

# Parallel Matrix Computations Using a Reconfigurable Pipelined Optical Bus

Keqin Li<sup>1</sup>

*Department of Mathematics and Computer Science, State University of New York,  
New Paltz, New York 12561-2499*  
E-mail: [li@mcs.newpaltz.edu](mailto:li@mcs.newpaltz.edu)

Yi Pan

*Department of Computer Science, University of Dayton, Dayton, Ohio 45469-2160*  
E-mail: [pan@hype.cps.udayton.edu](mailto:pan@hype.cps.udayton.edu)

and

Si Qing Zheng

*Department of Computer Science, University of Texas at Dallas,  
Richardson, Texas 75083-0688*  
E-mail: [sizheng@utdallas.edu](mailto:sizheng@utdallas.edu)

Received March 2, 1998; revised June 14, 1999; accepted June 15, 1999

---

We present fast and cost-efficient parallel algorithms for a number of important and fundamental matrix computation problems on linear arrays with reconfigurable pipelined optical bus systems. These problems include computing the inverse, the characteristic polynomial, the determinant, the rank, the  $N$ th power, and an LU- and a QR-factorization of a matrix and solving linear systems of equations. Our algorithms provide a wide range of performance–cost combinations. Compared with known results, the running time of parallel solutions to all these problems can be reduced by a factor of  $O(\log N)$  while costs are maintained under  $o(N^4)$ . © 1999 Academic Press

*Key Words:* characteristic polynomial; cost; determinant; linear system of equations; LU-factorization; matrix inversion; matrix multiplication; optical pipelined bus; processor array; rank; time complexity.

---

<sup>1</sup>To whom correspondence should be addressed. Fax: (914) 257-3571.

## 1. INTRODUCTION

It has long been recognized that a number of important matrix problems are quite closely related, in the sense that up to polynomial changes in the size of the matrices and constant factor changes in the running time of the algorithms, the amount of parallel time needed to solve these problems is the same. In particular, it is well known that finding the inverse, the characteristic polynomial, the determinant, the rank, the  $N$ th power, and an LU-factorization of a matrix and the product of a matrix chain, etc., can be reduced to each other, such that all of them have the same parallel computing complexity (cf. [3] and Section 2.4.5 of [10] for proofs of the above claim). In addition to the parallel time complexity, the cost of an algorithm, which is the product of the execution time and the number of processors employed by the algorithm, is widely used to measure the efficiency of the algorithm. Currently the fastest parallel algorithms for these problems (except LU-factorization) run in  $O((\log N)^2)$  time, by using  $O(N^4)$  processors connected by commonly used static networks such as meshes of trees and hypercubes [10]. Since the reduction among these problems may significantly increase the problem size, and hence, the number of processors, the costs are not preserved under these reductions. To keep the processor complexity at  $O(N^4)$ , the best algorithm for LU-factorization has time complexity  $O((\log N)^3)$  on static networks.

While the problem of finding tight time and cost bounds of each of these problems remains open, answering the following two questions appears to be challenging:

Q1. Can the parallel time complexities of these problems (except LU-factorization) be reduced to  $o((\log N)^2)$  and to  $o((\log N)^3)$  for LU-factorization?

Q2. Can the costs of parallel algorithms for these problems be reduced to  $o(N^4)$ ?

In answering question Q1, we notice that the major hurdle is that all algorithms for these problems ultimately use a matrix multiplication algorithm as a subroutine, and the best matrix multiplication algorithm has time complexity  $O(\log N)$ , even on theoretical models like parallel random access machines. The parallelism in matrix multiplication can be explored to such extent that the  $N^2$  vector products can be calculated simultaneously. The  $O(\log N)$  bottleneck is due to the summation of the  $N$  values in an inner vector product, which cannot be done any faster on existing computing models. Theoretically, the answer to question Q2 is affirmative. It is well known that there exist matrix multiplication algorithms on PRAMs that run in  $O(\log N)$  time by using  $O(N^\beta)$  processors, where  $O(N^\beta)$ ,  $2 < \beta < 3$ , is the time complexity of the best sequential algorithm for matrix multiplication [21]. The fact that the known smallest value of  $\beta$  is less than 2.3755 [4] implies that the costs of parallel PRAM algorithms for many matrix problems are less than  $O(N^4)$ . However, such a sequential algorithm is quite sophisticated, and its implementation is not considered practical even on sequential machines. Consequently, its parallelization on realistic parallel systems is far from feasible.

Recently, there have been significant advances in optical interconnections. Fiber optic communication technologies offer a combination of high bandwidth, predictable message delay, low interference and error probability, and gigabit transmission capacity. Based on the characteristics of fiber optical communications, a number of

researchers have proposed using optical interconnections to connect processors in a parallel computer system [1, 2, 6, 11, 18, 23, 26, 27]. In such a system, messages can be transmitted concurrently on a pipelined optical bus, by taking the advantages of unidirectional message transmission and predictable propagation delay. It is now feasible to integrate both optical message communication and electronic data computation in massively parallel processing systems. Many parallel algorithms from numerous application domains have been proposed for systems with optical interconnections [7, 12–17, 19, 22, 24].

It is clear that optical buses have created an entirely new parallel computing model and opened up a broader avenue of parallel algorithm design. Consider a linear array with a reconfigurable pipelined bus system (LARPBS) [13, 18, 19]. First, pipelined optical buses can support concurrent accesses by many processors in a single bus cycle and are able to transmit a large volume of data among processors simultaneously for various communication patterns. Second, an LARPBS can also be reconfigured into disjoint subsystems, each being an independent LARPBS of smaller size. This feature supports parallel implementation of divide-and-conquer computations. These capabilities have led to parallelization of nontrivial algorithms, especially the following result.

R0. Strassen's algorithm [25] for multiplying two  $N \times N$  matrices can be implemented on an LARPBS with  $O(N^{2.8074})$  processors in  $O(\log N)$  time [13].

Such an implementation is very difficult (if not impossible) on existing networks with electronic interconnections or systems with electronic buses. In addition to communication channels with tremendous capacity and flexibility, optical buses can serve as active computing agents. One excellent example is that  $N$  values can be added up in a constant number of bus cycles on linear arrays with a reconfigurable pipelined bus system, where computations are essentially done by appropriate timing of optical signals. This implies that

R1. Multiplying two  $N \times N$  matrices can be done on an LARPBS in  $O(1)$  time, using  $N^3$  processors [13].

Combining R0 and R1, we know that

R2. Multiplying two  $N \times N$  matrices can be done on an LARPBS in sub-logarithmic time  $O((\log N)^\delta)$ , using  $O(N^3 / (\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^3)$  for  $0 < \delta \leq 1$  [13].

With results R1 and R2, we are able to give affirmative answers to questions Q1 and Q2 on realistic parallel computing systems.

