SUCCESSIVE APPROXIMATION IN PARALLEL GRAPH ALGORITHMS*

Donald Fussell and Ramakrishna Thurimella

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-88-42 November 1988

* This work was supported in part by the Office of Naval Research under Contracts N00014-86-K-0597 and N00014-86-K-0763.

Successive Approximation in Parallel Graph Algorithms[1]

Donald Fussell
Ramakrishna Thurimella

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712.

Abstract

The notion of successive approximation is introduced in the context of parallel graph algorithms. The implementation of graph algorithms on Leighton's mesh of trees network model is considered. The implementations that have appeared so far in the literature are relatively straightforward. A common characteristic of these algorithms is that, in each iteration, for each vertex v, at most one edge is selected from the edges incident on v. This selection is based purely on local information such as the weights of the edges incident v or the labels of the neighboring vertices of v etc. As this sort of information appears on the same row of a mesh, these algorithms lend themselves to a direct implementation. In this paper we present an implementation of the open ear decomposition algorithm of Maon, Schieber and Vishkin. Some applications of open ear decomposition include parallel planarity testing, triconnectivity and 4-connectivity testing. This algorithm is different from the other algorithms considered for implementation on a mesh of trees in that a direct implementation is ruled out due to the communication problems posed by the network. Our implementation uses a technique of successive approximation. The process starts by finding an open ear decomposition of a subgraph of at most $2n$ edges: the edges of two edge-disjoint forests of G. Each subsequent iteration uses the decomposition from the previous step to obtain an open ear decomposition of an enlarged subgraph. This enlarged subgraph consists of the edges that received an ear label in the previous step together with at least as many new ones. Therefore the process converges in $O(\log n)$ iterations. The decomposition algorithm for each iteration can be distributed on the network. The whole algorithm takes $O(\log^3 n)$ time using $O(n/\log n \times n/\log n)$ processors. Assuming adjacency matrix representation of the graph, the achieved speedup is $O(\log n)$ factor off the optimal, which is the best known.

---

[1]This work was supported in part by the Office of Naval Research under Contracts N00014-86-K-0597 and N00014-86-K-0763.

1 Introduction

Parallel algorithms for graph problems have been the subject of much interest in recent years. Since graph algorithms are of such widespread utility as building blocks for other algorithms, it is of particular interest to determine efficient ways to perform these computations on universal parallel machines i.e. those machines which can simulate any other parallel machine built using the same hardware resources within a polylog time factor. The most commonly studied universal machine is the Parallel RAM (PRAM) shared memory model, which is the most powerful parallel machine. In recent years algorithms for finding minimum spanning forests, connected components and biconnected components in $O(\log n)$ time with $O(n+e)$ processors have been found that run on CREW (Concurrent Read Exclusive Write) PRAM [12],[13].

While PRAMs are useful theoretical models for studying parallel algorithms, they are of limited interest in practice due to the difficulty of constructing shared memory machines with very large numbers of processors. While not as powerful as shared memory machines, network models are of interest in practical terms. A network model is a restrictive parallel computation model which explicitly models communication and granularity constraints. The communication constraint is due to the bounded degree of the vertices of the network. That is each processor in the network can communicate to only a constant number of other processors in unit time. The granularity constraint is caused by the conflicts that arise when several processors want to access the variables that are stored in the same local memory. Such machines include Leighton's mesh of trees and tree of meshes [6] among others. See Fig. 1.

In this paper, the design of graph algorithms on the mesh of trees model is considered. We argue that the idea of an approximate solution arises naturally while implementing graph algorithms on networks and suggest, therefore, a method of successive approximation as an implementation technique in this context.

The following algorithms have been considered for implementation on a mesh of trees. Leighton [7] has shown that the algorithm of Hirschberg and Volper [2] for finding connected components and minimum spanning forest of an n vertex graph can be implemented on a mesh of trees of size $n \times n$ to run in $O(\log^2 n)$ time. Huang [3] later improved the processor-bound to $O(n^2/\log^2 n)$. Additionally, he gives implementations on a mesh of trees that have the same time and processor complexity for: a) the biconnected component algorithm of Tarjan and Vishkin; b) an algorithm that finds a directed spanning forest. All these algorithms, with the exception of the biconnected component algorithm, have the following property in common: *in each iteration, for each vertex v, an edge incident on v is picked based purely on local information.* Implementation of an iteration of these algorithms typically involves, a row of processors performing together a simple operation on the values stored in the local memory of the processors on that row. In the case of the minimum spanning tree algorithm, for example, each vertex v picks an edge incident on it that has the least weight. In an implementation that means finding the minimum of at

most n values that are stored on same row of the mesh. These operations, usually, take no more than $\log n$ steps. In the case of the biconnected component algorithm, Tarjan and Vishkin make an important observation about the blocks of a graph which implies the property mentioned above. See the construction of G'' for details [13].

An open ear decomposition of an undirected graph, roughly speaking, is a partition of $E(G)$ into an ordered collection of edge disjoint simple paths that satisfy certain requirements. Ear decomposition was defined by Whitney in [14] in 1932. Recently, Lovasz showed [8] that the problem of ear decomposition has a parallel algorithm which runs in poly-log time using a polynomial number of processors and is therefore in the class NC. Later Maon, Schieber and Vishkin (MSV) [9], and Miller and Ramachandran [10] gave algorithms for finding an open ear decomposition which run in $O(\log n)$ time using $n+m$ processors on a CRCW PRAM. We consider, in this paper, an implementation of MSV's algorithm. There are two important reasons for this choice: first, it does not satisfy the property mentioned in the previous paragraph and therefore poses challenging implementation problems; second, the idea of partitioning the edges into open ears has been found to be extremely useful in the design of parallel graph algorithms.

Maon et al. conjecture that, Depth First Search (DFS) being "inherently sequential", open ear decomposition may prove to be as useful in the design of parallel graph algorithms as DFS in the design of sequential graph algorithms. In fact, parallel graph algorithms that are based on open ear decomposition include algorithms for testing a graph for planarity [5], triconnectivity [1], [11] and 4-connectivity [4].

