Parallel Matrix Multiplication on a Linear Array with a Reconfigurable Pipelined Bus System

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

Victor Y. Pan†
Dept. of Mathematics and Computer Science
Lehman College, CUNY
Bronx, New York 10468-1589
VPAN@lcvax.lehman.cuny.edu

Abstract

The known fast sequential algorithms for multiplying two $N \times N$ matrices (over an arbitrary ring) have time complexity $O(N^\alpha)$, where $2 < \alpha < 3$. The current best value of α is less than 2.3755. We show that for all $1 \leq p \leq N^\alpha$, multiplying two $N \times N$ matrices can be performed on a p-processor linear array with a reconfigurable pipelined bus system (LARPBS) in $O(N^\alpha/p + (N^2/p^{2/\alpha}) \log p)$ time. This is currently the fastest parallelization of the best known sequential matrix multiplication algorithm on a distributed memory parallel system.

1 Introduction

Many sequential algorithms have been proposed for matrix multiplication, one of the most fundamental problems in sciences and engineering. The standard sequential algorithm takes $O(N^3)$ operations to multiply two $N \times N$ matrices. Since Strassen's remarkable discovery of his $O(N^{2.8074})$ algorithm [17], 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 exponent is $\alpha < 2.3755$ [3].

The standard algorithm for matrix multiplication, Strassen's algorithm of [17] with $\alpha < 2.8074$, and its Winograd's variation are used extensively in practice [5]. Other known asymptotically fast algorithms are not practically useful because of large overhead constants hidden in the big-O notation (this applies to all the known algorithms with $\alpha < 2.78$), or because of the requirement of increasing memory space [6]. It is still plausible that faster practical algorithms for matrix multiplication will appear, which should motivate theoretical importance of parallelization of matrix multiplication algorithms, and in the case of Strassen's algorithm and its Winograd variation, such a parallelization should already have practical value.

On shared memory multiprocessors, Strassen's algorithm has been parallelized [2]. Furthermore, it is well known that two $N \times N$ matrices can be multiplied under a CREW PRAM in $O(\log N)$ time by using $O(N^{\alpha+\varepsilon})$ processors, for any fixed positive ε, as soon as we have an algorithm using $O(N^\alpha)$ arithmetic time for $N \times N$ matrix multiplication [1, 15]. As a matter of fact, based on any of the known algorithms for $N \times N$ matrix multiplication running in $O(N^\alpha)$ time, we yield $O(\log N)$ parallel time using $O(N^\alpha/\log N)$ arithmetic processors under the PRAM model.

*Supported by National Aeronautics and Space Administration and the State University of New York through the NASA/University Joint Venture (JOVE) in Space Science Program under Grant NAG8-1313.

†Supported by National Science Foundation under Grant CCR 9625344 and PSC CUNY Award 668365.

On distributed memory multicomputers (which are considered more practical), research has essentially focused on the parallelization of the standard method. 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 $N^2 \leq p \leq N^3$ [4]. It was also reported that matrix multiplication can be done in constant time on a reconfigurable mesh with N^4 processors [16]. Such an implementation, though very fast, is far from cost-optimal.

To have fast and processor efficient parallel algorithms, it is necessary to consider non-standard algorithms. To the best of the authors' knowledge, all $O(N^\alpha)$ sequential algorithms with $\alpha < 3$ have not been fully parallelized on any distributed memory systems, since these systems do not have sufficient capability to support complicated communication efficiently. In [4], 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 $1 \leq p \leq N^2$. This algorithm is valid only in a small interval of p, which is not cost-optimal (as compared to the $O(N^\alpha)$ sequential algorithm), and the shortest execution time is reached when $p = N^2$ is $O(N)$, 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 the fast sequential matrix multiplication algorithm on distributed memory systems, more powerful communication mechanism is required. It is clear that all existing realistic static networks with electronic connections have limited communication capability in supporting fast parallelization of an $O(N^\alpha)$ algorithm, where $\alpha < 3$.