In this paper, we consider solving 17 fundamental matrix problems on LARPBS. These problems and the time and processor complexities of our solutions are listed in Table 1. Problems (1)–(7) are basic matrix operations, whose algorithms are repeatedly used in other algorithms. Problems (8)–(17) include a number of fundamental matrix operations. Our results provide the following answers to questions Q1 and Q2.

A1. Problems (6)–(15) can all be solved in  $O(\log N)$  time, and problems (16)–(17) in  $O((\log N)^2)$  time, using no more than  $O(N^4)$  processors. Thus,

**TABLE 1**  
**The Problems and Time/Processor Complexities**

| No.  | Problem                                   | Time complexity          | Processor complexity                     |
|------|---|--------------------------|--|
| (1)  | Matrix transposition                      | $O(1)$                   | $O(N^2)$                                 |
| (2)  | Vector chain addition                     | $O(1)$                   | $O(N^2)$                                 |
| (3)  | Matrix chain addition                     | $O(1)$                   | $O(N^3)$                                 |
| (4)  | Matrix-vector multiplication              | $O(1)$                   | $O(N^2)$                                 |
| (5)  | Matrix multiplication                     | $O((\log N)^\delta)$     | $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$ |
| (6)  | Matrix powers                             | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (7)  | Matrix chain product                      | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (8)  | Lower triangular matrix inversion         | $O((\log N)^{1+\delta})$ | $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$ |
| (9)  | Upper triangular matrix inversion         | $O((\log N)^{1+\delta})$ | $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$ |
| (10) | The characteristic polynomial of a matrix | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (11) | The determinant of a matrix               | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (12) | The rank of a matrix                      | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (13) | Arbitrary matrix inversion                | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (14) | Solving linear systems of equations       | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (15) | Krylov matrix                             | $O((\log N)^{1+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (16) | LU-factorization of a matrix              | $O((\log N)^{2+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |
| (17) | QR-factorization of a matrix              | $O((\log N)^{2+\delta})$ | $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ |

compared with known results, the running time of parallel solutions to all these problems mentioned in Table 1 can be reduced by a factor of  $O(\log N)$  using these algorithms.

A2. Problems (6)–(15) can be solved in  $o((\log N)^2)$  time, and problems (16)–(17) can be solved in  $o((\log N)^3)$  time, with cost  $o(N^4)$  on LARPBS.

## 2. THE LARPBS COMPUTING MODEL

A pipelined optical bus system uses optical waveguides instead of electrical wires 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 [2, 11]. 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, multi-casting, global aggregation, and irregular communication patterns.

A linear array with a reconfigurable pipelined bus system (LARPBS) consists of  $N$  processors  $P_1, P_2, \dots, P_N$  connected by an optical bus. In addition to the tremendous communication capabilities, an LARPBS can be partitioned into  $k \geq 2$  independent subarrays LARPBS<sub>1</sub>, LARPBS<sub>2</sub>, ..., LARPBS<sub>k</sub>, such that LARPBS<sub>j</sub> contains processors  $P_{i_{j-1}+1}, P_{i_{j-1}+2}, \dots, P_{i_j}$ , where  $0 = i_0 < i_1 < i_2 \dots < i_k = N$ . 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 [13, 18] for an elaborated exposition and [27] for similar reconfigurable pipelined optical bus architectures).

As in many other synchronous parallel computing systems, an LARPBS computation 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. This complexity measure implies that a bus cycle takes constant time, and this assumption has been adopted widely in the literature. (Remark: To avoid controversy, let us emphasize that in this paper, by “ $O(f(p))$  time” we mean  $O(f(p))$  bus cycles for global communication plus  $O(f(p))$  number of local arithmetic/logic operations.)

For ease of algorithm development and specification, a number of basic communication, data movement, and global operations on the LARPBS model implemented using the coincident pulse processor addressing technique [2, 11, 23] have been developed [13, 18]. Each of these primitive operations can be performed in a constant number of bus cycles. These powerful primitives that support massive parallel communications, plus the reconfigurability of the LARPBS model, make the LARPBS very attractive in solving problems that are both computation and communication intensive, such as matrix manipulations. Optical buses are not only communication channels among the processors, but also active components and agents of certain computations, e.g., global data aggregations. The following primitive operations on LARPBS are used in this paper, and our algorithms are developed using these operations as building blocks.

*One-to-one communication.* Assume that processors  $P_{i_1}, P_{i_2}, \dots, P_{i_m}$  are senders, and processors  $P_{j_1}, P_{j_2}, \dots, P_{j_m}$  are receivers. In particular, processor  $P_{i_k}$  sends a value in its register  $R(i_k)$  to the register  $R(j_k)$  in  $P_{j_k}$ . The operation is represented as

```
for  $1 \leq k \leq m$  do in parallel
   $R(j_k) \leftarrow R(i_k)$ 
endfor
```

(Note that we use  $R(i)$  to denote both the name and the content of register  $R(i)$ .)

*Broadcasting.* Here, we have a source processor  $P_i$ , which sends a value in its register  $R(i)$  to all the  $N$  processors:

$$R(1), R(2), R(3), \dots, R(N) \leftarrow R(i)$$

*Multiple multicasting.* In a multicasting operation, we have a source processor  $P_i$ , which sends a value in its register  $R(i)$  to a subset of the  $N$  processors  $P_{j_1}, P_{j_2}, \dots, P_{j_m}$ ,

$$R(j_1), R(j_2), \dots, R(j_m) \leftarrow R(i)$$

Assume that we have  $g$  disjoint groups of destination processors,  $G_k = \{P_{j_{k,1}}, P_{j_{k,2}}, P_{j_{k,3}}, \dots\}$ ,  $1 \leq k \leq g$ , and there are  $g$  senders  $P_{i_1}, P_{i_2}, \dots, P_{i_g}$ . Processor  $P_{i_k}$  has value  $R(i_k)$  to be broadcast to all the processors in  $G_k$ , where  $1 \leq k \leq g$ . Since there are  $g$  simultaneous multicasting, we have a multiple multicasting operation, which is denoted as

```
for  $1 \leq k \leq g$  do in parallel
     $R(j_{k,1}), R(j_{k,2}), R(j_{k,3}), \dots \leftarrow R(i_k)$ 
endfor
```

It is important to point out that the  $g$  groups of destination processors must be disjoint.

