Information Processing Letters 28 (1988) 61–65
North-Holland

24 June 1988

# AN IMPROVED PARALLEL ALGORITHM THAT COMPUTES THE BFS NUMBERING OF A DIRECTED GRAPH *

Hillel GAZIT and Gary L. MILLER

*Department of Computer Science, University of Southern California, University Park, Los Angeles, CA 90089-0782, U.S.A.*

Communicated by G.R. Andrews
Received 3 September 1987
Revised 24 February 1988

This paper presents a parallel algorithm that computes the *breadth-first search* (BFS) numbering of a directed graph in $O(\log^2 n)$ time using $M(n)$ processors on the exclusive-read exclusive-write (EREW) parallel random access machine (PRAM) model, where $M(n)$ denotes the number of processors needed to multiply two $n \times n$ integer matrices over the ring $(\mathbb{Z}, +, \times)$ in $O(\log n)$ time. The best known bound for $M(n)$ is $O(n^{2.376})$ (Coppersmith and Winograd, 1987). The algorithm presented in their paper uses fewer processors than the classical algorithm for BFS that employs matrix powering over the semiring (dioid) $(\mathbb{N}, \min, +)$, using $O(\log n)$ time and $O(n^3)$ processors on the concurrent-read concurrent-write (CRCW) model, or using $O(\log^2 n)$ time and $n^3/\log n$ processors on the EREW model.

*Keywords*: Single source, breadth-first search, fast matrix multiplication, parallel algorithm

## 1. Introduction

The single source *breadth-first search* (BFS) of a graph $G = (V, E)$ with respect to a vertex $s$ is an assignment of labels to the vertices of $G$ such that the label on vertex $v$ is the distance from $s$ to $v$. This problem is also known as the single-source, shortest-path problem of unit-length edges. With these labels, a BFS tree of the graph can be built using $|E|$ processors in $O(1)$ time. BFS is one of the basic paradigms for the development of efficient sequential algorithms. It can easily be seen that the sequential time of the problem is $O(n + m)$, where $n$ and $m$ are the number of vertices and edges, respectively [1]. On the other hand, a parallel BFS seems to require a substantial number of processors. The classic naive parallel algorithm views $G$ as an incidence matrix $M$ over the semiring $(\mathbb{N}, \min, +)$, where $\mathbb{N}$ denotes the set of natural numbers and repeatedly squares $M$, returning $M^n$. This yields an algorithm using $O(\log n)$ time and $n^3$ processors on the concurrent-read concurrent-write (CRCW) model. [1] A similar algorithm in the exclusive-read exclusive-write (EREW) mode takes $O(\log^2 n)$ time and uses $n^3/\log n$ processors. This processor count arises because all known algorithms for matrix multiplication over this semiring require $n^3$ operations.

If one is able to work over a ring instead of a semiring, one can use one of many processor-efficient algorithms—at least in the limit as $n$ goes to infinity. The simple $O(n^3)$ sequential algorithm for matrix multiplication over a ring was first improved by Strassen [5] to obtain a complexity of $O(n^{2.81})$. This upper bound has been progressively improved upon by the work of many researchers.

* This research was supported in part by the National Science Foundation under Grant No. DCR-8514961.

[1] The $n$ min operations can be performed in $O(1)$ time using $n$ processors, since the size of each value is $\leqslant n$. This fact is relatively easy to see, but we do not have a proper reference for it.

For a good summary on the subject, see [4]. So far, the best-known result for matrix multiplication is $O(n^{2.376})$ obtained by Coppersmith and Winograd [3]. All of these algorithms can be implemented in parallel, with optimal speed-up, in $O(\log n)$ time.

In this paper, $M(n)$ denotes the number of processors needed for matrix multiplication over the integers of two $n \times n$ matrices in $O(\log n)$ time. Thus, the best-known value for $M(n)$ is $O(n^{2.376})$ [3].

These algorithms use the fact that there exists an inverse element with respect to a $+$ operator. Since the semiring $(\mathbb{N}, \min, +)$ does not have additive (for min) inverses, these algorithms cannot be applied directly. This paper shows how to use the fast matrix multiplication algorithms indirectly.

Our basic algorithm uses the EREW parallel random access machine (PRAM) model, in which concurrent *reads* or *writes* to the same memory location are not permitted. It is assumed that processors can add, multiply, and compare $\log n$ bit integers in unit time.

