September 1991

UILU-ENG-91-2241
ACT-118

Applied Computation Theory

IMPROVED PROCESSOR BOUNDS FOR PARALLEL ALGORITHMS FOR WEIGHTED DIRECTED GRAPHS

Nancy Amato

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1a. REPORT SECURITY CLASSIFICATION Unclassified	**1b. RESTRICTIVE MARKINGS** None
2a. SECURITY CLASSIFICATION AUTHORITY	**3. DISTRIBUTION/AVAILABILITY OF REPORT** Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-91-2241 (ACT-118)	**5. MONITORING ORGANIZATION REPORT NUMBER(S)**

6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	**6b. OFFICE SYMBOL** *(If applicable)* N/A	**7a. NAME OF MONITORING ORGANIZATION**
6c. ADDRESS *(City, State, and ZIP Code)* 1101 W. Springfield Ave. Urbana, IL 61801		**7b. ADDRESS** *(City, State, and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	**8b. OFFICE SYMBOL** *(If applicable)*	**9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER**
8c. ADDRESS *(City, State, and ZIP Code)*		**10. SOURCE OF FUNDING NUMBERS**

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

11. TITLE *(Include Security Classification)*
Improved Processor Bounds for Parallel Algorithms for Weighted Directed Graphs

12. PERSONAL AUTHOR(S)
Amato, Nancy

13a. TYPE OF REPORT Technical	**13b. TIME COVERED** FROM ________ TO ______	**14. DATE OF REPORT** *(Year, Month, Day)* September 1991	**15. PAGE COUNT** 13

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	weighted directed graph, single-source shortest paths, parallel algorithms, matrix multiplication

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

This paper presents a parallel algorithm that solves the single-source shortest path (SSSP) problem for a weighted digraph $G = (V, E)$ in time $O(\log^2 n)$ using $M(n)$ processors on an exclusive-read exclusive-write parallel random access machine (EREW PRAM), where $n = |V|$, edge weights are drawn from the set $[0, 1, 2, \ldots, k]$, for some fixed constant k, and $M(n)$ is the number of processors necessary to multiply two $n \times n$ integer matrices over a ring in $O(\log n)$ time. This algorithm is a generalization of the result of Gazit and Miller [GM 88] for the SSSP problem on an unweighted digraph. We then show how our solution of the SSSP problem for a weighted digraph can be used to solve a number of digraph problems in polylog time using $M(n)$ processors; all previous NC algorithms for these problems required $\Theta(n^3)$ processors (to within a polylog factor). The digraph problems we consider are finding a minimum weight branching, finding an ear-decomposition, and the transitive reduction problem.

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

Improved Processor Bounds for Parallel Algorithms for Weighted Directed Graphs

Nancy Amato
Department of Computer Science
University of Illinois at Urbana-Champaign Urbana, IL 61801

Abstract

This paper presents a parallel algorithm that solves the single-source shortest path (SSSP) problem for a weighted digraph $G = (V, E)$ in time $O(\log^2 n)$ using $M(n)$ processors on an exclusive-read exclusive-write parallel random access machine (EREW PRAM), where $n = |V|$, edge weights are drawn from the set $[0, 1, 2, \ldots, k]$, for some fixed constant k, and $M(n)$ is the number of processors necessary to multiply two $n \times n$ integer matrices over a ring in $O(\log n)$ time. This algorithm is a generalization of the result of Gazit and Miller [GM 88] for the SSSP problem on an unweighted digraph. We then show how our solution of the SSSP problem for a weighted digraph can be used to solve a number of digraph problems in polylog time using $M(n)$ processors; all previous NC algorithms for these problems required $\Theta(n^3)$ processors (to within a polylog factor). The digraph problems we consider are finding a minimum weight branching, finding an ear-decomposition, and the transitive reduction problem.

1 Introduction

We present a parallel algorithm that solves the single-source shortest path problem (SSSP) for a weighted digraph $G = (V, E)$ in time $O(\log^2 n)$ using $M(n)$ processors on an exclusive-read exclusive-write parallel random access machine (EREW PRAM), where $n = |V|$, edge weights are drawn from the set $[0, 1, 2, \ldots, k]$, for some fixed constant k, and $M(n)$ is the number of processors necessary to multiply two $n \times n$ integer matrices over a ring in $O(\log n)$ time (currently, $M(n) = n^{2.376}$ [CW 87]). The best previous parallel solution to this problem was essentially the computation of the transitive closure of the incidence matrix of the digraph, and was implemented by using a parallelization of the standard sequential matrix multiplication algorithm; this approach required time $O(\log^2 n)$ and $O(n^3/\log n)$ processors in the EREW and CREW PRAM models or $O(\log n)$ time and $O(n^3)$ processors in the