*Element pair-wise operations.* Let  $P_i, P_{i+1}, \dots, P_j$  be a consecutive group of processors. We use  $R[i..j]$  as an abbreviation of the registers  $R(i), R(i+1), \dots, R(j)$ .  $R[i..j]$  can be used to store a vector, or a matrix in the row-major or column-major order. Assume that  $A$  and  $B$  are two matrices of size  $N$ . All array elements are in a domain with a binary operator  $\oplus$ . Elements  $a_{ij}$  and  $b_{ij}$  are stored in  $R[m + (i-1)N + j - 1]$  and  $R[n + (i-1)N + j - 1]$ , respectively. Then  $C = A \oplus B$ , where  $c_{ij} = a_{ij} \oplus b_{ij}$ , can be done as follows, where  $c_{ij}$  is found in  $R[m + (i-1)N + j - 1]$ :

```
for  $1 \leq k \leq N^2$  do in parallel
     $R(m+k-1) \leftarrow R(m+k-1) \oplus R(n+k-1)$ 
endfor
```

*Global aggregation.* Assume that every processor  $P_i$ , where  $1 \leq i \leq N$ , has a register  $R(i)$  which holds a value. We need to calculate  $R(1) + R(2) + \dots + R(N)$ , and save the result in  $R(1)$ . The operation is represented as

$$R(1) \leftarrow R(1) + R(2) + \dots + R(N)$$

The following results have been proven in [13, 18].

**THEOREM 0.** *One-to-one communication, broadcasting, multiple multicasting, element pair-wise operation, integer (of bounded magnitude) aggregation, and real value (of bounded precision and magnitude) aggregation all take  $O(1)$  bus cycles in the LARPBS computing model.*

The extension of global aggregation to unbounded values has been discussed in [13]. In particular, it was shown that the summation of  $N$  integers can be calculated in  $O(\log \log M)$  time, using  $O(N \log M / \log \log M)$  processors in the LARPBS computing model, where  $M$  is the maximum magnitude, and the summation of  $N$  real values, with precision up to  $2^{-(p+q-1)}$  and magnitude on the order of  $M = 2^{2^q}$ , can be calculated in  $O(\log \log M + \log p)$  time, using  $O(Np \log p)$  processors in the LARPBS computing model. Since all real machines have finite word length, which implies bounded magnitude and precision, we will not consider unbounded magnitude and precision in this paper.

The primitive operations can be directly used for some simple matrix manipulations. For instance, the one-to-one communication can be used for transposing a matrix. Assume that we have an LARPBS with  $N^2$  processors  $P_1, P_2, \dots, P_{N^2}$ , and each processor has a register  $A(k)$ , where  $1 \leq k \leq N^2$ . A matrix  $A = (a_{ij})_{N \times N}$  is stored in the linear array in the row-major order; that is,  $A((i-1)N + j) = a_{ij}$ , for all  $1 \leq i, j \leq N$ . Then, our algorithm for matrix transposition simply does the following:

```

for  $1 \leq i, j \leq N$  do in parallel
     $A((i-1)N + j) \leftarrow A((j-1)N + i)$ 
endfor

```

After the above one-to-one communication, which takes one bus cycle, we have  $A((i-1)N + j) = a_{ji}$ , for all  $1 \leq i, j \leq N$ .

**THEOREM 1.** *Matrix transposition can be done in  $O(1)$  time using  $N^2$  processors.*

The aggregation operation can be used to calculate the summations of  $N$  vectors or  $N$  matrices. Given  $N$   $N$ -dimensional vectors  $\mathbf{v}_i = (v_{i1}, v_{i2}, \dots, v_{iN})$ , where  $1 \leq i \leq N$ , the vector chain addition problem is to calculate  $\mathbf{v} = (v_1, v_2, \dots, v_N)$ , where  $v_j = v_{1j} + v_{2j} + \dots + v_{Nj}$ , for all  $1 \leq j \leq N$ . Assume that the  $N$  vectors are stored in  $N^2$  processors, where each processor has a register  $V(k)$ ,  $1 \leq k \leq N^2$ , such that  $\mathbf{v}_i$  is scattered in  $V(i), V(i+N), \dots, V(i+N^2-N)$ ; that is,  $V(i+(j-1)N) = v_{ij}$ , for all  $1 \leq i, j \leq N$ . We first configure the LARPBS into  $N$  subarrays LARPBS $_j$ , such that LARPBS $_j$  contains processor  $P_{(j-1)N+1}, P_{(j-1)N+2}, \dots, P_{jN}$ , where  $1 \leq j \leq N$ . The  $N$  processors in LARPBS $_j$  compute  $v_j$  by invoking the aggregation operation and save the result  $v_j$  in  $V((j-1)N+1)$ . Then, all the  $v_j$ 's are moved to  $V(1), V(2), \dots, V(N)$  by a one-to-one communication operation. The complete vector chain addition algorithm is described below:

```

for  $1 \leq j \leq N$  do in parallel
     $V((j-1)N + 1) \leftarrow V((j-1)N + 1) + V((j-1)N + 2) + \dots + V(jN)$ 
endfor
for  $1 \leq j \leq N$  do in parallel
     $V(j) \leftarrow V((j-1)N + 1)$ 
endfor

```

It is clear that the above algorithm requires two bus cycles.

**THEOREM 2.** *Vector chain addition can be done in  $O(1)$  time using  $N^2$  processors.*

Given  $N$  matrices  $A_1, A_2, \dots, A_N$ , where  $A_k = (a_{ij}^{(k)})_{N \times N}$ , the matrix chain addition problem is to calculate  $A = (a_{ij})_{N \times N}$ , where  $a_{ij} = a_{ij}^{(1)} + a_{ij}^{(2)} + \dots + a_{ij}^{(N)}$ , for all  $1 \leq i, j \leq N$ . Assume that we have  $N^3$  processors, and matrix  $A_k$  is put into  $P_{(k-1)N^2+1}, P_{(k-1)N^2+2}, \dots, P_{kN^2}$ , in the row-major order, where  $1 \leq k \leq N$ ; that is,  $A((k-1)N^2 + (i-1)N + j) = a_{ij}^{(k)}$ , for all  $1 \leq i, j, k \leq N$ . We first rearrange the data via a one-to-one communication, such that all  $a_{ij}^{(1)}, a_{ij}^{(2)}, \dots, a_{ij}^{(N)}$  are packed together. We then reconfigure the LARPBS into  $N^2$  subarrays LARPBS $_{ij}$ , such that

LARPBS<sub>*ij*</sub> contains processor  $P_{((i-1)N+(j-1))N+k}$ ,  $1 \leq k \leq N$ , and such that  $A(((i-1)N+(j-1))N+k) = a_{ij}^{(k)}$ , where  $1 \leq i, j, k \leq N$ . LARPBS<sub>*ij*</sub> is used to calculate  $a_{ij}$  by aggregating  $(a_{ij}^{(1)}, a_{ij}^{(2)}, \dots, a_{ij}^{(N)})$ . Finally, all the  $a_{ij}$ 's are moved to the first  $N^2$  processors. The matrix chain addition algorithm, which takes the three bus cycles, is given as follows:

```

for  $1 \leq i, j, k \leq N$  do in parallel
   $A(((i-1)N+(j-1))N+k) \leftarrow A((k-1)N^2+(i-1)N+j)$ 
endfor
for  $1 \leq i, j \leq N$  do in parallel
   $A(((i-1)N^2+(j-1))N+1) \leftarrow \sum_{k=1}^N A(((i-1)N^2+(j-1))N+k)$ 
endfor
for  $1 \leq j \leq N$  do in parallel
   $A((i-1)N+j) \leftarrow A(((i-1)N^2+(j-1))N+1)$ 
endfor

```

**THEOREM 3.** *Matrix chain addition can be done in  $O(1)$  time using  $N^3$  processors.*

In the remainder of the paper, we will show that many matrix problems can be solved using the primitive operations defined in this section and the known algorithms for other matrix problems as subroutines. It seems that the description of these algorithms will be quite tedious if we insist on specifications down to the basic operation level, which is clearly unnecessary. Thus, we will outline the algorithms, provide as many details as possible, and justify their time and processor complexities.