A graph has an open ear decomposition iff it is biconnected [14]. An ear decomposition of a connected, bridgeless graph G is considered to be open if the number of closed ears in that decomposition is equal to the number of blocks of that graph. Therefore an ear decomposition in which the number of closed ears is greater than the number of blocks of G is, in a sense, only an approximation to an open ear decomposition. In other words, between two ear decompositions of a graph, the one with the least number of closed ears is better. Our algorithm starts by finding an open ear decomposition of a subgraph G' that is the union of a spanning tree T and a spanning forest F , respectively, of G and $G-T$. Notice that if G is bridgeless then so is G'. The open ear decomposition of G' can be made an ear decomposition of G by making every edge of $G-G'$ a single-edge ear. But such an ear decomposition of G is not open if the number of blocks of G' is greater than that of G. However, it is only our initial approximation. We guarantee that each iteration reduces the number of closed ears by a fraction. Notice that G' can have at most $\lfloor 2(n-1)/3 \rfloor$ blocks as it has no more than $2(n-1)$ edges, and any biconnected component should have at least 3 edges. Therefore the process converges in $O(\log n)$ iterations. In essence, we start with an ear decomposition that can have up to $O(n)$ closed ears and reduce the number of closed ears by a fraction in each iteration. In addition, each iteration uses the "approximate" open ear decomposition from the previous iteration in a crucial way when reducing the number of closed ears. Hence the name successive approximation.

Our implementation runs in $O(\log^3 n)$ time on a mesh of trees of size $O(n/\log n \times$

$n/\log n$). The speedup achieved, assuming the adjacency matrix representation of the graph, is $O(\log n)$ factor off the optimal, which is the best known.

2 Definitions and Notation

$V(G)$ and $E(G)$ stand, respectively, for the vertex set and the edge set of a graph G. Assume $|V(G)| = n$ and $|E(G)| = e$. The subgraph G' *induced* by $V'(G) \subseteq V(G)$ is the subgraph consisting of the edges (x, y) of G where $\{x, y\} \subseteq V'(G)$. For any two graphs G_1 and G_2 with $V(G_2) \subseteq V(G_1)$, $G_1 + G_2$ ($G_1 - G_2$) refers to the graph with $V(G_1)$ as the vertex set and $E(G_1) \cup E(G_2)$ (resp. $E(G_1) - E(G_2)$) as the edge set.

A vertex (edge) in a connected graph is an *articulation point* (resp. *bridge*) if its removal results in disconnecting the graph. A connected graph G is *r-connected* if at least r vertices must be removed to disconnect the graph. A *biconnected graph* (or a *block*) is 2-connected.

An *ear decomposition starting with a node* P_0 of an undirected graph G is a partition of $E(G)$ into an ordered collection of edge-disjoint simple paths $P_0, P_1, ..., P_k$ such that P_1 is a simple cycle starting and ending at P_0, and each end point P_i, $1 \leq i \leq k$, is contained in some P_j for some $j < i$. Each of these paths P_i is an *ear*. P_0 is called the *root* of the decomposition and is referred to as r. If the two end nodes of a path P_i do not coincide, then P_i is an *open ear*, otherwise the ear is *closed*. An *open ear decomposition* is one in which every ear P_i, $1 < i \leq k$, is open. An ear is a *single-edge ear* if it consists of a single edge, otherwise it is a *many-edge ear*. An (open) ear decomposition (open ear decomposition) exists for a graph G if G is bridgeless (resp. biconnected) [14]. By an open ear decomposition of a bridgeless graph G, that is not biconnected, we mean an ear decomposition in which there is exactly one closed ear for each block of G. See Fig. 2 (Fig. 3) for an example of an open ear decomposition of a graph that is biconnected (resp. bridgeless but not biconnected).

Let G be some connected graph that is not necessarily biconnected. Let G' be a subgraph such that $V(G) = V(G')$ and $E(G') \subseteq E(G)$. The subgraph G' is called a *biconnectivity preserving subgraph* if it satisfies the following condition. For any two nodes u and v in G, if there are two vertex disjoint paths between u and v in G then there are two vertex disjoint paths between u and v in G'.

3 Open Ear Decomposition on a Mesh of trees

3.1 MSV's Algorithm

The following algorithm, from [9], finds an ear decomposition of an undirected graph on a CRCW PRAM. Assume each edge $(x, y) \in E(G)$ has a serial number $1 \leq serial(x, y) \leq e$.

Algorithm 1 *ear-decomposition*

1. Find a spanning tree T and fix a vertex to be the root r of the tree . Also, find the level of each vertex in the tree with respect to r.
2. For each nontree edge (u, v), find $lev(lca(u, v))$: the level of the least common ancestor of u and v in T. The ear number $ear(u, v)$ of (u, v) is a 2-tuple $\langle lev(lca(u, v)), serial(u, v)\rangle$.
3. Assume we have a lexicographic ordering on the 2-tuples, defined the usual way. Now, for each $(p, q) \in E(T)$, $ear(p, q)$ is the minimum of $\{ear(u, v) \mid (u, v) \in E(G_T)$ and the fundamental cycle created by (u, v) when included in T contains $(p, q)\}$

Applying the above algorithm may result in some closed ears. Refer to [9] for an example that illustrates this phenomenon. Let *open-ear*(u, v) represent the ear label of the edge (u, v) in an open ear decomposition. It turns out that by having a meaningful second component, instead of an arbitrary serial number as in the above algorithm, it is possible to obtain an open ear decomposition. The following algorithm, from [9], is based on this idea.

Algorithm 2 *open-ear-decomposition*

1. Find an ear decomposition that is not necessarily open.
2. Construct n bipartite graphs $H_{v_1}, H_{v_2}, ..., H_{v_n}$ corresponding to the n vertices $v_1, v_2, ..., v_n$ as follows. For $x \in \{v_1, ..., v_n\}$ define $V(H_x)$ to be the union of $V'(H_x)$ and $V''(H_x)$ where $V'(H_x) = \{[u, v] \mid (u, v) \in E(G - T)$ and $lca(u, v) = x\}$ and $V''(H_x) = \{[w, x] \mid w$ is a son of x in $T\}$.