CRCW PRAM model. The algorithm presented here is a generalization of the $O(\log^2 n)$ time, $M(n)$ processor EREW PRAM algorithm due to Gazit and Miller [GM 88] for the SSSP problem in an unweighted digraph. In addition, we provide a proof of correctness for the generalized algorithm that is simpler than the proof given in [GM 88] for the original algorithm; this new proof has the added benefit that it provides a more intuitive characterization of how the algorithm computes the shortest paths. Using our solution of the SSSP problem for a weighted digraph, we show that a number of digraph problems can be solved in polylog time using $M(n)$ processors; all previous NC algorithms for these problems required at least $\Theta(n^3)$ processors (to within a polylog factor). The digraph problems we consider are finding an ear-decomposition, the transitive reduction problem, and finding a minimum weight branching where the edge weights are non-negative integers bounded by some constant k.

In this paper we use the parallel random access machine (PRAM) model of parallel computation, in which it is assumed that each processor has random access to any shared memory location in unit time. In the EREW PRAM model, different processors may not access the same common memory location when reading or writing; in the CREW and CRCW PRAM models, concurrent reads and concurrent reads and writes, respectively, are allowed. The commonly held view is that a PRAM algorithm should attempt to solve the problem in time $O(\log^k n)$ for some constant k (often referred to as polylog time) using $O(n^{O(1)})$ processors, where n is the size of the input; the class NC consists of those problems that can be solved within these bounds. In general, an NC algorithm is said to be *optimal* if the product of the time and the number of processors used (processor-time product) equals the sequential complexity of the the problem. Similarly, an NC algorithm is said to be *efficient* if the processor-time product exceeds the sequential complexity of the problem by at most a polylog factor. For a detailed treatment of the various PRAM models and the class NC consult [KR 90].

Although efficient parallel algorithms exist for many problems concerned with undirected graphs, the same is not true when dealing with directed graphs (digraphs). The main reason for this disparity is that reachability information (which vertices are reachable from other vertices) is often required and, although there are various optimal techniques for searching undirected graphs (see e.g., [KR 90]), the most economical method known today to determine if one vertex is reachable from another in a digraph is to compute the transitive closure of its incidence matrix by repeated matrix multiplication. The standard matrix multiplication algorithm, working over the semiring $(N, \min, +)$, requires $\Theta(n^3)$ operations. When working over a ring, or Boolean matrices that may be embedded in a ring, there exist more economical techniques requiring $O(n^\alpha)$ operations; currently the best of these achieves $\alpha = 2.376$ [CW 87]. All of these sequential techniques may be implemented efficiently in parallel; the standard algorithm requires $O(1)$ time and $O(n^3)$ processors on a CRCW PRAM, or $O(\log n)$ time and $O(n^3/\log n)$ processors on an EREW or CREW PRAM, and the more economical algorithms require $O(\log n)$ time and $O(n^\alpha)$ processors on an EREW or CREW PRAM (unfortunately these algorithms are not amenable to implementation on CRCW PRAMs). It is a common convention to denote n^α by $M(n)$.

Thus, although the parallel solution of many problems dealing with digraphs seems to require the transitive closure of the relevant incidence matrices, if the computation can be restricted to matrices over a ring, or Boolean matrices that can be embedded in a ring, then fewer processors will be required yielding a lower processor-time product. The first contribution to this effort was made by Gazit and Miller [GM 88] when they provided an $O(\log^2 n)$ time, $M(n)$ processor EREW PRAM algorithm for solving the SSSP problem for an unweighted digraph $G = (V, E)$, where $|V| = n$. In this paper we extend their result to include weighted digraphs in which the edges have weights drawn from the set $[0, 1, 2, \ldots, k]$, for some constant k. In addition, we provide a proof of correctness for the generalized algorithm that is simpler than the proof given in [GM 88] for the original algorithm; this new proof has the added benefit that it provides a more intuitive characterization of how the algorithm computes the shortest paths. We then explain how our solution of the SSSP problem for a weighted digraph can be used to reduce the previous known processor bounds for a number of digraph problems to $M(n)$ from $\Theta(n^3)$ (within a polylog factor) without increasing the running time; the problems we consider are finding an ear-decomposition, the transitive reduction problem, and finding a minimum weight branching in a digraph in which the edge weights are non-negative integers bounded by some constant k.

