数理解析研究所講究録
992 巻 1997 年 114-121

An $O(log\ n)$ parallel algorithm for constructing a spanning forest on Trapezoid graphs

本間 宏利 (Hirotoshi Honma)† 増山 繁 (Shigeru Masuyama)‡

† Department of Information Engineering,
Kushiro National College of Technology,
Kushiro-shi, Hokkaido 084, JAPAN

‡ Department of Knowledge-Based Information Engineering,
Toyohashi University of Technology
Toyohashi-shi, Aichi 441, JAPAN

Abstract

Let $G = (V, E)$ be a simple graph with n vertices, m edges and p connected components. The problem of constructing a spanning forest is to find a spanning tree for each connected component of G. For a simple graph, Chin et al.[1] demonstrated that a spanning forest can be found in $O(log^2\ n)$ time using $O(n^2/log^2\ n)$ processors. In this paper, we propose an $O(log\ n)$ time parallel algorithm with $O(n)$ processors on the EREW PRAM for constructing a spanning forest on trapezoid graphs.

1 Introduction

Given a simple graph $G = (V, E)$ with n vertices, m edges and p connected components, the spanning forest problem is to find a spanning tree for each connected component of G. If $p = 1$ for G, i.e., G is connected, the spanning forest problem is equivalent to the spanning tree problem of finding a connected subgraph which is a tree and contains all the vertices of G. These problems have applications to electrical power demand problem or computer network design problem etc. A spanning tree and a spanning forest can be found in linear time using, for example, the depth-first search. In recent years a large number of studies have been made to parallelize known sequential algorithms. The spanning tree problem can be solved in $O(log\ n)$ time with $O(log\ n + m)$ processors on CRCW PRAM (Concurrent-Read Concurrent-Write Parallel Random Access Machine) by Klein[5] et al.'s algorithm. Moreover, Chin[1] et al. demonstrated that a spanning forest can be found in $O(log^2\ n)$ time using $O(n^2/log^2\ n)$ processors for simple graphs. In general, it is known that more efficient or optimal parallel algorithms can be developed by restricting classes of graphs. For instance, Wang[7] et al. proposed an optimal parallel algorithm for constructing a spanning tree on *permutation graphs*[2] which runs in $O(log\ n)$ time using $O(n/log\ n)$ processors on the EREW PRAM (Exclusive-Read Exclusive-Write Parallel Random Access Machine). In this paper, we propose an efficient parallel algorithm which runs in $O(log\ n)$ time with $O(n)$ processors for constructing a spanning forest by restricting the class of graphs to *trapezoid graphs*[6].

We next illustrate the trapezoid graph. There are two horizontal lines, called the *top channel* and the *bottom channel*, respectively. Each channel is labeled with consecutive integer values 1,2,...,$2n$ (where n is the number of trapezoids). A *trapezoid* T_i is defined by four corner points $[a_i, b_i, c_i, d_i]$ where a_i, b_i ($a_i < b_i$) lie on the top channel and c_i, d_i ($c_i < d_i$) lie on the bottom channel, respectively. Without loss of generality, we assume that each trapezoid has four corner points and all corner points are distinct[6]. The geometric representation described above is called a *trapezoid diagram T*.

