A Parallel Algorithm for Functions of Triangular Matrices *

Ç. K. Koç and B. Bakkaloğlu
Department of Electrical & Computer Engineering
Oregon State University
Corvallis, Oregon 97331

Abstract

We present a new parallel algorithm for computing arbitrary functions of triangular matrices. The presented algorithm is the first one to date requiring polylogarithmic time, and computes an arbitrary function of an $n \times n$ triangular matrix in $O(\log^3 n)$ time using $O(n^6)$ processors. The algorithm requires the eigenvalues of the input matrix be distinct, and makes use of the commutativity relationship between the input and output matrices.

1 Introduction

Computing a function $f(A)$ of an $n \times n$ matrix A is an important problem in linear algebra, engineering and applied mathematics. There are several methods including computing Jordan decomposition, Schur decomposition, and approximation methods such as Taylor expansion, rational Padé approximations, etc. The Jordan decomposition approach seems to have computational difficulties unless A is diagonalizable and has a well-conditioned matrix of eigenvectors. On the other hand, the Schur decomposition is more stable and can be easily applied for matrix function evaluation. If $A = QTQ^H$ is the Schur decomposition of a full matrix A, then we have $f(A) = Qf(T)Q^H$, where T is a triangular matrix. Thus, an effective algorithm for finding the matrix valued functions of triangular matrices is needed. A complicated explicit expression for $f(T)$ is known [5, 4]. Given the upper triangular matrix $T = \{t_{ij}\}$ with $\lambda_i = t_{ii}$ and the function $f(T) = \{f_{ij}\}$ defined on $\mathcal{R}$, we have $f_{ij} = 0$ for $i > j$, $f_{ii} = f(\lambda_i)$, and also for $i < j$

$$f_{ij} = \sum_{(s_0,\ldots,s_k)\in S_{ij}} t_{s_0,s_1} t_{s_1,s_2} \cdots t_{s_{k-1},s_k} f[\lambda_{s_0},\ldots,\lambda_{s_k}] \ ,$$

where S_{ij} is a set of distinct sequence of integers such that $s_0 = i < s_1 < \ldots < s_k = j$, $1 \leq k \leq j-i$, and $f[\lambda_{s_0},\ldots,\lambda_{s_k}]$ is the kth order divided difference of f at $\{\lambda_{s_0},\ldots,\lambda_{s_k}\}$. Unfortunately, computing the upper triangular matrix function $F = f(T)$ using this method requires $O(2^n)$ arithmetic operations, which is computationally prohibitive for large matrices [5].

2 Parlett's Algorithm

The first practical (in terms of the required number of arithmetic operations) algorithm for computing an arbitrary function of an upper triangular matrix is given by Parlett [9]. The algorithm

*This work is supported in part by the National Science Foundation under grant ECS–9312240.

Computing, 57(1)85-92, 1996.

is derived using the property that the matrices T and F commute:

$$FT = TF \ . \tag{1}$$

Parlett shows that by expanding the matrix multiplication and solving for f_{ij} in the above, we obtain the summation formula

$$f_{ij} = t_{ij} \frac{f_{jj} - f_{ii}}{t_{jj} - t_{ii}} + \frac{1}{t_{jj} - t_{ii}} \sum_{k=i+1}^{j-1} (t_{ik} f_{kj} - f_{ik} t_{kj}) \ . \tag{2}$$

Parlett's algorithm starts with computing the main diagonal entries of F. Since the main diagonal entries t_{ii} are the eigenvalues of T, f_{ii} is calculated by applying f to each t_{ii}, i.e., $f_{ii} = f(t_{ii})$. After computing the main diagonal entries, the algorithm computes the superdiagonals one at a time, using the summation expression (2). The number of arithmetic operations required to compute an element of the Lth superdiagonal is easily calculated as $4L$. Since there are $(n - L)$ elements in the Lth superdiagonal, the computation of each superdiagonal requires $4(n - L)L$ arithmetic operations. Thus, assuming a single scalar function evaluation requires K arithmetic operations, Parlett's algorithm requires a total of $Kn + \frac{2}{3}(n^3 - n)$ arithmetic operations to compute all elements of the upper triangular matrix F. However, we must remark that if T has close or multiple eigenvalues, this algorithm will give inaccurate results. Alternative methods for dealing with the repeated eigenvalue case can be found in [9, 5].

