中華民國八十四年全國計算機會議論文集
Proceedings of National Computer Symposium 1995

# 辨識序列平行圖的平行演算法
# Parallel Decomposition of Gneralized-Series-Parallel Graphs

謝孫源 何錦文
**Sun-Yuan Hsieh and Chin-Wen Ho**

國立中央大學資訊工程研究所
**Department of Computer Science & Information Engineering**
**National Central University**
**Chung-Li 32054, R.O.C.**

## 摘要

在此篇論文中，我們設計一個有效率的平行演算法去辨識一般化序列平行圖，並建立能夠表示此種圖形之建構方法的一種資料結構：分解樹。此平行演算法在可同時讀寫的平行隨機存取器只花費$O(\log n)$的時間和$C(m,n)$個處理器。$C(m,n)$是在對數時間 (logarithmic time) 找出一個具有$m$個邊和$n$個節點的圖形之相連部份(connected components) 所使用的處理器個數。如果能夠輸入一般化序列平行圖的分解樹，則在此種圖形上所探討的圖形理論問題都能夠有效地求解。同時，在此篇論文中我們也導出一些關於此種圖形的重要性質。

關鍵詞：平行演算法、一般化序列平行圖、分解樹、可同時讀寫的平行隨機存取器。

## Abstract

An efficient parallel algorithm for constructing a decomposition tree of given generalized-series-parallel (*GSP*) graphs is presented. It takes $O(\log n)$ time with $C(m,n)$ processors on a CRCW PRAM, where $C(m,n)$ is the number of processors required to find connected components of a graph with $m$ edges and $n$ vertices in logarithmic time. The class of *GSP* graphs belongs to the decomposable graphs which can be represented by their decomposition trees. Given a decomposition tree of a *GSP* graph there are many graph-theoretic problems can be solved efficiently. In this paper, we also derive some interesting properties for *GSP* graphs based on their structure characterizations.

**Key words**: parallel algorithm, generalized-series-parallel graphs, decomposition tree, CRCW PRAM,

## 1. Introduction

The generalized-series-parallel (*GSP*) graphs are those graphs which can be obtained from a set of single-edges by applying recursively the series, parallel and generalized-series compositions. Such graphs belong to a class of $k$-terminal graphs defined in [13], since each *GSP* graph $G$ has two special vertices called terminals and satisfies the condition that $G$ is generated by above composition rules acting only at terminals. The *GSP* graphs contain series-parallel (*SP*) graphs, outerplanar graphs, trees, unicyclic graphs, *CN*-trees, *C*-trees, *2*-trees, cacti and filaments (square, triangular and hexagonal) [13].

The $k$-terminal graphs are also called decomposition graphs which can be decomposed into a set of primitive graphs by a certain set of composing rules [3]. The decomposition structure of the given decomposable graph can be represented by a decomposition tree in which each leave represents a primitive graph and each internal node represents an appropriate composition operation. Given a *GSP* graph in the form of its decomposition tree, there exists linear-time sequential algorithms for solving many graph-theoretic problems such as the maximum cut set, the maximum cardinality of a minimal dominating set, etc [9]. Furthermore, Bern, Lawler and Wong [3] shows that by providing a decomposition tree for the given decomposable graph many combinatorial optimization problems for finding an optimal subgraph $H$ can be solved in linear time by a dynamic programming approach if the desired subgraph $H$ satisfying a property that is "regular" with respect to the composition rules. Such problems include the maximum independent set, maximum matching set and minimum dominating set problems [3].

For parallel computations, Yain [14] presents a cost optimal parallel algorithm for solving above "regular" optimal subgraph problems by applying binary tree contraction technique [1] to a decomposition tree of the given decomposable graph, which takes $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM, where $n$ is the size of the decomposition tree of the problem. Consequently, this implies that a wide class of "regular" optimal subgraph problems on *GSP* graphs can be solved optimally. Thus the problem of constructing decomposition trees is crucial for sequnetial and parallel *GSP* graph computations.

The sequential $O(n)$ time algorithm of recognition *GSP* graphs is presented in [13], where $n$ is the number of vertices of input graph. In this paper, we develop a parallel strategy for constructing a decomposition tree of the given graph $G$ if $G$ is recognized as a *GSP* graph in our algorithm. The time complexity of the algorithm is $O(\log n)$ and the number of processors used is $C(m,n)$, where $C(m,n)$ is the number of processors required to compute connected components of a graph with $m$ edges and $n$ vertices in logarithmic time. The best result for $C(m,n)$ is $O((m+n)\alpha(m,n)/\log n)$, where $\alpha$ is inverse ackermann function [4]. Our algorithm runs on a deterministic parallel random access machine that permits concurrent reads and concurrent writes (CRCW) in its shared memory and, in case of a write conflict, allows an arbitrary processor to success [11].