   Refer to the vertices of $V'(H_x)$ and $V''(H_x)$ as *nontree* and *tree* vertices, respectively. The edges are added as shown in the following. Let (w, x) be the first edge in the unique path in T from $lca(u, v)$ to u. Then the edge $([u, v], [w, x]) \in E(H_x)$.
3. For each H_{v_i}, $1 \le i \le n$, compute the connected components and a spanning forest.
4. Root the spanning tree in each component of H_x at a vertex $[w, x]$ such that

   (a) $[w, x]$ is a tree vertex in H_x, i.e. $[w, x] \in V''(H_x)$ and

   (b) $ear(w, x).1 < lev(x)$ where $ear(w, x).1$ is the first component of the ear label of (w, x) found in Step 1.

   For the proof of existence of such a tree vertex $[w, x]$, refer to the main lemma in [9].
5. Preorder the vertices of the spanning tree.

6. Define *open-ear*(u,v) for all nontree edges as follows. Let $lca(u,v) = w$ and let $pre_H([u,v])$ be the preorder label $[u,v] \in V'(H_w)$ received in the previous step. Then *open-ear*$(u,v) = \langle lev(w), pre_H([u,v])\rangle$.

7. Find the ear labels of tree edges using the new ear labels of nontree edges by repeating Step 3 of Algorithm 1.

3.2 An Algorithm on a Mesh of Trees

The difficulty in implementing MSV's open ear decomposition algorithm directly is that it requires the least common ancestor (lca) of potentially $O(n^2)$ pairs of vertices. More precisely, for a given graph G and a spanning tree T of G, MSV's algorithm requires the value of $lca(x,y)$ for each $(x,y) \in E(G-T)$. If the graph is dense, then the value of $|e|$ and, hence, $|e| - |E(T)|$ is $O(n^2)$. That is, assuming the load can be distributed, each row needs to compute the lca of $O(n)$ pairs of vertices in polylog time. There appears to be no way of performing this computation efficiently.

Assuming each row cannot perform more than a constant number of lca computations in logarithmic time, the two conditions that need to be satisfied in order to have a direct implementation MSV's algorithm are: first, there are at most $O(n)$ nontree edges; secondly, there is a way to distribute the lca computation of $O(n)$ pairs uniformly over the network. The following argument shows even if the first condition is met, i.e. G is sparse, it is not always possible to satisfy the second condition due the communication problems posed by the network model. Notice that a mesh of trees of size $n \times n$ has a square mesh of size $\sqrt{n} \times \sqrt{n}$ the upper left corner. The processors from this smaller mesh are connected to the rest of the processors of the mesh by only $2\sqrt{n}$ links. Now consider a graph G whose adjacency matrix is such that all the entries belong to the square of size $\sqrt{n}$ located in the upper left corner of the mesh. Obviously G is sparse. If lca computation needs to be distributed uniformly over the network, then $O(n)$ information needs to flow from the upper left corner to the rest of the mesh. But since there are only $2\sqrt{n}$ links to carry this information it takes at least $\Omega(\sqrt{n})$ time to distribute the data.

One way to get around this difficulty would be to insist that each node in the graph have no more than a constant number of edges incident on it. But constant degree graphs are too restrictive to be of any practical interest. We circumvent this problem by focusing on the subgraphs G' of G where $E(G')$ can be partitioned into a spanning tree T and two spanning forests F, F'. The tree and the two forests are such that their edge sets are mutually disjoint. Hence the only nontree edges of G' are the edges of F and F'. The load distribution is achieved by assigning to row i the task of finding the lca of (i,x) and (i,y) where x and y are the parents of node i, respectively, in F and F'.

The following algorithm employs the technique of successive approximation to implement MSV's algorithm on a mesh of trees. Corollary 1 and Observation 1 are crucial to the understanding of the algorithm that follows. Corollary 1 follows from Whitney's theorem

(Theorem 1)[14] while Observation 1 can be deduced from the definition of ears. Assume (x_i, y_i) and (u_i, v_i) to be the end edges of P_i, for all $1 \leq i \leq k$, where P_i is a many-edge ear of an ear decomposition. Let G_m be the subgraph of G consisting of the edges of the many-edge ears i.e. the edges of P_i, for all $1 \leq i \leq k$.

Theorem 1 (Whitney) *A graph has an open ear decomposition iff it is biconnected.*

Corollary 1 *G_m is a biconnectivity preserving subgraph.*

Observation 1 *G_m consists of two edge disjoint forests: a) a spanning tree of G; b) a forest of G. Specifically, $E(G_m) - \{(x_i, y_i) \mid 1 \leq i \leq k\}$ is the edge set of a spanning tree, say T_x, of G and, furthermore, $\{(x_i, y_i) \mid 1 \leq i \leq k\}$ constitutes a forest, say F_x, of G.*

Algorithm 3 *OED-on-a-mesh-of-trees*

1. Find a spanning tree T and a spanning forest F, respectively, in G and $G - T$.
2. Find an open ear decomposition of $T + F$. Make an edge (x, y) of $G - (T + F)$ a single-edge ear provided there is a block of $T + F$ that contains both x and y.
3. Find the subgraph G' by discarding those edges of G that received an ear label.
4. *while* $E(G') \neq \emptyset$ *do*
   - (a) Find a spanning forest F' in G'.
   - (b) Find an open ear decomposition of $T + F + F'$. Make an edge (x, y) of $G - (T + F + F')$ a single-edge ear provided there is a block of $T + F + F'$ that contains both x and y.
   - (c) Find the subgraph G' by discarding those edges of G that received an ear label.
   - (d) Redefine T and F to be, respectively, T_x and F_x (Observation 1).

Fig. 4 illustrates the different stages of the above algorithm.

3.2.1 Correctness

Theorem 2 *The loop in the above algorithm terminates in at most $O(\log n)$ iterations. Moreover, $T + F$ becomes a biconnectivity preserving subgraph of G at that time.*

Proof : Assume G is biconnected; otherwise, apply the following argument independently for each block B substituting the subgraphs of G, T, F and F' that are induced by $V(B)$, respectively, for G, T, F and F'. Given that G biconnected, it is sufficient to prove that $T+F$ would be biconnected within $\log n$ (or $\log m$ where m is the number of the vertices in the largest block) iterations, because, once $T+F$ becomes biconnected, then every edge of G would receive an ear label in Step 4b of Algorithm 3 and hence $E(G')$ would be empty.

We claim that if $T+F$ has k blocks, $k > 1$, then $T+F+F'$ can have at most $\lfloor k/2 \rfloor$ blocks. Therefore, Step 4b of Algorithm 3 reduces the number of blocks of $T+F$ by at least half. Since $T+F$ initially has no more than $\lfloor 2(n-1)/3 \rfloor$ blocks, the loop has to terminate in $O(\log n)$ iterations.