2 Preliminaries

In many of our algorithms we use well known parallel techniques such as pointer jumping, list ranking and the Euler tour technique as subroutines. Pointer jumping is an operation on linked lists in which, for every element in the list, the pointer to the next element in the list is replaced by a pointer to the next element's next element. Thus, after $\log n$ iterations the first element will have a pointer to the last element. The Euler tour technique on trees is used with list ranking to obtain a rooted spanning tree corresponding to a depth first search traversal of the tree. For a detailed explanation of these and other techniques used in this paper consult [KR 90].

In this paper, we will use a notation introduced in [R 88] to refer to the time and processor bounds when the processor count is dominated by the number of processors necessary to multiply two $n \times n$ Boolean matrices. The bounds will be reported as parallel time $t(n)$ using $Q(n)$ processors when the algorithm would run in time $t(n)$ with $O(n^3)$ processors on a CRCW PRAM and would run in time $O(t(n) \cdot \log n)$ with $O(n^\alpha)$ processors on an EREW or CREW PRAM.

There will be two different kinds of matrix multiplication that we will refer to in the

following section, and they will be represented by the following symbols:
$*$: Matrix multiplication over a ring (or Boolean matrices embedded in a ring).
$\bullet$: Matrix multiplication over the semiring $(N \cup \{\infty\}, \min, +)$.

3 The Single-Source Shortest Path Problem

In a digraph $G = (V, E)$ with root r, the single-source shortest path (SSSP) problem is to find for every vertex $v \neq r$ the length of the shortest path from r to v. Since the solution of the SSSP problem can be translated into a single-source breadth-first search (BFS) tree of the digraph G in constant time using $O(|E|)$ processors, the SSSP and BFS problems can be viewed as slightly different instances of the same problem; we will refer to this problem as the SSSP/BFS problem. In this section, n denotes $|V|$.

When working with matrices over the semiring $(N, \min, +)$, the SSSP/BFS problem can be solved in $O(\log n)$ time using $O(n^3)$ processors on a CRCW PRAM (or $O(\log^2 n)$ time with $O(n^3/\log n)$ processors on a CREW or EREW PRAM) by using repeated $(\min, +)$ matrix multiplication; this technique actually solves the all-pairs shortest path problem. Gazit and Miller have given an algorithm that will solve the SSSP/BFS problem for an unweighted digraph in time $O(\log n)$ using $Q(n)$ processors by performing the computations over a ring [GM 88]. We now show how their method may be extended to solve the problem for a weighted digraph, where weights of zero and positive integers, less than some constant k, are allowed; this generalized algorithm also runs in time $O(\log n)$ using $Q(n)$ processors.

In the algorithm for solving the SSSP/BFS problem for a weighted digraph G we deal with edges with weights larger than one and edges with weight zero separately. In particular, we remove the necessity of considering edges with weights larger than one by transforming the digraph G into an equivalent digraph G' that has edges with weights of only zero and one; an important property of this transformation is that the new digraph G' has only $O(n)$ vertices. When considering edges of weight zero, we first identify all paths consisting only of edges with weight zero, and then concatenate these paths with all incident edges of weight one to identify all paths of weight zero or one. In the following discussion we use $wt(v_i, v_j)$ and $d(i, j)$ to refer to the weight of the edge (v_i, v_j) and the shortest distance between the vertices v_i and v_j, respectively, in G (or the equivalent digraph G').

Algorithm 1: SSSP/BFS in a Weighted Digraph
Input: A digraph G with edge weights $w \in \{0, 1, \ldots, k\}$, for some fixed k, and a source s.
Output: The lengths of the shortest paths from s to all other vertices in G.

1. Create the digraph $G' = (V', E')$; initially $V' = V$ and $(x, y) \in E'$ if $wt(x, y)$ is 0 or 1 in G (this same weight is retained in G'). For each vertex $v \in V$, for each edge

weight $2 \leq w \leq k$ in G, add $w-1$ new vertices to V' forming a path of length $w-1$ originating at vertex v (each edge on the path has weight 1); denote the other endpoint of this path as v_w. Introduce edge (v_w, u) of weight 1 in G' for every vertex u such that $wt(v,u) = w$ in G. Let $n' = |V'|$.

2. Consider the subgraph $C = (V_C, E_C)$ of G' formed by including only those edges of weight 0; let C be the Boolean incidence matrix of this subgraph (i.e., $C[x,y] = 1$ if $(x,y) \in E_C$ or $x = y$, $1 \leq x, y \leq n'$). Compute $C^{n'}$, the transitive closure of C. Pre and post multiply the Boolean incidence matrix of G' by $C^{n'}$ to form the matrix B_0.

