Scalable Parallel Matrix Multiplication on Distributed Memory Parallel Computers

Keqin Li
Department of Mathematics and Computer Science
State University of New York
New Paltz, New York 12561-2499
li@mcs.newpaltz.edu

Abstract

Consider any known sequential algorithm for matrix multiplication over an arbitrary ring with time complexity $O(N^\alpha)$, where $2 < \alpha \le 3$. We show that such an algorithm can be parallelized on a distributed memory parallel computer (DMPC) in $O(\log N)$ time by using $N^\alpha/\log N$ processors. Such a parallel computation is cost optimal and matches the performance of PRAM. Furthermore, our parallelization on a DMPC can be made fully scalable, that is, for all $1 \le p \le N^\alpha/\log N$, multiplying two $N \times N$ matrices can be performed by a DMPC with p processors in $O(N^\alpha/p)$ time, i.e., linear speedup and cost optimality can be achieved in the range $[1..N^\alpha/\log N]$. This unifies all known algorithms for matrix multiplication on DMPC, standard or non-standard, sequential or parallel. Extensions of our methods and results to other parallel systems are also presented. The above claims result in significant progress in scalable parallel matrix multiplication (as well as solving many other important problems) on distributed memory systems, both theoretically and practically.

1 Introduction

Matrix multiplication is a building block in many matrix operations and for solving graph theory problems. Due to its fundamental importance in sciences and engineering, much effort has been devoted to finding and implementing fast matrix multiplication algorithms. The standard sequential algorithm has $O(N^3)$ time complexity in multiplying two $N \times N$ matrices. Since Strassen's $O(N^{2.8074})$ algorithm was discovered [39], successive progress has been made to develop fast sequential matrix multiplication algorithms with time complexity $O(N^\alpha)$, where $2 < \alpha < 3$. The current best value of α is less than 2.3755 [7].

As for parallel implementation of matrix multiplication algorithms, the standard algorithm can be easily parallelized, and Strassen's algorithm has been parallelized on shared memory multiprocessors [5]. Furthermore, any known sequential $O(N^\alpha)$ matrix multiplication algorithm can be parallelized on a CREW PRAM in $O(\log N)$ time by using $O(N^\alpha/\log N)$ processors [4, 33, 35]. This significant result has been used extensively in many other parallel algorithms. Unfortunately, the PRAM model as well as large scale shared memory systems are practically limited due to their difficulty in realization.

On distributed memory multicomputers (which are considered more practical), research has essentially focused on the parallelization of the standard algorithm. It was shown that matrix multiplication can be done in $O(N^3/p + \log(p/N^2))$ time on a hypercube with p processors, where $N^2 \le p \le N^3$ [8]. However, it is not clear how non-standard algorithms can be implemented efficiently on commonly used electronic static networks such as hypercubes and meshes, which have limited network connectivity and communication capability. In fact, none of the $O(N^\alpha)$ sequential algorithms with $\alpha < 3$ has been fully parallelized on any distributed memory systems with static interconnection networks, since these networks do not have sufficient capability to support complicated communication efficiently. In [8], it was shown that matrix multiplication can be done in $O(N^\alpha/p^{(\alpha-1)/2})$ time on a hypercube with p processors, where $1 \le p \le N^2$, and α is the exponent of the best available $O(N^\alpha)$ sequential algorithm. This implementation is valid only in a small interval of p, and is not cost-optimal (as compared to the $O(N^\alpha)$ sequential algorithm). When $p = N^2$, we get the shortest execution time $O(N)$, which is very slow. The reason is that the $O(N^\alpha)$ algorithm is invoked sequentially to calculate submatrix products, and not parallelized at all.

To fully parallelize non-standard sequential matrix multiplication algorithms on distributed memory systems, more powerful support for communication patterns used in efficient parallelization of these algorithms is required. One promising approach is to use fiber optical buses as interconnections [21]. For instance, on Linear Arrays with Reconfigurable Pipelined Bus Systems (LARPBS, a kind of distributed memory systems employing optical interconnections), one recent breakthrough was reported in [22].

(R1) Matrix multiplication (where we assume that matrix elements are integers of bounded magnitude, or floating-point values of bounded precision and magnitude) can be performed by an LARPBS in $O((\log N)^\delta)$ time by using $O(N^3/1.1428^{(\log N)^\delta})$ processors, where $0 \le \delta \le 1$ [22]. This implies that matrix multiplication can be done in $O(1)$ time using N^3 processors, and in $O(\log N)$ time using $O(N^{2.8074})$ processors.

This was then the fastest parallel implementation of non-standard matrix multiplication algorithms on distributed memory systems.

In reality, it is not feasible to use, say, N^α processors where $\alpha > 2$, for large scale matrix multiplications. Typically, the number of processors available is far less than what is required. Therefore, it is necessary to design parallel algorithms that can be executed on p processors, where p is arbitrarily chosen in certain range. An important issue in parallel computing is the scalability of a parallel algorithm on a parallel system. Though there is no unified definition of scalability, scalability essentially measures the ability to maintain speedup that is linearly proportional to the number of processors.

To be more specific, when the number of processors is less than the maximum required, the parallel time complexity can generally be represented as $O(T(N)/p + T_{\rm comm}(N,p))$, where N is the problem size, p is the number of processors available, $T(N)$ is the time complexity of the sequential algorithm being parallelized, and $T_{\rm comm}(N,p))$ is the overall communication overhead of a parallel implementation. We say that a parallel implementation is *scalable* in the range $[1..p^*]$ if linear speedup can be achieved for all $1 \le p \le p^*$. The implementation is *highly scalable* if p^* is as large as $\Theta(T(N)/(T^*(N)(\log N)^k))$ for some constant $k \ge 0$, where $T^*(N)$ is the best possible parallel time. The implementation is *fully scalable* if $k = 0$. A fully scalable parallel implementation means that the sequential algorithm can be fully parallelized, and communication overhead in parallelization is negligible.

The initial results of scalable parallelization of sequential matrix multiplication algorithms were reported in [25].

(R2) There is a fully scalable parallelization of the standard algorithm (where we assume that matrix elements are integers of bounded magnitude, or floating-point values of bounded precision and magnitude, or boolean values), i.e., for all $1 \le p \le N^3$, multiplying two $N \times N$ matrices can be performed on a p-processor LARPBS in $O(N^3/p)$ time.

(R3) There is a highly scalable parallelization of Strassen's algorithm (applicable to an arbitrary ring) with $k = 2.4771$. In particular, for all $1 \le p \le N^{2.8074}$, multiplying two $N \times N$ matrices can be performed on a p-processor LARPBS in $O(N^{2.8074}/p + (N^2/p^{0.7124})\log p)$ time, which implies that when $p = O(N^{2.8074}/(\log N)^{3.4771})$, linear speedup and cost-optimality can be achieved. (Notice that $T^*(N) = O(\log N)$ is the best possible parallel time for Strassen's algorithm.)

Continuing the investigation along this line, we recently made the following progress, namely, obtaining highly scalable implementation of any known sequential matrix multiplication algorithm [19].