Let the blocks of $T+F$ be $B_1, B_2, ..., B_k$, for some $k > 1$. We claim that for every B_i there exists a B_j, $1 \leq i \neq j \leq k$, such that $V(B_i)$ and $V(B_j)$ belong to a single block in $T+F+F'$. That is, each block of $T+F$ gets merged with at least one other block of $T+F$ when the edges of F' are included in $T+F$. In other words, the number of blocks goes down by at least half in each iteration. Refer to Fig. 4. The graph G has three biconnected components. The number of blocks of $T+F$ is six. Two of the these six blocks are biconnectivity preserving subgraphs of the corresponding blocks of G; the vertices of the other four blocks of $T+F$ are in a single block in G. These four of the six blocks of $T+F$ become two blocks when the the edges of F' are introduced, and thus reduce the total block count to four.

Consider an articulation point $v \in V(B_i)$, for some B_i, $1 \leq i \leq k$, of $T+F$. The deletion of v from $T+F$ results in more than one component: let C_1 be the connected (not necessarily biconnected) component that contains the vertices $V(B_i) - \{v\}$; let C_2 be any other connected component. Define $covering\text{-}edges(v) = \{(c,d) \mid c \in V(C_1)$ and $d \in V(C_2)\}$. Our assumption that G is biconnected guarantees that $covering\text{-}edges(v) \neq \emptyset$. Let T_1 be a spanning tree of $T+F$ rooted at y.

Assume $E(F') \cap covering\text{-}edges(v) \neq \emptyset$, and let $(x,y) \in E(F') \cap covering\text{-}edges(v)$. Consider the fundamental cycle created by adding (x,y) to T_1. We claim that there is a vertex a on the cycle such that $a \in V(B_i) - \{v\}$. As y is on the cycle and y belongs to B_j, for some j, $1 \leq j \neq i \leq k$, we can conclude that B_i and B_j are in the same block in $T+F+F'$. The vertex x has to be a descendant of v in T_1 for the following reason. If x is not a descendant of v in T_1, then the path from y to x in T_1 does not contain v. But that implies x and y are in the same component when v is deleted from $T+F$, contradicting that $C1 \neq C2$. We will show that the vertex a that we claimed to exist is the child of v that appears on the tree path P_1 from v to x. It remains to show that $a \in V(B_i) - \{v\}$. Clearly there exists a vertex $s \in V(B_i) - \{v\}$ such that s is a child of v in T_1. As s and x belong to $V(C_1)$ there exists a path P_2 from s to x such that $v \notin V(P_2)$. Let t be the first common vertex of the two directed paths that are obtained from P_1 and P_2 by directing them into x. If $t = s$ then $a = s$, and hence $a \in V(B_i) - \{v\}$. Otherwise, the vertices v, s, t and a are on a simple cycle in $T+F+F'$ where the cycle consists of the vertices of P_1 from v to t together with the vertices of P_2 from t to s. Therefore s and a are in the same

block. As we assumed $s \in V(B_i) - \{v\}$ we conclude that $a \in V(B_i) - \{v\}$.

Assume $E(F') \cap covering\text{-}edges(v) = \emptyset$. Clearly v is an articulation point in $T+F+F'$. Let Q be the path between x and y in F'. The vertex v should appear on Q or else v would not be an articulation point in $T+F+F'$. Let (v,w) be the edge of Q where $w \in V(C_1)$. Notice that if an edge (p,q) belongs to F' then p and q belong to different blocks of $T+F$. As $(v,w) \in E(F')$ and $v \in V(B_i)$ we can conclude that $w \notin V(B_i)$. Assume $w \in V(B_m)$, for some $B_m \neq B_i$. Now consider the fundamental cycle created by adding (v,w) to T_1. Using an argument similar to that of the previous paragraph, we conclude that B_i is merged with B_m.

In either case, the vertices of B_i are merged with the vertices of at least one other block of $T+F$ when the edges of F' are added to $T+F$. $\square$

4 Implementation

4.1 Overview

Algorithms that run in $O(\log^2 n)$ time on a mesh of trees of size $O(n/\log n \times n/\log n)$ for finding a spanning tree, directing a tree, finding a preorder labeling of a directed tree, finding the connected components and finding the biconnected components of a graph are given by Huang [3]. Our goal is to show that Algorithm 3 can be implemented to run in $O(\log^3 n)$ time using no more than $O(n/\log n \times n/\log n)$ processors. Step 1 and Step 4a can be implemented as shown in [3]. Step 3, Step 4c and Step 4d are trivial given Step 2 and Step 4b. Step 2 and Step 4b have two parts: first, finding an open ear decomposition; second, deciding if an edge should be a single-edge ear for that decomposition. Step 2 is a special case of Step 4b with $E(F') = \emptyset$. Section 4.4 shows how to implement the first part of Step 4. The following argument applies to the second part.

The task of any biconnected component algorithm is to find *block* labeling for the edges such that, for any two edges (x,y) and (u,v), $block(x,y)$ is equal to $block(u,v)$ iff (x,y) and (u,v) are in the same block. Each block of G has exactly one closed ear in any open ear decomposition of G by Whitney's theorem. Given an open ear decomposition of G and a biconnected component algorithm we can easily modify the algorithm so that the output $block(u,v)$ is equal to the ear label of the closed ear of the block to which (u,v) belongs. Extend the *block* labeling from the edges of G to the vertices of G by defining $block(u)$ to be the minimum of the *block* labels of the edges incident on u. Now, an edge (u,v) connects two vertices u and v in the same block iff either $block(u) = block(v)$ or u is the starting and the ending vertex of the closed ear whose label is $block(v)$. Fig. 3. illustrates the *block* labeling of the edges and the vertices of a graph G.

Ear labels for the edges that do not connect two vertices of the same block can be found as shown below. Assume each edge (u,v) has a serial number $serial(u,v)$. For example,

this could be the position of the (u,v)th entry in the lower (assuming $u > v$) triangular matrix of the adjacency matrix when going from top to bottom and from left to right. For (u,v), assign $ear(u,v) = \langle n+1, serial(u,v)\rangle$. Since no node is at a level greater than n in T, the first component of the ear labels of edges of $T+F+F'$ is less than n. Hence the ear labels of single-edge ears are different from the ear labels of $T+F+F'$ as desired. Each single-edge ear has a distinct label as the second component is unique for that edge.