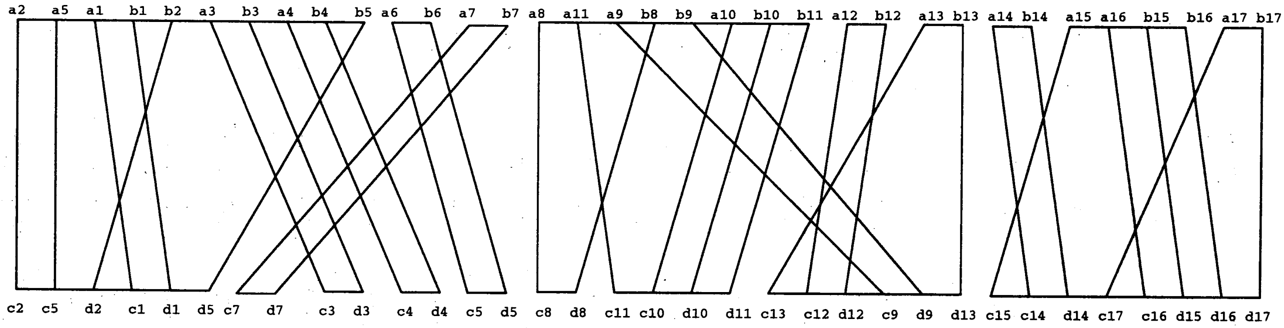

Figure 1: Trapezoid diagram T.

Figure 1 shows a trapezoid diagram T consisting of seventeen trapezoids. We assume that trapezoids are labeled in increasing order of their corner points b_i's, i.e., $i < j$ if $b_i < b_j$. An undirected graph $G = (V, E)$ is called a trapezoid graph if there exists a trapezoid diagram T satisfying

$V = \{i \mid \text{vertex } i \text{ corresponds to trapezoid } T_i\}$,

$E = \{(i, j) \mid \text{trapezoids } T_i \text{ and } T_j \text{ intersect in trapezoid diagram } T\}$.[6]

Input of trapezoid diagram consists of array $T_T[1:2n]$ of corner points, array $P_T[1:2n]$ of *corner point numbers* each of which is assigned to each corner point on the top channel and array $T_B[1:2n]$ of corner points, array $P_B[1:2n]$ of corner point numbers each of which is assigned to each corner point on the bottom channel. Table 1 shows $T_T[1:2n]$, $P_T[1:2n]$, $T_B[1:2n]$, $P_B[1:2n]$ for trapezoid diagram T shown in Figure 1. The trapezoid graph G corresponding to the trapezoid diagram T illustrated in Figure 1 is shown in Figure 2. The class of trapezoid graphs includes two well-known classes of intersection graphs[2], the class of *permutation graphs*[2] and the class of *interval graphs*[2]. The former is obtained by setting $a_i = b_i$ and $c_i = d_i$ for all i, and the latter is obtained by setting $a_i = c_i$ and $b_i = d_i$ for all i, respectively.

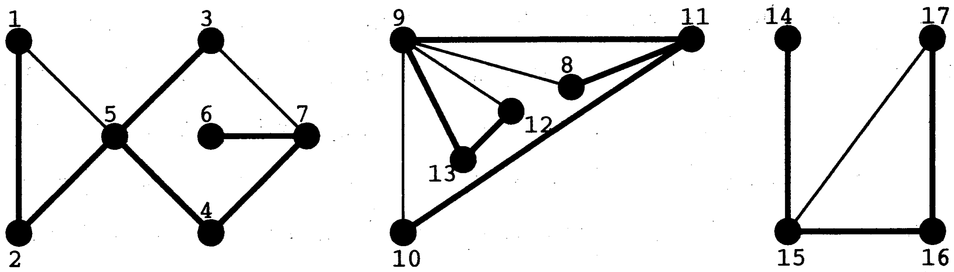

Figure 2: Trapezoid graph G and Spanning Forest of G

Table 1: Arrays T_T, P_T, T_B, P_B.

T_T	a_2	a_5	a_1	b_1	b_2	a_3	b_3	a_4	b_4	b_5	a_6	b_6	a_7	b_7	a_8	a_{11}	a_9
P_T	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
T_B	c_2	c_5	d_2	c_1	d_1	d_5	c_7	d_7	c_3	d_3	c_4	d_4	c_5	d_5	c_8	d_8	c_{11}
P_B	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