### 3. MATRIX-VECTOR AND MATRIX MULTIPLICATION

Assume that we have an  $N \times N$  matrix  $A = (a_{ij})$  and an  $N$ -dimensional vector  $\mathbf{v} = (v_1, v_2, \dots, v_N)$ . Matrix  $A$  is stored in the row-major order in processors  $P_1, P_2, \dots, P_{N^2}$ , such that  $A((i-1)N+j) = a_{ij}$ , for all  $1 \leq i, j \leq N$ . Vector  $\mathbf{v}$  is put in processors  $P_1, P_2, \dots, P_N$ , such that  $V(j) = v_j$ , for all  $1 \leq j \leq N$ . The  $N^2$  processors will be reconfigured into  $N$  subarrays LARPBS<sub>*i*</sub>,  $1 \leq i \leq N$ , which contain processors  $P_{(i-1)N+j}$ ,  $1 \leq j \leq N$ . To calculate the matrix-vector product  $A\mathbf{v} = \mathbf{x} = (x_1, x_2, \dots, x_N)$ , our algorithm performs the following steps:

- First, the vector  $\mathbf{v}$  is made available to all subarrays LARPBS<sub>*i*</sub>,  $1 \leq i \leq N$ , via multiple multicasting in one bus cycle.
- Second, processor  $P_{(i-1)N+j}$  calculates  $a_{ij}v_j$  locally, where  $1 \leq i, j \leq N$ , in constant time.
- Third, LARPBS<sub>*i*</sub>,  $1 \leq i \leq N$ , aggregates  $x_i = a_{i1}v_1 + a_{i2}v_2 + \dots + a_{iN}v_N$  in one bus cycle.
- Finally, the  $x_i$ 's are moved to processors  $P_i$ ,  $1 \leq i \leq N$ , using a one-to-one communication.

Clearly, we can have the following claim.

**THEOREM 4.** *Matrix-vector product can be calculated in  $O(1)$  time, by using  $N^2$  processors.*

Matrix multiplication is perhaps the most important subproblem in many other matrix manipulations. A number of parallel matrix multiplication algorithms have been developed on a linear array with a reconfigurable pipelined bus system. In [13], it is shown that matrix multiplication can be performed

- in  $O(N)$  time by using  $N^2$  processors;
- in  $O(1)$  time by using  $N^3$  processors;
- in  $O(\log N)$  time by using  $O(N^{2.8074})$  processors;
- in  $O(\log N + N^{1-\varepsilon})$  time by using  $O(N^{2+0.8074\varepsilon})$  processors,  $0 \leq \varepsilon \leq 1$ ;

and most noteworthy, the following result, which leads to performance enhancement for many matrix manipulation algorithms.

**THEOREM 5.** *There is a matrix multiplication algorithm  $MM_\delta$  on LARPBS, which runs in  $O((\log N)^\delta)$  time, by using  $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost of  $MM_\delta$  is  $o(N^3)$  for  $0 < \delta \leq 1$ .*

Now, let us consider how to calculate the powers  $A, A^2, A^3, \dots, A^N$  of an  $N \times N$  matrix  $A$ . We require an LARPBS with  $N\rho(N, \delta)$  processors, where

$$\rho(N, \delta) = O\left(\frac{N^3}{1.1428^{(\log N)^\delta}}\right) \quad (1)$$

is the number of processors used in the matrix multiplication algorithm  $MM_\delta$ . The LARPBS will be reconfigured into  $N$  subarrays LARPBS $_q$ ,  $1 \leq q \leq N$ , where LARPBS $_q$  consists of processors  $P_{(q-1)\rho(N, \delta)+k}$ , for all  $1 \leq k \leq \rho(N, \delta)$ . Array  $A$  is initially held by the first  $N^2$  processors of LARPBS $_1$ . Without loss of generality, we assume that  $N = 2^n$  is a power of 2. Our algorithm, which calculates the first  $N$  powers of  $A$ , performs the following two major steps.

- First, LARPBS $_1$  computes the powers  $A^{2^1}, A^{2^2}, \dots, A^{2^{n-1}}$ , one by one, such that  $A^{2^d} = A^{2^{d-1}} \times A^{2^{d-1}}$  is obtained by invoking algorithm  $MM_\delta$ , where  $1 \leq d \leq n-1$ . All these results are also sent to all LARPBS $_q$ ,  $1 \leq q \leq N$ , using the multiple multicasting operation.

- Second, if  $q = c_{n-1}2^{n-1} + c_{n-2}2^{n-2} + \dots + c_02^0$ , then LARPBS $_q$  calculates  $A^q = M_{n-1} \times M_{n-2} \times \dots \times M_0$ , where  $M_d = A^{2^d}$  if  $c_d \neq 0$ , and  $M_d = I_N$  if  $c_d = 0$ , and  $I_N$  is the  $N \times N$  identity matrix.

Notice that both steps essentially perform  $n-1 = O(\log N)$  matrix multiplications. Thus, the overall time and processor complexities depend on those of matrix multiplication.