For the result pertaining to a BFS from more than one source, the CRCW PRAM model will be used, in which concurrent writes to the same memory location are permitted only if all processors try to write the same value.

Our algorithm replaces $\log n$ matrix multiplications over the semiring $(\min, +)$ by $\log n$ arithmetic matrix multiplications over the ring $(+, \times)$ and $\log n$ vector–matrix multiplication operations. Matrix multiplication has an obvious lower bound of $O(n^2)$; therefore, the processor time product of our BFS algorithm cannot be as good as the sequential time of $O(|E|)$.

## 2. Notation

Let $G = (V, E)$ be a directed graph with $n$ vertices and $m$ edges. Assume that the vertex set is the set of integers, $\{1, 2, \ldots, n\}$. The *distance* from vertex $u$ to $v$ is the number of edges in the shortest directed path from $u$ to $v$, and equals infinity $(\infty)$ if no path exists between $u$ and $v$. The distance is denoted by $d(u, v)$.

The semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$ shall be used, where $\mathbb{N}$ denotes the set of natural numbers and infinity $(\infty)$ satisfies the following natural rules:

$$\infty + k = \infty, \qquad \infty \cdot k = \infty, \qquad \infty \min k = k.$$

Matrix multiplication over the above semiring is denoted by $*$.

## 3. Computing the BFS of a directed graph

The algorithm presented in this section computes the BFS of a directed graph $G(V, E)$ from some given vertex $s$. The algorithm is comprised of three parts. In the first part, $\lceil \log n \rceil$ matrices of size $n \times n$ are computed; the nonzero entries determine those pairs of vertices which are a distance of at most $2^i$ apart. In the second part, the algorithm rewrites the information from the first part in terms of distances. In the third part, the actual distances are computed.

These three parts are combined into Procedure BFS described below. The BFS algorithm takes a graph $G(V, E)$ and a vertex $s$ as input and computes a vector $D$ of length $n$ such that $D[w]$ equals the distance from $s$ to $w$.

**Procedure BFS**

**program** BFS($G(V, E)$, $s$)
  *First-Approximation*($G(V, E)$);
  *Find-Distances*($s$);
**end** BFS

### 3.1. First approximation

In this section, an approximation of the distance between all pairs of vertices in $G$ is computed. It is well known that Boolean matrix multiplication can be reduced to integer multiplication (see [1, pp. 242–243] and [2]). The model presented in this paper assumes that $\log n$ bit numbers can be multiplied in $O(1)$ time. Thus, we begin by viewing the incidence matrix of $G$ as a Boolean matrix:

$$B_0[u, v] = \begin{cases} 1 & \text{if } (u, v) \in E \text{ or } u = v, \\ 0 & \text{otherwise.} \end{cases}$$

By letting $B_{i+1} = B_i^2$, the first $\lceil \log n \rceil$ matrices can be computed by doubling-up over the Boolean semiring. This produces $B_1, \ldots, B_{\lceil \log n \rceil}$, satisfying

$$B_i[u, v] = \begin{cases} 1 & distance(u, v) \leqslant 2^i, \\ 0 & \text{otherwise.} \end{cases}$$

There are $\lceil \log n \rceil$ matrix products computed over the Boolean semiring. Each product requires at most $O(\log n)$ time and $M(n)$ processors. Thus, $O(\log^2 n)$ time and $M(n)$ processors are needed to compute these matrices.

New values are substituted into the entries of each matrix $B_i$, and the resulting new matrices are viewed over the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$. This computation will give the matrices $M_1, \ldots, M_{\lceil \log n \rceil}$ as defined below:

$$M_i[u, v] = \begin{cases} 0 & \text{if } u = v, \\ 2^i & \text{if } distance(u, v) \leqslant 2^i \\ & \text{and } u \neq v, \\ \infty & \text{otherwise.} \end{cases}$$

Given the matrix $B_i$, $M_i$ can be computed in constant time using at most $n^2$ processors. Thus, the distance between any pair of vertices has been approximated up to a factor of 2. The next subsection will show how one can efficiently combine this information to get the BFS of $G$.

### 3.2. Computing the distances

In the previous two parts of the algorithm, approximation to the distances between any two vertices in the graph were found. It was shown that $M_i[u, v] \neq \infty$ if and only if the distance between $u$ and $v$ is at most $2^i$. The least $i$ can be found such that $M_i[u, v] = 2^i$, which implies that the distance between $u$ and $v$ is between $2^{i-1} + 1$ and $2^i$.