3. Next, approximations of the distances between all pairs of vertices are calculated. Compute the first $\lceil \log n' \rceil$ matrices of the form $B_{i+1} = B_i * B_i = B_i^2$. For each matrix B_i, construct a matrix M_i as follows; first set $M_i[x,y] = \infty$, next set $M_i[x,y] = 2^i$ if $B_i[x,y] = 1$, and finally set $M_i[x,y] = 0$ if $C^{n'}[x,y] = 1$, for $1 \leq x, y \leq n'$.

4. This step consists of $\lceil \log n' \rceil + 1$ stages so that after the i^{th} stage (i decreasing), $d(s,x)$ has been computed if it is a multiple of 2^i; specifically, these distances are represented by a row vector V_i of length n' in which $V_i[x] = d(s,x)$ if and only if $d(s,x)$ is a multiple of 2^i, $1 \leq x \leq n'$. Initially, $V_{\lceil \log n' \rceil + 1}[s] = 0$ and $V_{\lceil \log n' \rceil + 1}[j] = \infty$ for all $j \neq s$. For $i = \lceil \log n' \rceil$ downto 0 let $V_i = V_{i+1} \bullet M_i$.

The algorithm of [GM 88] for solving the SSSP/BFS problem in an unweighted digraph essentially consists of steps 3 and 4 of the above algorithm, with n replacing n', the incidence matrix of G replacing B_0, and the condition $x = y$ replacing the condition $C^{n'}[x,y] = 1$.

Theorem 1: Algorithm 1 may be implemented in time $O(\log n)$ using $Q(n)$ processors.

Proof: In Step 1, for each of the n original vertices in G, we add an additional $O(k^2)$ vertices to G', for a total of $n' = O(k^2 n) = O(n)$ vertices; this procedure can be accomplished in $O(\log n)$ time using $O(k^2 n) = O(n)$ processors for each of the n vertices of G. Clearly the graph G' preserves the distances among the vertices of G; note that the obvious method of edge subdivision would not work because this approach could potentially add $O(|E|k) = O(n^2)$ vertices to G' and would require $Q(n^2)$ processors for Boolean matrix multiplication rather than the desired $Q(n)$. The transitive closure of the matrix C in step 2 and of the B_i matrices in step 3 can be computed in time $O(\log n)$ using $Q(n)$ processors. The transformation of the B_i matrices into the M_i matrices in step 3 can be accomplished in time $O(\log n)$ using $O(n^2)$ processors. Each of the multiplications over the semiring $(N \cup \{\infty\}, \min, +)$ in step 4 can be performed by making n' copies of V, computing n'^2 "+" operations, and then computing n' "min" operations, each on a set of n' numbers. Thus, step 4 can be implemented in time $O(\log^2 n)$ using $O(n^2 / \log n)$ processors. Hence, the total complexity of Algorithm 1 is $O(\log n)$ time using $Q(n)$ processors. □

The correctness of Algorithm 1 is established by the following theorem which also provides a new proof of the correctness of the algorithm presented in [GM 88] for SSSP/BFS in an unweighted digraph. This theorem has the added benefit that its proof is simpler and shorter than the corresponding proof of its counterpart in [GM 88]; the main reason for this is because this theorem only requires that $V_i[x] = d(s,x)$ if $d(s,x)$ is a multiple of 2^i, whereas the theorem in [GM 88] requires that $V_i[x] = 2^i\lceil d(s,x)/2^i\rceil$, $1 \leq x \leq n$. Furthermore, although this theorem is weaker, its proof provides a better characterization of the progress that the algorithm makes in computing the shortest paths; specifically, if $0 < d(s,y) < 2^i$, $1 \leq y \leq n'$ then the values in $V_i[y]$ are irrelevant to the computation at stage i (i decreasing) because if $d(s,x)$ is a multiple of 2^i but not of 2^{i+1}, then the algorithm calculates $d(s,x)$ in stage i from some previously calculated $d(s,w)$ where $d(s,w)$ is a multiple 2^{i+1} and $d(w,x) = M_i[w,x] = 2^i$.

Theorem 2: If $d(s,j) = c \cdot 2^i$ then $V_i[j] = d(s,j)$, $1 \leq j \leq n'$, $0 \leq i \leq \lceil \log n' \rceil$, $0 \leq c \leq 2^{\lceil \log n' \rceil - i}$.

Proof: It is immediate to verify (i) $M_i[x,y] = \min_{l \in \{0,2^i,\infty\}}[l | d(x,y) \leq l]$, $1 \leq x,y \leq n'$, and (ii) $d(s,x) \leq V_i[x] \leq V_{i+1}[x]$, $1 \leq x \leq n'$ and $0 \leq i \leq \lceil \log n' \rceil$; these will be referred to as Facts (i) and (ii), respectively.