**THEOREM 6.** *The first  $N$  powers  $A, A^2, A^3, \dots, A^N$  of a matrix  $A$  can be calculated in  $O((\log N)^{1+\delta})$  time, by using  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

Given  $N$  matrices  $A_1, A_2, \dots, A_N$ , where  $A_k = (a_{ij}^{(k)})_{N \times N}$ , the matrix chain product problem is to calculate  $A = A_1 \times A_2 \times \dots \times A_N$ . Our algorithm for obtaining  $A$  is actually the standard binary tree method, and the overall running time is a factor of  $O(\log N)$  more than that of matrix multiplication. Thus, similar to Theorem 6, we have

**THEOREM 7.** *A matrix chain product  $A = A_1 \times A_2 \times \dots \times A_N$  can be obtained in  $O((\log N)^{1+\delta})$  time, using  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

#### 4. INVERSION OF LOWER AND UPPER TRIANGULAR MATRICES

Let  $A$  be an  $N \times N$  lower triangular matrix that is invertible; i.e., all the elements on  $A$ 's main diagonal are nonzeros. We partition  $A$  into four submatrices of equal size  $N/2 \times N/2$ ,

$$A = \begin{bmatrix} A_1 & 0 \\ A_3 & A_2 \end{bmatrix}.$$

Since  $A_1$  and  $A_2$  are also invertible lower triangular matrices, we have

$$A^{-1} = \begin{bmatrix} A_1^{-1} & 0 \\ -A_2^{-1}A_3A_1^{-1} & A_2^{-1} \end{bmatrix}.$$

Therefore,  $A^{-1}$  can be obtained by inverting  $A_1$  and  $A_2$  recursively and then multiplying  $A_1^{-1}$  and  $A_2^{-1}$  by  $A_3$ . Similarly, if  $A$  is an  $N \times N$  upper triangular matrix that is invertible, we partition  $A$  into four submatrices of equal size  $N/2 \times N/2$ ,

$$A = \begin{bmatrix} A_1 & A_3 \\ 0 & A_2 \end{bmatrix}.$$

Thus,  $A_1$  and  $A_2$  are also invertible upper triangular matrices, and

$$A^{-1} = \begin{bmatrix} A_1^{-1} & -A_1^{-1}A_3A_2^{-1} \\ 0 & A_2^{-1} \end{bmatrix}.$$

The above discussion yields the following method for inverting a lower (upper) triangular matrix.

**A METHOD FOR LOWER (UPPER) TRIANGULAR MATRIX INVERSION.** (1) Recursively calculate  $A_1^{-1}$  and  $A_2^{-1}$ .

(2) Compute  $-A_2^{-1}A_3A_1^{-1}(-A_1^{-1}A_3A_2^{-1})$  by using the matrix multiplication algorithm MM.

This method reduces the lower/upper triangular matrix inversion problem to matrix multiplication. Without loss of generality, we assume that  $N = 2^n$  is a power of 2. The recursion can be unwound into  $n + 1$  iterations such that a sequence of

matrices  $A_0, A_1, A_2, \dots, A_n$  is calculated. Initially,  $A_0$  is obtained from  $A$  by inverting the elements on the main diagonal. This is the base of the recursion. In general,  $A_k$  is obtained from  $A_{k-1}$  by further calculating  $2^{n-k}$  submatrices of size  $2^{k-1}$ , where  $1 \leq k \leq n$ . Finally,  $A_n = A^{-1}$ . To calculate  $A_k$ , a linear array is reconfigured into  $2^{n-k}$  subarrays for  $2^{n-k}$  simultaneous invocation of submatrix multiplications. To this end, certain data movements are required such that the entries of a submatrix are packed together. Fortunately, the communication patterns for this purpose are quite regular.

It is clear that to calculate  $A_k$ ,  $1 \leq k \leq n$ , there are  $2^{n-k}$  submatrices of size  $2^{k-1}$  that can be computed in parallel, and each requires only two matrix multiplications. The time complexity is

$$O\left(\sum_{k=1}^n (k-1)^\delta\right) = O(n^{\delta+1}) = O((\log N)^{\delta+1}).$$

The total number of processors required is

$$2^{n-k} \rho(2^{k-1}, \delta) = 2^{n-k} O\left(\frac{2^{3k-3}}{1.1428^{(k-1)^\delta}}\right) = O\left(\frac{N^3}{1.1428^{(\log N-1)^\delta}}\right),$$

where  $\rho(2^{k-1}, \delta)$  is defined in Eq. (1). Thus, we have

**THEOREM 8.** *The inverse of a lower triangular matrix can be obtained in  $O((\log N)^{1+\delta})$  time, with number of processors  $O(N^3/(\frac{8}{7})^{(\log N-1)^\delta})$ , for  $0 \leq \delta \leq 1$ . The cost is  $o(N^3)$  for  $0 < \delta \leq 1$ .*

Similarly, we have

**THEOREM 9.** *The inverse of an upper triangular matrix can be obtained in  $O((\log N)^{1+\delta})$  time, with number of processors  $O(N^3/(\frac{8}{7})^{(\log N-1)^\delta})$ , for  $0 \leq \delta \leq 1$ . The cost is  $o(N^3)$  for  $0 < \delta \leq 1$ .*

## 5. DETERMINANTS, CHARACTERISTIC POLYNOMIALS, AND RANKS

We use  $\det(A)$  to denote the determinant of a matrix  $A = (a_{ij})_{N \times N}$ . The characteristic polynomial of a matrix  $A$  is represented as

$$\phi_A(\lambda) = \det(\lambda I_N - A) = \lambda^N + c_1 \lambda^{N-1} + c_2 \lambda^{N-2} + \dots + c_{N-1} \lambda + c_N,$$

where  $I_N$  is the  $N \times N$  identity matrix. The trace  $\text{tr}(A)$  of a matrix  $A = (a_{ij})_{N \times N}$  is the sum of the entries on  $A$ 's main diagonal; i.e.,  $\text{tr}(A) = a_{11} + a_{22} + \dots + a_{NN}$ .

The following classical result [9] is the basis of a parallel algorithm for obtaining  $\phi_A(\lambda)$  and  $\det(A)$ .

**LEVERRIER'S LEMMA.** The coefficients  $c_1, c_2, \dots, c_N$  of the characteristic polynomial of a matrix  $A$  satisfy  $S(c_1, c_2, \dots, c_N)^T = (s_1, s_2, \dots, s_N)^T$ , where

$$S = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ s_1 & 2 & 0 & \cdots & 0 & 0 \\ s_2 & s_1 & 3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ s_{N-2} & s_{N-3} & s_{N-4} & \cdots & N-1 & 0 \\ s_{N-1} & s_{N-2} & s_{N-3} & \cdots & s_1 & N \end{bmatrix},$$

and  $s_k = \text{tr}(A^k)$ , for all  $1 \leq k \leq N$ . ■

Based on Leverrier's lemma, Csanky devised the following method for calculating the characteristic polynomial of a matrix  $A$  [5]. Since  $\phi_A(0) = \det(-A) = c_N$ , i.e.,  $\det(A) = (-1)^N c_N$ , this algorithm can also be used to calculate  $\det(A)$ .

**CSANKY'S STRATEGY FOR CHARACTERISTIC POLYNOMIAL.** (1) Calculate  $A^2, A^3, A^4, \dots, A^N$ .

(2) Calculate  $s_k = \text{tr}(A^k)$ , for all  $1 \leq k \leq N$ .

(3) Find  $S^{-1}$ , where  $S$  is the lower triangular matrix in Leverrier's lemma.

(4) Calculate the matrix-vector product  $S^{-1}[s_1, s_2, \dots, s_N]^T$  to obtain  $c_1, c_2, \dots, c_N$ .