Recently, fiber optical buses have emerged as promising networks [11]. 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 [12]. 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 [7] and sorting [9]. Numerous parallel algorithms using optical interconnection networks have been developed recently [7, 8, 9, 10].

On a linear array with a reconfigurable pipelined bus system (LARPBS) proposed in [12], Strassen's algorithm has been parallelized and has execution time $O((\log N)^\delta)$ by using $O(N^3/1.1428^{(\log N)^\delta})$ processors, where $0 \leq \delta \leq 1$ [8]. 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 is thus far the fastest parallel matrix multiplication algorithm on distributed memory models.

In this paper, we show that for all $1 \leq p \leq N^\alpha$, multiplying two $N \times N$ matrices (over an arbitrary ring) 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 algorithm achieves linear speedup and is cost-optimal. This is currently the fastest and most processor efficient parallelization of the best known sequential matrix multiplication algorithms on a distributed memory parallel system. In particular, for all $1 \leq p \leq N^{2.3755}$, multiplying two $N \times N$ matrices can be performed on a p-processor LARPBS in $O(N^{2.3755}/p + (N^2/p^{0.8419}) \log p)$ time, and linear speedup can be achieved for p as large as $O(N^{2.3755}/(\log N)^{6.3262})$. Furthermore, multiplying two $N \times N$ matrices can be performed by an LARPBS with $O(N^\alpha)$ processors in $O(\log N)$ time. This matches the performance of PRAM. Also, it is clear that the processor complexity is substantially reduced as compared with that in [8].

2 The LARPBS Computing Model

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 signal (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. 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.

A linear array with a reconfigurable pipelined bus system (LARPBS) consists of N processors P_1, P_2, P_3, ..., P_N, connected by an 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 [12] 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.

Perhaps the best way to understand the LARPBS computing model is to inspect the primitive operations that it can efficiently support. A number of basic communication, data movement, and aggregation operations on the LARPBS model implemented using the coincident pulse processor addressing technique have been developed [8, 12]. Each of these primitive operations can be performed in a constant number of bus cycles.

One-to-One Communication. Assume that processors P_{i_1}, P_{i_2}, ..., P_{i_q} are senders, and processors P_{j_1}, P_{j_2}, ..., P_{j_q} are receivers. In particular, processor P_{i_k} sends a value x_{i_k} to P_{j_k}, for all $1 \leq k \leq q$ simultaneously.

Broadcasting. Here, we have a source processor P_i, who sends a value x to all the N processors $P_1, P_2, P_3, ..., P_N$.

Multicasting. In a multicasting operation, we have a source processor P_i, who sends a value x to a subset of the N processors P_{j_1}, P_{j_2}, ..., P_{j_m}.

Multiple Multicasting. Assume that we have g disjoint groups of destination processors, $G_k = \{P_{j_{k,1}}, P_{j_{k,2}}, P_{j_{k,3}}, ...\}$, $1 \leq k \leq g$, and there are g senders P_{i_1}, P_{i_2}, ..., P_{i_g}. Processor P_{i_k} has value x_{i_k} to be broadcast to all the processors in G_k, where $1 \leq k \leq g$.

Global Summation. Suppose that every processor P_j holds a numerical value v_j, $1 \leq j \leq N$, where v_j is an integer or a floating-point value with finite magnitude and precision, we need to calculate the summation $v_1 + v_2 + v_3 + \cdots + v_N$. The summation is finally saved in P_1.

It has been a common practice in algorithm analysis to assume that a single manipulation (e.g., an arithmetic operation) takes constant time. This essentially implies that numerical values have finite magnitude and precision; otherwise, a manipulation either takes longer time, or requires extra hardware support. Therefore, our assumption in the global summation is quite reasonable. However, since in this paper, we are dealing with matrix multiplication on an arbitrary ring, we need the following general aggregation operation.

Global Aggregation. Suppose that every processor P_j holds a value v_j, $1 \leq j \leq N$, where v_j is in an arbitrary set S with a binary associative operator $\oplus$, we need to calculate $v_1 \oplus v_2 \oplus v_3 \oplus \cdots \oplus v_N$. The result of the aggregation is finally saved in P_1. We say that the size of the aggregation is N.