T_T	b_8	b_9	a_{10}	b_{10}	b_{11}	a_{12}	b_{12}	a_{13}	b_{13}	a_{14}	b_{14}	a_{15}	a_{16}	b_{15}	b_{16}	a_{17}	b_{17}
P_T	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
T_B	c_{10}	d_{10}	d_{11}	c_{13}	c_{12}	d_{12}	c_9	d_9	d_{13}	c_{15}	c_{14}	d_{14}	c_{17}	c_{16}	d_{15}	d_{16}	d_{17}
P_B	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34

2 Parallel Algorithm

In this section we propose a parallel algorithm for constructing a spanning forest of trapezoid graphs. The algorithm can be parallelized by applying pointer jumping technique[3][4] and parallel prefix computation[3][4]. Algorithm CSF (Construction of Spanning Forest) for constructing a spanning forest of a trapezoid graph is presented as follows:

Algorithm CSF

Input: Arrays $T_T[1:2n]$, $P_T[1:2n]$, $T_B[1:2n]$, $P_B[1:2n]$.

Output: A spanning forest F^* of G. Initially F^* be a graph with n vertices and no edge.

(Step 1) [Construction of arrays $P_a[1:n]$,$P_b[1:n]$,$P_c[1:n]$,$P_d[1:n]$.]
(1) If $T_T[i]$ is corner point 'a_j', $P_T[i]$ is stored to $P_a[j]$, otherwise (i.e., $T_T[i]$ is 'b_j') $P_T[i]$ is stored to $P_b[j]$ in parallel for i,$1 \leq i \leq 2n$.
(2) If $T_B[i]$ is corner point 'c_j', $P_B[i]$ is stored to $P_c[j]$, otherwise (i.e., $T_B[i]$ is 'd_j') $P_B[i]$ is stored to $P_d[j]$ in parallel for i,$1 \leq i \leq 2n$.

Table 2 shows the result obtained by applying Step 1 to Table 1. Each of $P_a[1:n]$,$P_b[1:n]$,$P_c[1:n]$,$P_d[1:n]$ is an array having corner point numbers assigned to corner points 'a','b','c','d' for each trapezoid T_i, $1 \leq i \leq n$ on trapezoid diagram T, respectively.

Table 2: Arrays P_a, P_b, P_c, P_d.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P_a	3	1	6	8	2	11	13	15	17	20	16	23	25	27	29	30	33
P_b	4	5	7	9	10	12	14	18	19	21	22	24	26	28	31	32	34
P_c	4	1	9	11	2	13	7	15	24	18	17	22	21	28	27	31	30
P_d	5	3	10	12	6	14	8	16	25	19	20	23	26	29	32	33	34

(Step 2) [Construction of arrays $L_a[1:n]$,$L_c[1:n]$,$R_d[1:n]$.]
(1) Let $L_a[i]$ be min($P_a[n]$,$P_a[n-1]$,...,$P_a[i]$) in parallel for i, $1 \leq i \leq n$.
(2) Let $L_c[i]$ be min($P_c[n]$,$P_c[n-1]$,...,$P_c[i]$) in parallel for i, $1 \leq i \leq n$.
(3) Let $R_d[i]$ be max($P_c[1]$,$P_c[2]$,...,$P_c[i]$) in parallel for i, $1 \leq i \leq n$.

(Step 3) [Construction of arrays $S_a[1:n]$ and $C[1:n]$.]
Initially $C[i] := 0$ for all i.
(1) If $P_a[i] = L_a[i]$, let $S_a[i]$ be a pointer to i (*self-loop*), otherwise, let $S_a[i]$ be a pointer to $i+1$ in parallel for i, $1 \leq i \leq n$.
Then, we apply pointer jumping technique to $S_a[i]$ in parallel for i, $1 \leq i \leq n$.
(2) If $P_b[i] > L_a[i+1]$, then $C[i] := S_a[i+1]$ and $F^* := F^* \cup \{(i, S_a[i+1])\}$ in parallel for i,$1 \leq i \leq n-1$.

