Solving Graph Theory Problems Using Reconfigurable Pipelined Optical Buses

Keqin Li[1], Yi Pan[2], and Mounir Hamdi[3]

[1] Dept. of Math & Computer Sci., State Univ. of New York, New Paltz, NY 12561
li@mcs.newpaltz.edu
[2] Dept. of Computer Science, University of Dayton, Dayton, OH 45469
pan@cps.udayton.edu
[3] Dept. of Computer Sci., Hong Kong Univ. of Sci. & Tech., Kowloon, Hong Kong
hamdi@cs.ust.hk

Abstract. We solve a number of important and interesting problems from graph theory on a linear array with a reconfigurable pipelined optical bus system. Our algorithms are based on fast matrix multiplication and extreme value finding algorithms, and are currently the fastest algorithms. We also distinguish the two cases where weights have bounded/unbounded magnitude and precision.

1 Introduction

It has been recognized that many important and interesting graph theory problems can be solved based on matrix multiplication. A representative work is a parallel matrix multiplication algorithm on hypercubes, which has time complexity $O(\log N)$ on an N^3-processor hypercube for multiplying two $N \times N$ matrices [3]. By using this algorithm, many graph problems can be solved in $O((\log N)^2)$ time on an N^3-processor hypercube. Since the summation of N values takes $O(\log N)$ time even on a completely connected network, further reduction in the time complexity seems difficult due to the limited communication capability of static networks used in distributed memory multicomputers.

The performance of parallel algorithms on shared memory multiprocessors, e.g., PRAMs, can be much better. For instance, on a CRCW PRAM, a boolean matrix product can be calculated in constant time by using $O(N^3)$ processors. This implies that the transitive closure and many related problems of directed graphs can be solved in $O(\log N)$ time by using N^3 processors [5]. Since a general matrix multiplication takes $O(\log N)$ time even on a PRAM, the all-pairs shortest problem requires $O((\log N)^2)$ time, no matter how many processors are used.

Using reconfigurable buses, a number of breakthrough have been made. For instance, a constant time algorithm for matrix multiplication on a reconfigurable mesh was reported in [18]. However, the number of processors used is as many as N^4. It is also known that the transitive closure as well as other related problems can be solved in constant time on a reconfigurable mesh [20]. However, the

algorithm only works for undirected graphs, while the problem is defined on directed graphs.

The recent advances in optical interconnection networks have inspired a great interest in developing new parallel algorithms for classic problems [9]. Pipelined optical buses can support massive volume of data transfer simultaneously and realize various communication patterns. An optical bus can also implement some global operations such as calculating the summation and finding the extreme values of N data items in constant time. Furthermore, an optical interconnection can be reconfigured into many subsystems which can be used simultaneously to solve subproblems. The reader is referred to [10, 16] for more detailed discussion on these issues.

In this paper, we solve graph theory problems on the LARPBS (linear arrays with reconfigurable pipelined bus system) computing model. (LARPBS was first proposed in [15, 16], and a number of algorithms have been developed on LARPBS [4, 7, 8, 10–12, 14, 15, 17].) We use matrix multiplication algorithms as subroutines. We show that when the weights are real values with bounded magnitude and precision, the all-pairs shortest paths problem for weighted directed graphs can be solved in $O(\log N)$ time by using $O(N^3)$ processors. If the weights have unbounded magnitude and precision, the problem can be solved in $O(\log N)$ time with high probability by using $O(N^3)$ processors. The transitive closure problem for directed graphs can be solved in $O(\log N)$ time by using $O(N^3/\log N)$ processors. Other related problems are also discussed. These algorithms are currently the fastest.

2 Reconfigurable Pipelined Optical Buses

A pipelined optical bus system uses optical waveguides instead of electrical signals to transfer messages among electronic processors. In addition to the high propagation speed of light, there are two important properties of optical pulse transmission on an optical bus, namely, unidirectional propagation and predictable propagation delay. These advantages of using waveguides enable synchronized concurrent accesses of an optical bus in a pipelined fashion [2, 6]. Such pipelined optical bus systems can support a massive volume of communications simultaneously, and are particularly appropriate for applications that involve intensive regular or irregular communication and data movement operations such as permutation, one-to-one communication, broadcasting, multicasting, multiple multicasting, extraction and compression. It has been shown that by using the coincident pulse addressing technique, all these primitive operations take $O(1)$ bus cycles, where the bus cycle length is the end-to-end message transmission time over a bus [10, 16]. (*Remark.* To avoid controversy, let us emphasize that in this paper, by "$O(f(p))$ time" we mean $O(f(p))$ bus cycles for communication plus $O(f(p))$ time for local computation.)