Parlett's algorithm first computes the main diagonal elements of the matrix F by performing n independent scalar function evaluations. Provided that we have n processors available, this step can be performed in time for a single function evaluation. The remaining elements of the upper triangular matrix can be obtained by computing each super diagonal in parallel. This parallel algorithm has n phases; a superdiagonal vector is computed at each phase using all the available processors. If there are n processors available, we can compute an arbitrary function of an $n \times n$ upper triangular matrix in $O(n^2)$ time. However, it is also possible to compute the matrix function in less than $O(n^2)$ time using more than $O(n)$ processors. To see this, we note that the maximum length of the summation formula (2) is equal to $n - 2$. Using $O(n)$ processors, each summation can be obtained in $O(\log n)$ time. Obtaining the summation for all the elements on the same superdiagonal would require $O(n^2)$ processors and $O(\log n)$ time, and by repeatedly using these $O(n^2)$ processors on all superdiagonals we obtain the matrix function in $O(n \log n)$ time. It is an open question whether Parlett's algorithm can further be parallelized, more specifically whether a parallel algorithm requiring polylogarithmic time can be obtained, which uses Parlett's summation.

3 The Divide-and-Conquer Algorithm

A divide-and-conquer algorithm making use of the commutativity relationship of Equation (1) has been proposed in [6]. This algorithm is of the same order of complexity as Parlett's algorithm, but the block structure of the algorithm makes it favorable to Parlett's method for computers with two levels of memory. We will now show that this algorithm can also be efficiently parallelized. Let $n = 2k$ and the matrices T and F be partitioned as

$$T = \begin{bmatrix} T_1 & T_2 \\ 0 & T_3 \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} F_1 & F_2 \\ 0 & F_3 \end{bmatrix} \ ,$$

Computing, 57(1)85-92, 1996.

respectively. Here $T_1, F_1 \in \mathcal{C}^{k\times k}$ and $T_3, F_3 \in \mathcal{C}^{k\times k}$ are upper triangular, and $T_2, F_2 \in \mathcal{C}^{k\times k}$ are full matrices. Here we use the commutativity relationship (1), and expand the matrix equation $FT = TF$ in terms of the products of the matrix blocks as

$$\begin{aligned} T_1F_1 &= F_1T_1 \ , \\ T_3F_3 &= F_3T_3 \ , \\ T_1F_2 + T_2F_3 &= F_1T_2 + F_2T_3 \ . \end{aligned}$$

Since T_1 and T_3 are upper triangular, we have $F_1 = f(T_1)$ and $F_3 = f(T_3)$. Assuming F_1 and F_2 are already computed, we define $C = F_1T_2 - T_2F_3$, and proceed to solve the matrix equation

$$T_1F_2 - F_2T_3 = C \tag{3}$$

in order to calculate F_2. This matrix equation is known as the Sylvester equation [5]. Let λ_i and μ_i for $i = 1, 2, \ldots, k$ be the distinct eigenvalues (diagonal elements) of T_1 and T_3. The Sylvester equation (3) has a unique solution F_2 if and only if $\lambda_i \neq \mu_j$ for all i and j. This unique solution can be found using Bartels-Stewart algorithm [1] or Kronecker product method [2], both of which require $O(n^3)$ arithmetic operations. A detailed analysis of the solution for the specific case of upper triangular coefficient matrices has been given in [6].

The new matrix function evaluation algorithm is a recursive algorithm, however, it can be 'unrolled' to obtain a non-recursive algorithm. The progression of the algorithm is similar to the inversion of triangular matrices in [8]. Unwinding the recursion to the lowest level and then building back up again, we produce a simple $\log(n)$-phase algorithm for finding $f(T)$. Let n be a power of 2, i.e., $n = 2^d$. The non-recursive algorithm first applies the function f to the main diagonal. After obtaining the scalar function of the main diagonal, in the first phase the algorithm solves a scalar Sylvester equation which is a linear equation in one unknown $f_{i,i+1}$,

$$t_{ii}f_{i,i+1} - f_{i,i+1}t_{i+1,i+1} = f_{ii}t_{i,i+1} - t_{i,i+1}f_{i+1,i+1} \ \text{for} \quad i = 1, 3, 5, \ldots n-1 \ .$$

Prior to the kth step, the evaluation of $n/2^{k-1}$ matrix blocks (of dimension $2^{k-1} \times 2^{k-1}$) in the main diagonal have been completed. During the kth step, the algorithm uses these $n/2^{k-1}$ matrix blocks in pairs, and solves $n/2^k$ Sylvester equations in order to obtain $n/2^k$ matrix blocks (of dimension $2^k \times 2^k$) required for the next step. The total number of arithmetic operations for the unrolled divide-and-conquer algorithm can be given as

$$T(n) = Kn + \sum_{k=0}^{d-1} \frac{n}{2^k} S(2^k) + \frac{n}{2^{k+1}} U(2^k) = Kn + \frac{2n^3}{3} + \frac{n^2}{2} - \frac{7n}{6} \ ,$$

where $S(n)$ is the number of arithmetic operations required to solve a Sylvester matrix equation of size n, and $U(n)$ is the number of arithmetic operations needed to compute the $n \times n$ matrix C using $C = F_1T_2 - T_2F_3$, which are found as $S(n) = 2n^3$ and $U(n) = 2n^3 + n^2$ [6].