Step (1) involves the calculation of the first  $N$  powers of  $A$  (Theorem 6). Step (2) can be implemented using the aggregation operation, plus certain data movements. Step (3) invokes the lower triangular matrix inversion algorithm (Theorem 8). Finally, Step (4) is a matrix-vector multiplication (Theorem 4). It is clear that the time and processor complexities of Csanky's method are dominated by Step (1).

**THEOREM 10.** *The characteristic polynomial of a matrix can be obtained in  $O((\log N)^{1+\delta})$  time, using  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

**THEOREM 11.** *The determinant of a matrix can be calculated in  $O((\log N)^{1+\delta})$  time, and the number of processors is  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ , where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

The rank of a matrix  $A$ ,  $\text{rank}(A)$ , is the number of nonzero rows (or columns) in the row-reduced (or column-reduced) echelon form of  $A$ . It is well known that  $\text{rank}(A) = \text{rank}(A^T A)$ , where  $A^T$  is the transpose of  $A$  (or conjugate transpose of  $A$  for complex matrices), and  $A^T A$  is similar to a diagonal matrix whose elements are the roots of the characteristic polynomial  $\phi_{A^T A}(\lambda)$ . Therefore,  $\text{rank}(A)$  is the number of nonzero roots of  $\phi_{A^T A}(\lambda)$ . This leads to the following algorithm for finding  $\text{rank}(A)$  [8].

**AN ALGORITHM FOR CALCULATING MATRIX RANK.** (1) Get the matrix  $A^T A$ .

(2) Calculate  $\phi_{A^T A}(\lambda) = c_0 \lambda^N + c_1 \lambda^{N-1} + c_2 \lambda^{N-2} + \cdots + c_{N-1} \lambda + c_N$ .

(3) Find  $\text{rank}(A) = N - i$ , where  $i$ ,  $0 \leq i \leq N$ , is the largest integer such that  $c_{N-i} \neq 0$  and  $c_{N-i+1} = c_{N-i+2} = \cdots = c_N = 0$ .

In the above algorithm, Step (1) performs a matrix transposition (Theorem 1) and a matrix multiplication (Theorem 5). Step (2) invokes Csanky's method for computing characteristic polynomial (Theorem 10). Step (3) can be implemented in a few bus cycles (i.e., constant time) by simple data testing, comparison, and movement.

**THEOREM 12.** *The rank of a matrix can be found in  $O((\log N)^{1+\delta})$  time, and the number of processors is  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ , where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

## 6. INVERSION OF ARBITRARY MATRICES

Inverting an arbitrary matrix  $A$  is closely related to the calculation of the characteristic polynomial  $\phi_A(\lambda)$ , as revealed by the following well-known theorem from linear algebra.

**CAYLEY-HAMILTON THEOREM.** *Let  $\phi_A(\lambda) = \lambda^N + c_1\lambda^{N-1} + c_2\lambda^{N-2} + \dots + c_{N-1}\lambda + c_N$  be the characteristic polynomial of matrix  $A$ . Then*

$$\phi_A(A) = A^N + c_1A^{N-1} + c_2A^{N-2} + \dots + c_{N-1}A + c_NA^0$$

*is the  $N \times N$  zero matrix.*

The Cayley-Hamilton theorem implies that

$$A(A^{N-1} + c_1A^{N-2} + c_2A^{N-3} + \dots + c_{N-1}I_N) = -c_NI_N.$$

Hence, the inverse of a matrix  $A$  can be calculated using the equation,

$$A^{-1} = -\frac{1}{c_N}(A^{N-1} + c_1A^{N-2} + c_2A^{N-3} + \dots + c_{N-2}A + c_{N-1}I_N). \quad (2)$$

Csanky's method for calculating matrix inversion can be described as follows [5].

**CSANKY'S STRATEGY FOR MATRIX INVERSION.** (1) Calculate the characteristic polynomial of  $A$ , that is,  $c_1, c_2, \dots, c_N$ .

(2) Compute  $A^{-1}$  by using the identity in Eq. (2).

The time and processor complexities of Step (1) are given in Theorem 10. Step (2) involves the computation of the first  $N$  powers of  $A$  (Theorem 6) and matrix chain addition (Theorem 3). Hence, we have the following result.

**THEOREM 13.** *The inverse of a matrix can be calculated in  $O((\log N)^{1+\delta})$  time, and the number of processors is  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ , where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

## 7. LINEAR SYSTEMS OF EQUATIONS

Let  $A$  be a nonsingular  $N \times N$  matrix  $(a_{ij})_{N \times N}$  and let  $\mathbf{b} = (b_1, b_2, \dots, b_N)^T$  be an  $N$ -dimensional vector. The problem of solving a linear system of equations is to find

a vector  $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$  such that  $A\mathbf{x} = \mathbf{b}$ . It is clear that  $\mathbf{x} = A^{-1}\mathbf{b}$ ; i.e., solving the above linear system of equations can be accomplished by a matrix inversion (Theorem 13) and a matrix–vector multiplication (Theorem 4).

**THEOREM 14.** *A linear system of equations can be solved in  $O((\log N)^{1+\delta})$  time, and the number of processors is  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$ , where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

Let  $\mathbf{k}_j = A^j\mathbf{b}$ , where  $0 \leq j \leq N-1$ . Then, the matrix  $K(A, \mathbf{b}, N) = [\mathbf{k}_0, \mathbf{k}_1, \mathbf{k}_2, \dots, \mathbf{k}_{N-1}] = [\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \dots, A^{N-1}\mathbf{b}]$  is called the Krylov matrix defined by the matrix  $A$ , the vector  $\mathbf{b}$ , and the integer  $N$ . By the Cayley–Hamilton theorem, we have  $A(A^{N-1}\mathbf{b} + c_1A^{N-2}\mathbf{b} + c_2A^{N-3}\mathbf{b} + \dots + c_{N-1}\mathbf{b}) = -c_N\mathbf{b}$ , which implies that

$$\mathbf{x} = -\left(\frac{1}{c_N}\right)\mathbf{k}_{N-1} - \left(\frac{c_1}{c_N}\right)\mathbf{k}_{N-2} - \left(\frac{c_2}{c_N}\right)\mathbf{k}_{N-3} - \dots - \left(\frac{c_{N-1}}{c_N}\right)\mathbf{k}_0.$$

In other words,  $\mathbf{x}$  is a linear combination of the column vectors of the Krylov matrix  $K(A, \mathbf{b}, N)$ . Thus, we can first calculate the first  $N$  powers of  $A$  (Theorem 6) and then compute the Krylov matrix  $K(A, \mathbf{b}, N)$  using matrix–vector multiplication (Theorem 4). Once  $K(A, \mathbf{b}, N)$  is available,  $\mathbf{x}$  can be obtained via vector chain addition in constant time (Theorem 2). This proves the following result.