In addition to supporting fast communications, an optical bus itself can be used as a computing device for global aggregation. It was proven in [10, 16] that by using N processors, the summation of N integers or reals with bounded

magnitude and precision, the prefix sums of N binary values, the logical-or and logical-and of N Boolean values can be calculated in constant number of bus cycles.

A linear array with a reconfigurable pipelined bus system (LARPBS) consists of N processors P_1, P_2, ..., P_N connected by a pipelined optical bus. In addition to the tremendous communication capabilities, an LARPBS can also be partitioned into $k \geq 2$ independent subarrays LARPBS$_1$, LARPBS$_2$, ..., LARPBS$_k$, such that LARPBS$_j$ contains processors $P_{i_{j-1}+1}$, $P_{i_{j-1}+2}$, ..., P_{i_j}, where $0 = i_0 < i_1 < i_2 \cdots < i_k = N$. The subarrays can operate as regular linear arrays with pipelined optical bus systems, and all subarrays can be used independently for different computations without interference (see [16] for an elaborated exposition).

The above basic communication, data movement, and aggregation operations provide an algorithmic view on parallel computing using optical buses, and also allow us to develop, specify, and analyze parallel algorithms by ignoring optical and engineering details. These powerful primitives that support massive parallel communications plus the reconfigurability of optical buses make the LARPBS computing model very attractive in solving problems that are both computation and communication intensive.

3 Matrix Multiplication

The problem of matrix multiplication can be defined in a fairly general way. Let S be a set of data with two binary operators $\oplus$ and $\otimes$. Given two $N \times N$ matrices $A = (a_{ij})$ and $B = (b_{jk})$, where $a_{ij} \in S$, and $b_{jk} \in S$, for all $1 \leq i, j, k \leq N$, the product $C = AB = (c_{ik})$ is defined as

$$c_{ik} = (a_{i1} \otimes b_{1k}) \oplus (a_{i2} \otimes b_{2k}) \oplus \cdots \oplus (a_{iN} \otimes b_{Nk}),$$

for all $1 \leq i, k \leq N$.

Several methods that parallelize the standard matrix multiplication algorithm have been developed on LARPBS by using communication capabilities of optical buses [10]. As a matter of fact, we can establish the following general result.

Lemma 1. *The product of two $N \times N$ matrices can be calculated in $O(T)$ time by using N^2M processors, assuming that the aggregation $x_1 \oplus x_2 \oplus \cdots \oplus x_N$ can be calculated in $O(T)$ time by using $M \geq 1$ processors.* ∎

The Appendix gives the implementation details of the generic matrix multiplication algorithm of Lemma 1. Compared with the results in [10], Lemma 1 is more general in two ways. First, while only numerical and logical data are considered in [10], Lemma 1 is applicable to arbitrary data set S and operations $\oplus$ and $\otimes$. Second, Lemma 1 covers a wide range of processor complexity. For example, if S contains real numbers whose magnitude and precision are bounded, and $\otimes$ and $\oplus$ are numerical multiplication and addition, then the summation of N reals can be calculated in $O(N/M)$ time by M processors, where $1 \leq M \leq N$,

which implies that matrix multiplication can be performed in $O(N/M)$ time by N^2M processors for all $1 \leq M \leq N$. In particular, matrix multiplication can be performed

- in $O(N)$ time by N^2 processors;
- in $O(1)$ time by N^3 processors.

These results were developed in [10].

For boolean matrices, where S is the set of truth values, and $\otimes$ is the logical-and, and $\oplus$ is the logical-or, a different approach has been adopted. The method is to parallelize the Four Russians' Algorithm, with the following performance [7].