It may not be the case that all such kind of global aggregations can be implemented using optical signals in constant number of bus cycles. However, we can still use the ordinary binary tree method to find the aggregation in $O(\log N)$ time, where the data communications can be easily supported by an optical bus. If N is a constant, such an aggregation requires constant amount of time. Fortunately, in this paper, we only use *aggregations of constant sizes* on subarrays whose sizes are independent of matrix sizes, and hence, their constant execution time is *independent of the data set S and the definition of $\oplus$*. We will mention the size of each aggregation explicitly.

3 The Strategy

In this section, we show how to parallelize any fast sequential matrix multiplication algorithm, with one limitation, namely, the execution time is bounded from below by $O(\log N)$. This limitation is due to the recursive nature of the class of bilinear algorithms, not to the communication constraints on a parallel system. Assume that the known sequential algorithm for multiplying two $N \times N$ matrices has time complexity $O(N^\alpha)$.

We follow the approach [1, pages 315-316] and [14, 15]. With no loss of generality, we will consider the class of recursive bilinear algorithms for the evaluation of the matrix product $C = A \times B$, where $A = (a_{ij})$, $B = (b_{jk})$, and $C = (c_{ik})$ are $m \times m$ matrices. First of all, there is a basis bilinear computation that has three steps.

Step (a). In the first step, the values of $2R$ linear functions are computed, $L_u = \sum_{1 \leq i,j \leq m} f(i,j,u) a_{ij}$, and $L_u^* = \sum_{1 \leq j,k \leq m} f^*(j,k,u) b_{jk}$, where $u = 1, 2, ..., R$.

Step (b). Then, in the second step, we compute the R products $L_u L_u^*$, for all $1 \leq u \leq R$.

Step (c). Finally, in the third step, we calculate the m^2 outputs, $c_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk} = \sum_{u=1}^{R} f^{**}(k,i,u) L_u L_u^*$, for all $1 \leq i, k \leq m$.

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 any positive ε and any existent or plausible algorithm for $N \times N$ matrix multiplication running in $O(N^\alpha)$ arithmetic time, we may fix a natural m and a basis bilinear computation with $R \leq m^{\alpha+\varepsilon}$ [13]. 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 $\alpha = \log_2 7 < 2.8074$ [17], and the ones of [3] for $\alpha < 2.3755$, which are currently asymptotically fastest), we may

yield $R = m^\alpha$ in the associated basis bilinear construction. (Note that for fixed m and α, R is finite.)

The above bilinear algorithm can be used recursively. Let $A = (A_{ij})$, $B = (B_{jk})$, and $C = (C_{ik})$ be $N \times N$ matrices, where $N = m^n$. Assume that m is fixed, and $n \to \infty$. Each of these matrices are partitioned into m^2 submatrices A_{ij}, B_{jk}, C_{ik} of size $m^{n-1} \times m^{n-1}$. Then, the above computation is still applicable when all the a_{ij}'s, b_{jk}'s, and c_{ik}'s are replaced by submatrices. The recursive algorithm that computes $C = A \times B$ is described below.

Step (0). If the matrices are of size $m \times m$, compute the product $C = A \times B$ directly using the method above (i.e., Steps (a)–(c)), and return; otherwise, do Steps (1)–(3).

Step (1). Calculate $2R$ linear combinations of submatrices $L_u = \sum_{1 \le i,j \le m} f(i,j,u) A_{ij}$, and $L_u^* = \sum_{1 \le j,k \le m} f^*(j,k,u) B_{jk}$, for all $1 \le u \le R$, where L_u and L_u^* are $m^{n-1} \times m^{n-1}$ matrices.

Step (2). Calculate the R matrix products $L_u \times L_u^*$, for all $1 \le u \le R$.

Step (3). Compute $C_{ik} = \sum_{j=1}^{m} A_{ij} B_{jk} = \sum_{u=1}^{R} f^{**}(k,i,u)(L_u \times L_u^*)$, for all $1 \le i,k \le m$.