The statement is proven by induction on i, i decreasing. Since $V_{\lceil \log n' \rceil}$ is the s^{th} row of $M_{\lceil \log n' \rceil}$, the basis is established by Fact (i). We now assume the statement holds for $i+1$ and show that it holds for i; let v_j be a vertex such that $d(s,j) = c \cdot 2^i$.

We first note that if c is even, then $d(s,j) = c' \cdot 2^{i+1}$, for some integer c', and thus by the hypothesis $V_{i+1}[j] = d(s,j)$; this and Fact (ii) ensure that $V_i[j] = d(s,j)$. If c is odd, then there must be a vertex v_k that lies on a shortest path from s to v_j such that $d(k,j) = 2^i$ and $d(s,k) = d(s,j) - d(k,j) = c \cdot 2^i - 2^i = (c-1) \cdot 2^i$. Moreover, since $(c-1)$ is even, $d(s,k) = (c-1)/2 \cdot 2^{i+1}$ and thus by the hypothesis, $V_{i+1}[k] = d(s,k)$. Finally, $V_i[j] \leq V_{i+1}[k] + M_i[k,j] = (c-1) \cdot 2^i + 2^i = c \cdot 2^i = d(s,j)$ since, by Fact (i), $M_i[k,j] = 2^i$; this and Fact (ii) ensure that $V_i[j] = d(s,j)$. □

The proof of this theorem indicates that any additional generalization of this technique to include digraphs with polynomially bounded weights is unlikely without major alterations to the general algorithm. This is due to the fact that if $d(s,x) = c \cdot 2^i$, c odd, then when calculating $d(s,x)$ at stage i, the algorithm depends on the fact that there is some v_y that lies on a shortest path from s to v_x such that $d(s,y) = (c-1)/2 \cdot 2^{i+1}$ and $d(y,x) = 2^i$; clearly this is unlikely to be the case in a digraph with polynomially bounded edge weights. (The technique employed here would not work because G' could have as many as $O(nw_{\max}^2)$ vertices, where $w_{\max}$ is the maximum edge weight).

4 Minimum Weight Branchings of Digraphs

A branching (also known as a forward or in-branching) of a digraph is a rooted spanning subgraph such that every vertex except the root has indegree 1; in a reverse or out-branching, every vertex except the root has outdegree 1. In a weighted digraph $G = (V, E)$, a minimum weight branching is a collection of edges of E such that these edges form a branching of G and the sum of the weights on these edges is minimum. In [L 85] Lovász shows that this problem, in which the edge weights are polynomially bounded, is in NC by giving an algorithm that finds the branching in time $O(\log^2 n \log w)$ using $O(n^3)$ processors, where w is the number of bits used to represent the weights on the edges. The algorithm we present below finds such a branching in the digraph G in time $O(\log^2 n)$ using $Q(n)$ processors, where the weights of the edges are bounded by some constant k, and is based on the algorithm presented by Lovász.

Algorithm 2: Minimum Weight Branching of a Digraph
Input: A digraph G with edge weights $w \in \{0, 1, \ldots, k\}$, for some fixed k, and a root r.
Output: A minimum weight branching of G rooted at r.

1. Without loss of generality, assume that r has outdegree 1 such that the outgoing edge from r has a very large weight (if not, create such an r in $O(1)$ time). For each vertex v determine the minimum of the weights of its entering edges. If we subtract this minimum weight from the weight of each edge entering v we will shift the value of the minimum branching but the edges in a minimum branching will remain unchanged. We now have a graph in which every vertex has an entering edge with weight 0. Form the subgraph H by choosing an entering edge with weight 0 for every vertex.

2. If H is a branching, it has minimum weight and we are done. If not, there exists at least one directed cycle in H. Identify the directed cycles $H_1, H_2, \ldots, H_k$ of H. (Thus far the algorithm is the same as Edmonds' [E 67].)

3. For each cycle H_i, construct the set of vertices V_i that are reachable from H_i. For each H_i form a strongly connected component U_i by determining those vertices $v \in V_i$ whose distance to H_i is less than or equal to the distance from every vertex outside of V_i to H_i. Form a new graph G' by contracting the sets U_i into a single vertex u_i. Define the following weighting for the edges of G'

$$w'(x,y) = \begin{cases} w(x,y) - \beta_i + d_i(y) & \text{if } y \in U_i,\ 1 \le i \le k \\ w(x,y) & \text{otherwise} \end{cases}$$

where $w(x,y)$ $(w'(x,y))$ is the weight of edge (x,y) in G (G'), $d_i(y)$ is the shortest distance from y to H_i, and β_i is the shortest distance from any vertex outside of V_i to H_i.