4 The Parallel Divide-and-Conquer Algorithm

The divide-and-conquer type algorithm for computing an arbitrary function of an $n \times n$ triangular matrix has $\log(n)$ phases; however, at the kth phase a Sylvester equation of size $2^k \times 2^k$ needs to be solved. We now show how to parallelize the solution of the Sylvester equation. The proposed

Computing, 57(1)85-92, 1996.

parallel algorithm for solving the Sylvester equation is based on the Kronecker product algorithm. Let $A \in \mathcal{R}^{m\times m}$ and $B \in \mathcal{R}^{m\times m}$ be upper triangular matrices, and $C \in \mathcal{R}^{m\times m}$ be a full matrix. Then solving the Sylvester equation $AX + XB = C$ of size m is equivalent to solving the $m^2 \times m^2$ linear equation

$$H\mathcal{X} = \mathcal{C} \ , \tag{4}$$

where $\mathcal{X}$ and $\mathcal{C}$ are the $m^2 \times 1$ vectors formed by stacking the transposed rows of the $m\times m$ matrices X and C, respectively. Also H is an $m^2 \times m^2$ matrix such that $H = A \otimes I + I \otimes B^T$, where $\otimes$ is the Kronecker (or tensor) product. In terms of the matrix blocks the Kronecker product matrix can be represented as $H = T_1 \otimes I - I \otimes T_3^T$. For example, for $m = 4$, we have

$$H = \begin{bmatrix} a_{11}I + B^T & a_{12}I & a_{13}I & a_{14}I \\ 0 & a_{22}I + B^T & a_{23}I & a_{24}I \\ 0 & 0 & a_{33}I + B^T & a_{34}I \\ 0 & 0 & 0 & a_{44}I + B^T \end{bmatrix}$$

The structure of H can be exploited to design a parallel algorithm for the solution of the equation $H\mathcal{X} = \mathcal{C}$. This algorithm is similar to the parallel inversion of triangular matrices [3]. Let D be the $m^2 \times m^2$ diagonal matrix such that $d_{ii} = h_{ii}$ for $i = 1, 2, \ldots, m^2$. Let $J = D^{-1}H$, and $U = I - J$, where U is an $m^2 \times m^2$ matrix with diagonal elements all zero. It can easily be proven that $U^i = 0$ for $i \geq 2m - 1$. We will try to analyze this property of the block upper triangular matrix U. The general form of U is given as

$$U = \begin{bmatrix} L_{11} & a_{12}I & a_{13}I & \cdots & a_{1m}I \\ 0 & L_{22} & a_{23}I & \cdots & a_{2m}I \\ 0 & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & L_{mm} \end{bmatrix} ,$$

where the block diagonal element L_{ii} is a $m \times m$ lower triangular matrix with zero entries in the main diagonal. Let the kth power of U be given as

$$U^k = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1m} \\ 0 & P_{22} & \cdots & P_{2m} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & P_{mm} \end{bmatrix} ,$$

where $P_{ii} = L_{ii}^k$. The matrix L_{ii} is of the form

$$L_{ii} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ \times & 0 & 0 & \cdots & 0 \\ \times & \times & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \times & \cdots & \times & \times & 0 \end{bmatrix} ,$$

where $\times$ denotes the nonzero entries. The consecutive powers of L_{ii} is given as

$$L_{ii}^2 = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \times & 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \times & \cdots & \times & 0 & 0 \end{bmatrix} , \quad L_{ii}^{m-1} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \times & 0 & \cdots & 0 & 0 \end{bmatrix} , \quad L_{ii}^m = \mathbf{0} \ .$$

Computing, 57(1)85-92, 1996.

Therefore for $k \geq m$ the main diagonal matrix blocks of U^k are zero matrices, and the block structure of U^m becomes

$$U^m = \begin{bmatrix} 0 & P'_{12} & \cdots & P'_{1m} \\ 0 & 0 & \cdots & P'_{2m} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} .$$

Now we can easily show that

$$J^{-1} = I + U + U^2 + \cdots + U^{2m-2}$$

by multiplying the right hand side with $J = I - U$. Once J^{-1} is computed, we compute H^{-1} using $H^{-1} = J^{-1}D^{-1}$. A fast algorithm for evaluating the matrix polynomial $I + A + \cdots + A^{N-1}$ is given in [7]. This algorithm is based on the cyclic reduction technique and computes this sum using $2\lfloor \log_2 N \rfloor - 1$ matrix multiplications and $\lfloor \log_2 N \rfloor$ matrix additions. Since $O(m^3)$ processors suffice to multiply two $m \times m$ matrices in $O(\log m)$ time, a matrix product of size $m^2 \times m^2$ can be performed in $O(\log m)$ time using $O(m^6)$ processors. Therefore, the computation of J^{-1} requires $\log m \cdot (2\lfloor \log 2m - 1 \rfloor - 1) = O(\log^2 m)$ time by using $O(m^6)$ processors. The solution of Sylvester's equation is then computed using $H^{-1}\mathcal{C} = J^{-1}D^{-1}\mathcal{C}$ which requires an additional $O(\log m)$ time with $O(m^2)$ processors.