(R4) For any $O(N^\alpha)$ sequential matrix multiplication algorithm over an arbitrary ring with $2 < \alpha \le 3$, there is a highly scalable parallel implementation on LARPBS. That is, for all $1 \le p \le N^\alpha$, multiplying two $N \times N$ matrices can be performed on a p-processor LARPBS in $O(N^\alpha/p + (N^2/p^{2/\alpha})\log p)$ time. If the number of processors is $p = O(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$, our implementation achieves linear speedup and is cost-optimal. In particular, multiplying two $N \times N$ matrices can be performed by an LARPBS with $O(N^\alpha)$ processors in $O(\log N)$ time.

It is clear that (R3) is a special case of (R4), where the time complexity is substantially reduced as compared with that in [25]. Also, the processor complexity is substantially reduced as compared with that in [22]. This is currently the fastest and most processor efficient parallelization of the best known sequential matrix multiplication algorithm on a distributed memory system.

In this paper, we further strengthen result (R4). In fact, we consider a quite general model, called Distributed Memory Parallel Computers (DMPC), which includes a number of important models of distributed memory systems such as LARPBS as special cases.

(R5) For any $O(N^\alpha)$ sequential matrix multiplication algorithm over an arbitrary ring with $2 < \alpha \le 3$, there is a fully scalable parallel implementation on DMPC. That is, for all $1 \le p \le N^\alpha/\log N$, multiplying two $N \times N$ matrices can be performed by a DMPC with p processors in $O(N^\alpha/p)$ time, and linear speedup can be achieved in the range $[1..N^\alpha/\log N]$. In particular, multiplying two $N \times N$ matrices can be performed in $O(\log N)$ time by a DMPC with $N^\alpha/\log N$ processors. This matches the performance of PRAM.

The significance of result (R5) is three-fold. First, it strengthens result (R4) by making our parallel implementation from high scalability to full scalability. Second, result (R5) is applicable to a wider range of systems than LARPBS, since the DMPC model is weaker than LARPBS. Third, result (R5) unifies all known algorithms for matrix multiplication on DMPC, standard or nonstandard, sequential or parallel. Such a unification has rarely been seen for other problems.

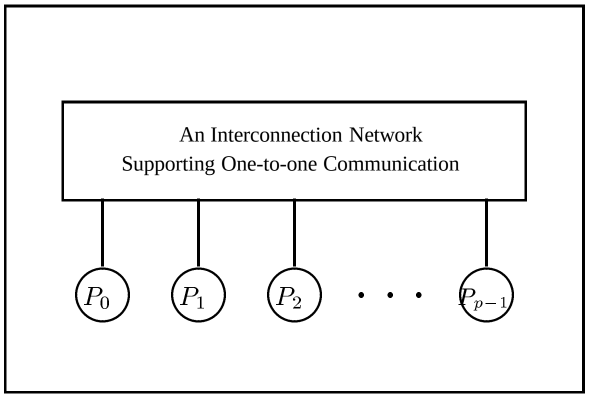
An Interconnection Network
Supporting One-to-one Communication
P_0
P_1
P_2
$\cdots$
P_{p-1}

Figure 1. A distributed memory parallel computer (DMPC).

Moreover, our methods and results can be extended to other parallel systems (see §6 for more details). All the above claims result in significant progress in scalable parallel matrix multiplication (as well as solving many other important problems) on distributed memory systems, both theoretically and practically.

2 Distributed Memory Parallel Computers

A *distributed memory parallel computer* (DMPC) consists of p processors $P_0, P_1, P_2, ..., P_{p-1}$ (see Figure 1). Each processor has its own local memory, and there is no global shared memory. Processors communicate with each other via message passing. The computations and communications in a DMPC are globally synchronized into steps. A step is either a computation step or a communication step. In a computation step, each processor performs a local arithmetic/logic operation, or, is idle. A computation step takes constant amount of time.

In a communication step, processors send and receive messages via an interconnection network. A communication step can be specified as

$$((\pi(0), v_0), (\pi(1), v_1), (\pi(2), v_2), ..., (\pi(p-1), v_{p-1})),$$

where for all $0 \le j \le p-1$, processor P_j sends a value v_j to processor $P_{\pi(j)}$, and π is a mapping $\pi : \{0, 1, 2, ..., p-1\} \rightarrow \{-1, 0, 1, 2, ..., p-1\}$. If processor P_j does not send anything during a communication step, then $\pi(j) = -1$ and v_j is undefined. It is required that for all $0 \le i \le p-1$, there is at most one j such that $\pi(j) = i$, i.e., each processor can receive at most one message in a communication step. Such a communication step is essentially *one-to-one communication*. It is assumed that the interconnection network connecting the p processors in a DMPC can realize any one-to-one communication in constant amount of time. In a busiest communication step, every processor sends a message to another processor, and $(\pi(0), \pi(1), \pi(2), ..., \pi(p-1))$ is a permutation of $(0, 1, 2, ..., p-1)$.

Based on the above discussion, the time complexity of a parallel computation on a DMPC is the sum of the numbers of computation and communication steps.

The DMPC model is functionally identical to the Module Parallel Computer (MPC) model which was used to study the capability of distributed memory systems in simulating Parallel Random Access Machines (PRAM) [27]. Using electronic interconnections, the interconnection network in a DMPC can be implemented using a static completely connected network (as in MPC), or a dynamic crossbar network, both can support a one-to-one communication step and arbitrary permutation in constant time. However, such implementations using electronic interconnections are very expensive and unrealistic when p is large.

Due to recent advances in optical interconnection technologies, the interconnection network in a DMPC can be implemented using optical interconnection networks. For instance, the Optical Model of Computation (OMC) was proposed in [10] (cf. [2, 11, 12] for more study on this model). In such a system, there is a processor layer and a deflection layer, both embedded in Euclidean planes. Interprocessor communications are performed using free space optical beams. An optical beam can carry and transmit information in constant time, independent of the distance covered. There are various techniques to implement the deflection layer (see [10] for detailed discussions). The optical beams function properly as long as each processor receives at most one message in a communication step, thus supporting arbitrary one-to-one communication.

Recently, fiber optical buses have emerged as promising networks [3, 6, 9, 13, 16, 26, 30, 37, 40]. Pipelined optical buses can support massive volume of data transfer simultaneously, and can implement various communication patterns. Furthermore, a system with optical buses can be reconfigured into independent subsystems, which can be used simultaneously to solve subproblems in parallel [30]. It is now feasible to build distributed memory systems that are no less powerful and flexible than shared memory systems in solving many problems, such as Boolean matrix multiplication [17] and sorting [24]. Numerous parallel algorithms using optical interconnection networks have been developed recently [1, 14, 17, 18, 20, 22, 23, 24, 28, 29, 31, 36, 38].