4. Form the subgraph H', in which each vertex has one entering edge with weight 0, from G' as follows. Note that, by the definition of β_i and w', each vertex $u_i \in G'$ has an entering edge with weight 0; choose one of these edges for each u_i and for all other vertices choose the same edge that was chosen in H. Recursively apply the above steps, beginning with step 2, to the new graph G', and the just formed subgraph H', to find a minimum weight branching in it.

We now show how the above algorithm may be implemented in time $O(\log^2 n)$ using $Q(n)$ processors. Step 1 may be accomplished in time $O(\log n)$ using $O(n^2)$ processors, and the subgraph H' in step 4 can be formed in $O(1)$ time using $O(n^2)$ processors by assigning n processors to every vertex.

In order to implement step 2 we first note that if we form a branching of each connected component of H there will be at most one non-tree edge in each branching because each vertex has indegree of at most one. In addition, these non-tree edges will identify any directed cycles that exist. We can find a spanning forest of the undirected version of H using an optimal parallel algorithm [KR 90]. Introducing the orientation of the edges into the undirected spanning forest will translate it into a branching of H, in which the connected components of H are identified. If none of the connected components has a non-tree edge H is a branching, otherwise we have identified the directed cycles in the connected components. Thus, step 2 can be implemented in time $O(\log^2 n)$ using $O(n)$ processors.

In order to form the new graph G' in step 3 we begin by considering each cycle H_i, its connected component V_i, and all the edges $(x, y) \in G$ where both x and y are contained in V_i. For each such group we compute the shortest distance from each vertex $v \in V_i$ to the cycle H_i. This is done by reversing the direction of all the edges and considering the cycle H_i as the single source for Algorithm 1. We now have $d_i(v)$, the shortest distance for every vertex $v \in V_i$ to the cycle H_i. We next calculate β_i, the shortest distance from some vertex $v \in V - V_i$ to the cycle H_i as follows. Each vertex $x \in V_i$ chooses the edge with minimum weight entering it from a vertex $v \in V - V_i$ and adds the weight of this edge to its distance from H_i; the minimum of these values is β_i. From this point the formation of the U_i sets and the new graph G' is trivial. The step can be accomplished in time $O(\log n)$ using $Q(n)$ processors.

It is established in [L 85] that if B' is a minimum weight branching in G' and B is a minimum weight branching in G, then $w'(B') = w(B) - \beta_1 - \beta_2 - \ldots - \beta_k$. This shows that if a minimum weight branching is found for the graph G' it can easily be extended to a minimum weight branching in G. In addition, it is shown in [L 85] that every directed cycle in H' will contain at least two vertices that were previously shrunk. This establishes that there will be at most a logarithmic number of iterations in this algorithm. Therefore, since we have $O(\log n)$ iterations, each taking time $O(\log n)$ and using $Q(n)$ processors, the entire algorithm will run in time $O(\log^2 n)$ using $Q(n)$ processors.

5 Applications of SSSP/BFS and Minimum Weight Branchings

An ear-decomposition of a digraph $G = (V, E)$ is a partition of E into an ordered collection of ears (edge-disjoint simple directed paths or simple directed cycles), $D = [P_0, P_1, \ldots, P_k]$, such that P_0 is a simple directed cycle and for $1 \leq i \leq n$, each endpoint of P_i belongs to a smaller numbered ear while the internal vertices of P_i do not belong to any smaller numbered ear. For undirected graphs, efficient ear-decomposition algorithms imply efficient solutions to a number of other problems such as 2-edge connectivity and biconnectivity [KR 90]. It is known that a digraph is strongly connected if and only if it has an ear-decomposition [L 85]; this and the prospect of other applications, as in the undirected case, has lead to interest in developing ear-decomposition techniques for strongly connected digraphs. Lovász shows that finding an ear-decomposition of a strongly connected digraph $G = (V, E)$ is in NC by giving an algorithm that runs in time $O(\log^2 n)$ using $O(n^4)$ processors [L 85]; the general strategy of this algorithm is to merge a forward and reverse branching of G by an assignment of labels to the edges so that edges with the same label designate an ear. In the appendix, we show that Lovász's algorithm can be implemented in time $O(\log n)$ using $Q(n)$ processors by using Algorithm 1 and a judicious combination of well known efficient parallel algorithms such as pointer jumping and the Euler tour technique [KR 90]. Lucas and Sackrowitz [LM 91] have proposed another $O(\log n)$ time, $Q(n)$ processor algorithm that is similar to that of Lovàsz but uses a simpler technique of labeling the edges; the advantage of their approach is that it only requires SSSP/BFS in an unweighted digraph, whereas that of Lovász uses SSSP/BFS in a weighted digraph.