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 (0)) when the size of the submatrices is $m \times m$. If sufficiently many processors are available, we can calculate the L_u's in parallel, and then all the L_u^*'s in parallel (cf. Step (1)). After the recursion in Step (2), all the C_{ik}'s are also computed in parallel (see Step (3)). Hence, each level takes constant time, and the overall time complexity is $O(\log N)$.

Under the PRAM model, the above computation only needs $R^n / \log N \le m^{(\alpha+\varepsilon)n} / \log N = N^{\alpha+\varepsilon} / \log N$ processors, for any fixed positive ε. Furthermore, for all so far available sequential algorithms for matrix multiplication, there is a parallelization which requires $O(N^\alpha / \log N)$ processors (see p.317 in [1]).

4 Implementation Details

In this section, we examine the implementation details of a bilinear algorithm on an LARPBS with $m^2 R^n$ processors. Our algorithm is called $\texttt{Fast}(n)$, which stands for the fastest bilinear algorithm parallelized on an LARPBS with $p_n = m^2 R^n$ processors P_1, P_2, ..., P_{p_n}, where p_n is the total number of processors required in the above bilinear algorithm.

The number of processors required to achieve the above maximum parallelism is analyzed as follows. When $n = 1$, we have the base case. Step (a) needs $m^2 R$ processors so that all the L_u's (and then all the L_u^*'s) are obtained in parallel. Step (b) only requires R processors. Step (c) takes $m^2 R$ processors so that all the c_{ik}'s are calculated in parallel. Hence, $p_1 = m^2 R$.

In general, when $n > 1$, Steps (1) and (3) require $m^2 R (m^{n-1})^2 = m^{2n} R$ processors, and Step (2) needs $p_{n-1} R$ processors. That is, $p_n = \max(m^{2n} R, p_{n-1} R)$. It can be proven by induction on n that $p_n = m^2 R^n$, for all $n \ge 1$. Therefore, for a fixed m, and $n \to \infty$, the total number of processors used for multiplying two $N \times N$ matrices, where $N = n^m$, is $p_n = O(R^n) = O(R^{\log_m N}) = O(N^{\log_m R}) = O(N^\alpha)$.

4.1 The Base Case

We first examine $\texttt{Fast}(1)$, that is, the base case, which calculates $C = A \times B$, where $A = (a_{ij})$, $B = (b_{jk})$, and $C = (c_{ik})$ are $m \times m$ matrices, in constant time by using $p_1 = m^2 R$ processors.

Assume that initially, the input matrices A and B are stored in the first m^2 processors in the row-major order, that is, elements a_{ij} and b_{ij} are stored in processor $P_{(i-1)m+j}$, for all $1 \le i,j \le m$, and when algorithm $\texttt{Fast}(1)$ completes, the output matrix C is stored in the first m^2 processors in the row-major order, that is, element c_{ij} is found in processor $P_{(i-1)m+j}$, for all $1 \le i,j \le m$,

For convenience, different index systems are used for the p_1 processors during the execution of algorithm $\texttt{Fast}(1)$. In Step (a), processors are also named as $P_{u,i,j}$, where $1 \le u \le R$, and $1 \le i,j \le m$, and the processors are mapped to the linear array using the lexicographical order, namely, $P_{u,i,j}$ corresponds to $P_{(u-1)m^2+(i-1)m+j}$. We use $P_{u,*,*}$ to denote a subarray with m^2 processors, i.e., the $P_{u,i,j}$'s for all $1 \le i,j \le m$. The p_1 processors will be divided into R subarrays, $P_{1,*,*}$, $P_{2,*,*}$, ..., $P_{R,*,*}$, such that the subarray $P_{u,*,*}$ is used to calculate L_u and L_u^*, where $1 \le u \le R$. There are three basic operations to compute the L_u's.