## 2. Preliminaries

Let $V(G)$ and $E(G)$ stand, respectively, for the vertice set and the edge set of an undirected graph $G$. Assume that $|V(G)| = n$ and $|E(G)| = m$. We denote an edge between $x$ and $y$ as $(x, y)$. An undirected graph $G = (V, E)$ is ***connected*** if there exists a path between any pair of vertices in $V$. A ***connected component*** for a graph $G$ is a maximal induced subgraph of $G$ which is connected. A vertice $v \in V$ is an ***articulation vertex*** or ***cut vertex*** of a connected undirected graph $G = (V, E)$ if the subgraph induced by $V-\{v\}$ is not connected. $G$ is ***biconnected*** if it contains no articulation point. A ***bicomponent*** (or ***block***) of $G$ is a maximal induced subgraph of $G$ which is biconnected. In this paper, the graphs we discussed are all connected.

The ***generalized-series-parallel*** (*GSP*) graphs are defined recursively as follows.

**Definition 1.** (1) A graph consisting of two vertices $u$ and $v$, and a single edge $(u,v)$ is a primitive *GSP* graph with terminals $u$ and $v$. (2) If $G1$ and $G2$ are two two-terminal *GSP* graphs with terminals $\{u1, v1\}$ and $\{u2, v2\}$, respectively, then the graph obtained by either of following three operations is a *GSP* graph:

(a) The ***series composition*** of $G1$ and $G2$ : identifying $v1$ with $u2$ and specifying $u1$ and $v2$ as the terminals of the resulting graph. (b) The ***parallel composition*** of $G1$ and $G2$: identifying $u1$ with $u2$ and $v1$ with $v2$, and specifying $u1$ and $v1$ as the terminals of the resulting graph. (c) The ***generalized-series composition*** of $G1$ and $G2$ : identifying $v1$ with $u2$ and specifying $u1$, $v1$ as the terminals of the resulting graph.

The family of ***series-parallel*** (*SP*) graphs consists of those *GSP* graphs that are obtained by using only the series and parallel compositions of Definition 1. A characterization of *SP* graphs can be obtained by the following definitions of two inverse operations of series and parallel compositions. Suppose that the degree of a vertice $w$ in $V(G)$ is two, the ***series reduction*** of two edges in series $e1 = (u, w)$ and $e2 = (w, v)$, is an operation of replacing $e1$ and $e2$ by a new edge $e = (u, v)$. The ***parallel reduction*** of two parallel edges (two edges with common end vertices) $e1 = (u, v)$ and $e2$ is an operation of replacing $e1$ and $e2$ by a new edge $e = (u, v)$.

**Lemma 2.1** [5, 8]. If $G$ is a connected *SP* graph then $G$ can be reduced to a single edge by a sequence of series and parallel reductions.

A *GSP* graph $G$ can be represented by a decomposition tree $T$ which is defined as follows.

**Definition 2.** (1) The tree consisting of a single vertex labeled by $e=(u,v)$ is a decomposition tree of the primitive *GSP* graph $G=(\{u,v\},\{(u,v)\})$. (2) Let $G$ be the *GSP* graph generated by some composition of two *GSP* graphs $G1$ and $G2$ and let $T1$ and $T2$ be the decomposition trees of $G1$ and $G2$ , respectively. Then, the decomposition tree $T$ of $G$ is the tree with the root $r$ labeled by an appropriate composition (may be "***G***", "***P***" or "***S***" depending on which composition is applied to generate $G$) and with $T1$ and $T2$ as the left and right child of $r$, respectively.

The definition of a decomposition tree of an *SP* graph is similar with Definition 2, but without internal node labeled by "*G*" since each *SP* graph is generated only by series and parallel compositions. Fig. 1. shows a *GSP* graph with its decomposition tree.

## 3. An Algorithm for Decomposing *GSP* Graphs.

In this section, we first discuss two important properties of *GSP* graphs and one result of *SP* graphs, then present our decomposition algorithm.