In this section, an attempt will be made to try to improve these estimates for a given vertex $s$ (the root of the BFS tree). An algorithm will be presented that computes the distance of $s$ from any other vertex. In the algorithm, at the beginning of each iteration $i$, the error in the estimated distances is at most $2^i$. At the end of the iteration, the error is reduced to $2^{i-1}$. Suppose that, at the end of iteration $i$, the distance from $s$ to $w$ is estimated to be $k \cdot 2^i$. During the $(i-1)$st iteration, it will be determined whether $d(s, w)$ lies in the range $(2k-2)2^{i-1}+1$ to $(2k-1)2^{i-1}$ or in the range $(2k-1)2^{i-1}+1$ to $(2k)2^{i-1}$. The former case holds if and only if $\exists u$ such that $(k-2)2^i < d(s, u) \leqslant (k-1)2^i$, and $d(u, w) \leqslant 2^{i-1}$. If this is the case, at the end of the $i$th iteration, the distance estimates of $u$ from $s$ must be $(k-1)2^i$, and $M_{i-1}[u, w] \neq \infty$. Therefore, if such a vertex $u$ exists, the distance estimates from $s$ to $w$ are corrected to the lower half of the range; otherwise, they are corrected to the upper half of the range.

In the *Find-Distances* algorithm, $D$ is a row vector of length $n$, and $*$ denotes matrix multiplication over the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$. All the distances in the graph can be computed from some vertex $s$ in the following way:

**Procedure *Find-Distances***

**for** $1 \leqslant u \leqslant n$ **in parallel do**
  $D[u] = \begin{cases} 0 & \text{if } u = s \\ \infty & \text{otherwise} \end{cases}$
**od**
**for** $i := \lceil \log n \rceil$ **downto** 0 **do**
  $D := D * M_i$
**od**
**end** *Find-Distances*

**3.1. Lemma.** *The complexity of Procedure Find-Distances in the* EREW *model is performed in* $O(\log^2 n)$ *time, using* $n^2/\log n$ *processors.*

**Proof.** There are $\log n$ iterations in Procedure *Find-Distances*, and each of them can be performed on the EREW model in the following way:

*Step* 1. Make $n$ copies of the vector $D$.

*Step* 2. Compute the $n^2$ + operations.

*Step* 3. Compute $n$ min operations, each on a set of $n$ numbers.

It is obvious that each operation can be performed in $\log n$ time using $n^2/\log n$ processors. □

**3.2. Corollary.** *The complexity of a BFS from a single source on the* EREW *model is* $O(\log^2 n)$ *time, using* $M(n)$ *processors.*

**Proof.** The first step involves performing $\log n$ matrix multiplications over the Boolean semiring. Each multiplication takes $O(\log n)$ time and uses $M(n)$ processors [1,2]. The complexity of the second step is bounded by Lemma 3.1. □

The accuracy of the algorithm must still be proven. The correctness of the *Find-Distances* algorithm can be proven by induction, by showing that

$$(M_i * M_{i-1} * \cdots * M_0)[u, v] = distance(u, v)$$

for $distance(u, v) \leqslant 2^{i+1} - 1$. The correctness of the algorithm is proven by showing that in every iteration for every element of $D$ there is at most one possible improvement. Thus, the time of the *Find-Distances* algorithm is reduced to $\log n$ using $n^2$ processors. This observation is important if one desires a BFS of the graph from more than one source without using more processors.

After iteration $i$ ($i$ decreasing) of Procedure *Find-Distances*, the row vector $D[w]$ is denoted by $D^i[w]$. As defined in Section 2, let $d(u, v)$ be the distance from $u$ to $v$.

**3.3. Theorem.** *After every iteration i, for every* $w \in V$,

$$D^i[w] = 2^i\lceil distance(s, w)/2^i\rceil.$$

**Proof.** The theorem is proven by induction on $i$ for $i$ running from $\lceil \log n \rceil$ to 0. Initially, $D^{\lceil \log n \rceil}(N)$ is in the $s$th row of $M_{\lceil \log n \rceil}$. Thus, the claim follows from the definition of $M_{\lceil \log n \rceil}$. The theorem is assumed correct for $i+1$ and proven for $i$ ($i$ decreasing). Note that

$$D^i[w] \geqslant 2^i\lceil d(s, w)/2^i\rceil.$$

Next, $D^i[w]$ is proven equal to some $D^{i+1}[u] + M_i[u, w]$. By the definition of $M[u, w]$,

$$M[u, w] \geqslant 2^i\lceil d(u, w)/2^i\rceil.$$

Therefore,

$$\begin{aligned} D^i[w] &= 2^{i+1}\lceil d(s, u)/2^{i+1}\rceil + 2^i\lceil d(u, w)/2^i\rceil \\ &\geqslant 2^i\lceil [d(s, u) + d(u, w)]/2^i\rceil \\ &\geqslant 2^i\lceil d(s, w)/2^i\rceil. \end{aligned}$$