Given a strongly connected digraph, the transitive reduction problem is to determine a *minimal* strongly connected spanning subgraph of it. This spanning subgraph is minimal in the sense that if any of its edges were removed it would not be strongly connected. A definition of this problem and algorithms, both parallel and sequential, for its solution are given in [GKRST 91]. The parallel algorithm runs in time $O(\log^4 n)$ with $O(n^3)$ processors on a CREW PRAM. The subtask in this algorithm responsible for the processor bound is the computation of a minimum weight branching in a digraph with edge weights of 0 and 1. Consequently, by using Algorithm 2 to find this branching the processors necessary can be reduced to $Q(n)$ without increasing the running time.

6 Acknowledgement

The author would like to thank Vijaya Ramachandran for many useful discussions and suggestions.

References

[CW 87] D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, *Proc. 19th Ann. ACM Symp. on Theory of Computing* May (1987), pp. 1-6.

[E 67] J. Edmonds, Optimum Branchings, *J. Res. Nat. Bureau of Standards* **71B** (1967), pp. 233-240.

[GM 88] H. Gazit and G.L. Miller, An Improved Parallel Algorithm That Computes the BFS Numbering of a Directed Graph, *Information Processing Letters* **28** (1988), pp. 61-65.

[GKRST 91] P. Gibbons, R. Karp, V. Ramachandran, D. Soroker, R. Tarjan, Transitive Reduction in Parallel via Branchings, *Journal of Algorithms* **12** (1991), pp. 110-125.

[KR 90] R. Karp, V. Ramachandran, Parallel Algorithms for Shared-Memory Machines, *Handbook of Theoretical Computer Science - Volume A*, J. Van Leeuwan, ed., Elsevier Science Publishers/The MIT Press, Amsterdam, (1990), pp. 869-941.

[L 85] L. Lovász, Computing Ears and Branchings in Parallel, *Proc. 26th Annual Symposium on Foundations of Computer Science* (1985), pp. 464-467.

[LM 91] J. Lucas and M. Sackrowitz, Private Communication, (1991).

[R 88] V. Ramachandran, Fast and Processor-Efficient Parallel Algorithms for Reducible Flow Graphs, Tech. report #UILU-ENG-88-2257 (ACT #103), Coordinated Science Lab, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801 (1988).

A Appendix - An Ear Decomposition Algorithm for Digraphs

An ear-decomposition of a digraph $G = (V, E)$ is a partition of E into an ordered collection of ears (edge-disjoint simple directed paths or simple directed cycles), $D = [P_0, P_1, \ldots, P_k]$, such that P_0 is a simple directed cycle and for $1 \leq i \leq n$, each endpoint of P_i belongs to a smaller numbered ear while the internal vertices of P_i do not belong to any smaller numbered ear. It is known that a digraph has an ear-decomposition if and only if it is strongly connected [L 85]. Lovász shows that finding an ear-decomposition of a strongly connected digraph $G = (V, E)$ is in NC by giving an algorithm that runs in time $O(\log^2 n)$ using $O(n^4)$ processors [L 85]. We show, by using Algorithm 1 and efficient parallel algorithms such as pointer jumping and the Euler tour technique [KR 90], that his algorithm can be implemented in time $O(\log n)$ using $Q(n)$ processors.

Since each edge belongs to exactly one ear, we can numerically label the edges in E so that these labels uniquely identify an ear-decomposition by assigning the same label to all edges in the same ear and requiring that the label of ear P_i be less than the label of ear P_{i+1}. The following algorithm constructs such a labeling of a strongly connected digraph $G = (V, E)$ corresponding to an ear-decomposition of G. Its correctness is shown in [L 85]. The general strategy of this algorithm is to construct the ear-decomposition by merging a forward and reverse branching of G from some root r. From these branchings we will have an $r \rightarrow v$ and a $v \rightarrow r$ path for each $v \in V$, which if combined form a cycle. In general, there will be some redundancy in these cycles that must be removed to ensure that the internal vertices of an ear are not contained in any previous ear. The following algorithm avoids this redundancy by assigning a priority hierarchy to the cycles and including only those portions of these cycles that are not contained in some ear of higher priority.

Algorithm: Parallel Ear-Decomposition
Input: A strongly connected digraph $G = (V, E)$.
Output: An ear-decomposition of G.

1. Pick a root r and construct a forward branching, $B = (V_B, E_B)$, of G rooted at r. Label the edges in $E - E_B$ arbitrarily with unique integers referred to as ψ labels. For each vertex $v \in V$ let F_v denote the unique $r \rightarrow v$ path in B.