4.2 About the model

P_{ij} stands for the the processor on the ith row and j^{th} column of the mesh. The input is assumed to be an $n \times n$ adjacency matrix. In the following we list the five most commonly used operations on a mesh of trees.

[A] *Route Information* : For each row (column) i, send a value from P_{ix} (resp. P_{xi}) P_{iy} (resp. P_{yi}) for some $1 \leq x, y \leq n$.

[B] *Broadcast*: For each row (column) i, broadcast a value from a processor on row (resp. column) i to all the processors on row (resp. column) i.

[C] *Find MIN, MAX or SUM*: For each row (column) i, find the minimum, maximum or sum of the values stored on that row (resp. column).

[D] *Rank rows and columns* : For each row (column) i, rank the entries of an adjacency matrix in left to right (resp. top to bottom) order from 1 to number of entries on row (column) i.

[E] *Do local operations*: Perform arithmetical/logical operations on the values stored in the local memory.

It is easy to see that these operations can be performed in $O(\log n)$ time on a mesh of trees of size $n \times n$. In fact, the processor-time product can be improved. Assume the matrix is divided into $\log n \times \log n$ blocks of submatrices of size $n/\log n \times n/\log n$. For $1 \leq i, j \leq \log n$, store the $[i,j]$th block in the local memory of P_{ij}. Now, by pipelining we can achieve a time and processor complexity of $O(\log^2 n)$ and $n^2/\log^2 n$, respectively. See [3] for details.

Refer to [3] for the implementation of tree operations such as *root a tree* T at a given vertex r, find the level $lev(v)$ of every vertex with respect to r in T, perform a *transitive closure on a directed tree*, find $parent(v)$ for each vertex v in T, find a preorder label $pre(v)$ for each vertex v in a rooted tree T, find $high(v)$ for a given preorder labeling and so on.

The implementation is described in terms of operations where each such operation is one of the five types mentioned above. The types corresponding to these operations are

listed at the end of each step. For the sake of clarity, we assume that the size of the mesh is $n \times n$. As it is possible to pipeline each individual operation of every step of our implementation, we can keep the processor count to $O(n^2/log^2 n)$ without increasing the running time.

4.3 Ear Decomposition of $T + F + F'$

1. Perform preliminary tree computations.

Root T, F and F' at some arbitrary vertices. Find the parent of v in T, F and F'. Denote them by $parent_T(v)$ $parent_F(v)$ and $parent_{F'}(v)$, respectively. Also, find $pre(v)$, $high(v)$ and $lev(v)$ for all v in T. Store these values on all the processors that are on row v and column v. Direct the edges of T from a vertex to its children and perform a transitive closure on T. Assume the transitive closure algorithm marks at the (i,j)th position of the adjacency matrix whether j is an ancestor/descendant of i in T.

2. Ear label edges of F.

For $(i,j) \in E(F)$, assume that j is the parent of i in F. We will find the ear label for (i,j) on row i and route the output $ear(i,j)$ to P_{ij} and P_{ji}. To find the ear label of (i,j) we need the least common ancestor of i and j in T and the level of the least common ancestor. We know the level from the previous step. In the following we show how to find $lca(i,j)$.

Assume, without loss of generality, that $pre(j) > pre(i)$. We claim that $lca(i,j) = a$ where $pre(a)$ is the maximum of $\{pre(k) \mid pre(k) < pre(i) \;\&\; high(k) \geq high(j)\}$

Notice that for a vertex k, if $high(k) > high(j)$, then there are two possibilities: k is on the tree path from the root to j; or k comes after j in the given preorder. As $pre(k) < pre(i) < pre(j)$ we can rule out the second possibility and conclude that k is on the tree path from the root to j. In addition, k should appear on the tree path from the root to i for the following reason. Observe that, for any l, if $pre(l) < pre(i)$ and if l does not occur on the path from i to the root, then $high(l) < pre(i)$. But as $high(k) \geq high(j) \geq pre(j) > pre(i)$ we conclude that k should appear also on the path from i to the root. Therefore k is a common ancestor of i and j. Finally, the claim follows since the common ancestor with the highest preorder label is the least common ancestor.

We use the above characterization to find the least common ancestor in our implementation.

(i) Find $lca(i,j)$

(a) Broadcast j and $high(j)$ from P_{ij} to all the processors on row i. [B]

(b) Find the maximum $\langle pre(h), h\rangle$ of the 2-tuples $\{\langle pre(k), k\rangle \mid pre(k) < pre(i)$ and $high(k) \geq high(j)\}$ on the ith row. Notice that at P_{ik}, $1 \leq k \leq n$, we have $pre(i)$, $pre(k)$, $high(k)$ and $high(j)$. The $lca(i,j)$ is h. [C],[E]

(c) Broadcast h to all the processors on row i from the root of the ith row. [B]

(ii) Find an ear label for (i,j)

(a) Broadcast $lev(h) = l$ to all the processors on row i from P_{ih}. [B]

(b) Send l from P_{ii} to P_{ji}. [A]

(c) Assign $ear(i,j) = \langle l, serial(i,j)\rangle$ at P_{ij} and P_{ji}. [E]

3. *Ear label edges of F'.*

Similar to the above step.

4. *Ear label edges T.*

First, we find an ear label for $(i,j) \in E(T)$ assuming that the only nontree edges are the edges of F. Next, we repeat the same steps with F' in place of F and find another ear label for (i,j). The minimum of the two ear labels is the correct ear label of (i,j).

Consider $ear(u,v)$ where $u = parent_T(v)$. Any nontree edge (i,j) where exactly one of $\{i,j\}$ is a descendant of v in T creates a cycle containing both (i,j) and (u,v) when included in T. Also note that a vertex k is descendant of v in T *iff* $pre(v) < pre(k) < high(v)$.

The computations necessary for finding $ear(u,v)$ are performed on the vth row, where u is the parent of v in T. The following steps assume that j is the parent of i in F.