- For all $1 \le i,j \le m$, processor $P_{1,i,j}$ sends a_{ij} to $P_{u,i,j}$, for all $2 \le u \le R$. This is actually a multiple multicasting operation. Thus, matrix A is available to each subarray $P_{u,*,*}$, where $1 \le u \le R$.
- Processor $P_{u,i,j}$ performs one local calculation for $f(i,j,u) a_{ij}$, for all $1 \le i,j \le m$, and $1 \le u \le R$.
- The value L_u is then calculated by the m^2 processors in $P_{u,*,*}$ via an aggregation of size m^2 for the summation $\sum_{1 \le i,j \le m} f(i,j,u) a_{ij}$. To this end, it is necessary to reconfigure the original LARPBS with p_1 processors into R independent subsystems LARPBS_1, LARPBS_2, ..., LARPBS_R, where LARPBS_u contains processors in $P_{u,*,*}$, for all $1 \le u \le R$.

The L_u^*'s can be obtained in a similar way. Let us assume that after Step (a), L_u and L_u^* are hold by processor $P_{u,1,1}$, for all $1 \le u \le R$.

In Step (b), there are two basic operations.

- Processor $P_{u,1,1}$ sends L_u and L_u^* to processor P_u, for all $1 \le u \le R$ in parallel. This can be done using two one-to-one communications.
- Once processor P_u receives L_u and L_u^*, it can compute the product $L_u L_u^*$ using local computation.

After Step (b) is finished, processor P_u has the values $L_u L_u^*$, for all $1 \le u \le R$.

In Step (c), processors are called $P_{i,k,u}$, where $1 \le i,k \le m$, and $1 \le u \le R$. The subarray $P_{i,k,*}$, which contains processors $P_{i,k,1}$, $P_{i,k,2}$, ..., $P_{i,k,R}$, is used to compute c_{ik}, for all $1 \le i,k \le m$.

- First, for all $1 \le u \le R$, processor P_u sends the product $L_u L_u^*$ to processor $P_{i,k,u}$, for all $1 \le i,k \le m$, via a multiple multicasting operation.
- Second, processor $P_{i,k,u}$ computes the value $f^{**}(k,i,u) L_u L_u^*$ locally.
- Third, the summation $c_{ik} = \sum_{u=1}^{R} f^{**}(k,i,u) L_u L_u^*$ is assembled by the processors in $P_{i,k,*}$ using an aggregation operation of size R. The result c_{ik} is then stored in $P_{i,k,1}$, for all $1 \le i,k \le m$. System reconfiguration is required in this step.
- Finally, we need to perform a one-to-one communication to bring c_{ik} from $P_{i,k,1}$ to $P_{(i-1)m+k}$, for all $1 \le i,k \le m$, to meet the output data layout requirement.

It is clear that Steps (a), (b), and (c) can all be implemented using primitive data movement and communication operations, aggregations of constant sizes, and local operations, in constant number of bus cycles. Hence, we reach the following conclusion.

Theorem 1. Multiplying two $m \times m$ matrices can be performed in constant time on an LARPBS with $p_1 = m^2 R$ processors. ∎

4.2 The General Case

Now, let us look at algorithm $\texttt{Fast}(n)$ in the general case where $n > 1$. As mentioned before, the number of processors used by algorithm $\texttt{Fast}(n)$ is $p_n = m^2 R^n$. Basically, we will show how each level of the recursion can be implemented in constant amount of time.

We assume that the input matrices A and B, as well as the output matrix C are all arranged in the shuffled-row-major order in the first N^2 processors of an LARPBS. Essentially, this means that if $A = (A_{ij})$ of size $m^n \times m^n$ is divided into blocks A_{ij} of size $m^{n-1} \times m^{n-1}$, then the blocks are arranged in the row-major order. Furthermore, the arrangement of the blocks are defined recursively, and the base matrices of size $m \times m$ are in the row-major order.

In Step (0), we reach the base case. Therefore, algorithm $\texttt{Fast}(1)$ is executed in constant time using p_1 processors to multiply two base matrices of size $m \times m$.