Under such circumstances, a new model called Linear Array with Reconfigurable Pipelined Bus Systems (LARPBS) was proposed in [29, 30]. An LARPBS consists of p processors connected by a pipelined optical bus system, which 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 signal 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 [6, 16, 37]. Such pipelined optical bus systems can support a massive volume of communications simultaneously, and are particularly appropriate for applications that involve intensive communication operations such as broadcasting, one-to-one communication, multicasting, global aggregation, and irregular communication patterns. 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 $-1 = i_0 < i_1 < i_2 \cdots < i_k = p-1$. 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 [30] for an elaborated exposition). As in many other parallel computing systems, a computation on LARPBS is a sequence of alternate global communication and local computation steps. The time complexity of an algorithm is measured in terms of the total number of bus cycles in all the communication steps, as long as the time of the local computation steps between successive communication steps is bounded by a constant and independent of the problem size. In addition to one-to-one communication, an LARPBS can also support broadcast, multicast, multiple multicast, and even global aggregation in constant time. Hence, LARPBS is a more powerful model than DMPC.

3 Bilinear Algorithms

All existing sequential matrix multiplication algorithms over an arbitrary ring fall into the category of *bilinear algorithms* (see [4, pages 315-316] and [33, 34, 35]). Let $A = (A_{ij})$, $B = (B_{jk})$, and $C = (C_{ik})$ be $N \times N$ matrices, where $N = m^n$. Assume that m is a fixed constant, and $n \rightarrow \infty$. Each of these matrices are partitioned into m^2 submatrices A_{ij}, B_{jk}, C_{ik} of size $m^{n-1} \times m^{n-1}$. A bilinear algorithm for computing $C = A \times B$ can be recursively described in Figure 2. In the above computation, all the $f(i,j,u)$'s, $f^*(j,k,u)$'s, and $f^{**}(k,i,u)$'s are constants. The value R is called the rank of the algorithm. For all the known algorithms for $N \times N$ matrix multiplication running in $O(N^\alpha)$ time for $\alpha \leq 3$ (including the standard algorithm for $\alpha = 3$, Strassen's algorithm for $\alpha = \log_2 7 < 2.8074$ [39], Pan's algorithm for $\alpha = \log_{70} 143640 < 2.7952$ [32], and the ones of [7] for $\alpha < 2.3755$, which are currently asymptotically fastest), we may yield $R = m^\alpha$ in the associated bilinear construction. Note that for fixed m and α, R is finite.

Step (B) (Basis) If $n = 0$, i.e., the matrices are of size 1×1, compute the product $C = A \times B$ directly, and return; otherwise, do the following steps.

Step (D) (Division) Calculate the $2R$ linear combinations of submatrices

$$L_u = \sum_{1 \leq i,j \leq m} f(i,j,u) A_{ij},$$

and

$$L_u^* = \sum_{1 \leq j,k \leq m} f^*(j,k,u) B_{jk},$$

for all $1 \leq u \leq R$, where L_u and L_u^* are $m^{n-1} \times m^{n-1}$ matrices.

Step (R) (Recursion) Calculate the R matrix products $M_u = L_u \times L_u^*$, for all $1 \leq u \leq R$.

Step (C) (Combination) Compute

$$C_{ik} = \sum_{j=1}^{m} A_{ij} \times B_{jk} = \sum_{u=1}^{R} f^{**}(k,i,u) M_u,$$

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

Figure 2. The recursive bilinear algorithm.

The above recursive algorithm has $n = \lceil \log_m N \rceil$ levels. (A few extra dummy rows and columns are introduced if N is not a power of m.) The recursion reaches its base case (see Step (B)) when the size of the submatrices is 1×1. In general, a bilinear algorithm has three steps. If sufficiently many processors are available, we can calculate the L_u's in parallel, and then all the L_u^*'s in parallel in the division step (see Step (D)). After the recursion step (see Step (R)), all the C_{ik}'s are also computed in parallel in the combination step (see Step (C)). Hence, each level takes constant time, and the overall time complexity is $O(\log N)$, which is the best possible. The execution time cannot be further reduced, due to the recursive nature of the class of bilinear algorithms, not to the communication constraints on a parallel system.

For all so far available sequential algorithms for matrix multiplication, there is a parallelization under the CREW PRAM model, which requires $O(N^\alpha / \log N)$ processors (see p.317 in [4]).

4 Implementation on DMPC

The recursive bilinear algorithm can be unrolled into an iterative algorithm. Let us label Steps (D), (R), and (C) in the lth level of the recursion as Step (D$_l$), (R$_l$), and (C$_l$) respectively, where $1 \leq l \leq n$. The computation proceeds in the following way:

$$\text{Step } (\text{D}_1) \rightarrow \text{Step } (\text{D}_2) \rightarrow \cdots \rightarrow \text{Step } (\text{D}_n)$$
$$\rightarrow \text{Step (B)} \rightarrow \text{Step } (\text{C}_n) \rightarrow \cdots \rightarrow \text{Step } (\text{C}_2) \rightarrow \text{Step } (\text{C}_1).$$

The recursive step (R$_l$), $1 \leq l \leq n$, is actually

$$\text{Step } (\text{D}_{l+1}) \rightarrow \text{Step } (\text{D}_{l+2}) \rightarrow \cdots \rightarrow \text{Step } (\text{D}_n) \rightarrow$$

$$\text{Step (B)} \to \text{Step (C}_n) \to \cdots \to \text{Step (C}_{l+2}) \to \text{Step (C}_{l+1}).$$

Given matrices $L(u_1, u_2, ..., u_l)$ and $L^*(u_1, u_2, ..., u_l)$ of size $N/m^l \times N/m^l$, Step (R$_l$) obtains their product

$$M(u_1, u_2, ..., u_l) = L(u_1, u_2, ..., u_l) \times L^*(u_1, u_2, ..., u_l),$$

for all $1 \le u_1, u_2, ..., u_l \le R$. Therefore, we can imagine that the entire bilinear algorithm is actually Step (R$_0$) which, given two matrices $L() = A$ and $L^*() = B$, computes their product $C = M() = L() \times L^*() = A \times B$.

To compute the product $M(u_1, u_2, ..., u_l)$ in Step (R$_l$), where $0 \le l \le n-1$, we first execute Step (D$_{l+1}$), i.e., partitioning $L(u_1, u_2, ..., u_l)$ and $L^*(u_1, u_2, ..., u_l)$ into submatrices $L_{ij}(u_1, u_2, ..., u_l)$ and $L^*_{jk}(u_1, u_2, ..., u_l)$ of size $N/m^{l+1} \times N/m^{l+1}$, and calculating the following matrices,

$$L(u_1, u_2, ..., u_{l+1}) = \sum_{1 \le i,j \le m} f(i, j, u_{l+1}) L_{ij}(u_1, u_2, ..., u_l),$$

and

$$L^*(u_1, u_2, ..., u_{l+1}) = \sum_{1 \le j,k \le m} f^*(j, k, u_{l+1}) L^*_{jk}(u_1, u_2, ..., u_l),$$

for all $1 \le u_{l+1} \le R$. Then, all the matrix products

$$M^*(u_1, u_2, ..., u_{l+1}) = L(u_1, u_2, ..., u_{l+1}) \times L^*(u_1, u_2, ..., u_{l+1})$$

are computed by Step (R$_{l+1}$). Finally, in Step (C$_{l+1}$), we get $M(u_1, u_2, ..., u_l)$, which consists of submatrices $M_{ik}(u_1, u_2, ..., u_l)$, where

$$M_{ik}(u_1, u_2, ..., u_l) = \sum_{u_{l+1}=1}^{R} f^{**}(k, i, u_{l+1}) M(u_1, u_2, ..., u_{l+1}),$$

for all $1 \le i, k \le m$. When $l = n$, Step (R$_n$) is actually Step (B), and the $M(u_1, u_2, ..., u_n)$'s are calculated directly.