Lemma 2. *The product of two $N \times N$ boolean matrices can be calculated in $O(1)$ time by using $O(N^3/\log N)$ processors.* ∎

4 Finding Extreme Values

One instance of $\oplus$ is to find the minimum of N data. This operation is used in several graph problems.

Let S be the set of real values. The following result is obvious, since even the radix sorting algorithm can be implemented on an LARPBS in constant time by using N processors when the magnitude and precision are bounded [16].

Lemma 3A. *The minimum value of N data with bounded magnitude and precision can be found in $O(1)$ time by using N processors.* ∎

When the reals have unbounded magnitude or precision, different approaches are required. One approach is to use more processors. It is obvious that by using N^2 processors, the minimum can be found in constant time by making all the possible comparisons [17]. The method can be easily generalized to the following, using the same method in PRAM [19].

Lemma 3B. *The minimum value of N data with unbounded magnitude and precision can be found in $O(1)$ time by using $N^{1+\delta}$ processors, where $\delta > 0$ is any small constant.* ∎

By using the above method in Lemma 3B as a subroutine, the well known doubly logarithmic-depth tree algorithm [5] has been implemented on LARPBS, that can find the minimum of N data in $O(\log\log N)$ time by using N processors [16]. The number of processors can easily be reduced by a factor of $O(\log\log N)$.

Lemma 3C. *The minimum value of N data items with unbounded magnitude and precision can be found in $O(\log\log N)$ time, by using $N/\log\log N$ processors.* ∎

The third method to handle general real values with unbounded magnitude and precision is to use randomization, as shown by the following lemma [17, 19].

Lemma 3D. *The minimum value of N data with unbounded magnitude and precision can be found in $O(1)$ time with high probability (i.e., with probability $1 - O(1/N^\alpha)$ for some constant $\alpha > 0$) by using N processors.* ∎

By Lemma 1 and Lemmas 3A-3D, we have

Theorem 4. *When $\oplus$ is the "min" operation, the product of two $N \times N$ matrices can be calculated in $O(1)$ time by using N^3 processors, if the matrix entries are of bounded magnitude and precision. For matrix entries of unbounded magnitude and precision, the problem can be solved*

- *in $O(1)$ time by using $N^{3+\delta}$ processors;*
- *in $O(\log\log N)$ time by using $N^3/\log\log N$ processors;*
- *in $O(1)$ time with high probability by using N^3 processors.* ∎

5 Repeated Squaring

It turns out that to solve graph theory problems, we need to calculate the Nth power of an $N \times N$ matrix A. This can be obtained by $\lceil \log N \rceil$ successive squaring, i.e., calculating A^2, A^4, A^8, and so on. Such a computation increases the time complexities in Theorem 4 by a factor of $O(\log N)$.

Theorem 5. *When $\oplus$ is the "min" operation, the Nth power of an $N \times N$ matrix can be calculated in $O(\log N)$ time by using N^3 processors, if the matrix entries are of bounded magnitude and precision. For matrix entries of unbounded magnitude and precision, the problem can be solved*

- *in $O(\log N)$ time by using $N^{3+\delta}$ processors;*
- *in $O(\log N \log\log N)$ time by using $N^3/\log\log N$ processors;*
- *in $O(\log N)$ time with high probability by using N^3 processors.* ∎

The last claim in Theorem 5 needs more explanation. In one matrix multiplication, there are N^2 simultaneous computation of minimum values using the Monte Carlo method in Lemma 3D. Since the algorithm in Lemma 3D has failure probability $O(1/N^\alpha)$, the failure probability of one matrix multiplication is $O(N^2/N^\alpha) = O(1/N^{\alpha-2})$. Since this Monte Carlo matrix multiplication is performed for $\lceil \log(N-1) \rceil$ times, the success probability of the all-pairs shortest paths computation is

$$\left(1 - O\left(\frac{1}{N^{\alpha-2}}\right)\right)^{\log N} = 1 - O\left(\frac{\log N}{N^{\alpha-2}}\right) = 1 - O\left(\frac{1}{N^{\alpha-2-\epsilon}}\right),$$

for any $\epsilon > 0$. The above argument implies that we need α (in Lemma 3D) to be no less than, say, 3. Fortunately, this can be easily achieved because a Monte Carlo algorithm which runs in $O(T(N))$ time with probability of success $1 - O(1/N^\alpha)$ for some constant $\alpha > 0$ can be turned into a Monte Carlo algorithm which runs in $O(T(N))$ time with probability of success $1 - O(1/N^\beta)$ for any large constant $\beta > 0$ by running the algorithm for $\lceil \beta/\alpha \rceil$ consecutive times and choosing a one that succeeds without increasing the time complexity.