(Step 4) [Construction of arrays $S_c[1:n]$.]
(1) If $P_c[i] = L_c[i]$, let $S_c[i]$ be a pointer to i (*self-loop*), otherwise, let $S_c[i]$ be a pointer to $i+1$ in parallel for i, $1 \leq i \leq n$.
Then, we apply pointer jumping technique to $S_c[i]$ in parallel for i, $1 \leq i \leq n$.
(2) If $P_d[i] > L_c[i+1]$ and $C[i] = 0$, then $C[i] := S_c[i+1]$ and $F^* := F^* \cup \{(i, S_c[i+1])\}$ in parallel for i, $1 \leq i \leq n-1$.

(Step 5) [Construction of arrays $S_d[1:n]$.]
(1) If $P_d[i] = R_d[i]$, let $S_d[i]$ be a pointer to i (*self-loop*), otherwise, let $S_d[i]$ be a pointer $i-1$ in parallel for i, $1 \leq i \leq n$.
Then, we apply pointer jumping technique to $S_d[i]$ in parallel for i, $1 \leq i \leq n$.
(2) If $R_d[i] > L_c[i+1]$ and $C[i] = 0$, then $C[S_c[i+1]] := S_d[i]$ and $F^* := F^* \cup \{(S_c[i+1], S_d[i])\}$ in parallel for i, $1 \leq i \leq n-1$.
(3) Change F^* to be an undirected graph by neglecting the direction of each edge in F^*.

Table 3 shows the result obtained by applying Steps 2,3,4,5 for Table 2. Figure 2 shows the spanning forest $F^* = (V, E')$ constructed by Algorithm CSF for trapezoid graph G, where

V={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17},
E'={(1,2),(2,5),(3,5),(4,5),(6,7),(7,4),(8,11),(9,11),(10,11),(12,13),(13,9),(14,15),(15,16),(16,17)}.

Table 3: Arrays $L_a, L_c, R_d, S_a, S_c, S_d, C$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P_a	3	1	6	8	2	11	13	15	17	20	16	23	25	27	29	30	33
L_a	1	1	2	2	2	11	13	15	16	16	16	23	25	27	29	30	33
S_a	2	2	5	5	5	6	7	8	11	11	11	12	13	14	15	16	17
P_b	4	5	7	9	10	12	14	18	19	21	22	24	26	28	31	32	34
P_c	4	1	9	11	2	13	7	15	24	18	17	22	21	28	27	31	30
L_c	1	1	2	2	2	7	7	15	17	17	17	21	21	27	27	30	30
S_c	2	2	5	5	5	7	7	8	11	11	11	13	13	15	15	17	17
P_d	5	3	10	12	6	14	8	16	25	19	20	23	26	29	32	33	34
R_d	5	5	10	12	12	14	14	16	25	25	25	25	26	29	32	33	34
S_d	1	1	3	4	4	6	6	8	9	9	9	9	13	14	15	16	17
C	2	5	5	5	0	7	4	11	11	11	0	13	9	15	16	17	0

3 The correctness and complexity of Algorithm CSF

Before proving the correctness of Algorithm CSF, note that notation (v, w) where v, w are vertices, is used for both directed and undirected edges. Note also that we sometimes use abbreviated expressions like "$(i, S_a[i])$ is an edge of trapezoid graph G" which means "directed edge $(i, S_a[i])$ corresponds to an undirected edge of trapezoid graph G", and "a connected graph is constructed" which means "a graph which is connected by neglecting the direction of edges", whenever no confusion may arise. Furthermore recall that F^* is directed until Step 5-(3) is executed, but F^* is regarded as an undirected graph by neglecting the direction of edges when we refer to connected components of F^*. Finally, note that T is a rooted tree (in-tree) when we refer to the root of T.

Lemma 1

For i,j, $1 \leq i < j \leq n$, if $P_b[i] > L_a[j]$, $(i, S_a[j])$ is an edge of trapezoid graph G after the execution of Step 3.

For i,j, $1 \leq i < j \leq n$, if $P_d[i] > L_c[j]$, $(i, S_c[j])$ is an edge of trapezoid graph G after the execution of Step 4.