We now discuss how the above iterative bilinear algorithm can be implemented on a DMPC. Our main result of this section is the following claim. The rest of the section is devoted to the proof Theorem 1.

Theorem 1. *Multiplying two $N \times N$ matrices can be performed in $O(\log N)$ time by a DMPC with N^α processors.* ∎

We use the notation $P[s : t]$ to represent the group of t processors with consecutive indices starting from s, i.e., P_s, P_{s+1}, P_{s+2}, ..., P_{s+t-1}. All the matrices and their submatrices are of size m^l, where $0 \le l \le n$. A matrix X of size $m^l \times m^l$ is stored in a processor group $P[s : m^{2l}]$ in the shuffled-row-major order. That is, $X = (X_{ij})$ is partitioned into m^2 submatrices of size $m^{l-1} \times m^{l-1}$, and X_{ij} is stored in the processor group

$$P[s + ((i-1)m + (j-1))m^{2(l-1)} : m^{2(l-1)}]$$

in the shuffled-row-major order, where $1 \le i, j \le m$.

Based on the assumption that the interconnection network in a DMPC can support one-to-one communication in one step, it is not difficult to see the following lemmas.

Lemma 1. *A processor group $P[s : m^{2l}]$ that holds an $m^l \times m^l$ matrix X can send X to another processor group $P[s' : m^{2l}]$ in one communication step. Furthermore, let $P[s_u : m^{2l}]$, where $1 \le u \le R$, be R disjoint processor groups, and processor group $P[s_u : m^{2l}]$ holds an $m^l \times m^l$ matrix X_u, where $1 \le u \le R$. Let $P[s'_u : m^{2l}]$, where $1 \le u \le R$, be R disjoint processor groups. Then, in one communication step, $P[s_u : m^{2l}]$ can send X_u to $P[s'_u : m^{2l}]$, for all $1 \le u \le R$ in parallel.* ∎

Lemma 2. *Let $P[s_u : m^{2l}]$, where $1 \le u \le R$, be R disjoint processor groups, and processor group $P[s : m^{2l}]$ holds an $m^l \times m^l$ matrix X. The matrix X can be broadcast to all $P[s_u : m^{2l}]$, $1 \le u \le R$, in $(\lceil \log R \rceil + 1)$ communication steps.* ∎

Lemma 3. *If a processor group $P[s : m^{2l}]$ holds d $m^l \times m^l$ matrices $X_1, X_2, ..., X_d$, the linear combination $f_1 X_1 + f_2 X_2 + \cdots + f_d X_d$, where f_1, f_2, ..., f_d are constants, can be obtained in $(2d-1)$ computation steps.* ∎

Lemma 4. *Let $P[s_u : m^{2l}]$, where $1 \le u \le R$, be R disjoint processor groups. If processor group $P[s_u : m^{2l}]$ holds an $m^l \times m^l$ matrix X_u, where $1 \le u \le R$, the linear combination $f_1 X_1 + f_2 X_2 + \cdots + f_R X_R$, where f_1, f_2, ..., f_R are constants, can be obtained in $\lceil \log R \rceil$ communication steps, and $\lceil \log R \rceil + 1$ computation steps, such that the result is saved in $P[s_1 : m^{2l}]$.* ∎

Let us first examine how the $L(u_1, u_2, ..., u_l)$'s can be computed in Step (D$_l$), where $1 \le l \le n$. The $L(u_1, u_2, ..., u_l)$'s can be arranged in the lexicographical order, where $0 \le l \le n$. Each $L(u_1, u_2, ..., u_l)$ is assigned an order number $\beta(u_1, u_2, ..., u_l)$ in the range $[0..R^l - 1]$:

$$\beta(u_1, u_2, ..., u_l) = (u_1 - 1)R^{l-1} + (u_2 - 1)R^{l-2} + \cdots + (u_l - 1).$$

Matrix $L(u_1, u_2, ..., u_l)$ is then stored in

$$P\left[\beta(u_1, u_2, ..., u_l)\left(\frac{N}{m^l}\right)^2 : \left(\frac{N}{m^l}\right)^2\right]. \tag{1}$$

In Step (D$_l$), where $1 \le l \le n$, we basically calculate the $L(u_1, u_2, ..., u_l)$'s from the $L(u_1, u_2, ..., u_{l-1})$'s. Let $L(u_1, u_2, ..., u_{l-1})$, which has size $(N/m^{l-1}) \times (N/m^{l-1})$, be partitioned into m^2 submatrices of size $(N/m^l) \times (N/m^l)$, i.e., the $L_{ij}(u_1, u_2, ..., u_{l-1})$'s, where $1 \le i, j \le m$. The submatrix $L_{ij}(u_1, u_2, ..., u_{l-1})$ is stored in

$$P\left[\beta(u_1, u_2, ..., u_{l-1})\left(\frac{N}{m^{l-1}}\right)^2 + ((i-1)m + (j-1))\left(\frac{N}{m^l}\right)^2 : \left(\frac{N}{m^l}\right)^2\right], \tag{2}$$

which is made available to the processors specified in Eq. (1), for all $1 \le u_l \le R$. When the processor group specified in Eq. (1) collect all the $L_{ij}(u_1, u_2, ..., u_{l-1})$'s, $L(u_1, u_2, ..., u_l)$ can be computed using the following equation:

$$L(u_1, u_2, ..., u_l) = \sum_{1 \le i,j \le m} f(i, j, u_l) L_{ij}(u_1, u_2, ..., u_{l-1}). \tag{3}$$

Our algorithm for Step (D$_l$) is given in Figure 3.

It is clear that Lines 3–4 can be implemented in $(\lceil \log R \rceil + 1)$ communication steps (see Lemma 2). Thus, the sequential for-loop in Lines 2–5 takes $m^2(\lceil \log R \rceil + 1)$ communication steps. The computation in Line 7, which involves a linear combination of matrices defined in Eq. (3), can be done in $(2m^2 - 1)$ computation steps (see Lemma 3). Thus, the parallel for-loop in Lines 6–8 takes $(2m^2 - 1)$ computation steps. The overall parallel for-loop in Lines 1–9 requires $m^2(\lceil \log R \rceil + 1) + (2m^2 - 1) <$

```
1. for all $(u_1, u_2, ..., u_{l-1})$, $1 \leq u_1, u_2, ..., u_{l-1} \leq R$, do in parallel
2.   for $1 \leq i, j \leq m$ do
3.     The processor group specified by Eq. (2) broadcast $L_{ij}(u_1, u_2, ..., u_{l-1})$
4.     to the processor groups specified by Eq. (1), for all $1 \leq u_l \leq R$.
5.   end do
6.   for all $1 \leq u_l \leq R$ do in parallel
7.     The processor group specified by Eq. (1) calculate $L(u_1, u_2, ..., u_l)$ using Eq. (3).
8.   end do in parallel
9. end do in parallel
```

Figure 3. Details of Step (D$_l$) to compute the $L(u_1, u_2, ..., u_l)$'s.