2. Reassign the weights of the edges; 0 for edges in E_B, and 1 otherwise. Form a reverse branching $R = (V_R, E_R)$ by finding, for every $v \in V$, the shortest $v \rightarrow r$ path in G; let R_v denote the unique $v \rightarrow r$ path in R.

3. Extend the ψ labels to the edges in E_B by giving an unlabeled edge the same label

as the first labeled edge on the path back to the root in R from the vertex which the unlabeled edge enters.

4. Assign λ labels to the edges of the form

$$\lambda(u,v) = (|(E(R_v) \cup (u,v)) - E_B|, |(E(F_v) \cup (u,v)) - E_R|, \psi(u,v))$$

   where $E(R_v)$ $(E(F_v))$ denotes the edges in R_v (F_v). These λ labels uniquely determine an ear-decomposition of the graph G; those edges with the same label form the ears, and the ordering of the ears is determined by the value of the labels with priority assigned to the first, second and third components of the labels in that order.

5. If desired, sort the λ labels to obtain the ordering of the ears.

We will now show how each of these steps may be implemented. In every case, optimal parallel algorithms exist for all techniques used except for the solution of the SSSP problem which requires $Q(n)$ processors. In order to construct the branching in step 1 we assign a weight of 1 to each edge in E and find the shortest path from r to every other vertex in $v \in V$. For each vertex $v \in V$ we mark the last edge in this path as the edge entering v in B. This can be done using Algorithm 1.

The rational behind the reassignment of weights in step 2 is to ensure that the final labels that determine the ears assigned in step 4 will be properly ordered. This is shown in [L 85]. The reverse branching R is easily computed by reversing the directions of all edges $e \in E$ while maintaining the new weights assigned above and applying Algorithm 1.

In order to extend the ψ labels in step 3 to the edges in B we will think of vertices as being labeled. If a vertex has an outgoing edge that is labeled it will assign that same label to itself. An edge in B will be assigned the label of the vertex that it enters. Our strategy will be to isolate the relevant edges that are contained in the R_v path for some vertex v and to determine the first edge on this path that is not contained in B. We note that any edge $e \in B$ that is not in R cannot be in the unique vertex to root path R_v, for any $v \in V$. Furthermore, for every internal vertex in B, there is at most one outgoing edge in B that is also in R. Therefore, we can form linked lists of edges in B by disregarding the edges in $E_B - E_R$. The label of every vertex in these newly formed linked lists will be either the label of an outgoing edge of that vertex or of the first vertex in the list below it that is labeled. (From the structure of B and R we are assured that every leaf of B has an outgoing edge in $E - E_B$). We can further decompose these lists by assigning every vertex with a labeled outgoing edge that same label and removing the edge leaving that vertex in B, if there is one. We are now left with a set of linked lists of vertices in B in which the bottom vertex in the list is labeled and all other vertices are unlabeled. Now we simply use the pointer jumping technique to propagate the label from the bottom vertex in each list to all other vertices in the list. This whole process can be accomplished optimally in time $O(\log n)$ using $O(n/\log n)$ processors.

When calculating the λ labels in step 4, we note that we already have the third component of label, but we must compute the first and second. The computation of the first component is trivial. When we were forming the reverse branching R we assigned the edges in B a weight of 0 and all other edges a weight of 1. Thus, when we found the shortest vertex to root path, R_v, for each vertex $v \in V$ we were simply counting the number of edges in this path that were not in B. Consequently, for an edge (u, v) we determine the first component by finding the distance from v to the root and adding one to this distance if (u, v) is not in B. For the second component we need to determine, for all vertices $u \in V$, the number of edges in the F_u path that are not in R. We can use the Euler tour technique on the tree B to solve this problem optimally. We begin with the tree B and for every edge (u, v) we add an edge (v, u). This gives us an Euler tour of B. We now assign a weight of 1 to those edges that are in $B - R$ and a weight of -1 to their corresponding back edges. For those edges that are in R, and their corresponding back edges, we assign a weight of 0. Now the sum of the weights of the edges on the path from the root to the first occurrence of any vertex v along the Eulerian tour will be the number of edges in the path F_v that are not in R, as desired. These sums can be calculated optimally by list ranking in time $O(\log n)$ with $O(n/\log n)$ processors.

All portions of the first four steps of the algorithm are accomplished optimally except the matrix multiplications in the SSSP computations which require $Q(n)$ processors and time $O(\log n)$. Consequently, an ear-decomposition of a strongly connected digraph G can be obtained in time $O(\log n)$ using $Q(n)$ processors. If desired, the λ labels may be sorted so that the ears may be output in order.