6 All-Pairs Shortest Paths

Let $G = (V, E, W)$ be a weighted directed graph, where $V = \{v_1, v_2, ..., v_N\}$ is a set of N vertices, E is a set of arcs, and $W = (w_{ij})$ is an $N \times N$ weight matrix, i.e., w_{ij} is the distance from v_i to v_j if $(v_i, v_j) \in E$, and $w_{ij} = \infty$ if $(v_i, v_j) \notin E$. We assume that the weights are real numbers, whose magnitude and precision can be bounded or unbounded. The all-pairs shortest paths problem is to find $D = (d_{ij})$, an $N \times N$ matrix, where d_{ij} is the length of the shortest path from v_i to v_j along arcs in E.

Define $D^{(k)} = (d_{ij}^{(k)})$ be an $N \times N$ matrix, where $d_{ij}^{(k)}$ is the length of the shortest path from v_i to v_j that goes through at most $(k-1)$ intermediate vertices. It is clear that $D^{(1)} = W$, and $D^{(k)}$ can be obtained from $D^{(k/2)}$ by

$$d_{ij}^{(k)} = \min_l (d_{il}^{(k/2)} + d_{lj}^{(k/2)}),$$

where $k > 1$, and $1 \leq i, j, l \leq N$. Such a computation can be treated as a matrix multiplication problem $D^{(k)} = D^{(k/2)} D^{(k/2)}$, where $\otimes$ is the numerical addition operation $+$, and $\oplus$ is the "min" operation. It is also clear that $D = D^{(N-1)}$, so that we can apply Theorem 5.

Theorem 6. *The all-pairs shortest paths problem for a weighted directed graph with N vertices can be solved in $O(\log N)$ time by using N^3 processors, if the weights are of bounded magnitude and precision. For weights of unbounded magnitude and precision, the problem can be solved*

- *in $O(\log N)$ time by using $N^{3+\delta}$ processors;*
- *in $O(\log N \log\log N)$ time by using $N^3 / \log\log N$ processors;*
- *in $O(\log N)$ time with high probability by using N^3 processors.* ∎

6.1 Related Problems

It is well known that [1,3] there are many other interesting graph theory problems very closely related to the all-pairs shortest paths problem in the sense that the solution to the all-pairs shortest paths problem can be used to easily assemble the solutions to these problems. The following corollary gives a list of such problems, and the reader is referred to [3] for more details of these problems. All these problems can be solved on LARPBS with the time and processor complexities specified in Theorem 6.

Corollary 7. *All the following problems on a weighted directed graph with N vertices can be solved with the same time and processor complexities in Theorem 6: radius, diameter, and centers, bridges, median and median length, shortest path spanning tree, breadth-first spanning tree, minimum depth spanning tree, least median spanning tree, max gain, topological sort and critical paths.* ∎

7 Minimum Weight Spanning Tree

Let $G = (V, E, W)$ be a weighted undirected graph, where $V = \{v_1, v_2, ..., v_N\}$ is a set of N vertices, E is a set of edges, and $W = (w_{ij})$ is an $N \times N$ weight matrix, i.e., $w_{ij} = w_{ji}$ is the cost of the edge $\{v_i, v_j\} \in E$, and $w_{ij} = \infty$ if $\{v_i, v_j\} \notin E$. It is assumed that the edge costs are distinct, with ties broken using the lexicographical order. The minimum weight spanning tree problem is to find the unique spanning tree of G such that the sum of costs of the edges in the tree is minimized.