$m^2 \log R + 4m^2 - 1$ steps. Since m and R are constants, Step (D$_l$) has constant running time to compute all the $L(u_1, u_2, ..., u_l)$'s, for all $1 \leq l \leq n$. The total running time of Steps (D$_1$) to (D$_n$) is $O(\log N)$.

The size of the matrix $L(u_1, u_2, ..., u_l)$ is $(N/m^l) \times (N/m^l)$, where $1 \leq l \leq n$. Since there are R^l $L(u_1, u_2, ..., u_l)$ matrices, the total number of elements in the $L(u_1, u_2, ..., u_l)$'s is $R^l(N/m^l)^2 = N^2(R/m^2)^l = N^2 m^{(\alpha-2)l}$, which is the number of processors used in Steps (D$_l$) and (C$_l$). As the computation proceeds from Steps (D$_1$) to (D$_n$), the number of processors required increases, which reaches its maximum, i.e., N^α, when $l = n$. Hence, the total number of processors required in Steps (D$_1$) to (D$_n$) is N^α.

The computation of the $L^*(u_1, u_2, ..., u_l)$'s in Step (D$_l$) can be done in a similar way, which requires the same amount of execution time and the same number of processors.

It is clear that Step (B) takes one local computation step.

In Step (C$_l$), where $1 \leq l \leq n$, we calculate $M(u_1, u_2, ..., u_{l-1})$ based on the $M(u_1, u_2, ..., u_l)$'s, for all $1 \leq u_1, u_2, ..., u_{l-1} \leq R$, using the following equation,

$$M_{ik}(u_1, u_2, ..., u_{l-1}) = \sum_{u_l=1}^{R} f^{**}(k, i, u_l) M(u_1, u_2, ..., u_{l-1}, u_l). \tag{4}$$

Our algorithm for Step (C$_l$) is given in Figure 4. Again, for all $0 \leq l \leq n$, the $M(u_1, u_2, ..., u_l)$'s are arranged in the lexicographical order, and $M(u_1, u_2, ..., u_l)$ is stored in the processor group specified in Eq. (1). Lines 2–4 involves a linear combination in Eq. (4), which can be obtained in $\lceil \log R \rceil$ communication steps, and $\lceil \log R \rceil + 1$ computation steps (see Lemma 4). Hence, the parallel for-loop in Lines 1–5 requires $2\lceil \log R \rceil + 1$ computation and communication steps. Lines 7–8 only perform a one-to-one communication, which can be done in one communication step (see Lemma 1), and the parallel for-loop in Lines 6–9 requires one communication step. The entire Step (C$_l$) takes $2(\lceil \log R \rceil + 1)$ computation and communication steps, i.e., constant amount of time. The total running time of Steps (C$_n$) to (C$_1$) is $O(\log N)$. The number of processors required in Step (C$_l$) is the same as Step (D$_l$), for all $1 \leq l \leq n$. It is clear that as the computation proceeds from Steps (C$_n$) to (C$_1$), the number of processors required decreases.

Summarizing all the above discussions, we obtain Theorem 1.

```
1. for all $(u_1, u_2, ..., u_{l-1})$, $1 \leq u_1, u_2, ..., u_{l-1} \leq R$, do in parallel
2.   The processor groups specified by Eq. (1), where $1 \leq u_l \leq R$, compute
3.   $M_{ik}(u_1, u_2, ..., u_{l-1})$ using Eq. (4), and save the result in
4.   $P\left[\beta(u_1, u_2, ..., u_{l-1}, 1)\left(\frac{N}{m^l}\right)^2 : \left(\frac{N}{m^l}\right)^2\right]$.   (5)
5. end do in parallel
6. for all $(u_1, u_2, ..., u_{l-1})$, $1 \leq u_1, u_2, ..., u_{l-1} \leq R$, do in parallel
7.   The processor group specified by Eq. (5) send $M_{ik}(u_1, u_2, ..., u_{l-1})$ to
8.   the processor group specified by Eq. (2)
9. end do in parallel
```

Figure 4. Details of Step (C$_l$) to compute the $M(u_1, u_2, ..., u_l)$'s.

5 Processor Reduction and Scalability

In this section, we show that the number of processors in Theorem 1 can be reduced to $N^\alpha / \log N$, while the running time is still $O(\log N)$. Thus, the performance of matrix multiplication on DMPC matches the performance of CREW PRAM.

The size of the matrix $L(u_1, u_2, ..., u_l)$ is $(N/m^l) \times (N/m^l)$, where $1 \leq l \leq n$. Since there are R^l $L(u_1, u_2, ..., u_l)$ matrices, the total number of elements in the $L(u_1, u_2, ..., u_l)$'s is $R^l(N/m^l)^2 = N^2(R/m^2)^l = N^2 m^{(\alpha-2)l}$. Let l^* be defined as the largest integer such that

$$N^2 m^{(\alpha-2)l^*} \leq \frac{N^\alpha}{\log N}.$$

Then, we have

$$l^* = \left\lfloor \frac{(\alpha-2)\log N - \log\log N}{(\alpha-2)\log m} \right\rfloor.$$

In Step (D$_l$), $l^* + 1 \leq l \leq n$, several passes are required to compute the $L(u_1, u_2, ..., u_l)$'s. The number of $L(u_1, u_2, ..., u_l)$ matrices that can be accommodated by $p = N^\alpha / \log N$ processors in one pass is $K_l = \lceil p/(N/m^l)^2 \rceil$. For all $0 \leq l \leq n$, matrices $L(u_1, u_2, ..., u_l)$ and $M(u_1, u_2, ..., u_l)$ are stored in

$$P\left[\left(\beta(u_1, u_2, ..., u_l) \bmod K_l\right)\left(\frac{N}{m^l}\right)^2 : \left(\frac{N}{m^l}\right)^2\right].$$

The number of passes required to compute all the $L(u_1, u_2, ..., u_l)$'s in Step (D$_l$) is $\lceil R^l / K_l \rceil$. Hence, the total number of passes in Steps (D$_1$) to (D$_n$) is

$$\begin{aligned} \sum_{l=1}^{n} \lceil R^l / K_l \rceil &\leq \sum_{l=1}^{n} (R^l / K_l + 1) \\ &\leq \sum_{l=1}^{n} \left(\frac{R^l}{p/(N/m^l)^2} + 1 \right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{l=1}^{n} \left(\frac{R^l (N/m^l)^2}{N^\alpha / \log N} + 1 \right) \\
&= \left(\frac{\log N}{N^{\alpha-2}} \right) \sum_{l=1}^{n} \left(\frac{R}{m^2} \right)^l + \log N \\
&= \left(\frac{\log N}{N^{\alpha-2}} \right) \left(\frac{(R/m^2)^{n+1} - 1}{R/m^2 - 1} \right) + \log N \\
&= O\left(\left(\frac{\log N}{N^{\alpha-2}} \right) (m^{\alpha-2})^n + \log N \right) \\
&= O(\log N).
\end{aligned}
$$

Taking the $L^*(u_1, u_2, ..., u_l)$'s into consideration, the running time of Steps (D$_1$) to (D$_n$) is at most doubled. The number of passes in Steps (C$_n$) to (C$_1$) is also $O(\log N)$, which can be obtained in a similar way. The number of passes in Step (B) is $\lceil R^n / K_n \rceil = O(\log N)$. Since each pass takes constant amount of time, the overall time complexity of our parallelized bilinear algorithm is $O(\log N)$.