For i,j, $1 \leq i < j \leq n$, if $R_d[i] > L_c[j]$, $(S_c[j], S_d[i])$ is an edge of trapezoid graph G after executing Step 5

Proof. We first give a condition for (i, j) to exist between two distinct vertex i and j $(i < j)$ in trapezoid graph G. By the definition of trapezoid graph, there exists (i, j) between two distinct vertex i and j in G if and only if trapezoid T_i and T_j intersect in trapezoid diagram T. If trapezoid T_i and T_j intersect, it satisfies either $P_b[i] > P_a[j]$ on the top channel or $P_d[i] > P_c[i]$ on the bottom channel. Therefore, edge (i, j) exists between i and j in G if and only if (1) is satisfied:

$$(i-j)(P_b[i] - P_a[j]) < 0 \textit{ or } (i-j)(P_d[i] - P_c[j]) < 0. \tag{1}$$

By the assumption that $i < j$ and $P_b[i] > L_a[j]$ we obtain

$$(i-j)(P_b[i] - L_a[j]) < 0. \tag{2}$$

After executing Step 4-(1) $S_a[j]$ has value k_1 ($k_1 \geq j$) which satisfies $L_a[j] = P_a[k_1]$. Besides, by the definition that $L_a[j]$ =min$(P_a[j], P_a[j+1], ..., P_a[n])$ we obtain

$$S_a[j] \geq j,$$

$$L_a[j] = L_a[S_a[j]] = P_a[S_a[j]].$$

By applying the above to (2), we obtain

$$(i - S_a[j])(P_b[i] - P_a[S_a[j]]) < 0. \tag{3}$$

(3) means that there exists an edge between vertex i and $S_a[j]$ in G. Therefore $(i, S_a[j])$ is an edge in a trapezoid graph G. A similar discussion proves that $(i, S_c[j])$ is an edge and $(S_c[j], S_d[i])$ is an edge in G □

Lemma 2 *If array $C[1:n]$ has q '0' elements after executing Step 4, F^* has n vertices, $n-q$ edges and q connected components such that each connected component is a tree with root i, where $C[i] = 0$.* □

Proof. After executing Step 4, $C[n]$ obviously has value '0'. We consider a vertex i such that $C[i] = 0, C[i+1], C[i+2], ..., C[n-1] \neq 0, C[n] = 0$. If such i does not exist, G is connected (i.e., $p = 1$). Now we assume G has more than one connected components (i.e., $p > 1$). Then, since $C[n-1] \neq 0$, there exists an edge $(n-1, n)$ incident to vertex $n-1$ and n. And also, since $C[n-2] \neq 0$, there exists an edge incident to vertex $n-2$ and incident to either vertex $n-1$ or n. In this way, there exists an edge between vertex j and one among vertices $j+1$,$j+2$,...,n for each vertex j, $i+1 \leq j \leq n-1$. On the other hand, since $C[i] = 0$, there exists no edge between vertex i and vertex j where $j \geq i+1$. Thus, a connected graph having $n-i$ vertices from $i+1$ to n, and $n-i-1$ edges is constructed. By the definition of a tree, this subgraph of G is a tree with root n. Similarly, we can construct other trees with root j which corresponds to $C[j]=0$ for remaining vertex set $\{1,2,...,i\}$ where $1 \leq j \leq i$. Since $C[1:n]$ has q '0' elements, we can finally construct q distinct trees in F^*. By Lemma 1, edges constructed by Steps 3,4 are edges of trapezoid graph G. Therefore F^* is a subgraph of G with q connected components, n vertices, $n-q$ edges and each connected component is a tree with root i where $C[i]=0$. □

Lemma 3 *After executing Step 5, F^* is a spanning forest of G.* □

Proof. It is easy to see that F^* is a spanning forest of G if and only if F^* is a spanning subgraph of G where each of connected components of F^* is a tree and there exists no edge in G which connects two distinct connected components of F^*. We call this condition, condition 1 and prove that F^* constructed after executing Step 5 satisfies this condition.