It was shown in [13] that the minimum weight spanning tree problem can be solved in the same way as that of the all-pairs shortest paths problem. Let the cost of a path be the highest cost of the edges on the path. Define $C^{(k)} = (c_{ij}^{(k)})$ to be an $N \times N$ matrix, where $c_{ij}^{(k)}$ is the shortest path from v_i to v_j that passes through at most $(k-1)$ intermediate vertices. Then, we have $c_{ij}^{(1)} = w_{ij}$, and

$$c_{ij}^{(k)} = \min_l(\max(c_{il}^{(k/2)}, c_{lj}^{(k/2)})),$$

for all $k > 1$, and $1 \leq i, j, l \leq N$. Such a computation can be treated as a matrix multiplication problem $C^{(k)} = C^{(k/2)} C^{(k/2)}$, where $\otimes$ is the "max" operation, and $\oplus$ is the "min" operation. Once $C^{(N-1)}$ is obtained, it is easy to determine the tree edges, namely, $\{v_i, v_j\}$ is in the minimum weight spanning tree if and only if $c_{ij}^{(N-1)} = w_{ij}$.

Theorem 8. *The minimum weight spanning tree problem for a weighted undirected graph with N vertices can be solved in $O(\log N)$ time by using N^3 processors, if the weights are of bounded magnitude and precision. For weights of unbounded magnitude and precision, the problem can be solved*

- *in $O(\log N)$ time by using $N^{3+\delta}$ processors;*
- *in $O(\log N \log\log N)$ time by using $N^3/\log\log N$ processors;*
- *in $O(\log N)$ time with high probability by using N^3 processors.* ∎

The spanning tree problem for undirected graphs is a special case of the minimum weight spanning tree problem in the sense that $w_{ij} = w_{ji} = 1$ for an edge $\{v_i, v_j\} \in E$. Since all the weights are of bounded magnitude, we have

Theorem 9. *The spanning tree problem for an undirected graph with N vertices can be solved in $O(\log N)$ time by using N^3 processors.* ∎

8 Transitive Closure

Let $G = (V, E)$ be a directed graph, and A_G be the adjacency matrix of G, which is an $N \times N$ boolean matrix. Let A_G^* be the adjacency matrix of G's transitive closure. By applying Lemma 2, the following result was shown in [7].

Theorem 10. *The transitive closure of a directed graph with N vertices can be found in $O(\log N)$ time by using $O(N^3/\log N)$ processors.* ∎

9 Strong Components

A strong component of a directed graph $G = (V, E)$ is a subgraph $G' = (V', E')$ of G such that there is path from every vertex in V' to every other vertex in V' along arcs in E', and G' is maximal, i.e., G' is not a subgraph of another strong component of G.

To find the strong components of G, we first calculate $A_G^* = (a_{ij}^*)$, where $a_{ij}^* = 1$ if there is a path from v_i to v_j, and $a_{ij}^* = 0$ otherwise. Then, v_i and v_j are in the same component if and only if $a_{ij}^* = a_{ji}^* = 1$, for all $1 \leq i, j \leq N$. Based on A_G^*, we construct $C = (c_{ij})$, where $c_{ij} = 1$ if v_i and v_j are in the same component, and $c_{ij} = 0$ otherwise. If the strong components are represented in such a way that every vertex v_i remembers the vertex $v_{s(i)}$ with the smallest index $s(i)$ in the same component, then $s(i)$ is the minimum j such that $c_{ij} = 1$, where $1 \leq j \leq N$. The construction of C and the finding of the $s(i)$'s can be performed on an N^2-processor LARPBS in $O(1)$ time.

Theorem 11. *The strong components of a directed graph with N vertices can be found in $O(\log N)$ time by using $O(N^3/\log N)$ processors.* ∎

The connected component problem for undirected graphs is just a special case of the strong component problem for directed graphs.

Theorem 12. *The connected components of an undirected graph with N vertices can be found in $O(\log N)$ time by using $O(N^3/\log N)$ processors.* ∎

10 Summary and Conclusions

We have considered fast parallel algorithms on the model of linear array with a reconfigurable pipelined bus system for the following important graph theory problems:

- All-pairs shortest paths;
- Radius, diameter, and centers;
- Bridges;
- Median and median length;
- Shortest path spanning tree;
- Breadth-first spanning tree;
- Minimum depth spanning tree;
- Least median spanning tree;
- Max gain;
- Topological sort and critical paths;
- Minimum weight spanning tree;
- Spanning tree;
- Transitive closure;
- Strong components;
- Connected components.