**THEOREM 15.** *The Krylov matrix defined by the matrix  $A$ , the vector  $\mathbf{b}$ , and the integer  $N$  can be calculated, and hence, the linear system of equations  $A\mathbf{x} = \mathbf{b}$  can be solved, in  $O((\log N)^{1+\delta})$  time, using  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

## 8. LU- AND QR-FACTORIZATIONS

The LU-factors of matrix  $A$  contain a nonsingular lower triangular matrix  $L$  and a nonsingular upper triangular matrix  $U$  such that  $A = LU$ . Suppose that the LU-factors of  $A$  exist. We divide  $A$ ,  $L$ , and  $U$  into  $N/2 \times N/2$  blocks as

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \times \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}.$$

The above identity implies that  $A_{11} = L_{11}U_{11}$ ,  $A_{12} = L_{11}U_{12}$ ,  $A_{21} = L_{21}U_{11}$ ,  $A_{22} = L_{21}U_{12} + L_{22}U_{22}$ . The following method for LU-factorization is due to Victor Pan [20].

- PAN'S METHOD FOR LU-FACTORIZATION.**
- (1) Compute  $A_{11}^{-1}$ .
  - (2) Calculate  $X_1 = A_{11}^{-1}A_{12}$ ,  $X_2 = A_{21}A_{11}^{-1}$ , and  $X_3 = A_{21}A_{11}^{-1}A_{12}$ .
  - (3) Set  $X_4 = A_{22} - X_3$ . (It can be verified that  $X_4 = L_{22}U_{22}$ .)
  - (4) Recursively LU-factorize  $A_{11}$  to get  $L_{11}$  and  $U_{11}$ , and recursively LU-factorize  $X_4$  to get  $L_{22}$  and  $U_{22}$ .
  - (5) Calculate  $L_{21} = X_2L_{11}$ , and  $U_{12} = U_{11}X_1$ .

Let  $T(N)$  be the time complexity of the above algorithm. Then, we have

$$T(N) = T\left(\frac{N}{2}\right) + O((\log N)^{1+\delta}) = O\left(\sum_{l=0}^{\log N} (\log N - l)^{1+\delta}\right),$$

which gives  $T(N) = O((\log N)^{2+\delta})$ . As for number of processors, let us unwind the recursion, such that there are  $2^k$  simultaneous LU-factorizations of matrices with size  $N/2^k$ . We notice that Step (1) requires  $(N/2^k) \rho(N/2^k, \delta)$  processors. Each of Steps (2) and (5) involves at most two parallel matrix multiplications, and hence,  $\rho(N/2^k, \delta)$  processors. Step (3) needs only  $O(N^2)$  processors. Hence, the total number of processors is

$$2^k O\left(\frac{(N/2^k)^4}{1.1428^{(\log(N/2^k))^\delta}}\right) = O(N\rho(N, \delta)). \quad (3)$$

**THEOREM 16.** *LU-factorization of a matrix can be performed in  $O((\log N)^{2+\delta})$  time, using  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$  for  $0 < \delta \leq 1$ .*

The QR-factors of matrix  $A$  contain an orthogonal matrix  $Q$  (that is,  $Q$  satisfies  $Q^T Q = I_N$ ) and a nonsingular upper triangular matrix  $R$  such that  $A = QR$ . It is clear that if  $A = QR$ , then  $A^T A = R^T Q^T QR = R^T R$ . Since  $R$  is a nonsingular upper triangular matrix,  $R^T$  is a nonsingular lower triangular matrix. In other words,  $R^T$  and  $R$  are LU-factors of  $A' = A^T A$ . The above discuss suggests the following method.

**A METHOD FOR QR-FACTORIZATION.** (1) Calculate  $A^T$ .

(2) Compute  $A' = A^T A$ .

(3) LU-factorize  $A'$  to get  $R$ .

(4) Find  $R^{-1}$ .

(5) Calculate  $AR^{-1}$  to obtain  $Q$ .

The time and processor complexities of Steps (1), (2), (4), and (5) are given by Theorems 1, 5, and 13, respectively. Clearly, Step (3), whose complexities are given in Theorem 16, dominates the time and processor complexities of QR-factorization.

**THEOREM 17.** *QR-factorization of a matrix can be performed in  $O((\log N)^{2+\delta})$  time, using  $O(N^4/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^4)$ , for all  $0 < \delta \leq 1$ .*

## 9. ITERATIVE METHODS

The inverse of a matrix can also be obtained using Newton's iterative method [10]. The strategy is to use a sequence of matrices  $X_0, X_1, X_2, \dots$  as approximations to  $A^{-1}$ , where

$$X_k = 2X_{k-1} - X_{k-1}AX_{k-1},$$

for all  $k \geq 1$ . Let  $R_k = I_N - AX_k$  be the residual matrix of  $X_k$ , which measures how far  $X_k$  is from  $A^{-1}$ . It can be verified that  $R_k = R_{k-1}^2$ ; that is,  $R_k = R_0^{2^k}$ , for all  $k \geq 1$ , which means that  $X_k$  converges very rapidly to the zero matrix as long as  $X_0$  is a good initial approximation to  $A^{-1}$ . In particular, if  $\|R_0\|_2 \leq 1 - 1/p(N)$ , where  $p(N)$  is a polynomial of  $N$ , then after  $O(\log N)$  iterations, the first  $N$  bits of every entry of  $A^{-1}$  can be obtained, which is already enough for most applications. It is also known that if

$$X_0 = \frac{1}{\text{tr}(A^T A)} A^T,$$

then

$$\|R_0\|_2 \leq 1 - \frac{1}{N(\|A\|_2 \cdot \|A^{-1}\|_2)^2},$$

where  $\|A\|_2 \cdot \|A^{-1}\|_2$  is the condition number of  $A$ ,  $\|A\|_2$  is a matrix norm defined as

$$\|A\|_2 = \max_{\|\mathbf{x}\|_2 \neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2},$$

and  $\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \dots + x_N^2}$  is the Euclidean norm of a vector  $\mathbf{x} = (x_1, x_2, \dots, x_N)^T$ . It is clear that each iteration only performs matrix multiplication and addition. Thus, the number of processors can be reduced by a factor of  $O(N)$ , compared with Theorem 13.

**THEOREM 13'.** *If the condition number of  $A$  is a polynomial of  $N$ , then  $A^{-1}$  can be calculated in  $O((\log N)^{1+\delta})$  time, using  $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^3)$ , for all  $0 < \delta \leq 1$ .*

Theorem 13' immediately implies the following.