In Step (1), only $p'_n = N^2 R = m^{2n} R < m^2 R^n = p_n$ processors are required. These p'_n processors are divided into $m^2 R$ groups, and each group of $m^{2(n-1)}$ processors is called a super-processor, which can be used to store a block A_{ij} in the shuffled-row-major order. Let us label these super-processors as $\mathcal{P}_{u,i,j}$, where $1 \le u \le R$, and $1 \le i, j \le m$. The subarray of super-processors $\mathcal{P}_{u,*,*}$ calculates matrices L_u and L'_u, for all $1 \le u \le R$. It is clear that in the shuffled-row-major order, super-processor $\mathcal{P}_{1,i,j}$ holds the block A_{ij}, for all $1 \le i, j \le m$.

- For all $1 \le i, j \le m$, super-processor $\mathcal{P}_{1,i,j}$ sends A_{ij} to $\mathcal{P}_{u,i,j}$, for all $2 \le u \le R$. This is accomplished by a multiple multicasting operation. Matrix A is then available to each subarray $\mathcal{P}_{u,*,*}$, where $1 \le u \le R$.
- Super-processor $\mathcal{P}_{u,i,j}$ computes $f(i,j,u)A_{ij}$ using local calculation, for all $1 \le i, j \le m$, and $1 \le u \le R$.
- The matrix L_u of size $m^{n-1} \times m^{n-1}$ is then calculated by the m^2 super-processors in $\mathcal{P}_{u,*,*}$. To this end, in addition to reconfiguration of the original LARPBS into R independent subsystems of super-processors, it is necessary to perform data rearrangement within each subsystem, and to reconfigure each subsystem into $m^{2(n-1)}$ sub-subsystems, such that each sub-subsystem (with m^2 processors) calculates one element in L_u. There are three operations to obtain the submatrix summation $\sum_{1 \le i,j \le m} f(i,j,u)A_{ij}$, a one-to-one communication to bring the corresponding submatrix elements together into sub-subsystems, parallel aggregations of size m^2 in the sub-subsystems, and a one-to-one communication to put the results to the right processors. Detailed specification is omitted here.

The matrix L^*_u can be obtained in a similar way. Let us assume that after Step (1), L_u and L^*_u are hold by super-processor $\mathcal{P}_{u,1,1}$ in the shuffled-row-major order, for all $1 \le u \le R$.

In Step (2), all the p_n processors are employed. These p_n processors will be divided into R subsystems LARPBS_1, LARPBS_2, ..., LARPBS_R, and each subsystem LARPBS_u has p_{n-1} processors used to calculate the matrix product $L_u \times L^*_u$, for all $1 \le u \le R$. For convenience, processors are also grouped into super-processors.

- Super-processor $\mathcal{P}_{u,1,1}$ sends L_u and L^*_u to the first super-processor in LARPBS_u, for all $1 \le u \le R$ in parallel, using two one-to-one communications.
- LARPBS_u computes the product $L_u \times L^*_u$ by recursively invoking algorithm $\texttt{Fast}(n-1)$ and using p_{n-1} processors, where $1 \le u \le R$.

After Step (2) is finished, the first super-processor in LARPBS_u has the matrices $L_u \times L^*_u$, for all $1 \le u \le R$.

In Step (3), again, only p'_n processors are used. Once more, the p'_n processors are grouped into super-processors $\mathcal{P}_{i,k,u}$, where $1 \le i, k \le m$, and $1 \le u \le R$, and each super-processor has $m^{2(n-1)}$ processors. These $m^2 R$ super-processors are divided into m^2 subarrays $\mathcal{P}_{i,k,*}$, and $\mathcal{P}_{i,k,*}$ is used to compute one submatrix C_{ik}, where $1 \le i, k \le m$. Each subarray $\mathcal{P}_{i,k,*}$ has R super-processors $\mathcal{P}_{i,k,1}$, $\mathcal{P}_{i,k,2}$, ..., $\mathcal{P}_{i,k,R}$, where $\mathcal{P}_{i,k,u}$ holds the matrix product $L_u \times L^*_u$, for all $1 \le u \le R$.