Our algorithms are based on fast matrix multiplication and extreme value finding algorithms, and are currently the fastest algorithms.

Appendix. Implementation Details

We now present the implementation details of the generic matrix multiplication algorithm of Lemma 1. The algorithm calculates the product $C = AB = (c_{ik})$ of two $N \times N$ matrices $A = (a_{ij})$ and $B = (b_{jk})$ in $O(T)$ time by using N^2M processors, assuming that the aggregation $x_1 \oplus x_2 \oplus \cdots \oplus x_N$ can be calculated in $O(T)$ time by using M processors.

For convenience, we label the N^2M processors using triplets (i, j, k), where $1 \leq i, j \leq N$, and $1 \leq k \leq M$. Processors $P(i, j, k)$ are ordered in the linear array using the lexicographical order. Let $P(i, j, *)$ denote a group of consecutive processors $P(i, j, 1)$, $P(i, j, 2)$, ..., $P(i, j, M)$. It is noticed that the original system can be reconfigured into N^2 subsystems, namely, the $P(i, j, *)$'s. Each processor $P(i, j, k)$ has three registers $\mathtt{A}(i, j, k)$, $\mathtt{B}(i, j, k)$, and $\mathtt{C}(i, j, k)$.

We start with the case where $M \geq N$. The input and output data layout are specified as follows. Initially, elements a_{ij} and b_{ji} are stored in registers $\mathtt{A}(1, i, j)$ and $\mathtt{B}(1, i, j)$ respectively, for all $1 \leq i, j \leq N$. If we partition A into N row vectors A_1, A_2, ..., A_N, and B into N column vectors B_1, B_2, ..., B_N,

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{bmatrix}, \qquad B = [B_1, B_2, ..., B_N],$$

then the initial data distribution is as follows:

$$\begin{array}{cccc} P(1,1,*) & P(1,2,*) & \cdots & P(1,N,*) \\ (A_1, B_1) & (A_2, B_2) & \cdots & (A_N, B_N) \end{array}$$

where other processors are not shown here for simplicity, since they do not carry any data initially. When the computation is done, c_{ij} is found in $\mathtt{C}(1, i, j)$. If we divide C into N row vectors C_1, C_2, ..., C_N,

$$C = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix},$$

then the output layout is

$$\begin{array}{cccc} P(1,1,*) & P(1,2,*) & \cdots & P(1,N,*) \\ (C_1) & (C_2) & \cdots & (C_N) \end{array}$$

Such an arrangement makes it easy for C to be used as an input to another computation, e.g., repeated squaring.

The algorithm proceeds as follows. In Step (1), we change the placement of matrix A in such a way that element a_{ij} is stored in $\mathtt{A}(i, 1, j)$, for all $1 \leq$

A Generic Matrix Multiplication Algorithm on LARPBS

```
for 1 ≤ i, j ≤ N do in parallel // Step (1). One-to-one communication.
    A(i, 1, j) ← A(1, i, j)
endfor

for 1 ≤ i, k ≤ N do in parallel // Step (2). Multiple multicasting.
    A(i, 2, k), A(i, 3, k), ..., A(i, N, k) ← A(i, 1, k)
endfor
for 1 ≤ j, k ≤ N do in parallel
    B(2, j, k), B(3, j, k), ..., B(N, j, k) ← B(1, j, k)
endfor

for 1 ≤ i, j, k ≤ N do in parallel // Step (3). Local computation.
    C(i, j, k) ← A(i, j, k) ⊗ B(i, j, k)
endfor

for 1 ≤ i, j ≤ N do in parallel // Step (4). Aggregation.
    C(i, j, 1) ← C(i, j, 1) ⊕ C(i, j, 2) ⊕ ··· ⊕ C(i, j, N)
endfor

for 1 ≤ i ≤ N, 2 ≤ j ≤ N, do in parallel // Step (5). One-to-one communi.
    C(1, i, j) ← C(i, j, 1)
endfor
```