**THEOREM 14'.** *If the condition number of  $A$  is a polynomial of  $N$ , a linear system of equations  $A\mathbf{x} = \mathbf{b}$  can be solved in  $O((\log N)^{1+\delta})$  time, and the number of processors is  $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$ , where  $0 \leq \delta \leq 1$ . The cost is  $o(N^3)$  for  $0 < \delta \leq 1$ .*

Since the time and processor complexities of our LU- and QR-factorization methods are determined by that of matrix inversion, using an argument similar to that in Eq. (3), we know that the number of processors for LU- or QR-factorization can also be reduced by a factor of  $O(N)$  by using the iterative matrix inversion method (Theorem 13').

**THEOREM 16'.** *If the condition number of  $A$  is a polynomial of  $N$ , LU-factorization of a matrix can be performed in  $O((\log N)^{2+\delta})$  time, using  $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^3)$  for  $0 < \delta \leq 1$ .*

**THEOREM 17'.** *If the condition number of  $A$  is a polynomial of  $N$ , QR-factorization of a matrix can be performed in  $O((\log N)^{2+\delta})$  time, using  $O(N^3/(\frac{8}{7})^{(\log N)^\delta})$  processors, where  $0 \leq \delta \leq 1$ . The cost is  $o(N^3)$  for  $0 < \delta \leq 1$ .*

## 10. FINAL REMARKS

In this paper, we present fast and cost-efficient parallel algorithms on linear arrays with reconfigurable pipelined optical bus systems for a number of important and fundamental matrix computation problems. We show that our algorithms provide a wide range of performance–cost combinations, which have not been achieved previously on existing parallel computation models, including theoretical PRAM models. Compared with previously known best parallel algorithms, our algorithms have an  $O(\log N)$  reduction in time, while maintaining their cost below  $o(N^4)$ .

Our results demonstrate that the LARPBS is a very powerful model. In addition to its high communication bandwidth, an LARPBS supports versatile communication patterns, and its communication reconfigurability constitutes an integral part of a parallel computation. These features allow a large degree of parallelism in a computational problem to be exploited (with lower cost than in other systems), which most other machine models cannot achieve.

## ACKNOWLEDGMENTS

Keqin Li's research was supported by the National Aeronautics and Space Administration and the Research Foundation of the State University of New York through the NASA/University Joint Venture in Space Science Program under Grant NAG8-1313. Yi Pan's work was supported in part by the National Science Foundation under Grant CCR-9211621, the Air Force Avionics Laboratory, Wright Laboratory, Dayton, Ohio, under Grant F33615-C-2218, and an Ohio Board of Regents Investment Fund Competition Grant. Si Qing Zheng was partially supported by the National Science Foundation under Grant ECS-9626215 and Louisiana Grant LEQSF (1996-99)-RD-A-16.

## REFERENCES

1. A. F. Benner, H. F. Jordan, and V. P. Heuring, Digital optical computing with optically switched directional couplers, *Opt. Engrg.* **30** (1991), 1936–1941.
2. D. Chiarulli, R. Melhem, and S. Levitan, Using coincident optical pulses for parallel memory addressing, *IEEE Comput.* **30** (1987), 48–57.
3. S. A. Cook, A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64** (1985), 2–22.
4. D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, *J. Symbolic Comput.* **9** (1990), 251–280.
5. L. Csanky, Fast parallel matrix inversion algorithms, *SIAM J. Comput.* **5** (1976), 618–623.
6. Z. Guo, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, Pipelined communications in optically interconnected arrays, *J. Parallel Distrib. Comput.* **12** (1991), 269–282.
7. M. Hamdi and Y. Pan, Efficient parallel algorithms on optically interconnected arrays of processors, in “IEE Proceedings—Computers and Digital Techniques,” Vol. 142, pp. 87–92, 1995.
8. O. H. Ibarra, S. Moran, and L. E. Rosier, A note on the parallel complexity of computing the rank of order  $n$  matrices, *Inform. Process. Lett.* **11**, 4–5 (1980), 162.
9. U. J. J. Le Verrier, Sur les variations seculaires des elements elliptiques des sept planets principales, *J. Math. Pures Appl.* **5** (1840), 220–254.
10. T. Leighton, “Introduction to Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes,” Morgan Kaufmann, San Mateo, CA, 1992.

11. S. Levitan, D. Chiarulli, and R. Melhem, Coincident pulse techniques for multiprocessor interconnection structures, *Appl. Optics* **29** (1990), 2024–2039.
12. K. Li, Constant time boolean matrix multiplication on a linear array with a reconfigurable pipelined bus system, *J. Supercomput.* **11**, 4 (1997), 391–403.
13. 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 Trans. Parallel Distrib. Systems* **9**, 8 (1998), 705–720.
14. K. Li, Y. Pan, and S.-Q. Zheng, Eds., “Parallel Computing Using Optical Interconnections,” Kluwer Academic, Boston, MA, 1998.
15. 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 *J. Supercomput.*
16. Y. Pan and M. Hamdi, Efficient computation of singular value decomposition on arrays with pipelined optical buses, *J. Network Comput. Applications* **19** (1996), 235–248.
17. Y. Pan, M. Hamdi, and K. Li, Efficient and scalable quicksort on a linear array with a reconfigurable pipelined bus system, *Future Generation Comput. Systems* **13**, 6 (1998), 501–513.
18. Y. Pan and K. Li, Linear array with a reconfigurable pipelined bus system—concepts and applications, *J. Inform. Sci.* **106**, 3–4 (1998), 237–258.
19. Y. Pan, K. Li, and S.-Q. Zheng, Fast nearest neighbor algorithms on a linear array with a reconfigurable pipelined bus system, *J. Parallel Algorithms Appl.* **13** (1998), 1–25.
20. V. Pan, Complexity of parallel matrix computations, *Theoret. Comput. Sci.* **54** (1987), 65–85.
21. V. Pan and J. Reif, Efficient parallel solution of linear systems, in “Proc. of the 7th ACM Symposium on Theory of Computing,” pp. 143–152, 1985.
22. S. Pavel and S. G. Akl, Matrix operations using arrays with reconfigurable optical buses, *J. Parallel Algorithms Applications* **8** (1996), 223–242.
23. C. Qiao and R. Melhem, Time-division optical communications in multiprocessor arrays, *IEEE Trans. Comput.* **42** (1993), 577–590.
24. S. Rajasekaran and S. Sahni, Sorting, selection, and routing on the array with reconfigurable optical buses, *IEEE Trans. Parallel Distrib. Systems* **8**, 11 (1997), 1123–1131.
25. V. Strassen, Gaussian elimination is not optimal, *Numer. Math.* **13** (1969), 354–356.
26. C. Tocci and H. J. Caulfield, “Optical Interconnection—Foundations and Applications,” Artech House, Norwood, MA, 1994.
27. S.-Q. Zheng and Y. Li, Pipelined asynchronous time-division multiplexing optical bus, *Opt. Engrg.* **36**, 12 (1997), 3392–3400.