A characterization of *GSP* graphs can be derived by three specified operations: the series, parallel and generalized-series reductions, which can be viewed as the inverse operations of series, parallel and generalized-series compositions (the series and parallel reductions are defined in Section 2). The ***generalized-series reduction*** of two edges $e1=(u,v)$ and $e2=(v,w)$, where $w$ is a ***pendent*** vertex (the vertex with one degree), is an operation of replacing $e1$ and $e2$ by a new edge $e=(u,v)$.

Suppose that *T* is a decomposition tree of the given *GSP* graph *G*. Consider the following scheme: Finding some internal node *u* of *T* whose left and right child represent two edges *e1* and *e2* of *G*. Then applying some appropriate reduction to *e1* and *e2* according to the label of *u* (that is, if *u* is labeled by "***G***" then the generalized-series reduction is applied, the other cases for "***P***" and "***S***" can be described similarily). By definition of reduction, *e1* and *e2* are replaced by the new edge *e*. Thus the original graph *G* become another "smaller" graph and has the decomposition tree obtained from *T* by replacing the subtree rooted at *u* whose two children are *e1* and *e2* by the new leave node *e*. Clearly, if we repeat executing above scheme then *G* can be finally reduced to a single edge. Thus we conclude that a decomposition tree of *G* corresponds to a reducing sequence which can reduce *G* to a single edge. Note that such reducing sequence is not unique.

Conversely, given a reducing sequence δ which can reduce *G* to a single edge, we can construct the unique decomposition tree *T* corresponding to δ. The construction of *T* follows the process that reduces *G* to a single edge by δ. We assume that during the reduction process each edge is associated with a tree structure and when the reduction process terminates, the tree associated with the single remaining edge is the decomposition tree of *G*. At the beginning, each edge *e* of *G* is associated with a tree consisting of a single node labeled by *e*. When a generalized-series reduction is applied to two edges *e1* and *e2* we create a new node *u* labeled by "***G***" and let the trees associated with *e1* (resp. *e2*) be the left (resp. right) child of *u*. The cases for applying a series or parallel reduction is described similarily. Combining above results we have the following lemma.

**Lemma 3.1.** A graph *G* is a *GSP* graph if and only if it can be reduced to a single edge by a sequence of series, parallel and generalized-series reductions.

Assume that *G* is a *GSP* graph, then *G* can be generated by a sequence of compositions $\delta 1, \delta 2, \ldots \delta k$. If some $\delta i$ is the generalized-series composition applied to two *GSP* graphs *G1* and *G2* with terminals $\{u1, v1\}$ and $\{u2, v2\}$, respectively, the vertex $v1$ (= $u2$) will be a cut vertex of *G*. From the above observation, it is easy to derive the following characterization of *GSP* graphs.

**Lemma 3.2** [13]. A graph *G* is a *GSP* graph if and only if each block of *G* is an *SP* graph.

Eppstein [6] presents an efficient parallel algorithm for recognizing biconnected *SP* graphs. Given a biconnected *SP* graph *G* with an open ear decomposition $D = \{P0, P1, \ldots, Pr\text{-}1\}$( the detail implementation of finding an open ear decomposition is described in [16]) the algorithm can construct a decomposition tree, corresponding to a reducing sequence which can reduce *G* to *P0*, where *P0* is an edge of *G*. By the property of open ear decompositions and the algorithm for constructing them [16], we can easily modify the algorithm to construct an open ear decomposition with *P0* being any arbitrarily selected edge. From these observations, the following result is obtained.

**Lemma 3.3.** Given a biconnected *SP* graph *G* and an arbitrarily selected edge *e* of *G*. We can construct in parallel a decomposition tree corresponding to a reducing sequence δ, such that *G* can be reduced to *e* by δ.

According to Lemma 3.2 and the algorithm presented in [6] we can easily recognize *GSP* graphs, but for constructing their decomposition trees efficiently we need some useful strategy described in our algorithm. Before preceding to present our algorithm, we provide the following definitions which are necessary for the construction of a decomposition tree of a *GSP* graph. Suppose that *a1*, *a2*,..., *ak* are the cut vertices of *G* and *B1*, *B2*,..., *Bl* are the blocks of *G*. The block-cut vertex tree $BT$ is defined as follows [2]. The vertex set of $BT$ is {$a1$, $a2$,..., $ak$, $b1$, $b2$,..., $bl$} and ($ai$, $bj$) is an edge of $BT$ if and only if $ai$ is a vertex of $Bj$. In addition, we call each $bi$ (resp. $ai$) a ***block-vertex*** (resp. ***non-block vertex***) of $BT$. Let $BT'$ be the rooted tree by selecting one block-vertex $br$ of $BT$ as the root. Then, for each block-vertex $bi$ $(i \neq r)$ there is a unique directed path $P$ from $bi$ to the root $br$. If $bj$ is the first block-vertex of $BT'$ in which $bi$ encounters in $P$, we call $bj$ the ***parent*** of $bi$ (denote as $Par(bi)$) and $Bj$ (resp. $Bi$) is the ***parent block*** (resp. a ***child block***) of $Bi$ (resp. $Bj$). Specially, we denote $Child(Bi)$ as the set of child blocks of $Bi$ and call the block with no child block as the ***leave block***.