$i, j \leq N$. This can be accomplished by a one-to-one communication. After such replacement, we have the following data distribution,

$$\begin{array}{cccc}
P(1,1,*) & P(1,2,*) & \cdots & P(1,N,*) \\
(A_1, B_1) & (A_2, B_2) & \cdots & (A_N, B_N) \\
\\
P(2,1,*) & P(2,2,*) & \cdots & P(2,N,*) \\
(A_2, -) & (-,-) & \cdots & (-,-) \\
\\
\vdots & \vdots & \ddots & \vdots \\
\\
P(N,1,*) & P(N,2,*) & \cdots & P(N,N,*) \\
(A_N, -) & (-,-) & \cdots & (-,-)
\end{array}$$

where the linear array of N^2M processors are logically arranged as a two dimensional $N \times N$ array, and each element in the array stands for a group of M processors $P(i, j, *)$. The symbol "$-$" means that the A and B registers are still undefined.

In Step (2), we distribute the rows of A and columns of B to the right processors, such that processors $P(i, j, *)$ hold A_i and B_j, for all $1 \leq i, j \leq N$. This can be performed using multiple multicasting operations. After multicasting, the

data distribution is as follows:

$$\begin{array}{cccc}
P(1,1,*) & P(1,2,*) & \cdots & P(1,N,*) \\
(A_1,B_1) & (A_1,B_2) & \cdots & (A_1,B_N) \\
\\
P(2,1,*) & P(2,2,*) & \cdots & P(2,N,*) \\
(A_2,B_1) & (A_2,B_2) & \cdots & (A_2,B_N) \\
\\
\vdots & \vdots & \ddots & \vdots \\
\\
P(N,1,*) & P(N,2,*) & \cdots & P(N,N,*) \\
(A_N,B_1) & (A_N,B_2) & \cdots & (A_N,B_N)
\end{array}$$

At this point, processors are ready to compute. In Step (3), $P(i,j,k)$ calculates $a_{ik} \otimes b_{kj}$.

Then, in Step (4), the values $\mathtt{C}(i,j,k)$, $1 \leq k \leq N$, are aggregated by the M processors in $P(i,j,*)$, for all $1 \leq i,j \leq N$, and the result c_{ij} is in $\mathtt{C}(i,j,1)$. Here, for the purpose of multiple aggregations, the original system is reconfigured into N^2 subsystems $P(i,j,*)$'s, for $1 \leq i,j \leq N$. After calculation, the data distribution is as follows:

$$\begin{array}{cccc}
P(1,1,1) & P(1,2,1) & \cdots & P(1,N,1) \\
(c_{11}) & (c_{12}) & \cdots & (c_{1N}) \\
\\
P(2,1,1) & P(2,2,1) & \cdots & P(2,N,1) \\
(c_{21}) & (c_{22}) & \cdots & (c_{2N}) \\
\\
\vdots & \vdots & \ddots & \vdots \\
\\
P(N,1,1) & P(N,2,1) & \cdots & P(N,N,1) \\
(c_{N1}) & (c_{N2}) & \cdots & (c_{NN})
\end{array}$$

Finally, in Step (5), one more data movement via one-to-one communication brings the c_{ij}'s to the right processors.

It is clear that Steps (1), (2), (3), and (5) are simple local computations or primitive communication operations, and hence, take $O(1)$ time. Step (4) requires $O(T)$ time. Thus, the entire algorithm can be executed in $O(T)$ time.

In general, when $M \geq 1$ and $M \leq N$, to store a vector in a group $P(i,j,*)$ of consecutive processors $P(i,j,1)$, $P(i,j,2)$, ..., $P(i,j,M)$, we can divide a vector of length N into sub-vectors of length $\lceil N/M \rceil$ such that each processor $P(i,j,k)$ is assigned a sub-vector. This increases the time of Steps (1), (2), (3), and (5) by a factor of $O(N/M)$. Since $T = \Omega(N/M)$, the complete algorithm still takes $O(T)$ time.