- For all $1 \le u \le R$, the first super-processor in LARPBS_u of Step (2) sends the matrix product $L_u \times L^*_u$ to all $P_{i,k,u}$, where $1 \le i, k \le m$.
- Super-processor $\mathcal{P}_{i,k,u}$ calculates $f^{**}(k,i,u)(L_u \times L^*_u)$ by local computation, where $1 \le i, k \le m$, and $1 \le u \le R$.
- Subarray $\mathcal{P}_{i,k,*}$ computes the submatrix summation $\sum_{u=1}^{R} f^{**}(k,i,u)(L_u \times L^*_u)$, where $1 \le i, k \le m$, through appropriate data rearrangement and aggregations of size R. The summation is stored in super-processor $\mathcal{P}_{i,k,1}$.
- Super-processor $\mathcal{P}_{i,k,1}$ sends the result C_{ik} to the right place via a one-to-one communication, for all $1 \le i, k \le m$.

It is clear that except the recursion in Step (2), all other steps take constant number of bus cycles. Since each of the $O(\log N)$ levels of the recursion requires constant amount of time, algorithm $\texttt{Fast}(n)$ has time complexity $O(\log N)$. Thus, we have the following claim.

Theorem 2. Multiplying two $N \times N$ matrices can be performed in $O(\log N)$ time on an LARPBS with $O(N^\alpha)$ processors. ∎

5 Fewer Processors: $1 \le p \le N^\alpha$

In the last section, we provide the implementation details of parallelization of the fastest sequential matrix multiplication algorithm when there are sufficiently many processors. In reality, the number of processors available is not always enough for an application. In this section, we examine the case where the number of processors p is arbitrarily chosen in the range $[1..N^\alpha]$.

It is clear that in the implementation of Section 4, processors perform computation and communication at the element level, that is, processors

- send to each other one matrix element at a time;
- calculate element product locally;
- and aggregate element summations.

(Remark: Communication and aggregation of submatrices among super-processors are actually done by individual processors at the element level.) All the above operations take constant bus cycles. When there are fewer processors, it is necessary for processors to perform computation and communication at the submatrix level.

Let q be an integer such that $q^\alpha \le p$, i.e., $q = \lfloor p^{1/\alpha} \rfloor$, and $s = \lceil N/q \rceil$. All the matrices $A = (A_{ij})$, $B = (B_{jk})$, and $C = (C_{ik})$ are partitioned into submatrices A_{ij}, B_{jk}, C_{ik} of size $s \times s$. Our algorithm is called $\texttt{Fast}(n,p)$, which is essentially the same as $\texttt{Fast}(n)$, except that we imagine that each of these submatrices is a single element. Therefore, processors

- send to each other one submatrix at a time;
- calculate submatrix product locally;
- and aggregate submatrix summations.

Each communication (i.e., one-to-one and multiple multicasting) or aggregation at the submatrix level can be realized by s^2 communication or aggregation at the element level. This implies that the execution times of Steps (1), (2), and (3) are augmented by a factor of s^2. In Step (0), i.e., the base case, each processor calculates submatrix multiplication sequentially, which takes $O(s^\alpha)$ time. It seems that the matrix sizes are reduced from $N \times N$ to $q \times q$, if all submatrices are treated as single elements. This implies that the number of levels of recursion is $\lceil \log_m q \rceil$. Therefore, the overall execution time $T(N, p, \alpha)$ of algorithm $\texttt{Fast}(n, p)$ to multiply two $N \times N$ matrices using p processors by parallelizing the best $O(N^\alpha)$ sequential algorithm is $T(N, p, \alpha) = O(s^\alpha + s^2 \log q) = O(N^\alpha/p + (1/\alpha)(N^2/p^{2/\alpha}) \log p)$. The above discussion essentially proves the following theorem.

Theorem 3. For all $1 \leq p \leq N^\alpha$, 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. ∎

Theorem 2 is a special case of Theorem 3 by taking $p = N^\alpha$.

Corollary 1. Multiplying two $N \times N$ matrices can be performed on an LARPBS in $O(\log N)$ time, using N^α processors. In particular, multiplying two $N \times N$ matrices can be performed on an LARPBS in $O(\log N)$ time, using $N^{2.3755}$ processors. ∎

Notice that Corollary 1 is well known on the PRAM model [15].