(a) Broadcast $pre(i)$, $pre(j)$ and $ear(i,j)$ to all the processors on column i from P_{ii}. [B]

(b) For all v, $1 \leq v \leq n$, broadcast on the vth row $pre(v)$ and $high(v)$ from P_{vv}. This step is described in interval notation. Find at P_{vi} if

$$pre(i) \in [pre(v), high(v)) \text{ and } pre(j) \notin [pre(v), high(v))$$
$$\text{or}$$
$$pre(j) \in [pre(v), high(v)) \text{ and } pre(i) \notin [pre(v), high(v))$$

Find the minimum of the ear labels $ear(i,j)$ of all edges (i,j) satisfying the above condition. [B],[C],[E]

(c) At root of the vth row we have $ear(u,v)$. Broadcast this value to all the processors on the vth row. [B]

(d) At P_{vu} store this label as the ear label of the tree edge uv. Send $ear(u,v)$ from P_{vv} to P_{uv}. [A]

4.4 Open Ear Decomposition of $T + F + F'$

Recall that H_is are the bipartite graphs of Algorithm 2. The key thing to notice is that the sum of the sizes of the vertex sets of H_i's, $1 \leq i \leq n$, is $O(n)$ as explained in the following. Note that $|V''(H_x)|$ is equal to the number of children of x in T. Therefore, the total number of tree vertices

$$\sum_{x \in V(G)} |V''(H_x)| = |E(T)|$$

is $n - 1$. Consider the number of nontree vertices. A nontree vertex $[u, v]$ corresponding to the nontree edge (u, v) is a member of exactly one $V''(H_x)$ because the least common ancestor of u and v in T has precisely one value (in this case x). Therefore, the total number of nontree vertices is

$$\sum_{x \in V(G)} |V'(H_x)| = |E(F)| + |E(F')|$$

no more than $2(n - 1)$. It is clear from the above discussion that for $i \neq j$, $1 \leq i, j \leq n$, $V(H_i) \cap V(H_j) = \emptyset$. Therefore we can treat

$$\bigcup_{i=1}^{n} H_i$$

as a single graph H.

Implementation of every step of Algorithm 2, except Step 2 and Step 3, can be found Section 4.3. An implementation of Step 3 that runs in $O(\log^2 n)$ time on a mesh of trees of size $O(n/\log n \times n/\log n)$ can be found in [3]. It remains to show how to implement Step 2 i.e. how to build an adjacency matrix for H.

Assume we have a mesh of trees of size $3n \times 3n$. In the following we give only a partial construction of H for the sake of clarity. The construction given below builds the part of the adjacency matrix that consists of the tree and nontree vertices corresponding to the edges of T and F. The construction uses the first $2n$ rows and $2n$ columns. If full implementation is desired then substitute F' in place of F, repeat the steps that involve F, and fill the rows and columns that are between $2n$ and $3n$.

1. Label the rows and columns

Let row i and column i, for $1 \leq i \leq n$, represent the nontree vertex [i,j] where $j = parent_F(i)$. Let row $(n + i)$ and column $(n + i)$ represent the tree vertex [i,k] where $k = parent_T(i)$. Recall that the values $parent_T(i)$ and $parent_F(i)$, for $1 \leq i \leq n$, are stored in the local memory at P_{ii}. Let (i, j) and k be as defined in the previous paragraph.

(a) Mark each processor with the row and the column position at which the processor is located. Also, broadcast the value n to all the processors. [D]

(b) Broadcast the value $[i, k]$ to all the processors on the ith row. Send this value from $P_{(n+i),i}$ to $P_{(n+i),(n+i)}$. [A],[B]

(c) For $1 \leq l \leq 2n$, broadcast the row and column IDs i.e. $[i, j]$ for $1 \leq l \leq n$ and $[i, k]$ for $n < l \leq 2n$. [B],[E]

2. Mark the entries for edges

Assume, for a tree vertex $[i, j]$, $[a, b]$ is a nontree vertex such that $lca(i, j) = a$ and (a, b) is the first edge in the unique path from a to i in T. This step involves verifying that j is not an ancestor of i in T, and assigning a value of 1 to the $([i, j], [a, b])$th and $([a, b], [i, j])$th entry of the adjacency matrix of H.

(a) Check if j is an ancestor of i in T as shown in Step 4b of Section 4.3. If j is not an ancestor of i in T then broadcast the value $lca(i, j) = a$ (Step 2(i) of Section 4.3) and $pre(a)$ to all the processors on the ith row. [B]

(b) Identify the vertex b where $pre(b)$ is the minimum of $\{pre(c) \mid pre(c) \in (pre(a), pre(i))$ and $high(c) \geq high(i)\}$. Perform the minimum computation on the ith row (recall that a, $pre(a)$, $pre(i)$ and $high(i)$ are stored at all the processors on the ith row. In addition, each P_{ic}, $1 \leq c \leq n$, has $pre(c)$ and $high(c)$). [C],[E]

(c) Broadcast $[a, b]$ to all the processors on the ith row and the ith column from the root of the ith row tree. Set the edge entry to 1 at $P_{[i,j],[a,b]}$ and $P_{[a,b],[i,j]}$. [B]

5 Processor-Time Complexity

Theorem 3 *Open ear decomposition of an undirected graph G can be found in $O(\log^3 n)$ time on a $O(n/\log n \times n/\log n)$ mesh of trees.*

Proof: It is clear from the explanation given in Section 4.2 that each step of Section 4.3 and Section 4.4 takes no more than $O(\log^2 n)$ time on a mesh of trees of size $O(n/\log n \times n/\log n)$. Therefore, an open ear decomposition of $T + F + F'$ can be found within these bounds. Furthermore, each step of Algorithm 3 can be implemented within these bounds as argued in the first paragraph of Section 4.1. From Theorem 2, we know that the loop in Algorithm 3 terminates in at most $O(\log n)$ iterations. Hence, the processor complexity of Algorithm 3 is $O(n^2/\log^2 n)$ and time complexity is $O(\log^3 n)$. □

References

[1] D. Fussell and R. Thurimella, "Separation Pair Detection," *VLSI Algorithms and Architectures*, LNCS 319 (1988) pp. 149-159.

[2] D. S. Hirschberg and D. Volper, "A Parallel Solution for the Minimum Spanning Tree Problem," *Proc. 1983 Johns Hopkins Conf. on Information science and systems* (1983) pp. 680–684.

