J. Indian Inst. Sci., Nov.–Dec. 1987, 67, pp. 429–438
© Indian Institute of Science.

Algorithms for finding centers and medians of trees and graphs on a parallel computation model*

PRANAY CHAUDHURI†
Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur 721 302, India.

Abstract

This paper presents synchronized parallel algorithms for finding centers and medians of trees and graphs. The computation model used is a shared memory single instruction stream, multiple data stream computer that allows both read and write conflicts. Assuming that all the trees and graphs under investigation consist of n nodes, the time bound achieved by the algorithm for trees is $O(\log n)$ with n^2 processors whereas the same for graphs is $O(\log d.\log\log n)$ with $n^2\lceil n/\log\log n\rceil$ processors, where d is the graph diameter.

Key words: Parallel algorithms, graph, center, median, SM-SIMD computers, time complexity.

1. Introduction

Centers and medians provide useful topological information about the concerned graph and have applications in many real-life problems[1]. Dekel *et al*[2] proposed parallel algorithms of $O(\log^2 n)$ time complexity with n^3 processors for finding centers and medians of a graph on both cube-connected and perfect-shuffle computers. Recently, Korach *et al*[3] have proposed algorithms for finding centers and medians of networks on a distributed computation model in which the processors have no memory in common, and can communicate only by exchanging messages. In this paper, we present synchronized parallel algorithms for the problems of locating centers and medians of trees and graphs.

The computation model used is a shared memory single instruction stream, multiple data stream (SM-SIMD) computer. Simultaneous reading of several processors from the same shared memory location is allowed; simultaneous writing in the same memory location is also allowed provided all processors seek to write the same value. Such a model of computation is used by several researchers[4–7] for designing various parallel algorithms.

The time bound achieved by the algorithm for finding centers and medians of a tree is $O(\log n)$ with n^2 processors whereas the same for finding centers and medians of a graph is $O(\log d.\log\log n)$ using $n^2\lceil n/\log\log n\rceil$ processors, where n is the number of nodes of both the tree and the graph, and d is the graph diameter.

*First presented at the Platinum Jubilee Conference on Systems and Signal Processing held at the Indian Institute of Science, Bangalore, India, during December 11–13, 1986.

†Present address: Department of Computer Science, James Cook University, Towns Ville, Queensland 4811, Australia.

2. Notation

Given a graph $G = < V, E >$, where $|V| = n$, we denote by

(1) d the diameter of G;
(2) $l(i,j)$ the length of the shortest path between nodes i and j, $i, j \in V$;
(3) $s(i) = \max \{ l(i,j) \mid j \in V \}$ the separation number for $i \in V$;
(4) $c = \{ j \mid s(j) = \min \{ s(i) \mid i \in V \} \}$ the set of centers of G;
(5) $t(i) = \sum_{j \in V} l(i,j)$ the transmission number for $i \in V$;
(6) $m = \{ j \mid t(j) = \min \{ t(i) \mid i \in V \} \}$ the set of medians of G;

In a rooted tree $T(r) = < V_t, E_t >$, where $r \in V_t$ is the root and $|V_t| = n$, we denote by

(7) $PR(i)$ the immediate predecessor or parent of $i \in V_t$ (it is assumed that $PR(r) = \infty$);
(8) $L(i)$ the level of $i \in V_t$ (it is assumed that $L(r) = 0$);
(9) L the height of $T(r)$;
(10) $YCA(i,j)$ the youngest common ancestor of the pair $i, j \in V_t$.

For the definition of other graph theoretic terms used in this paper, see Christofides[1].

3. Centers and medians of trees

The parallel algorithm for finding centers and medians of a rooted tree $T(r) = < V_t, E_t >$ proposed in this section uses an array $A^k(i)$, $1 \leqslant i \leqslant n \in V_t$, $0 \leqslant k \leqslant L$, in which each row i contains a path from node i to root r in $T(r)$. Each entry of $A^k(i)$ gives the node number of kth ancestor of i. The concept of such an array was first introduced by Savage[8].

