Fast and Scalable Parallel Matrix Computations with Optical Buses (Extended Abstract*)

Keqin Li

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

Abstract. We present fast and highly scalable parallel computations for a number of important and fundamental matrix problems on linear arrays with reconfigurable pipelined optical bus systems. These problems include computing the Nth power, the inverse, the characteristic polynomial, the determinant, the rank, and an LU- and a QR-factorization of a matrix, and solving linear systems of equations. These computations are based on efficient implementation of the fastest sequential matrix multiplication algorithm, and are highly scalable over a wide range of system size. Such fast and scalable parallel matrix computations were not seen before on distributed memory parallel computing systems.

1 Introduction

Parallel matrix computations using optical buses have recently been addressed in [9], where a number of matrix manipulation problems are considered, including computations of the Nth power, the inverse, the characteristic polynomial, the determinant, the rank, and an LU- and a QR-factorization of an $N \times N$ matrix, and solving linear systems of equations. It is shown in [9] that compared with the best known results on distributed memory systems, the parallel complexities of these problems can be reduced by a factor of $O(\log N)$ on linear arrays with reconfigurable pipelined bus systems (LARPBS).

While speed is an important motivation of parallel computing, there is another issue in realistic parallel computing, namely, scalability, which measures the ability to maintain speedup linearly proportional to the number of processors [5]. In this paper, we present fast and highly scalable parallel computations for the above problems. These computations are based on efficient implementation of the fastest $O(N^\alpha)$ sequential matrix multiplication algorithm [2, 11], and are highly scalable (i.e., constant efficiency, cost-optimality, and linear speedup can be achieved) over a wide range of system size p. Such fast and scalable parallel matrix computations were not seen before on distributed memory parallel computing systems.

2 Scalable Parallelization

The time complexity of a parallel computation can be represented as $O(T(N)/p + T_{\text{comm}}(N,p))$, 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,

* A complete version of the paper is available as Technical Report #00-100, Dept. of Mathematics and Computer Science, SUNY at New Paltz, January 2000. See http://www.mcs.newpaltz.edu/tr.

and $T_{\text{comm}}(N,p)$) 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 and cost-optimality can be achieved for all $1 \leq p \leq p^*$. Clearly, p^* should be as large as possible, since this implies that the parallel implementation has the ability to be scaled over a large range of system size p. A parallel implementation is *highly scalable* if p^* is as large as $\Theta(T(N)/(T^*(N)(\log N)^k))$ for some constant $k \geq 0$, where $T^*(N)$ is the best possible parallel time. It is clear that not every sequential algorithm can be parallelized (by using sufficient processors) so that constant parallel execution time is achieved. If $T^*(N)$ is the best possible parallel time, the largest possible value for p^* is $\Theta(T(N)/T^*(N))$. High scalability means that p^* is very close to the largest possible except for a polylog factor of N. A highly scalable parallel implementation is *fully scalable* if $k = 0$, which means that the sequential algorithm can be fully parallelized, and communication overhead $T_{\text{comm}}(N,p)$ in parallelization is negligible.

3 Optical Buses

An LARPBS is a distributed memory system which consists of p processors P_1, P_2, ..., P_p linearly connected by a reconfigurable pipelined optical bus system. Each processor can perform ordinary arithmetic and logic computations, and interprocessor communication. All computations and communications are synchronized by bus cycles, so that an LARPBS is similar to an SIMD machine. In addition to the tremendous communication capabilities supported by a pipelined optical bus, an LARPBS can also be partitioned into several independent subarrays by reconfiguring an optical bus, and all subarrays can be used independently for different computations without interference (see [12] for detailed exposition). Due to reconfigurability, an LARPBS can also be viewed as an MIMD machine.

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.

A rich set of basic communication, data movement, and aggregation operations on the LARPBS model implemented using the coincident pulse processor addressing technique [1] have been developed [8, 12]. The following primitive operations on LARPBS are used in this paper, and our algorithms are described using these operations as building blocks.

One-to-One Communication. Processor P_{i_k} sends a value to P_{j_k}, for all $1 \leq k \leq q$ simultaneously.

Multiple Multicasting. Assume that we have g disjoint groups of destination processors G_k and g senders P_{i_k}, where $1 \leq k \leq g$. Processor P_{i_k} broadcasts a value to all the processors in G_k, for all $1 \leq k \leq g$ simultaneously.