By Lemma 2, F^* is a spanning subgraph of G after executing Step 4 and has $q(q \leq p)$ connected components $t_1,t_2,...,t_q$ which are arranged in increasing order of the number assigned to the root of each tree t_i, n vertices and $n-q$ edges.

We also denote each connected component of F^* constructed after executing Step 5 by $t'_1,t'_2,...,t'_p$. These connected components are constructed as follows.

For t_j,t_{j+1}, $1 \leq j \leq q-1$, if $P_d[i] > L_c[i+1]$ where i and $i+1$ correspond to the root vertex of t_j and the vertex of t_{j+1} having the minimum number, respectively, then $(S_c[i+1], S_d[i])$ is added to F^*. Note that $S_c[i+1]$ is in t_{j+1} and $S_d[i]$ is in one of t_k, $1 \leq k \leq j$, and $(S_c[i+1], S_d[i])$ is an edge incident to t_{j+1} and one of t_k, $1 \leq k \leq j$, furthermore, it is also an edge of G by Lemma 1. For each t_i, at most one edge is connected to each t_j where $j < i$. Hence, F^* is acyclic. As otherwise, any t_i has two edges connected to t_j,t_k $(j,k < i,\ j \neq k)$, which is a contradiction.

Therefore F^* is a spanning subgraph of G where each of connected components $t'_1,t'_2,...,t'_p$ of F^* is a tree, since the connection of two trees by one edge forms a tree by the property of a tree. On the other hand, unless $P_d[i] > L_c[i+1]$, it is clear that there exists no edge between t_{j+1} and one of t_k, $1 \leq k \leq j$ from definition of R_d and L_c. It means that there exists no edge in G connecting two distinct connected components of F^*. Therefore F^* satisfies condition 1 and is a spanning forest of G. □

We now analyze the complexity of Algorithm CSF. Step 1 can be executed in $O(log\ n)$ time using $O(n/log\ n)$ processors by applying Brent's scheduling principle[3][4]. Step 2 can be executed in $O(log\ n)$ time using $O(n/log\ n)$ processors by applying parallel prefix computation[3][4]. Steps 3,4,5-(1) can be executed in $O(log\ n)$ time using $O(n)$ processors by applying pointer jumping technique[3][4]. Steps 3,4,5-(2) can be executed in $O(log\ n)$ time using $O(n/log\ n)$ processors by applying Brent's scheduling principle. Above parallel algorithm design techniques can be executed on EREW PRAM. Hence we have the following theorem.

Theorem 1 *Algorithm CSF constructs a spanning forest of trapezoid graphs in $O(log\ n)$ time with $O(log\ n)$ processors on EREW PRAM.*

Acknowledgements

We would like to thank Ministry of Education, Science and Culture of Japan for awarding the first author a research fellowship at Toyohashi University of Technology, which enabled us to do this research.

References

[1] F. Y. Chin, J. Lam and I. Chen, Efficient parallel algorithms for some graph problems, *Communications of the ACM*, **25**, 9 (1982).

[2] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York (1988).

[3] A. Gibbons and W. Rytter, *Efficient parallel algorithms*, Cambridge University Press (1988).

[4] J. JáJá, *An Introduction to parallel algorithms*, Addison-Wesley Publishing Company (1992).

[5] P. Klein and C. Stein, A parallel algorithm for eliminating cycle in undirected graphs, *Information processing. letters.*, **34** (1990) 307-312.

[6] Y. D. Liang, Dominations in trapezoid graphs, *Information processing. letters.*, **52** (1994) 309-315.

[7] Yue-Li Wang, Hon-Chan Chen and Chen-Yu Lee, An $O(\log n)$ parallel algorithm for constructing a spanning tree on permutation graphs, *Information processing. letters.*, **56** (1995) 83-87.