Definition 3.1. On the set of nodes of a rooted tree $T(r)$ define a function 'A' as follows:

$$A(i) = \begin{cases} PR(i), \ \forall i \in V_t \wedge i \neq r, \\ \infty, \text{ else.} \end{cases}$$

Definition 3.2. On the set of nodes of a rooted tree $T(r)$ define a recursive function 'A^k' as follows:

$$A^k(i) = \begin{cases} A(A^{k-1}(i)), \ \forall i \in V_t \wedge k > 0, \\ i, \text{ else} \end{cases}$$

Theorem 3.3. Given a function A of a rooted tree $T(r)$, it is possible to compute $A^k(i)$, $1 \leqslant i \leqslant n$, $0 \leqslant k \leqslant L$, in $0(\log L)$ time with nL processors.

Proof.
Algorithm P:
Step 1: *for* $i: 1 \leqslant i \leqslant n$ *do*

$A^0(i) := i$; $A^1(i) := PR(i)$; *od*;

Step 2: *for* $w: 0 \leqslant w \leqslant \lceil \log L \rceil - 1$ *do*
for $x: 1 \leqslant x \leqslant 2^w$, $i: 1 \leqslant i \leqslant n$ *do*
$A^{2^w+x}(i) := A^{2^w}(A^x(i))$; *od*; *od*.

It is clear that Step 1 requires 0(1) time with $2n$ processors while Step 2 can be implemented in $0(\log L)$ time with nL processors. Combining the computational complexities of both Steps 1 and 2 of algorithm P, the theorem clearly follows.

Once $A^k(i)$, $1 \leqslant i \leqslant n$, $0 \leqslant k \leqslant L$ is constructed, level $L(i)$, $\forall i \in V_t$ can be obtained from this array $A^k(i)$ as $L(i) = \min\{k | A^k(i) = r \wedge 0 \leqslant k \leqslant L\}$.

Theorem 3.4. Given the function A^k of a rooted tree $T(r)$, it is possible to find level $L(i)$, $\forall i \in V_t$ in 0(1) time with $n(L+1)$ processors.

Proof.
Algorithm Q:

Step 1: *for* $i: 1 \leqslant i \leqslant n$, $k: 0 \leqslant k \leqslant L$ *do*
if $A^k(i) = r$ *then* $B(i,k) := 1$
else $B(i,k) := 0$; *od*;

Step 2: *for* $i: 1 \leqslant i \leqslant n$, $k: 0 \leqslant k \leqslant L$ *do*
if $k = 0$ *then* $C(i,0) := B(i,0)$
else $C(i,k) := B(i,k) - B(i,k-1)$; *od*;

Step 3: *for* $i: 1 \leqslant i \leqslant n$, $k: 0 \leqslant k \leqslant L$ *do*
if $C(i,k) = 1$ *then* $L(i) := k$; *od*.

It is obvious that each of the above steps requires 0(1) time with $n(L+1)$ processors and hence combining the computational complexities of all these steps of algorithm Q, the theorem follows.

Theorem 3.5. Youngest common ancestors for all pairs of $i, j \in V_t$ in a rooted tree $T(r)$ can be computed in $0(\log L)$ time with n^2 processors.

Proof.
Algorithm R:

Step 1: Construct array $A^k(i)$, $1 \leqslant i \leqslant n$, $0 \leqslant k \leqslant L$.

Step 2: Construct array $D(1:n, 0:L)$ as follows:
2.1: $D(i,k) := \infty$, $1 \leqslant i \leqslant n$, $0 \leqslant k \leqslant L$.
2.2: $D(i, L(i) - k) := A^k(i)$, $1 \leqslant i \leqslant n$, $0 \leqslant k \leqslant L(i)$.

Step 3: *for* $i: 1 \leqslant i \leqslant n$, $j: 1 \leqslant j \leqslant n$ *do*
$l(i,j) := 0$; $h(i,j) := L+1$; flag(i,j) := false;
while not flag(i,j) *do*
$m(i,j) := \lfloor (l(i,j) + h(i,j))/2 \rfloor$;
if $D(i, m(i,j)) = D(j, m(i,j))$ *then*

```
if D(i, m(i, j)) = x then h(i, j) := m(i, j)
else if D(i, m(i, j) + 1) = D(j, m(i, j) + 1)
  then h(i, j) := m(i, j)
  else YCA(i, j) := D(i, m(i, j));
    flag(i, j) := true; fi; fi; fi; od; od.
```

Step 1 requires $0(\log L)$ time with nL processors (Theorem 3.3) whereas array $D(1:n, 0:L)$ can be constructed in $0(1)$ time with $n(L+1)$ processors. Step 3 is a modified binary search algorithm and considering all pairs of $i, j \in V_t$, this step requires $0(\log L)$ time with n^2 processors. Therefore, the overall time complexity of algorithm Q is $0(\log L)$ with n^2 processors, which proves the theorem.