Global Aggregation. Suppose processor P_j holds a value v_j, $1 \leq j \leq p$. We need to calculate the summation $v_1 + v_2 + \cdots + v_p$, which is saved in P_1. It is assumed that the v_j's are integers or floating-point values with finite magnitude and precision, or boolean values (where + is replaced by logical operations). [1]

[1] It has been a common practice in algorithm analysis to assume that a single manipulation takes constant time. This essentially implies that numerical values have finite magnitude and precision; otherwise, a manipulation either takes non-constant time or requires extra hardware support. Therefore, our assumption in the global aggregation operation is quite reasonable.

All these communication, data movement, and global aggregation primitives can be performed on an LARPBS in constant number of bus cycles [8, 12]. We would like to point out that if each value is replaced by a block of s data, the execution time of the above primitive operations is simply increased to $O(s)$ by executing an operation for s times.

The above primitive operations can be directly used for some basic matrix manipulations. For instance, one-to-one communication can be used for transposing a matrix. We divide an $N \times N$ matrix $A = (a_{ij})_{N \times N}$ into submatrices $A = (A_{ij})_{\sqrt{p} \times \sqrt{p}}$, where each submatrix A_{ij} is of size $N/\sqrt{p} \times N/\sqrt{p}$. Assume that A is stored in a p-processor LARPBS in the row-major order, that is, A_{ij} is hold by $P_{(i-1)\sqrt{p}+j}$, for all $1 \leq i, j \leq \sqrt{p}$. By using one-to-one communication with blocks of data of size $s = N^2/p$, processor $P_{(i-1)\sqrt{p}+j}$ sends A_{ij} to processor $P_{(j-1)\sqrt{p}+i}$ simultaneously for all $1 \leq i, j \leq \sqrt{p}$.

Theorem 1. *Transposing an $N \times N$ matrix can be done on a p-processor LARPBS in $O(N^2/p)$ time. Our implementation for matrix transposition is fully scalable.*

The aggregation operation can be used to calculate the summation of N vectors. Given N N-dimensional vectors $\mathbf{v}_i = (v_{i1}, v_{i2}, ..., v_{iN})$, where $1 \leq i \leq N$, the vector chain addition problem is to calculate $\mathbf{v} = (v_1, v_2, ..., v_N)$, where $v_j = v_{1j} + v_{2j} + \cdots + v_{Nj}$, for all $1 \leq j \leq N$. By using one-to-one communication and the global aggregation operation, it is known [9] that the summation of q N-dimensional vectors can be calculated in $O(1)$ time on a qN-processor LARPBS. If $p < N$, each processor holds at most $\lceil N/p \rceil$ vectors. It takes $O(N^2/p)$ time for each processor to compute locally the summation of the vectors it holds. Then, the summation of the p partial sums can be obtained in $O(N)$ time by the p processors. If $p \geq N$, we will assign p/N processors to each vector $\mathbf{v}_i = (\mathbf{v}_{i,1}, \mathbf{v}_{i,2}, ..., \mathbf{v}_{i,p/N})$, such that each processor holds a vector segment $\mathbf{v}_{i,j}$ of size N^2/p. Then, the method in [9] is applied to these vector segments, and the execution time is increased by a factor of $O(N^2/p)$.

Theorem 2. *The summation of N N-dimensional vectors can be calculated on a p-processor LARPBS in $O(N^2/p)$ time. Our implementation for vector chain addition is fully scalable.*

Given N matrices A_1, A_2, ..., 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 $a_{ij} = a_{ij}^{(1)} + a_{ij}^{(2)} + \cdots + a_{ij}^{(N)}$, for all $1 \leq i, j \leq N$. By using one-to-one communication and the global aggregation operation, it is known [9] that the summation of N $s \times s$ matrices can be computed in $O(1)$ time on an s^2N-processor LARPBS. If $p < N$, each processor holds at most $\lceil N/p \rceil$ matrices. It takes $O(N^3/p)$ time for each processor to compute locally the summation of the matrices it holds. Then, the summation of the p partial sums can be obtained in $O(N^2)$ time by the p processors. If $p \geq N$, we will assign p/N processors to each matrix $A_k = (A_{ij}^{(k)})_{\sqrt{p/N} \times \sqrt{p/N}}$ such that each processor holds a submatrix $A_{i,j}$ of size $(N^{1.5}/\sqrt{p}) \times (N^{1.5}/\sqrt{p})$. Then, the method in [9] is applied to these submatrices, and the execution time is increased by a factor of $O(N^3/p)$.