Algorithm *Decomposing GSP*

Step 1. Find the blocks $B1$, $B2$,..., $Bk$of $G$.

Step 2. /* Prepare for constructing the decomposition tree of $G$ */

2-1. Construct the block-cut vertex tree $BT$ of $G$.

2-2. Transfer $BT$ to a rooted tree $BT'$ by selecting arbitrary one block-vertex $br$ as the root. /* thus $Br$ is the root block of $G$ */

/* The following Steps: 2-3, 2-4, 2-5 and Step 3 are executed in parallel for each block $Bi$ $(1 \le i \le k)$ */

2-3. Find the parent and child blocks for $Bi$ by using $BT'$.

2-4. For the cut vertex $v$ connecting to $Par(Bi)$ select one edge $e=(v,w)$ of $Bi$ and mark it as the ***main edge*** $ti$. For the root block $Br$ select arbitrary one edge as main edge.

2-5. For each cut vertex $v$ connecting to some child of $Bi$ select one edge $e=(u,v)$ of $Bi$ and mark it as the ***reducing edge*** $rv$. Computes the number $m$ of the edges $tj$'s, where $tj$ is the main edge of some child block of $Bi$ which is connected to $Bi$ by $v$. Then, constructs a left-skew binary tree structure $Rv$ with internal nodes labeled by "$G$", and with the leave nodes labeled by $tj$'s and $e$ as follows: ordering those edges $tj$'s from 1 to $m$ and constructing $m$ "***G***" nodes denoted by ***G1***, ***G2***,..., ***Gm***, such that the right child of each ***Gk*** $(1 \le k \le m-1)$ and ***Gm*** is the node ***Gk+1*** and $e$, respectively, and the left child of each ***Gk*** $(1 \le k \le m)$ is the node labeled by $tj$ (according to the ordering associated with it).

/* there may be some edge $e$ of $Bi$ which is marked as main edge and also reducing edge, but it does not effect the results of this algorithm */

Step 3. Apply the recognition of $SP$ graphs algorithm to $Bi$ and construct its decomposition tree $Ti$ corresponding to a reducing sequence which can reduce $Bi$ to its main edge. If one of the blocks is not an $SP$ graph, reject. /* $G$ is not a $GSP$ graph */

Step 4. /* Construct the decomposition tree $T$ of $G$ */

/* Step 4-1 and 4-2 are executed in parallel for each reducing eage and main edge, respectively */

4-1. For each reducing edge $rv=(u,v)$ of $Ti$, if $rv$ is also marked as the reducing edge for $u$, then replacing $rv$ by the root of $Ru$ and replacing the edge ($u$, $v$) (appears as some leave of $Ru$) by the root of $Rv$. Otherwise, replace $rv$ by the root of $Rv$.

4-2. Replace each $ti$ node by the root of $Ti$.

Fig. 2 shows the construction of a decomposition tree for a $GSP$ graph $G$ with four blocks $Bi$ $(1 \le i \le 4)$ in Fig.1. We first select $B1$ as the root in Step 2-2 and find $Child(B1)$ = $B2$ and $Child(B2)$ = {$B3$, $B4$} in Step 2-3. In Step 2-4, the edges $t1 = a$, $t2 = d$, $t3 = h$ and $t4 = g$ are selected as the main edges of $Bi$ $(1 \le i \le 4)$. Step 2-5 selects the edges $rv3 = b$ and {$rv4 = d$, $rv7 = f$} as the reducing edges of $B1$ and $B2$, respectively and then constructs the tree structures $Rv3$, $Rv4$ and $Rv7$ for the cut vertices $v3$, $v4$ and $v7$. Step 3 constructs in parallel the decomposition trees $Ti$'s $(1 \le i \le 4)$, where each $Ti$ corresponding to a reducing sequence which can reduce $Bi$ to its main edge $ti$. Finally, the decomposition tree $T$ of $G$ can be generated in Step 4 by replacing $rv3$, $rv4$ and $rv7$

by $Rv3$, $Rv4$ and $Rv7$ (since each reducing edge is marked for only one cut vertex), and replacing each $ti$ by the root of $Ti$ $(1 \le i \le 4)$.