The stepwise description of the parallel algorithm for finding centers and medians of a tree is given below.

Algorithm TREE_CENTER AND MEDIAN:

Input: Parent $PR(i)$ for each $i \in V_t$ in $T(r) = < V_t, E_t >$.

Output: Centers and medians of $T(r)$.

Step 1: Construct array $A^k(i)$, $1 \leqslant i \leqslant n$, $0 \leqslant k \leqslant L$.

Step 2: Compute level $L(i)$, $\forall i \in V_t$.

Step 3: Obtain for each pair of nodes $i, j \in V_t$, the youngest common ancestor $YCA(i,j)$.

Step 4: For each pair of nodes $i, j \in V_t$, compute $l(i, j)$ using the relation
$l(i, j) = L(i) + L(j) - 2^* L(YCA(i, j))$.

Step 5: Obtain $s(i) = \max\{l(i, j) | j \in V_t\}$, $\forall i \in V_t$ and identify each node $c \in V_t$ as the center of $T(r)$, provided $s(c) = \min\{s(i) | i \in V_t\}$.

Step 6: Compute $t(i) = \sum_{j \in V_t} l(i, j)$, $\forall i \in V_t$ and identify each node $m \in V_t$ as the median of $T(r)$, provided $t(m) = \min\{t(i) | i \in V_t\}$.

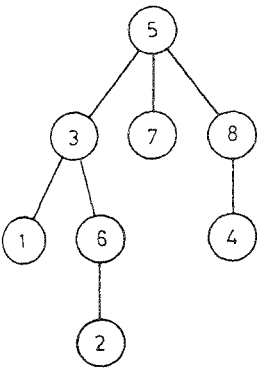
Theorem 3.6. Algorithm TREE_CENTER_AND_MEDIAN has a time complexity of $0(\log n)$ with n^2 processors.

Proof. The time complexity and processor requirement for each step of algorithm TREE_CENTER_AND_MEDIAN are summarised in Table I. From this, the theorem clearly follows.

Table 1

Step 1	$0(\log L)$	nL	Theorem 3.3
Step 2	$0(1)$	$n(L+1)$	Theorem 3.4
Step 3	$0(\log L)$	n^2	Theorem 3.5
Step 4	$0(1)$	n^2	Trivial
Steps 5 and 6	$0(\log n)$	n^2	Trivial

Algorithm TREE_CENTER_AND_MEDIAN is illustrated with the help of an example in fig. 1.



(a)

	0	1	2	3
1	1	3	5	∞
2	2	6	3	5
3	3	5	∞	∞
4	4	8	5	∞
5	5	∞	∞	∞
6	6	3	5	∞
7	7	5	∞	∞
8	8	5	∞	∞

(b)

Node (i)	s(i)	Set of centers of T(5)	t(i)	Set of medians of T(5)
1	4		18	
2	5		22	
3	3		12	
4	5	(3,5)	22	(3,5)
5	3		12	
6	4		16	
7	4		18	
8	4		16	

(c)

FIG. 1. (a) An arbitrary rooted tree $T(5)$ with $n=8$ (b) The array $A^5(i,k)$; $1 \leqq i \leqq 8$, $0 \leqq k \leqq 3$, corresponding to tree $T(5)$, (c) Centers and medians of $T(5)$ computed by algorithm TREE_CENTER_AND_MEDIAN.