Acknowledgments

Keqin Li was supported by National Aeronautics and Space Administration and the Research Foundation of State University of New York through NASA/University Joint Venture in Space Science Program under Grant NAG8-1313 (1996-1999). Yi Pan was supported by the National Science Foundation under Grants CCR-9211621 and CCR-9503882, the Air Force Avionics Laboratory, Wright Laboratory, Dayton, Ohio, under Grant F33615-C-2218, and an Ohio Board of Regents Investment Fund Competition Grant. Mounir Hamdi was partially supported by the Hong Kong Research Grant Council under the Grant HKUST6026/97E.

References

1. S.G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, New Jersey, 1997.
2. D. Chiarulli, R. Melhem, and S. Levitan, "Using coincident optical pulses for parallel memory addressing," *IEEE Computer*, vol. 30, pp. 48-57, 1987.
3. E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM Journal on Computing*, vol.10, pp.657-673, 1981.
4. M. Hamdi, C. Qiao, Y. Pan, and J. Tong, "Communication-efficient sorting algorithms on reconfigurable array of processors with slotted optical buses," to appear in *Journal of Parallel and Distributed Computing*.
5. J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.
6. S. Levitan, D. Chiarulli, and R. Melhem, "Coincident pulse techniques for multiprocessor interconnection structures," *Applied Optics*, vol. 29, pp. 2024-2039, 1990.
7. K. Li, "Constant time boolean matrix multiplication on a linear array with a reconfigurable pipelined bus system," *Journal of Supercomputing*, vol.11, no.4, pp.391-403, 1997.
8. K. Li and V.Y. Pan, "Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system," *Proceedings of IPPS/SPDP '99*, San Juan, Puerto Rico, April 12-16, 1999.
9. K. Li, Y. Pan, and S.-Q. Zheng, eds., *Parallel Computing Using Optical Interconnections*, Kluwer Academic Publishers, Boston, Massachusetts, 1998.
10. K. Li, Y. Pan, and S.-Q. Zheng, "Fast and processor efficient parallel matrix multiplication algorithms on a linear array with a reconfigurable pipelined bus system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 8, pp. 705-720, August 1998.
11. K. Li, Y. Pan, and S.-Q. Zheng, "Fast and efficient parallel matrix computations on a linear array with a reconfigurable pipelined optical bus system," in *High Performance Computing Systems and Applications*, J. Schaeffer ed., pp. 363-380, Kluwer Academic Publishers, Boston, Massachusetts, 1998.
12. K. Li, Y. Pan, and S.-Q. Zheng, "Efficient deterministic and probabilistic simulations of PRAMs on a linear array with a reconfigurable pipelined bus system," to appear in *Journal of Supercomputing*.
13. B.M. Maggs and S.A. Plotkin, "Minimum-cost spanning tree as a path-finding problem," *Information Processing Letters*, vol.26, pp.291-293, 1988.
14. Y. Pan and M. Hamdi, "Efficient computation of singular value decomposition on arrays with pipelined optical buses," *Journal of Network and Computer Applications*, vol.19, pp.235-248, July 1996.

15. Y. Pan, M. Hamdi, and K. Li, "Efficient and scalable quicksort on a linear array with a reconfigurable pipelined bus system," *Future Generation Computer Systems*, vol. 13, no. 6, pp. 501-513, June 1998.
16. Y. Pan and K. Li, "Linear array with a reconfigurable pipelined bus system – concepts and applications," *Journal of Information Sciences*, vol. 106, no. 3-4, pp. 237-258, May 1998.
17. Y. Pan, K. Li, and S.-Q. Zheng, "Fast nearest neighbor algorithms on a linear array with a reconfigurable pipelined bus system," *Journal of Parallel Algorithms and Applications*, vol. 13, pp. 1-25, 1998.
18. H. Park, H.J. Kim, and V.K. Prasanna, "An $O(1)$ time optimal algorithm for multiplying matrices on reconfigurable mesh," *Information Processing Letters*, vol.47, pp.109-113, 1993.
19. S. Rajasekaran and S. Sen, "Random sampling techniques and parallel algorithm design," in *Synthesis of Parallel Algorithms*, J.H. Reif, ed., pp.411-451, Morgan Kaufmann, 1993.
20. B.-F. Wang and G.-H. Chen, "Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems," *IEEE Transactions on Parallel and Distributed Systems*, vol.1, pp.500-507, 1990.