We first show the correctness of the algorithm. If $G$ is not a *GSP* graph, by Lemma 3.2 some block of $G$ is not an *SP* graph, then $G$ will be rejected in Step 4. Conversly, if $G$ is a *GSP* graph, the blocks of $G$ are all *SP* graphs by Lemma 3.2. We will show in the following claim that a decomposition tree $T$ of $G$ can be constructed correctly by our algorithm. This claim can be proved by induction on the number $k$ of blocks.

**Claim A**: If $G$ is a *GSP* graph. The algorithm can construct a decomposition tree $T$ of $G$, such that $T$ corrseponds to a reducing sequence which can reduce $G$ to the main edge of the root block $Br$ selected in our algorithm.

If $k = 1$, $G$ contains only one block which will be selected as the root $Br$ in Step 2-2. By Lemma 3.3, we can construct a decomposition tree corresponding to a reducing sequence which can reduce $Br$ to its main edge, and thus the claim is correct. Assume the claim is correct for any *GSP* graph with the number of blocks less than $k$. Now, let $G$ be a *GSP* graph with $k$ blocks $B1, B2, \ldots, Bk$. We select one block $Br$ as the root and consider the case of removing $Br$ from $G$. Then, the resulting graph contains several connected components $C1, C2, \ldots, Cm$, where $m < k$. Clearly, the number of blocks of each $Ci$ $(1 \le i \le m)$ is less than $k$. According to induction hypothesis, our algorithm can constrcut a decomposition tree $Tci$ of $Ci$ such that $Tci$ corresponds to a reducing sequence which can reduce $Ci$ to the main edge $ti = (ui,vi)$ of the root block $Bi$ of $Ci$. The block $Bi$ is some child block of $Br$ which is connected to $Br$ by the cut vertex $vi$. After reducing each $Ci$ to $ti$ (the edge $ti$ associated with the decomposition tree $Tci$) the graph $G$ is reduced to another graph $G'$, where $G'$ contains the block $Br$ and the edges $ti$'s (each of which is connected to $Br$ by $vi$). Note that the end vertex $ui$ of each $ti = (ui,vi)$ is a pendent vertex. Then, we reduce each $ti$ by applying a generalized series reduction to the edges $ti$ and some reducing edge $rvi$ of $Br$, where $rvi$ and $ti$ are two edges connected by $vi$. Such reduction can be represented by a tree structure $Rvi$ generated in Step 2-5. When each $ti$ $(1 \le i \le m)$ has been reduced, the resulting graph contains only one block $Br$. By Lemma 3.3, our algorithm can construct a decomposition tree $Tr$ corresponding to a reducing sequence which can reduce $Br$ to its main edge. Finally, replacing each reducing edge $rvi$ of $Tr$ by some tree structure and replacing each $ti$ by the root of $Tci$, the decomposition tree $T$ of $G$ can be generated in Step 4. This is a decomposition tree corresponding to a reducing sequence which can reduce $G$ to the main edge of $Br$. By induction we prove Claim A, and hence, we have the following theorem.

**Theorem 3.4.** The algorithm *decomposing GSP* can recognize a *GSP* graph and construct its decomposition tree correctly.

Now, we show that the time complexity of the algorithm *Decomposing GSP* is $O(\log n)$ time with $C(n,m)$ processors.

In Step 1, finding the blocks of $G$ takes $O(\log n)$ time with $C(m,n)$ processors on a CRCW PRAM [4].

In Step 2-1, the block-cut vertex tree $BT$ can be constructed in $O(1)$ time with $O(k)$ processors, where $k$ is the number of the blocks of $G$. This can be simulated in $O(\log k)$ time with $O(k/\log k)$ processors by Brent's theorem [11]. In the following steps, the implementations which take $O(1)$ time with $O(k)$ processors can apply above simulatting result.

Step 2-2 constructs the rooted tree $BT'$ from $BT$ by using the Eulerian tour technique described in [12]. It takes $O(\log k)$ time with $O(k/\log k)$ processors on an EREW PRAM.