4. Centers and medians of graphs

The most important task involved in finding the centers and medians of a graph $G = <V, E>$ is the computation of $l(i,j)$, $\forall i, j \in V$. For this purpose, we propose a parallel algorithm that computes $l(i,j)$, $\forall i, j \in V$ in $O(\log d . \log\log n)$ time with $n^2 \lceil n/\log\log n \rceil$ processors. This algorithm basically generates n shortest-path (SP) spanning trees, each rooted at the node $i \in V$, and simultaneously computes the level of each node $j \in V$ in each SP spanning tree rooted at $i \in V$.

4.1 *Generating n SP spanning trees*

We assume the graph $G = <V, E>$ to be an equivalent directed graph (digraph) denoted $G' < V, E' >$, where $E' = E \cup \{(j,i) \mid \forall i, j \in V \ni (i,j) \in E\}$, without any change in the adjacency matrix or adjacency lists of G. With respect to the digraph G', a tree rooted at a node $x \in V$ containing all nodes of G' reachable from x by an acyclic path of length $\leqslant 2^k$, where k is an integer satisfying $0 \leqslant k \leqslant \lceil \log d \rceil$, is denoted by $T(x,k)$ which preserves the SP property (SPP) defined as follows.

Definition 4.1.1. Let y, z be a pair of nodes in the tree $T(x,k)$ such that $(y,z)\in E'$ but (y,z) does not belong to $T(x,k)$. Then, the tree $T(x,k)$ is said to possess the SPP if one of the following is true:

(i) $L(y|T(x,k)) \geqslant L(z|T(x,k))-1$;

(ii) $L(y|T(x,k))=2^k$;

where, $L(z|T(x,k))$ represents the level of a node z in $T(x,k)$.

It is assumed that the digraph G' is available in the form of its adjacency relations, which in fact define the trees $T(x,0)$, $\forall x\in V$. Initially, for each $x\in V$, the trees $T(y,0)$, $\forall y\in T(x,0)$ are merged with the tree $T(x,0)$ to produce a new tree $T(x,1)$. This process of tree merging is repeated for $\lceil \log d\rceil$ times finally producing the tree $T(x,\lceil\log d\rceil)$, for each $x\in V$, which is an SP spanning tree of G' rooted at $x\in V$. The tree merging is carried out in such a manner that at any stage k, the tree $T(x,k)$ which is obtained from the trees $T(x,k-1)$ and all $T(y,k-1)$, $y\in T(x,k-1)$, $0<k\leqslant \lceil\log d\rceil$, preserves the SPP.

The detailed description of the algorithm for generating n SP spanning trees of G' is given below, where $PR(z|T(x,k))$ is used to denote the parent of a node z in $T(x,k)$.

Algorithm SP_SPANNING_TREES:

Input: The trees $T(x,0)$, $\forall x\in V$, specified by
$PR(y|T(x,0))$ and $L(y|T(x,0))$, $\forall y\in V$.

Output: The SP spanning trees $T(x,\lceil\log d\rceil)$, $\forall x\in V$,
specified by $PR(y|T(x,\lceil\log d\rceil))$ and $L(y|T(x,\lceil\log d\rceil))$, $\forall y\in V$.

```
1.  k := 0; /* initialize */
2.  while k < ⌈log d⌉ + 1 do /* repeat lines 3 through 17⌈log d⌉ times */
3.    p := 0;
4.    while p < ⌈log log n⌉ do /* initialize matrices */
5.      for each pair of x, y = 1, 2, ..., n do
6.        for z = (p⌈n/log log n⌉ + 1) to ((p+1)⌈n/log log n⌉) do
7.          if (z = x) ∧ (PR(y|T(x, k−1)) ≠ φ) then
8.            M_x^k(y, z) := L(y|T(x, k−1))
9.          else if (z ∈ T(x, k−1)) ∧ (PR(y|T(z, k−1)) ≠ φ) then
10.           M_x^k(y, z) := L(z|T(x, k−1)) + L(y|T(z, k−1))
11.         else M_x^k(y, z) := ∞; od; od;
12.     p := p + 1, od;
13    for each pair of x, y = 1, 2, ..., n do /* find the minimum of each row of each matrix
      and define new trees */
14.     find z_m such that M_x^k(y, z_m) = min {M_x^k(y, z) | z = 1, 2, ..., n};
15.     PR(y|T(x, k)) := PR(y|T(z_m, k−1));
16.     L(y|T(x, k)) := M_x^k(y, z_m); od;
17.   k := k + 1; od.
```