6 Cost-Optimality

We now show that algorithm $\texttt{Fast}(n, p)$ achieves linear speedup and is cost-optimal in a wide range of p for all the known $N \times N$ matrix multiplication algorithms. It is clear from Theorem 3 that when $p \ll N$, the term $O(N^\alpha/p)$ dominates the time complexity of $\texttt{Fast}(n, p)$. When p is close to N^α, the second term dominates $T(N, p, \alpha)$. Since $T(N, p, \alpha) = O(N^\alpha/p + (N^\alpha/p)^{2/\alpha} \log p)$, we write $p = N^\alpha/f(N)$, which yields $T(N, p, \alpha) = O(f(N) + (f(N))^{2/\alpha} \log N)$, where $f(N)$ is a small infinity. Clearly, when $f(N) = O((\log N)^{\alpha/(\alpha-2)})$, we have $T(N, p, \alpha) = O((f(N))^{2/\alpha} \log N)$. When p does not exceed $O(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$, the speedup of algorithm $\texttt{Fast}(n, p)$ is $S(N, p, \alpha) = O(p)$. When $p = \Omega(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$, we still have $S(N, p, \alpha) = \Omega(p/\log N)$. Similarly, the cost of algorithm $\texttt{Fast}(n, p)$ is $C(N, p, \alpha) = O(N^\alpha)$ for p as large as $O(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$. When $p = \Omega(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$, we have $C(N, p, \alpha) = O(p^{(\alpha-2)/\alpha} N^2 \log N)$. The highest cost is $O(N^\alpha \log N)$, when $p = N^\alpha$.

Corollary 2. If $p = O(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$, algorithm $\texttt{Fast}(n, p)$ achieves linear speedup and is cost-optimal. ∎

The following result is an immediate consequence of Theorem 3 and Corollary 2.

Corollary 3. For all $1 \leq p \leq N^{2.3755}$, multiplying two $N \times N$ matrices can be performed on a p-processor LARPBS in $O(N^{2.3755}/p + (N^2/p^{0.8419}) \log p)$ time. Hence, linear speedup can be achieved for $p = O(N^{2.3755}/(\log N)^{6.3262})$. ∎

7 Concluding Remarks

We have developed an efficient parallelization of the fastest sequential matrix multiplication algorithm on a linear array with a reconfigurable pipelined optical bus system. The algorithm has linear speedup and cost-optimality in a wide range of choice for the number of processors. It turns out that for parallel matrix multiplication, a distributed memory system with optical interconnections like LARPBS compares favorably with shared memory systems, where the concurrent read capability is replaced by highly efficient communications with predictable senders and receivers.

As indicated in Section 3, under the PRAM model, $O(N^\alpha/\log N)$ processors are enough to achieve $O(\log N)$ parallel execution time. While such a parallelization is not difficult under the PRAM model, where all the processors share infinite common memory, it is more involved to implement such a parallelization in a distributed memory model. However, we believe that by carefully designing data distribution and scheduling the computation and communication, it is possible to parallelize a sequential $O(N^\alpha)$ matrix multiplication algorithm on an LARPBS in $O(\log N)$ time by using $O(N^\alpha/\log N)$ processors. This will be our next step of investigation.

References

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

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

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

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

[5] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Maryland, 1996.

[6] J. Laderman, V.Y. Pan, and X.-H. Sha, "On practical acceleration of matrix multiplication," *Linear Algebra and Its Applications*, vol.162-164, pp.557-588, 1992.

[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, 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.

[9] 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*.

[10] 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, October 1998.

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

[12] Y. Pan and K. Li, "Linear array with a reconfigurable pipelined bus system – concepts and applications," *Information Sciences – An International Journal*, vol.106, no.3-4, pp.237-258, 1998.

[13] V. Pan, "How to multiply matrices faster," *Lecture Notes in Computer Science*, vol.179, Springer Verlag, Berlin, 1984.

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

[15] 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.

[16] 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.

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