Next,

$$D^i[w] \leqslant 2^i\lceil d(s, w)/2^i\rceil$$

is proven. If $d(s, w) = \infty$, the proof is complete, so it may be assumed that $d(s, w) < \infty$. The next procedure involves setting

$$d(s, w) = q \cdot 2^{i+1} + r, \quad \text{where } 0 \leqslant r < 2^{i+1}.$$

Two cases will be examined. In the first case, $r > 2^i$ or $r = 0$. In this case,

$$\begin{aligned} D^i[w] &\leqslant D^{i+1}[w] = 2^{i+1}\lceil d(s, w)/2^{i+1}\rceil \\ &= 2^i\lceil d(s, w)/2^i\rceil. \end{aligned}$$

In the second case, $0 < r \leqslant 2^i$. In this case there must exist a vertex $u$ such that $d(s, u) = q \cdot 2^{i+1}$ and $d(u, w) = r$. By the induction hypothesis, $D^{i+1}[u] = q \cdot 2^{i+1}$. By the definition of $M_i$ and the fact that $d(u, w) = r \leqslant 2^i$ it follows that $M_i[u, w] = 2^i$. Therefore, $D^i[w]$ is at most

$$(2q + 1)2^i = 2^i\lceil d(s, w)/2^i\rceil. \quad \square$$

**3.4. Corollary.** *After applying Procedure Find-Distances,* $d(s, k) = D[k]$.

**Proof.** Substitute $i = 0$ in Theorem 3.3 to get $D^0[w] = d(s, w)$. □

**3.5. Lemma.** *For every* $w \in V$ *and every iteration* $i$, $D^{i+1}[w] = D^i[w]$ *or* $D^{i+1}[w] - D^i[w] = 2^i$.

**Proof.** By Theorem 3.3,

$$D^{i+1}[w] = \lceil 2^{i+1} \cdot d(s, w)/2^{i+1}\rceil$$

and

$$D^i[w] = \lceil 2^i \cdot d(s, w)/2^i\rceil.$$

If $d(s, w)$ modulo $2^{i+1}$ is greater than $2^i$, then $D^i[w] = D^{i+1}[w]$; otherwise, $D^i[w] = D^{i+1}[w] - 2^i$. □

**3.6. Corollary.** *The n* min *operations in every matrix multiplication can be computed in* $O(1)$ *time on the* CRCW *model using* $n^2$ *processors.*

**Proof.** The proof immediately follows from Lemma 3.5. □

**3.7. Theorem.** *The Find-Distances algorithm can be computed on the* CRCW *model in* $O(\log n)$ *time using* $n^2$ *processors.*

**Proof.** In each iteration, $n^2$ + and $n$ min operations are computed. By Corollary 3.6, the claim of the theorem follows. □

Note that Theorem 3.7 implies that a BFS can be performed simultaneously from several vertices. By applying Procedure *Find-Distances* from each source vertex, Procedure BFS can be performed from $M(n) \log n/n^2$ vertices, in parallel, in $O(\log^2 n)$ time using $M(n)$ processors on the CRCW model.

## 4. Conclusions and open problems

A reduction of the BFS problem to the matrix multiplication problem has been demonstrated in this paper. This method significantly reduces the processor count without substantially increasing the running time.

Two related open questions remain:

**(1)** Can the algorithm be generalized in the case where the edges have polynomially bounded integer weights?

**(2)** Can the algorithm be generalized to solve the all-pairs, shortest-path problem?

## Acknowledgment

The authors would like to thank the referees for many insightful comments. We are also grateful to Esther Ashby for her editorial corrections.

## References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] S. Baase, *Computer Algorithms: Introduction to Design and Analysis* (Addison-Wesley, Reading, MA, 1983).

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

[4] V. Pan, *How to Multiply Matrices Faster* (Springer, Berlin, 1984).

[5] V. Strassen, Gaussian elimination is not optimal, *Numerische Mathematik* **13** (1969) 354–356.