An algorithm for finding the maximum of n numbers on the same model as considered in this paper is available[4]. It requires $0(n/p)$ time with $p(1 \leqslant p \leqslant \lceil n/\log\log n \rceil$ processors. Finding the minimum of n numbers is a logically equivalent problem to that of finding the maximum and hence line 14 of algorithm SP—SPANNING—TREES can be implemented in $0(\log\log n)$ time with $\lceil n/\log\log n \rceil$ processors. It can be verified easily that the body of the main *while*-block (lines 3 through 17) of algorithm SP—SPANNING—TREES requires an execution time of $0(\log\log n)$ with $n^2\lceil n/\log\log n \rceil$ processors. Since the body of the main *while*-block is to be repeated for $\lceil \log d \rceil$ times, we have the following.

Theorem 4.1.2. Algorithm SP—SPANNING—TREES requires $0(\log d.\log\log n)$ time with $n^2\lceil n/\log\log n \rceil$ processors.

The arguments regarding the validity of the algorithm SP—SPANNING—TREES are as follows. Since d is the diameter of the digraph G', every acyclic path between a pair of nodes consists of at most d edges. Consequently, the tree $T(x,\lceil \log d \rceil)$ for each $x \in V$, obtained as the output of the algorithm SP—SPANNING—TREES must consist of all nodes of G'. Finally, it can be proved by induction that the tree $T(x,\lceil \log d \rceil)$ for each $x \in V$, preserves the SPP.

4.2. *Finding centers and medians*

It follows from the property of the SP spanning tree of a graph G(or G') that the level of a node $j \in V$, in a SP spanning tree rooted at a node $i \in V$, is the length of the shortest path from i to j (i.e., $l(i,j)$) in G(or G'). Once $l(i,j)$, $\forall i,\ j \in V$ is available, the rest of finding centers and medians of a graph is straightforward. The following is a stepwise description of the parallel algorithm for finding centers and medians of a graph.

Algorithm GRAPH—CENTER—AND—MEDIAN:

Input: The trees $T(x,0)$, $\forall x \in V$, specified by $PR(y|T(x,0))$ and $L(y|T(x,0))$, $\forall y \in V$.

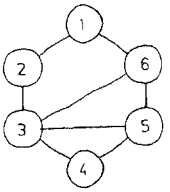
Output: Centers and medians of G.

Step 1: Generate n SP spanning trees, each rooted at a node $i \in V$, using algorithm SP—SPANNING—TREES. For each pair of nodes, $i,\ j \in V$, set $l(i,j) := L(j|T(i,\lceil \log d \rceil))$.

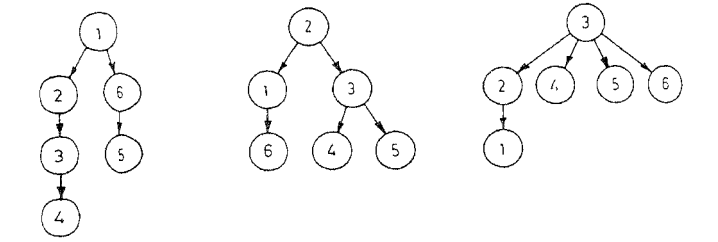
Step 2: Obtain $s(i) = \max\{l(i,j) | j \in V\}$, $\forall i \in V$. Mark each node $c \in V$ as the center of G, provided $s(c) = \min\{s(i) | i \in V\}$.

Table II

Step 1	$0(\log d.\log\log n)$	$n^2\lceil n/\log\log n \rceil$	Theorem 4.1.2
Step 2	$0(\log\log n)$	$n\lceil n/\log\log n \rceil$	Shiloach and Vishkin[4]
Step 3	$0(\log n)$	n^2	Trivial



(a)



(b)

Node (i)	s(i)	Set of centers of G	t(i)	Set of medians of G
1	3		9	
2	2		8	
3	2	(2, 3, 5, 6)	6	(3)
4	3		9	
5	2		7	
6	2		7	

(c)

FIG. 2. (a) A graph G; (b) SP spanning trees, each rooted at a node $i = 1, 2, \ldots, 6$ of G, generated at Step 1 of algorithm GRAPH_CENTER_AND_MEDIAN; (c) Centers and medians of G computed by algorithm GRAPH_CENTER_AND_MEDIAN.