In Step 2-3, the parent block $Bj$ of $Bi$ $(1 \le i \le k)$ can be found by using $BT'$ since $bj = par(par(bi))$. This can be done in constant time with $O(k)$ processors. For finding the child blocks of $Bi$, we maintain the table containing $k$ entries corresponding to $B1, B2, \ldots, Bk$ in which each entry has two fields that record the index $i$ of $Bi$ and the index of its parent block. We sort the entries by index values of their second fields, thus divide the table into several blocks $b(1), b(2), \ldots, b(m)$, $m < k$, such that the entries of $b(i)$ records all the bolcks of $G$ with the same parent block $Bi$. Thus the child blocks of $Bi$ can be found in constant time. We make use of the parallel sorting algorithm described in [7], which runs in $O(\log n)$ time with $O(n/\log n)$ processors on a CRCW PRAM to sort $n$ numbers in the range $[1, \ldots, n^{O(1)}]$.

In Step 2-4, the cut vertex $v$ in $Bi$ connecting to $Par(Bi)$ can be determined in $O(1)$ time and thus the main edge of $Bi$ $(1 \le i \le k)$ can be selected in constant time with $O(k)$ processors.

Step 2-5 first selects an edge $e = (u,v)$ of $Bi$ for each cut vertex $v$ connecting to some child of $Bi$, and mark it as a reducing edge $rv$. This can be done in constant time with $O(k)$ processors. Then, computes the number of the edges $tj$'s, where $tj$ is the main edge of some child block of $Bi$ which is connected to $Bi$ by $v$, by using optimal parallel prefix sum computation [10]. The ordering of the edges $tj$'s and making up a left-skew tree structure $Rv$ can be done by optimal parallel list ranking [10]. Thus Step 2-5 can be done in $O(\log n)$ time with $O(n / \log n)$ processors on an EREW PRAM.

In Step 3, the recognition of *SP* graphs takes $O(\log n)$ time within $C(n,m)$ processors [6].

In Step 4, for each reducing edge $rv = (u, v)$ of $Bi$, checking whether $rv$ is also the reducing edge selected by $u$ can be determined in constant time. It is clear that the other implementations of Step 4-1 and 4-2 can be achieved in $O(1)$ time with $O(k)$ processors. Hence, we have the following theorem.

**Theorem 3.5.** The algorithm *decomposing GSP* can recognize a *GSP* graph and construct its decomposition tree in $O(\log n)$ time with $C(m,n)$ processors on a CRCW PRAM.

## 4. Some Properties of *GSP* Graphs.

In this section, we derive some properties of *GSP* graphs from the results obtained in previous sections.

Suppose that $G$ is a *GSP* graph. Then, by definition there exists two special vertices $u$ and $v$ as the two terminals of $G$. From the proof of Lemma 3.1, we observe that $G$ is a *GSP* graph with terminals $\{u,v\}$ if $G$ can be reduced to a single edge $e = (u, v)$ by a sequence of series, parallel and generalized-series reductions. Hence, we could not select arbitrarily any two vertices as the terminals of the given *GSP* graph. Fig. 3. shows that $G$ is a *GSP* graph with terminals $\{v2, v3\}$ or $\{v3, v6\}$, but is not a *GSP* graph with terminals $\{v2, v5\}$ since no reducing sequence can reduce $G$ to the edge $e = (v2, v5)$.

**Theorem 4.1.** Let $G$ be a *GSP*. Then, for any edge $e = (u, v)$ of $G$, $G$ is a *GSP* graph with with terminals $\{u,v\}$.

*Proof.* Let $G$ be a *GSP* graph let $e = (u, v)$ be an edge of some block $Br$. If we select $Br$ as the root and mark $e$ as the main edge of $Br$. From the proof of Theorem 3.4, we can construct a decomposition tree corresponding to a reducing sequence which can reduce $G$ to the edge $e = (u, v)$. Hence, $G$ is a *GSP* graph with terminals $\{u, v\}$.

**Theorem 4.2.** Let $u$ and $v$ be two vertices of $G$. $G$ is a *GSP* graph with terminals $\{u, v\}$ if and only if $G' = G + (u, v)$ is a *GSP* graph.

*Proof.* If $G$ is a *GSP* graph with terminals $\{u, v\}$. It is clear that $G' = G + (u, v)$ is a *GSP* graph since $G'$ is generated by applying a parallel composition to $G$ and the edge $e' = (u, v)$ (each edge is a primitive *GSP* graph with terminals $u$ and $v$).