[3] M. A. Huang, "Solving Some Graph Problems with Optimal or Near Optimal Speedup on Mesh-of-trees Networks,"*Proc. 26th Annual IEEE Symp. on Foundations of Comp. Sci.* (1985) pp. 232–240.

[4] A. Kanevsky and V. Ramachandran, "Improved Algorithms for Four-connectivity," *Proc. 28th Annual IEEE Symp. on Foundations of Comp. Sci.* (1987) pp. 252-259.

[5] P.N. Klein and J.H. Reif, "An Efficient Parallel Algorithm for Planarity," *Proc. 27th Annual IEEE Symp. on Foundations of Comp. Sci.* (1986) pp. 465-477.

[6] T. Leighton, *Complexity Issues in VLSI*, Cambridge, Massachusetts, MIT Press, 1983, ch.6.1.

[7] T. Leighton, "Parallel Computation using Meshes of trees," *Proc. 1983 Int. Workshop on Graph-Theoretic Concepts in Computer Science* (1983).

[8] L. Lovasz, "Computing Ears and Branching in Parallel," *26th annual IEEE Symp. on Foundations of Comp. Sci.* (1985) pp. 464-467.

[9] Y. Maon, B. Schieber and U. Vishkin, "Parallel Ear Decomposition Search (EDS) and ST-numbering in Graphs," *Theoretical Computer Science*, 47 (1986) pp. 277-298.

[10] G.L. Miller and V. Ramachandran, "Efficient Ear Decomposition with Applications," *Manuscript*, MSRI, Berkeley, CA, Jan. 1986.

[11] G.L. Miller and V. Ramachandran, "A New Triconnectivity Algorithm and Its Parallelization," *Proc. 19th Annual Symp. on Theory of Computing* (1987) pp. 335–344.

[12] Y. Shiloach and U. Vishkin, "An $O(\log n)$ Parallel Connectivity Algorithm," *J. Algorithms* 2 (1981) pp. 57–63.

[13] R. E. Tarjan and U. Vishkin, "An Efficient Parallel Biconnectivity Algorithm," *SIAM J. Computing* 14 (1984) pp. 862-874.

[14] H. Whitney,"Non-separable and planar graphs," *Trans. Amer. Math. Soc.* 34 (1932) pp. 339-362.

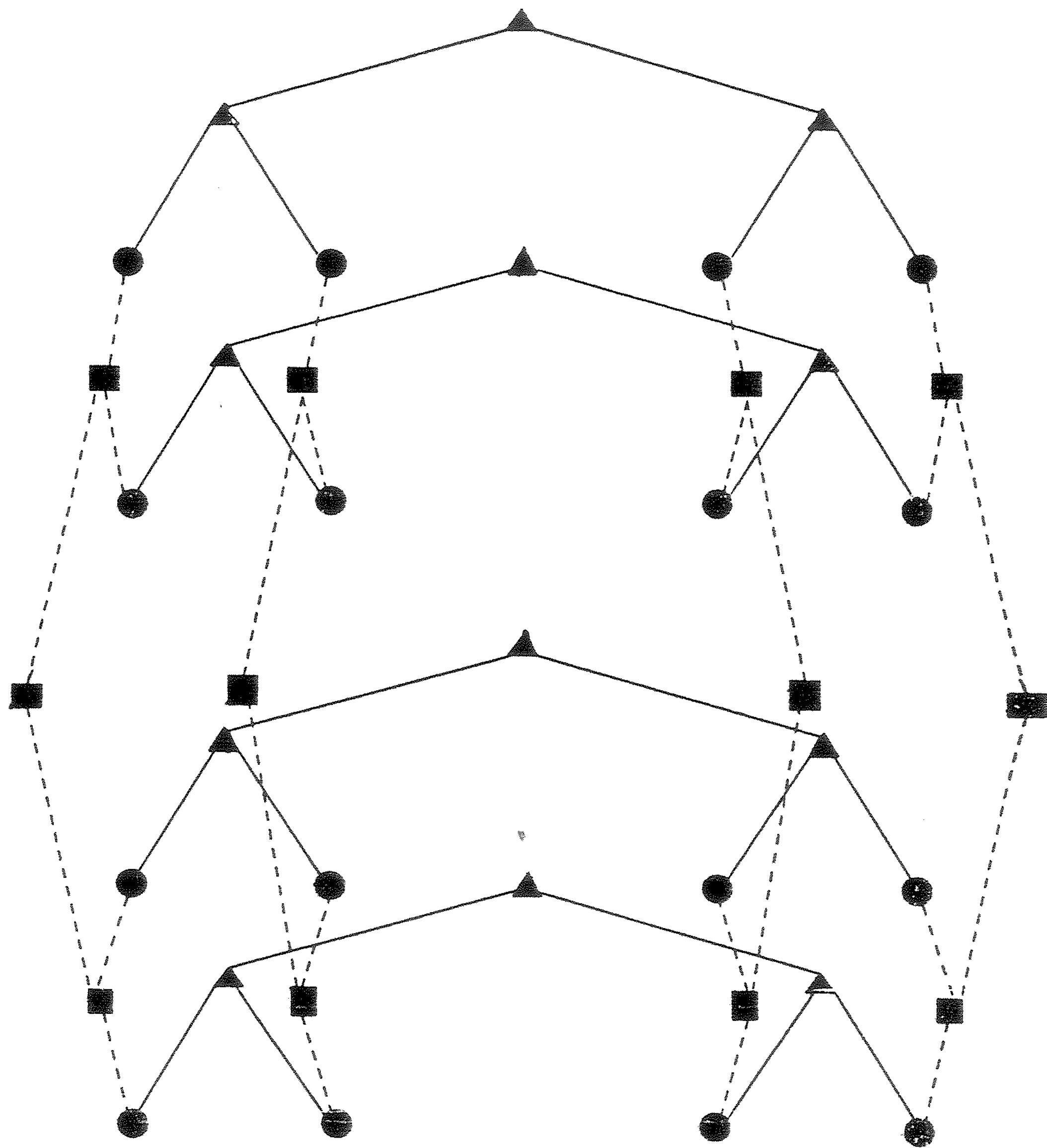


Fig. 1. The 4x4 mesh of trees.