Theorem 3. *The summation of N $N \times N$ matrices can be computed on a p-processor LARPBS in $O(N^3/p)$ time. Our implementation for matrix chain addition is fully scalable.*

In [9], it is shown that by using one-to-one communication, multiple multicasting, and the global aggregation operation, the product of an $N \times N$ matrix $A = (a_{ij})_{N \times N}$, and an N-dimensional vector $\mathbf{v} = (v_1, v_2, ..., v_N)$ can be obtained on an N^2-processor LARPBS in $O(1)$ time. It is clear that we can divide $A = (A_{ij})_{\sqrt{p} \times \sqrt{p}}$ into submatrices of size $(N/\sqrt{p}) \times (N/\sqrt{p})$ and $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{\sqrt{p}})$ into vector segments of size $N/\sqrt{p}$. Then, the method in [9] is applied to these submatrices and vector segments, and the execution time is increased by a factor of $O(N^2/p)$.

Theorem 4. *The product of an $N \times N$ matrix and an N-dimensional vector can be obtained on a p-processor LARPBS in $O(N^2/p)$ time. Our implementation for matrix-vector multiplication is fully scalable.*

4 Matrix Multiplication, Chain Product, and Powers

Matrix multiplication is definitely the most important subproblem in many other matrix manipulations. A number of parallel matrix multiplication algorithms have been developed on LARPBS [8, 10]. The following most noteworthy result, which leads to fast and scalable algorithms for many matrix operations, was shown in [7].

Theorem 5. *For all $1 \leq p \leq N^\alpha$, multiplying two $N \times N$ matrices can be performed on a p-processor LARPBS in*

$$T_{\text{mm}}(N, p) = O\left(\frac{N^\alpha}{p} + \frac{N^2}{p^{2/\alpha}} \log p\right)$$

time. In particular, multiplying two $N \times N$ matrices can be performed in $O(\log N)$ time on an LARPBS with N^α processors. Our implementation for matrix multiplication is scalable for $p = O(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$.

The last statement in the above theorem needs explanation. Assume that $p = N^\alpha/(\log N)^\beta$. Then, $T_{\text{mm}}(N, p) = O((\log N)^\beta + (\log N)^{1+2\beta/\alpha})$. It is clear that when $\beta \geq 1+2\beta/\alpha$, i.e., $\beta \geq \alpha/(\alpha-2)$, the first term dominates $T_{\text{mm}}(N, p)$. That is, $T_{\text{mm}}(N, p) = O(N^\alpha/p)$, which results in linear speedup and cost-optimality.

Given N matrices A_1, A_2, ..., A_N of size $N \times N$, the matrix chain product problem is to compute $A_1 \times A_2 \times \cdots \times A_N$. Given an $N \times N$ matrix A, the matrix powers problem is to calculate the first N powers of A, i.e., A, A^2, A^3, ..., A^N. For both problems, the sequential time complexity is $O(N^{\alpha+1})$.

Based on Theorem 5, the following result has been established in [6].

Theorem 6. *For all $1 \leq p \leq N^{\alpha+1}/2$, the product of N matrices of size $N \times N$ can be computed on a p-processor LARPBS in*

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

time. In particular, the product of N matrices of size $N \times N$ can be computed in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for matrix chain product is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

For matrix powers, the following result has been established in [6].

Theorem 7. *For all $1 \leq p \leq N^{\alpha+1}$, the first N powers of an $N \times N$ matrix can be computed on a p-processor LARPBS in*

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

time. Consequently, the first N powers of an $N \times N$ matrix can be computed in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for matrix powers is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

5 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} with 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 $N/2 \times N/2$, $A = \begin{bmatrix} A_1 & A_3 \\ 0 & A_2 \end{bmatrix}$. Then, 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 two. The recursion can be unwound into $n+1$ iterations, such that a sequence of matrices A_0, A_1, A_2, ..., A_n are 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, $A_n = A^{-1}$. The above method implies that the sequential time complexity is $\sum_{k=1}^{n} 2^{n-k}O((2^{k-1})^{\alpha}) = O(2^{n-\alpha}\sum_{k=1}^{n} 2^{(\alpha-1)k}) = O(2^{n-\alpha} \cdot 2^{(\alpha-1)(n+1)}) = O(2^{n\alpha-1}) = O(N^{\alpha})$, which is the same as matrix multiplication. We need to develop scalable parallelization of the above method for the number of processors p in the range $[1..N^{\alpha}]$.