Conversly, suppose that $G' = G + (u, v)$ is a *GSP* graph. Let the adding edge $e' = (u, v)$ be an edge in some block $Br$ of $G'$. If $G'$ contains only one block $Br$, then $G$ is an *SP* graph by Lemma 3.2. According to Lemma 3.3, we can construct a decomposition tree $T$ of $G'$ corresponding to a reducing sequence δ which can reduce $G'$ to the edge $e'$. Since $G'$ is biconnected, thus during the reducing process generated by δ, the resulting graph remains as being biconnected. From this observation, we can conclude that the root $r$ of $T$ must not be labeled by "*S*" (which contradicts that $G'$ is biconnected) and thus $r$ is labeled by "*P*". Moreover, consider the path of $T$ from the leave $e'$ to the root $r$. The internal nodes of such path are all labeled by "*P*" (if there exists some internal node labeled by "*S*" then vertex $u$ (or $v$) will be removed by a series reduction, and hence contradicts that $G'$ can be reduced to $e' = (u, v)$). Based on the structure of $T$, we can apply a reducing scheme described in the proof of Lemma 3.1, to reduce $G'$ to another graph. Such reduced graph has a decomposition tree $T'$ which can be obtained from $T$ by replacing each subtree of $T$ with the root labeled by "*S*", by an appropriate edge (this edge is generated by executing a reducing sequence corresponding to above subtree). Hence, all the internal nodes of $T'$ are labeled by "*P*". Then, we can generate another "equivalent" decomposition tree from $T'$: if $e'$ has been one child of the root $r$ of $T'$, then $T'$ is the decomposition tree we need, otherwise we can further transfer $T'$ to another tree, such that $e'$ is one child of $r$. Note that this transform is legal since the internal nodes of $T'$ have the same label, i.e. the same operators are applied in a reducing sequence corresponding to $T'$. Hence all the different binary decomposition trees generated by the same number of internal "*P*" nodes and the same set of leaves with $T'$ can be viewed as "equivalent". By above observation, this implies that there exists a decomposition tree $T$ of

$G'$ with the root $r$ labeled by "*P*", and $e'$ and the subtree *TG* are the two children of $r$, where *TG* corresponds to a reducing sequence which can reduce $G$ to a single edge $e=(u,v)$. Thus *TG* is a decomposition tree of $G$ and $G$ is a *GSP* graph with terminals $\{u, v\}$.

On the other hands, suppose that $G'$ contains more than one block. We can select *Br* as the root and select the edge $e'=(u,v)$ as the main edge in Step 2-4 of our algorithm, such that $e'$ can not also be selected as a reducing edge. This can be achieved since adding $e'$ to $G$ makes the number of the edges of *Br* larger than one. From the proof of Theorem 3.3, we know that a decomposition tree $T$ of $G'$ is constructed from the decomposition tree *Tr* of *Br* by replacing each reducing edge of *Br* by its corresponding tree structure, and then replacing the main edges of *Child*(*Br*) by their associated decomposition trees. Moreover, by the proof of Theorem 3.4, there exists a reducing sequence which can reduce $G'$ from the leaves to the root until *Br* is the only remaining block. Note that any reduction in above reducing sequence can not replace $e'$ since $e'$ is not a reducing edge of *Br*. Combining above observations and the result shown in previous paragraph, there exists a decomposition tree $T$ of $G'$ with the root $r$ labeled by "*P*" and with $e'$ and the subtree *TG* as the two children of $r$, such that *TG* corresponds to a reducing sequence which can reduce $G$ to a single edge $e=(u,v)$. Thus $G$ is a *GSP* graph with terminals $\{u,v\}$.

## 5. Conclusion.

In this paper, we present an effecient parallel algorithm to construct a decomposition tree for the given *GSP* graphs. It takes $O(\log n)$ time with $C(m,n)$ processors on a CRCW PRAM. From this algorithm, we can further obtain another result by considering the special input instance of the algorithm. Recall that trees are contained within the class of *GSP* graphs. If the input graph is known to be a tree, then its decomposition structure can be constructed by our algorithm without executing Step 1 and Step 3 since each block of trees contains only one edge. Moreover, because all of the steps in our algorithm can be done optimally except for above two steps, the decomposition structure of the given tree can be constructed in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM. According to the result shown in [14], that given a decomposition tree of a decomposable graph all problems satisfy the "regular" property can be solved by a cost optimal parallel algorithm. Combining these results we have the following conclusion: ***Any "regular" problems for the given tree can be solved in O(log n) time with O(n/log n) processors on an EREW PRAM.***

Yu, Tseng and Lin [15] shows that some problems for finding a maximun weight independent set, a maximum weight matching and a minimum weight dominating set on trees can be solved in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM. In fact, above three problems are all "regular", and thus it is a special case of our results.