Theorem 2. *Multiplying two $N \times N$ matrices can be performed in $O(\log N)$ time by a DMPC with $N^\alpha / \log N$ processors. Such a parallel computation is cost optimal and matches the performance of PRAM.* ∎

In the following, we show that our parallelized bilinear algorithm on a DMPC can be made fully scalable, that is, for all $1 \le p \le N^\alpha / \log N$, multiplying two $N \times N$ matrices can be performed by a DMPC with p processors in $O(N^\alpha / p)$ time, i.e., linear speedup can be achieved in the range $[1..N^\alpha / \log N]$. (As indicated in Section 3, for matrix multiplication using bilinear algorithms, we have $T^*(N) = O(\log N)$.)

It is clear that in the implementation mentioned in Theorem 2, processors perform computation and communication at the matrix element level. That is, processors

- calculate one element level addition or multiplication during a computation step in Steps (D$_1$) to (D$_n$) and Steps (C$_1$) to (C$_n$);
- calculate one element level multiplication during a computation step in Step (B);
- send/receive one element during a communication step in Steps (D$_1$) to (D$_n$) and Steps (C$_1$) to (C$_n$).

When there are fewer processors, it is necessary for processors to perform computation and communication at the block level. The idea is to partition the input/output matrices A, B, and C into blocks. When a matrix $X = (X_{ij})$ of size $N \times N$ is divided into blocks, i.e., the X_{ij}'s, of size $s \times s$, X is treated as a super-matrix of size $q \times q$, such that $N = qs$, and the blocks are treated as super-elements. Since the problem size is changed from N to q, the number of processors required in Theorem 2 is changed from $N^\alpha / \log N$ to $q^\alpha / \log q$. Let $r = \lceil \log_m q \rceil$. There are r levels of recursion. Then, processors perform computation and communication at the super-element (block) level, i.e., processors

- calculate one super-element level addition or a scalar-block multiplication during a computation step in Steps (D$_1$) to (D$_r$) and Steps (C$_1$) to (C$_r$);
- calculate one super-element level multiplication (i.e., multiplying matrices of size $s \times s$) during a computation step in Step (B);
- send/receive one super-element during a communication step in Steps (D$_1$) to (D$_r$) and Steps (C$_1$) to (C$_r$).

For super-matrices with super-elements, the running times in Lemmas 1–4 are increased by a factor of $O(s^2)$. This implies that the total running time of Steps (D$_1$) to (D$_r$) and Steps (C$_1$) to (C$_r$) is $O(s^2 \log q)$. Step (B) requires $O(s^\alpha \log q)$ time. Hence, multiplying two $q \times q$ super-matrices, where super-elements are $s \times s$ submatrices, can be performed by a DMPC with $q^\alpha / \log q$ processors in $O(s^\alpha \log q)$ time. Let q be the largest integer such that $q^\alpha / \log q \le p$. Hence, the overall running time is

$$
\begin{aligned}
O(s^\alpha \log q) &= O\left(\left(\frac{N}{q} \right)^\alpha \log q \right) \\
&= O\left(\left(\frac{N^\alpha}{p \log q} \right) \log q \right) = O\left(\frac{N^\alpha}{p} \right).
\end{aligned}
$$

Theorem 3. *For all $1 \le p \le N^\alpha / \log N$, multiplying two $N \times N$ matrices can be performed by a DMPC with p processors in $O(N^\alpha / p)$ time. In particular, For all $1 \le p \le N^{2.3755} / \log N$, multiplying two $N \times N$ matrices can be performed by a DMPC with p processors in $O(N^{2.3755} / p)$ time.* ∎

The first statement in Theorem 3 summarizes all sequential algorithms and parallelized bilinear algorithms for matrix multiplication on DMPC.

6 Extension to Other Parallel Systems

It is well known that multiplying two $N \times N$ matrices can be performed by a CREW PRAM with $N^\alpha / \log N$ processors in $O(\log N)$ time. Such performance has been achieved by a DMPC (see Theorem 2). As a matter of fact, Theorem 3, a generalized version of Theorem 2, holds for EREW PRAM, a weaker model of PRAM than CREW PRAM. The reason is that an EREW PRAM can easily simulate each communication step of a DMPC in constant time. For example, we can use a communication array

$$\texttt{c = (c[0], c[1], c[2], ..., c[}p-1\texttt{])}.$$

During a communication step specified as

$$((\pi(0), v_0), (\pi(1), v_1), (\pi(2), v_2), ..., (\pi(p-1), v_{p-1})),$$

processor P_j writes v_j to `c[`$\pi(j)$`]`, for all $0 \le j \le p-1$. Then all processors P_j read `c[`j`]` simultaneously. Clearly, there is no concurrent read/write.

Theorem 4. *For all $1 \le p \le N^\alpha / \log N$, multiplying two $N \times N$ matrices can be performed by a p-processor EREW PRAM in $O(N^\alpha / p)$ time.* ∎

Theorem 3 can also be extended to distributed memory systems with other interconnection networks that cannot support one-to-one communications in constant time.

Multiplying two $q \times q$ super-matrices, where super-elements are $s \times s$ submatrices, can be performed by a DMPC with $q^\alpha / \log q$ processors in $O(s^\alpha \log q + T(p) s^2 \log q)$ time, assuming that the parallel system can support one-to-one communication in $O(T(p))$ time. Let q be the largest integer such that $q^\alpha / \log q \le p$. This implies that $q^\alpha = \Theta(p \log q)$, $q = \Theta((p \log q)^{1/\alpha})$, and $\log q = \Theta(\log p)$. Hence, the overall running time is

$$
\begin{aligned}
&O(s^\alpha \log q + T(p) s^2 \log q) \\
&= O\left(\left(\frac{N}{q} \right)^\alpha \log q + T(p) \left(\frac{N}{q} \right)^2 \log q \right) \\
&= O\left(\left(\frac{N^\alpha}{p \log q} \right) \log q + T(p) \left(\frac{N^2}{(p \log q)^{2/\alpha}} \right) \log q \right) \\
&= O\left(\frac{N^\alpha}{p} + T(p) \left(\frac{N^2}{p^{2/\alpha}} \right) (\log p)^{1-2/\alpha} \right).
\end{aligned}
$$

Theorem 3 can be generalized as follows.