It is clear that in calculating 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. To calculate A_k, an LARPBS with p processors 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. If the p processors are evenly distributed to the 2^{n-k} subarrays, the time complexity for computing A_k is $T_{\rm mm}(2^{k-1}, p/2^{n-k})$, this gives the overall time complexity $T_{\rm tri}(N,p) = \sum_{k=1}^{n} T_{\rm mm}(2^{k-1}, p/2^{n-k})$. The following result is obtained by algebraic manipulations (see the complete version of the paper). Compared to matrix multiplication, our implementation only introduces a small overhead.

Theorem 8. *For all $1 \le p \le N^\alpha$, the inverse of a lower/upper triangular matrix can be calculated by a p-processor LARPBS in*

$$T_{\rm tri}(N,p) = O\left(\frac{N^\alpha}{p} + \frac{N^2}{p^{2/\alpha}}\log p + \log N \log\frac{p}{N}\right)$$

time. In particular, the inverse of a lower/upper triangular matrix can be calculated in $O((\log N)^2)$ time on an LARPBS with $N^\alpha/(\log N)^{\alpha/2}$ processors. Our implementation for lower/upper triangular matrix inversion is scalable for $p = O(N^\alpha/(\log N)^{\alpha/(\alpha-2)})$.

6 Determinants, Characteristic Polynomials, and Ranks

Let the determinant of a matrix A be denoted by $\det(A)$. The characteristic polynomial of a matrix A is defined as $\phi_A(\lambda) = \det(\lambda I_N - A) = \lambda^N + c_1\lambda^{N-1} + c_2\lambda^{N-2} + \cdots + c_{N-1}\lambda + c_N$, where I_N is the $N \times N$ identity matrix. The trace $\mathrm{tr}(A)$ of a matrix $A = (a_{ij})_{N\times N}$ is the sum of the entries on A's main diagonal, i.e., $\mathrm{tr}(A) = a_{11} + a_{22} + \cdots + a_{NN}$.

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

Leverrier's Lemma. *The coefficients c_1, c_2, ..., c_N of the characteristic polynomial of a matrix A satisfy*

$$S\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{N-1} \\ c_N \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_{N-1} \\ s_N \end{bmatrix}, \quad \text{where } S = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ s_1 & 2 & 0 & \cdots & 0 & 0 \\ s_2 & s_1 & 3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ s_{N-2} & s_{N-3} & s_{N-4} & \cdots & N-1 & 0 \\ s_{N-1} & s_{N-2} & s_{N-3} & \cdots & s_1 & N \end{bmatrix},$$

and $s_k = \mathrm{tr}(A^k)$, for all $1 \le k \le N$.

Based on Leverrier's Lemma, Csanky devised the following method for calculating the characteristic polynomial of a matrix A [3]. 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, ..., A^N;

(2) Calculate $s_k = \mathrm{tr}(A^k)$, 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 $S^{-1}[s_1, s_2, ..., s_N]^T$ to obtain c_1, c_2, ..., c_N.

Step (1) involves the calculation of the first N powers of A (Theorem 7). 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 complexity of Csanky's method is dominated by Step (1).

Theorem 9. *For all $1 \leq p \leq N^{\alpha+1}$, the characteristic polynomial of an $N \times N$ matrix can be obtained on a p-processor LARPBS in*

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

time. In particular, the characteristic polynomial of an $N \times N$ matrix can be obtained in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for characteristic polynomial is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

Theorem 10. *For all $1 \leq p \leq N^{\alpha+1}$, the determinant of an $N \times N$ matrix can be obtained on a p-processor LARPBS in*

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

time. Consequently, the determinant of an $N \times N$ matrix can be obtained in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for determinant is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

The rank of a matrix A, $\mathrm{rank}(A)$, is the number of nonzero rows (or columns) in the row-reduced (or column-reduced) echelon form of A. It is well known that $\mathrm{rank}(A) = \mathrm{rank}(A^TA)$, where A^T is the transpose of A (or conjugate transpose of A for complex matrices), and A^TA is similar to a diagonal matrix whose elements are the roots of the characteristic polynomial $\phi_{A^TA}(\lambda)$. Therefore, $\mathrm{rank}(A)$ is the number of nonzero roots of $\phi_{A^TA}(\lambda)$. This leads to the following algorithm for finding $\mathrm{rank}(A)$ [4].

