCW PRAM algorithm of Awerbuch and Shiloach except that Step 1 is modified to eliminate the priority concurrent write. Thus we describe the modified implementation of Step 1 only.

In Step 1 of the Priority algorithm, if more than one processor tries to hook a star with root i to a tree, then a priority write of the variable $P(i)$ occurs. Since there is a $P(i)$ for each vertex i, there are n cells into which processors may write. The $P(i)$'s are written by processors performing the hooking operation. Since processors performing the hooking operation correspond to edges leaving stars, as many as m processors may want to write into one $P(i)$.

In the Common algorithm, we first determine the processor of highest priority writing to each $P(i)$ and then have only that processor write. We begin with the direct implementation which requires solving the r-color minimization problem with m processors and n colors.

To maintain the O(log n) running time of the MST algorithm, Step 1 must run in time O(1). In Step 1, the Common PRAM algorithm simultaneously solves n 1-color minimization problems, one for each $P(i)$, using the algorithm of Section 4. Each problem requires n^{ε} cells and $n^{2\varepsilon}$ processors to obtain an O(1) time solution. During the first iteration, m processors are divided into n^{ε} groups.

During each iteration, a processor can determine the group to which it belongs since it knows its rank from the sort performed during the initialization phase. Each iteration reduces the number of contending processors by a factor of n^{ε}, and thus $O\left\lceil \frac{\log m}{\log n^{\varepsilon}} \right\rceil = O(1)$ iterations suffice. Since there are n problems, Step 1 requires a total of $mn+n^{1+2\varepsilon}$ processors and $n^{1+\varepsilon}$ cells. We now show how to reduce the number of processors.

In Step 1 of the Priority algorithm, each processor corresponding to an edge leaving a star writes to exactly one $P(i)$. Thus in the Common algorithm, each processor wanting to write is a possible winner for only one of the n 1-color minimization problems.

The absence of non-writing processors from the groups of processors formed during the solution of the 1-color minimization problem does not affect the outcome since the processors would not have written even if they were present. Thus each processor that wants to write needs only to participate in the solution of the 1-color minimization problem corresponding to the $P(i)$ it wants to write. Hence, for the n 1-color minimization problems, the algorithm requires a total of $m+n^{1+2\varepsilon}$ processors.

The remaining steps of the algorithm require $2m+n$ processors and $O(m+n)$ space. Thus we have a Common CRCW PRAM algorithm for the MST problem that runs in time $O(\log n)$ using $2m+n^{1+2\varepsilon}$ processors and $O(m+n^{1+\varepsilon})$ space. For graphs that are not connected, the algorithm can be easily modified to find a minimum weight spanning forest, and thus the connected components.

ACKNOWLEDGMENT

The author would like to thank Michael C. Loui for his helpful comments.

REFERENCES

[1] B. Awerbuch, Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election and related problems, *Proc. 19th Annual ACM Symp. on Theory of Computing*,

[2] B. Awerbuch and Y. Shiloach, New Connectivity and MSF Algorithms for Shuffle-Exchange Network and PRAM, *IEEE Trans. Computers*, **36** (1987), 1258-1263.

[3] R.B. Boppana, Optimal Separations Between Concurrent-Write Parallel Machines, *Proc. 21st Annual ACM Symp. on Theory of Computing*, (1989), 320-326.

[4] D. Cheriton and R.E. Tarjan, Finding Minimum Spanning Trees, *SIAM J. Comp.*, **5** (1976), 724-742.

[5] F.Y. Chin, J. Lam, and I. Chen, Efficient Parallel Algorithms for some Graph Problems, *Comm. ACM*, **25** (1982), 659-665.

[6] R. Cole, Parallel Merge Sort, *SIAM J. Comp.*, **17** (1988), 770-785.

[7] F.E. Fich, P. Ragde, and A. Wigderson, Relations Between Concurrent-Write Models of Parallel Computation, *SIAM J. Comp.*, **17** (1988), 606-627.

[8] F.E. Fich, P. Ragde, and A. Wigderson, Simulations Among Concurrent-Write PRAMs *Algorithmica*, **3** (1988), 43-51.

[9] M.L. Fredman, and R.E. Tarjan, Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, *J. ACM*, **34** (1987), 596-615.

[10] R. Gallager, P. Humblet, and P. Spira, A Distributed Algorithm for Minimum-Weight Spanning Trees, *ACM Trans. on Programming Languages and Systems*, **5** (1983), 66-77.

[11] R.L. Graham, and P. Hell, On the History of the Minimum Spanning Tree Problem, *Annals of the History of Computing*, **7** (1985), 43-57.

[12] D.S. Hirschberg, Parallel Graph Algorithms Without Memory Conflicts, *Proc. 20th Annual Allerton Conf. on Communications, Control, and Computing*, (1982), 257-263.

[13] A.C. Yao, An $O(|E| \log \log |V|)$ Algorithm for Finding Minimum Spanning Trees, *Info. Proc. Letters*, 4 (1975), 21-23.