Theorem 5. *For all $1 \le p \le N^\alpha / \log N$, multiplying two $N \times N$ matrices can be performed by a parallel system with p processors in*

$$O\left(\frac{N^\alpha}{p} + T(p)\left(\frac{N^2}{p^{2/\alpha}}\right)(\log p)^{(\alpha-2)/\alpha}\right)$$

time, assuming that the parallel system can support one-to-one communication in $O(T(p))$ time. In particular, the above time complexity is $O(T(p)\log N)$ when $p = N^\alpha / \log N$. $\blacksquare$

Corollary. *Let $T(p) = (\log p)^\beta$, where $\beta > 0$. If p is in the range $[1..p^*]$, where*

$$p^* = \Theta\left(\frac{N^\alpha}{(\log N)^{\alpha\beta/(\alpha-2)+1}}\right),$$

linear speedup can be achieved, i.e., our parallelized bilinear algorithm is highly scalable. $\blacksquare$

Corollary. *Let $T(p) = p^\beta$, where $\beta > 0$. If p is in the range $[1..p^*]$, where*

$$p^* = \Theta\left(\frac{N^{\alpha(\alpha-2)/(\alpha-2+\alpha\beta)}}{(\log N)^{(\alpha-2)/(\alpha-2+\alpha\beta)}}\right),$$

linear speedup can be achieved. In this case, our parallelized bilinear algorithm is not highly scalable. $\blacksquare$

For example, it is well known that the Beneš network connecting p processors can support any one-to-one communication in $T(p) = O(\log p)$ time, provided that a mapping $\pi : \{0, 1, 2, ..., p-1\} \to \{-1, 0, 1, 2, ..., p-1\}$ is known in advance, which is really the case for matrix multiplication, where all communication steps are well defined. The Beneš network belongs to the class of hypercubic networks, including the hypercube network and many of its constant-degree derivatives such as cube-connected cycles, shuffle-exchange, generalized shuffle-exchange, de Bruijn, k-ary de Bruijn, butterfly, wrapped butterfly, k-ary butterfly, Omega, Flip, Baseline, Banyan, and Delta networks. All these networks are computationally equivalent in the sense that they can simulate each other with only constant factor slow down [15]. Thus, all these hypercubic networks can support one-to-one communication in $T(p) = O(\log p)$ time.

Theorem 6. *Multiplying two $N \times N$ matrices can be performed by a DMPC with p processors connected by a hypercubic network in*

$$O\left(\frac{N^\alpha}{p} + \left(\frac{N^2}{p^{2/\alpha}}\right)(\log p)^{2(\alpha-1)/\alpha}\right)$$

time, where $1 \le p \le N^\alpha / \log N$. $\blacksquare$

Corollary. *The time complexity in Theorem 6 is a decreasing function of p, and reaches its minimum $O((\log N)^2)$ when $p = N^\alpha / \log N$.* $\blacksquare$

Corollary. *The time complexity in Theorem 6 implies that if*

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

linear speedup can be achieved, i.e., our parallelized bilinear algorithm is highly scalable for hypercubic networks. $\blacksquare$

Theorem 6 is much stronger than the ones in [8], and can be used to solve a variety of important problems including matrix manipulations and graph problems (see Chapter 9 of [1]), and results

d	p^*
1	$O(N^{0.3242}/(\log N)^{0.1365})$
2	$O(N^{0.5706}/(\log N)^{0.2402})$
3	$O(N^{0.7641}/(\log N)^{0.3217})$
4	$O(N^{0.9202}/(\log N)^{0.3874})$
5	$O(N^{1.0487}/(\log N)^{0.4415})$
6	$O(N^{1.1563}/(\log N)^{0.4868})$
7	$O(N^{1.2478}/(\log N)^{0.5253})$
8	$O(N^{1.3265}/(\log N)^{0.5584})$
9	$O(N^{1.3950}/(\log N)^{0.5872})$
10	$O(N^{1.4550}/(\log N)^{0.6125})$

Figure 5. The value p^* ($\alpha = 2.3755$).

in significant performance improvement in terms of time and processor complexities, flexibility, and scalability.

Another popular network is the k-ary d-cube network (i.e., the d-dimensional torus network), that has d dimensions and each dimension has size k. (In general, the sizes of the dimensions can be different.) The number of processors is $p = k^d$. The class of k-ary d-cube networks contain the rings ($k = p$, $d = 1$), tori ($k = \sqrt{p}$, $d = 2$), and hypercubes ($k = 2$, $d = \log p$) as special cases. It is well known that one-to-one communication can be realized in $T(p) = O(dp^{1/d})$ time [15].

Theorem 7. *Multiplying two $N \times N$ matrices can be performed by a DMPC with p processors connected by a k-ary d-cube network in*

$$O\left(\frac{N^\alpha}{p} + \left(\frac{dN^2}{p^{2/\alpha-1/d}}\right)(\log p)^{(\alpha-2)/\alpha}\right)$$

time, where $1 \le p \le N^\alpha / \log N$. (The above time complexity reduces to that in Theorem 6 when $d = \log p$.) $\blacksquare$

Corollary. *When $d = 1$, the time complexity in Theorem 7 reaches its minimum $O(N^{\alpha^2/(2(\alpha-1))}(\log N)^{(\alpha-2)/(2(\alpha-1))})$, when $p = O(N^{\alpha(\alpha-2)/(2(\alpha-1))}/(\log N)^{(\alpha-2)/(2(\alpha-1))})$. For $d \ge 2$, the above time complexity is a decreasing function of p, and reaches its minimum $O(dN^{\alpha/d}(\log N)^{(d-1)/d})$ when $p = N^\alpha / \log N$.* $\blacksquare$

Corollary. *The time complexity in Theorem 7 implies that for $d \ge 1$, if $p \le p^*$ where*

$$p^* = O\left(\frac{N^{\alpha(\alpha-2)d/((\alpha-2)d+\alpha)}}{d^{\alpha d/((\alpha-2)d+\alpha)}(\log N)^{(\alpha-2)d/((\alpha-2)d+\alpha)}}\right),$$

linear speedup can be achieved. $\blacksquare$

The value p^* given by the above Corollary is displayed in Figure 5, assuming that $\alpha = 2.3755$. Clearly, when d is a constant, our parallelized bilinear algorithm is not highly scalable for k-ary d-cube networks.

7 Closing Remarks

We have shown that any known sequential algorithm for matrix multiplication over an arbitrary ring with time complexity $O(N^\alpha)$, where $2 < \alpha \le 3$, can be parallelized on a distributed memory parallel computer (DMPC) in $O(\log N)$ time by using

$N^\alpha / \log N$ processors. Such a parallel computation is cost optimal and matches the performance of PRAM. Furthermore, our parallelization on a DMPC can be made fully scalable, that is, for all $1 \leq p \leq N^\alpha / \log N$, multiplying two $N \times N$ matrices can be performed by a DMPC with p processors in $O(N^\alpha / p)$ time, i.e., linear speedup and cost optimality can be achieved in the range $[1..N^\alpha / \log N]$. This unifies all known algorithms for matrix multiplication on DMPC, standard or non-standard, sequential or parallel. Extensions of our methods and results to other parallel systems are also presented. We believe that we have made significant progress in scalable parallel matrix multiplication (as well as solving many other important problems) on distributed memory systems, both theoretically and practically.