An Algorithm for Calculating Matrix Rank.

(1) Get the matrix A^TA;
(2) Calculate $\phi_{A^TA}(\lambda) = c_0\lambda^N + c_1\lambda^{N-1} + c_2\lambda^{N-2} + \cdots + c_{N-1}\lambda + c_N$;
(3) Find $\mathrm{rank}(A) = N - i$, 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 9). Step (3) can be implemented in $O(N/p)$ time by simple data testing, comparison, and movement.

Theorem 11. *For all $1 \leq p \leq N^{\alpha+1}$, the rank of an $N \times N$ matrix can be obtained on a p-processor LARPBS in*

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

time. Consequently, the rank of an $N \times N$ matrix can be obtained in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for matrix rank is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

7 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 $\phi_A(\lambda) = \lambda^N + c_1\lambda^{N-1} + c_2\lambda^{N-2} + \cdots + c_{N-1}\lambda + c_N$ be the characteristic polynomial of matrix A. Then $\phi_A(A) = A^N + c_1A^{N-1} + c_2A^{N-2} + \cdots + c_{N-1}A + c_NA^0$ is the $N \times N$ zero matrix.*

Cayley-Hamilton Theorem implies that $A(A^{N-1} + c_1A^{N-2} + c_2A^{N-3} + \cdots + c_{N-1}I_N) = -c_NI_N$. Hence, the inverse of a matrix A can be calculated using the following identity, $A^{-1} = -(1/c_N)(A^{N-1} + c_1A^{N-2} + c_2A^{N-3} + \cdots + c_{N-2}A + c_{N-1}I_N)$. Csanky's method for calculating matrix inversion can be described as follows [3].

Csanky's Strategy for Matrix Inversion.

(1) Calculate the characteristic polynomial of A, that is, c_1, c_2, ..., c_N;
(2) Compute A^{-1} by using the above identity.

The time complexity of Step (1) is given in Theorem 9. Step (2) involves the computation of the first N powers of A (Theorem 7) and matrix chain addition (Theorem 3). Hence, we have the following result.

Theorem 12. *For all $1 \leq p \leq N^{\alpha+1}$, the inverse of an $N \times N$ matrix can be obtained on a p-processor LARPBS in*

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

time. Consequently, the inverse of an $N \times N$ matrix can be obtained in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for matrix inversion is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

8 Linear Systems of Equations

Let A be a nonsingular $N \times N$ matrix $(a_{ij})_{N\times N}$, and $\mathbf{b} = (b_1, b_2, ..., b_N)^T$ be an N-dimensional vector. The problem of solving a linear system of equations is to find a vector $\mathbf{x} = (x_1, x_2, ..., x_N)^T$ 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 12) and a matrix-vector multiplication (Theorem 4).

Theorem 13. *For all $1 \leq p \leq N^{\alpha+1}$, a linear system of equations $A\mathbf{x} = \mathbf{b}$ can be solved on a p-processor LARPBS in*

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

time. Consequently, a linear system of equations $A\mathbf{x} = \mathbf{b}$ can be solved in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for solving a linear system of equations is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

Let $\mathbf{k}_j = A^j\mathbf{b}$, where $0 \leq j \leq N-1$. Then, the matrix $K(A,\mathbf{b},N) = [\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, ..., A^{N-1}\mathbf{b}]$ is called the Krylov matrix defined by the matrix A, the vector $\mathbf{b}$, and the integer N. By Cayley-Hamilton Theorem, we have $A^N\mathbf{b} + c_1A^{N-1}\mathbf{b} + c_2A^{N-2}\mathbf{b} + \cdots + c_{N-1}A\mathbf{b} + c_N\mathbf{b} = 0$, that is, $A(A^{N-1}\mathbf{b} + c_1A^{N-2}\mathbf{b} + c_2A^{N-3}\mathbf{b} + \cdots + c_{N-1}\mathbf{b}) = -c_N\mathbf{b}$, which implies that $\mathbf{x} = -(1/c_N)\mathbf{k}_{N-1} - (c_1/c_N)\mathbf{k}_{N-2} - (c_2/c_N)\mathbf{k}_{N-3} - \cdots - (c_{N-1}/c_N)\mathbf{k}_0$. 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 7), 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 (Theorem 2). This proves the following result.