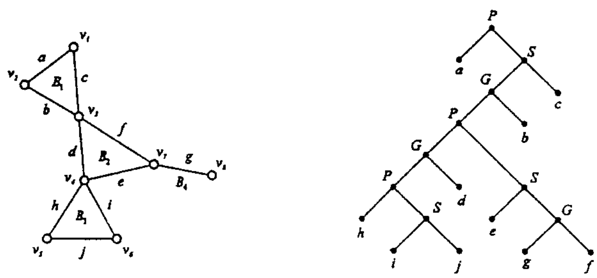

Fig. 1 A *GSP* graph with its decomposition tree.

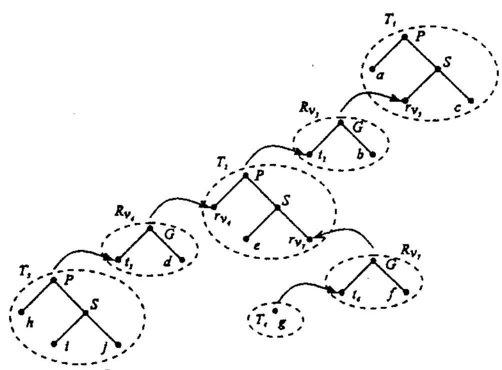

Fig. 2 The decomposition tree $T$ of a *GSP* graph shown in Fig. 1 can be constructed from step 4.

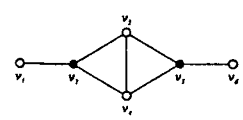

Fig. 3 $G$ is not a *GSP* graph with respect to $v_1$ and $v_4$.

## References


[1] K. Abrahamson, N. Dadoun, D. G. Kirkpatrick, and T. Przytycka, "A simple parallel tree contraction algorithm," J. Algorithms 10, 1989, pp. 287-302.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The design and analysis of computer algorithms," Addison-Wesley, Reading, MA, 1974.

[3] M. W. Bern, E. L. Lawler, and A. L. Wong, "Linear-time computation of optimal subgraphs of decomposable graphs," J. Algorithms 8, No.2 (1987), pp. 216-235.

[4] R. Cole and R. Thurimella, "Approximate parallel scheduling, II: application to optimal parallel graph algorithms in logrithmic time, " Inform. Comput. 91(1991), pp 1-47.

[5] R. J. Duffin, "Topology of series parallel networks, " J. Math. Appl. 10 (1965), pp. 303-318.

[6] D. Eppstein, "Parallel recognition of series-parallel graphs," Inform. Comput. , 98 (1992), 41-55

[7] S. Rajasekaran and J. Reif, "Optimal and Sublogarithmic time randomized parallel sorting algorithms," SIAM J. Comput 18, No. 3 (1989), pp. 594-607.

[8] X. He, "Efficient parallel algorithms for series parallel graphs," J. Algorithms 12, 1991, pp.409-430.

[9] E. Hare, S. Hedetniemi, R. Laskar, K. Peters and T. Wimer, "Linear-time computability of combinatorial problems on generalized-series-parallel graphs," Discrete Algorithms and Complexity, Academic Press, 1987, pp. 437-455.

[10] J. Ja'Ja' "An introduction to parallel algorithms," Addison Wesley, 1992.

[11] R. M. Karp and V. Ramachandran, "Parallel algorithms for shared memory machines", in Handbook of Theoretical Computer Science, North-Holland, Amsterdan, 1990, pp. 869-941.

[12] R. E. Tarjan and U. Vishkin. "Finding biconnected components and computing tree functions in logrithmic parallel time," SIAM J. Comput 14, No. 4 (1985), pp 862-874.

[13] T. V. Wimer and S. T. Hedetniemi, "K-terminal recursive families of graphs," Congressue Numerantium, 63(1988), pp. 161-176.

[14] Shi-Jim Yain, "Optimal parallel algorithms for decomposable graphs," Master thesis, 1993, Institute of Computer Science and Electrical Engineering, National Central University, Taiwan, R.O.C.

[15] Ming-Shing Yu, Lin-Yu Tseng, and Jiunn-Horng Lin, "Optimal parallel algorithm for some problems on trees," International Conference on Parallel Processing, 1992, pp. 160-163.

[16] Y. Maon, B. Schieber, and U. Viskin, "Parallel ear decomposition search (EDS) and s-t numbering in graphs", Theoret. Comput. Sci., 47(1986), pp. 277-298.