Step 3: Compute $t(i) = \sum_{j=1} l(i, j)$. $\forall i \in V$. Mark each node $m \in V$ as the median of G, provided $t(m) = \min \{t(i) \mid i \in V\}$.

Theorem 4.2.1. Algorithm GRAPH_CENTER_AND_MEDIAN runs in $O(\log d \cdot \log n)$ time with $n^2\lceil n/\log\log n\rceil$ processors.

Proof. Table II summarises the time and processor bounds of each step of algorithm GRAPH_CENTER_AND_MEDIAN, from which the theorem clearly follows.

Figure 2 illustrates algorithm GRAPH_CENTER_AND_MEDIAN with the help of an example.

5. Closing remarks

Since algorithm TREE_CENTER_AND_MEDIAN does not require simultaneous access to the same memory location by several processors during write operation, this algorithm also works on a SM SIMD computer without write conflict.

For digraphs two different separation numbers, namely, outseparation and inseparation numbers are defined. As a result, for digraphs, two types of centers, referred to as outcenters and incenters exist. Similarly, two different types of transmission numbers and hence two types of medians, *i.e.*, outmedians and inmedians are defined for digraphs[1]. The parallel algorithm GRAPH_CENTER_AND_MEDIAN can easily be modified for finding outcenters, incenters, outmedians and inmedians for digraphs.

An important outcome of this paper is a parallel algorithm that generates n SP spanning trees of an undirected graph or a strongly connected digraph in $O(\log d \cdot \log\log n)$ time with $n^2\lceil n/\log\log n\rceil$ processors. Moreover as a direct consequence of algorithm SP_SPANNING_TREES the minimum depth spanning tree algorithm can be developed. The existing parallel algorithm for the minimum depth spanning tree problem due to Dekel *et al*[2] requires $O(\log^2 n)$ time and n^3 processors on both cube-connected and perfect-shuffle computers.

The performance of a parallel algorithm is usually measured in terms of the cost of that algorithm which is the product between the parallel running time and the number of processors used. Clearly the cost of the algorithm GRAPH_CENTER_AND_MEDIAN for finding the centers and medians of graphs is $n^3 \log d$, whereas for the same problem the cost of Dekel *et al*'s[2] algorithm is $n^3 \log^2 n$. It may be noted that this problem can also be solved in $O(\log n)$ parallel time by using Kucera's[5] all-pairs shortest paths algorithm. Since Kucera's shortest paths algorithm requires n^4 processors the same will be required by the algorithm for finding centers and medians of graphs and hence its cost will be $n^4 \log n$ unlike $n^3 \log d$ as reported in this paper.

References

1. CHRISTOFIDES, N. *Graph theory: An algorithmic approach*, Academic Press, London, 1975.

2. DEKEL, E. NASSIMI, D. AND SAHNI, S. Parallel matrix and graph algorithms. *SIAM J. Computing*, 1981, **10**, 657–67

3. KORACH, E. ROTEM, D. AND SANTORO, N. Distributed algorithms for finding centers and medians in networks. *ACM Trans. Programming Languages Systems*, 1984, **6**, 380–401.

4. SHILOACH, Y. AND VISHKIN, U. Finding the maximum, merging, and sorting in a parallel computation mode *J. Algorithms*, 1981, **2**, 88–102.

5. KUCERA, L. Parallel computation and conflicts in memory access. *Inf. Processing Let.* 1982, **14**, 93–96.

6. CHAUDHURI, P. An $O(\log n)$ parallel algorithm for strong connectivity augmentatio problem. *Int. J. Computer Math.*, 1987, **22**, 187–197.

7. CHAUDHURI, P. AND GHOSH, R. K. Parallel algorithms for analyzing activity networks. *Bit*, 1986, **26**, 418–429.

8. SAVAGE, C. D. *Parallel algorithms for graph theoretic problems*. Ph. D. Thesis, CSL Rep ACT-4, University of Illinois, 1977.