Theorem 14. *For all $1 \leq p \leq N^{\alpha+1}$, the Krylov matrix defined by the matrix A, the vector $\mathbf{b}$, and the integer N, can be calculated on a p-processor LARPBS in*

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

time. Consequently, the Krylov matrix can be calculated in $O((\log N)^2)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^{\alpha/2}$ processors. Our implementation for Krylov matrix is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

9 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 follows: $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \times \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$. The above identity implies that $A_{11} = L_{11}U_{11}$, $A_{12} = L_{11}U_{12}$, $A_{21} = L_{21}U_{11}$, $A_{22} = L_{21}U_{12} + L_{22}U_{22}$.

Pan's Method for LU-Factorization [11].

(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 $U_{12} = U_{11}X_1$.

If $T(N)$ is the sequential time complexity of the above method, then we have $T(N) = 2T(N/2) + O((N/2)^{\alpha+1})$. It can be verified that $T(N) = O(N^{\alpha+1})$. Let $T_{\rm LU}(N,p)$ be the parallel time complexity of the above algorithm on an LARPBS. Then $T_{\rm LU}(N,p) = T_{\rm LU}(N/2,p/2) + T_{\rm inverse}(N/2,p) + 5T_{\rm mm}(N/2,p)$. The following result is obtained by algebraic manipulations (see the complete version of the paper).

Theorem 15. *For all $1 \le p \le N^{\alpha+1}$, LU-factorization of an $N \times N$ matrix can be performed on a p-processor LARPBS in*

$$T_{\rm LU}(N,p) = O\left(\frac{N^{\alpha+1}}{p} + \frac{N^{2(1+1/\alpha)}}{p^{2/\alpha}}\log p + (\log N)^3\right).$$

time. Consequently, LU-factorization of an $N \times N$ matrix can be performed in $O((\log N)^3)$ time on an LARPBS with $N^{\alpha+1}/(\log N)^\alpha$ processors. Our implementation for LU-factorization is scalable for $p = O(N^{\alpha+1}/(\log N)^{\alpha/(\alpha-2)})$.

(The part for QR-factorization is omitted due to space limitation.)

References

1. D. Chiarulli, R. Melhem, and S. Levitan, "Using coincident optical pulses for parallel memory addressing," *IEEE Computer*, vol. 30, pp. 48-57, 1987.
2. D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, pp. 251-280, 1990.
3. L. Csanky, "Fast parallel matrix inversion algorithms," *SIAM Journal on Computing*, vol. 5, pp. 618-623, 1976.
4. O.H. Ibarra, S. Moran, and L.E. Rosier, "A note on the parallel complexity of computing the rank of order n matrices," *Information Processing Letters*, vol. 11, no. 4,5, p. 162, 1980.
5. V. Kumar, *et al.*, *Introduction to Parallel Computing*, Benjaming/Cummings, 1994.
6. 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.
7. K. Li and V.Y. Pan, "Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system," *Proc. IPPS/SPDP '99*, pp. 31-35, April 1999.
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 Trans. on Parallel and Distributed Systems*, vol. 9, no. 8, pp. 705-720, 1998.
9. 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.
10. K. Li, Y. Pan, and S.-Q. Zheng, "Scalable parallel matrix multiplication using reconfigurable pipelined optical bus systems," *Proc. of 10th Int'l Conf. on Parallel and Distributed Computing and Systems*, pp. 238-243, October 1998.
11. V. Pan, "Complexity of parallel matrix computations," *Theoretical Computer Science*, vol. 54, pp. 65-85, 1987.
12. Y. Pan and K. Li, "Linear array with a reconfigurable pipelined bus system – concepts and applications," *Information Sciences*, vol. 106, no. 3-4, pp. 237-258, 1998.