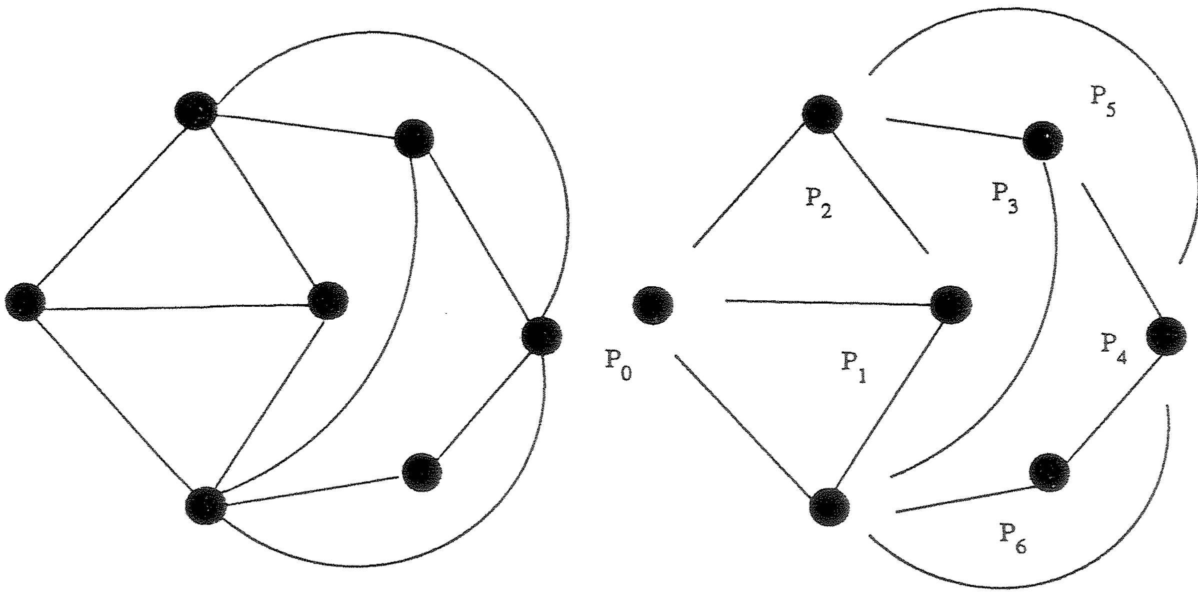
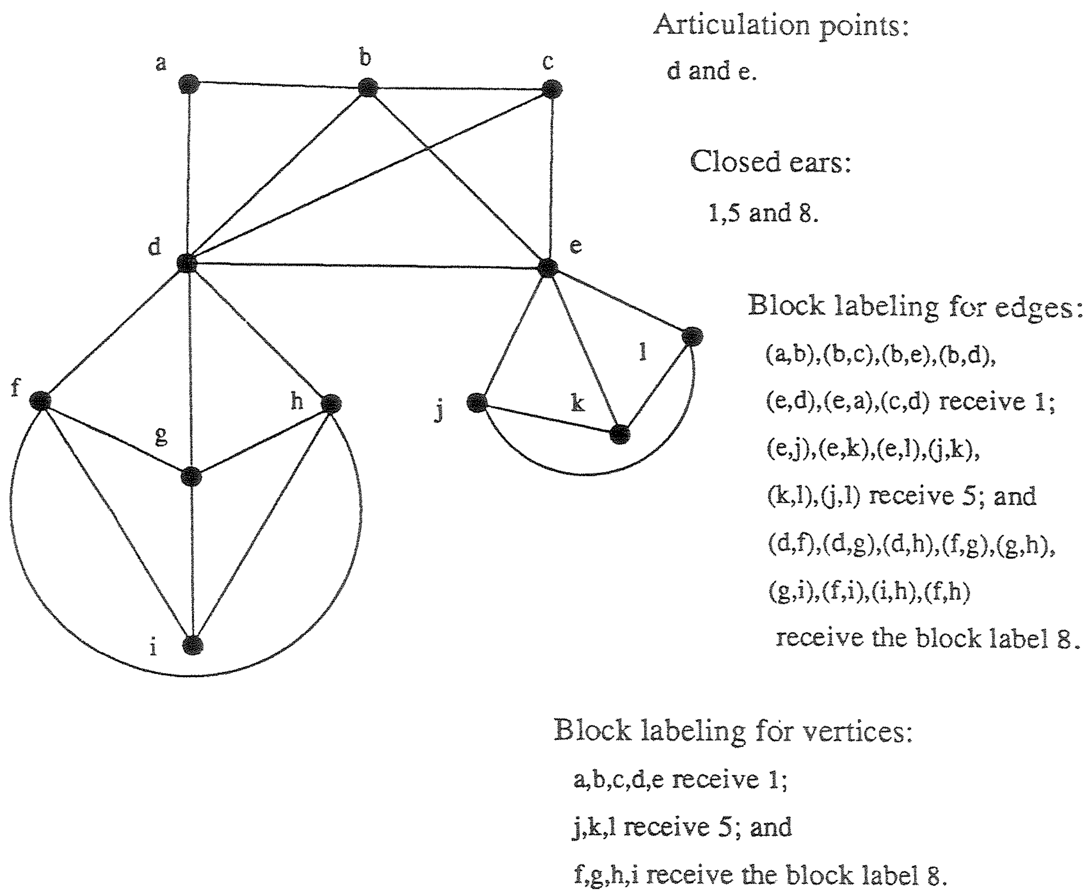


Fig. 2. A biconnected graph and its open ear decomposition



Ear labels:

P_0= <a>, P_1= <a,d,b,a>, P_2= <d,e,b>, P_3= <e,c,b>, P_4= <d,c>,

P_5= <e,k,l,e>, P_6= <e,j,k>, P_7= <j,l>, P_8= <d,g,i,h,d>, P_9= <d,f,g>,

P_{10} = <f,i,g>, P_{11} = <f,h>, P_{12} = <g,h> , P_{13}= <i,h>

Fig. 3. An open ear decomposition of a graph that is not biconnected.

a) A graph G.

b) T+F+G' where T:—— , F:----- and G':〰.

c) T+F+F' where T:—— , F:—— and F':-----. An OED where ⟶ stands for a many-edge ear; the rest are single-edge ears.

d) T_x+F_x where T_x:—— and F_x:-----.

Fig. 4. Different stages in the execution of Algorithm 3.