Acknowledgments

This research was partially supported by National Aeronautics and Space Administration and the Research Foundation of State University of New York through the NASA/University Joint Venture (JOVE) in Space Science Program under Grant NAG8-1313.

References

[1] S.G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, New Jersey, 1997.

[2] R.J. Anderson and G.L. Miller, “Optical communication for pointer based algorithms,” TR CRI 88-14, Computer Science Dept., University of Southern California, 1988.

[3] A.F. Benner, H.F. Jordan, and V.P. Heuring, “Digital optical computing with optically switched directional couplers,” *Optical Engineering*, vol. 30, pp. 1936-1941, 1991.

[4] D. Bini and V. Pan, *Polynomial and Matrix Computations, Vol.1, Fundamental Algorithms*, Birkhäuser, Boston, 1994.

[5] A.K. Chandra, “Maximal parallelism in matrix multiplication,” Report RC-6193, IBM T.J. Watson Research Center, Yorktown Heights, New York, October 1979.

[6] D. Chiarulli, R. Melhem, and S. Levitan, “Using coincident optical pulses for parallel memory addressing,” *IEEE Computer*, vol. 30, pp. 48-57, 1987.

[7] D. Coppersmith and S. Winograd, “Matrix multiplication via arithmetic progressions,” *Journal of Symbolic Computation*, vol. 9, pp. 251-280, 1990.

[8] E. Dekel, D. Nassimi, and S. Sahni, “Parallel matrix and graph algorithms,” *SIAM Journal on Computing*, vol. 10, pp. 657-673, 1981.

[9] P.W. Dowd, “Wavelength division multiple access channel hypercube processor interconnection,” *IEEE Transactions on Computers*, vol. 41, pp. 1223-1241, 1992.

[10] M.M. Eshaghian, “Parallel algorithms for image processing on OMC,” *IEEE Transactions on Computers*, vol. 40, pp. 827-833, 1993.

[11] M. Geréb-Graus and T. Tsantilas, “Efficient optical communication in parallel computers,” *Proceedings of 4th ACM Symposium on Parallel Algorithms and Architectures*, pp. 41-48, 1992.

[12] L.A. Goldberg, M. Jerrum, T. Leighton, and S. Rao, “Doubly logarithmic communication algorithms for optical-communication parallel computers,” *SIAM Journal on Computing*, vol. 26, pp. 1100-1119, 1997.

[13] Z. Guo, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, “Pipelined communications in optically interconnected arrays,” *Journal of Parallel and Distributed Computing*, vol. 12, pp. 269-282, 1991.

[14] M. Hamdi and Y. Pan, “Efficient parallel algorithms on optically interconnected arrays of processors,” *IEE Proceedings - Computers and Digital Techniques*, vol. 142, pp. 87-92, 1995.

[15] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Mateo, California, 1992.

[16] S. Levitan, D. Chiarulli, and R. Melhem, “Coincident pulse techniques for multiprocessor interconnection structures,” *Applied Optics*, vol. 29, pp. 2024-2039, 1990.

[17] 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.

[18] K. Li, “Fast and scalable parallel algorithms for matrix chain product and matrix powers on optical buses,” in *High Performance Computing Systems and Applications*, Kluwer Academic Publishers, Boston, Massachusetts, 1999.

[19] K. Li and V.Y. Pan, “Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system,” *Proceedings of IPPS/SPDP ’99* (*2nd Merged Symp. of 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*), pp. 31-35, San Juan, Puerto Rico, April 1999.

[20] K. Li, Y. Pan, and M. Hamdi, “Solving graph theory problems using reconfigurable pipelined optical buses,” *Lecture Notes in Computer Science*, vol. 1586, pp. 911-923, 1999. Also to appear in *Parallel Computing*.

[21] K. Li, Y. Pan, and S.Q. Zheng, eds., *Parallel Computing Using Optical Interconnections*, Kluwer Academic Publishers, Boston, Massachusetts, 1998.

[22] 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, 1998.

[23] K. Li, Y. Pan, S.Q. Zheng, “Parallel matrix computations using a reconfigurable pipelined optical bus,” *Journal of Parallel and Distributed Computing*, vol. 59, no. 1, pp. 13-30, October 1999.

[24] K. Li, Y. Pan, and S.Q. Zheng, “Efficient deterministic and probabilistic simulations of PRAMs on linear arrays with reconfigurable pipelined bus systems,” *Journal of Supercomputing*, vol. 15, no. 2, pp. 163-181, February 2000.

[25] K. Li, Y. Pan, and S.Q. Zheng, “Scalable parallel matrix multiplication using reconfigurable pipelined optical bus systems,” *Proceedings of 10th International Conference on Parallel and Distributed Computing and Systems*, pp. 238-243, Las Vegas, Nevada, October 1998.

[26] Y. Li, Y. Pan, and S.Q. Zheng, “Pipelined TDM optical bus with conditional delays,” *Optical Engineering*, vol. 36, no. 9, pp. 2417-2424, 1997.

[27] K. Mehlhorn and U. Vishkin, “Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories,” *Acta Informatica*, vol. 21, pp. 339-374, 1984.

[28] 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, 1996.

[29] 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, 1998.

[30] 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, 1998.

[31] 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.

[32] V. Pan, “New fast algorithms for matrix operations,” *SIAM Journal on Computing*, vol. 9, pp. 321-342, 1980.

[33] V. Pan, “Complexity of parallel matrix computations,” *Theoretical Computer Science*, vol. 54, pp. 65-85, 1987.

[34] V. Pan, “Parallel solution of sparse linear and path systems,” in *Synthesis of Parallel Algorithms*, J.H. Reif, ed., pp. 621-678, Morgan Kaufmann, San Mateo, California, 1993.

[35] V. Pan and J. Reif, “Efficient parallel solution of linear systems,” *Proceedings of 7th ACM Symposium on Theory of Computing*, pp. 143-152, May 1985.

[36] S. Pavel and S.G. Akl, “Matrix operations using arrays with reconfigurable optical buses,” *Journal of Parallel Algorithms and Applications*, vol. 8, pp. 223-242, 1996.

[37] C. Qiao and R. Melhem, “Time-division optical communications in multiprocessor arrays,” *IEEE Transactions on Computers*, vol. 42, pp. 577-590, 1993.

[38] S. Rajasekaran and S. Sahni, “Sorting, selection, and routing on the array with reconfigurable optical buses,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 11, pp. 1123-1132, 1997.

[39] V. Strassen, “Gaussian elimination is not optimal,” *Numerische Mathematik*, vol. 13, pp. 354-356, 1969.

[40] S.Q. Zheng and Y. Li, “Pipelined asynchronous time-division multiplexing optical bus,” *Optical Engineering*, vol. 36, no. 12, pp. 3392-3400, 1997.