The divide-and-conquer algorithm solves $n/2^k$ Sylvester equations of dimension $2^{k-1} \times 2^{k-1}$ at the kth step of the algorithm where $k = 1, 2, \ldots, \log(n)$. Let $m = 2^{k-1}$ be the size of the matrix blocks at the kth step of the algorithm. We have shown that the solution of an $m \times m$ Sylvester equation in $O(\log m)$ time requires $O(m^6)$ processors. At each phase of the algorithm $n/2m$ Sylvester equations can be solved at the same time. Therefore the total number of processors needed at the kth step of the algorithm can be found as $O((n/2m)m^6) = O(nm^5)$. Since the maximum value of m is $n/2$, the maximum number of processors is found as $O(n^6)$. On the other hand, the arithmetic complexity of each step of the algorithm depends on that of the half-sized problem plus the parallel solution of the linear system $H\mathcal{X} = \mathcal{C}$. The parallel block upper triangular linear system solution is shown to require $O(m^6)$ processors and $O(\log^2(2^{k-1}))$ time at each phase. The total number of arithmetic operations to compute an arbitrary function of a triangular matrix is found as

$$\sum_{k=1}^{\log n} \log^2(2^{k-1}) = \frac{1}{6} \log n(\log n - 1)(2\log n - 1) = O(\log^3 n) .$$

5 An Example

We will illustrate the algorithm by computing the square-root of the following 4×4 matrix

$$T = \begin{bmatrix} 16 & -15 & -76 & -14 \\ 0 & 1 & -50 & 14 \\ 0 & 0 & 81 & -44 \\ 0 & 0 & 0 & 4 \end{bmatrix} .$$

The main diagonal elements can be obtained by applying the square root function. The first superdiagonal is obtained by solving the scalar Sylvester equation, which is in fact a linear equation in one unknown:

$$f_{12} = \frac{f_{11}t_{12} - t_{12}f_{22}}{t_{11} - t_{22}} = -3 \text{ and } \quad f_{34} = \frac{f_{33}t_{34} - t_{34}f_{44}}{t_{33} - t_{44}} = -4 .$$

Computing, 57(1)85-92, 1996.

The matrix blocks are found as

$$T_1 = \begin{bmatrix} 16 & -15 \\ 0 & 1 \end{bmatrix}, \quad T_2 = \begin{bmatrix} -76 & -14 \\ -50 & 14 \end{bmatrix}, \quad T_3 = \begin{bmatrix} 81 & -44 \\ 0 & 4 \end{bmatrix}.$$

The computed blocks of the matrix F are

$$F_1 = \begin{bmatrix} 4 & -3 \\ 0 & 1 \end{bmatrix}, \quad F_3 = \begin{bmatrix} 9 & -4 \\ 0 & 2 \end{bmatrix}.$$

The Kronecker product matrix H is found as

$$H = T_1 \otimes I - I \otimes T_3^T = \begin{bmatrix} -65 & 0 & -15 & 0 \\ 44 & 12 & 0 & -15 \\ 0 & 0 & -80 & 0 \\ 0 & 0 & 44 & -3 \end{bmatrix},$$

and J becomes

$$J = D^{-1}H = \begin{bmatrix} 1 & 0 & 0.2308 & 0 \\ 3.6667 & 1 & 0 & -1.25 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -14.6667 & 1 \end{bmatrix}.$$

Removing the unity elements along the diagonal we obtain $U = I - J$. The inverse of J can be found using the power method as

$$J^{-1} = I + U + U^2 = \begin{bmatrix} 1 & 0 & -0.2308 & 0 \\ -3.6667 & 1 & 19.1795 & 1.25 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 14.6667 & 1 \end{bmatrix},$$

and H^{-1} becomes

$$H^{-1} = J^{-1}D^{-1} = \begin{bmatrix} -0.0154 & 0 & 0.0029 & 0 \\ 0.0564 & 0.0833 & -0.2397 & -0.4167 \\ 0 & 0 & -0.0125 & 0 \\ 0 & 0 & -0.1833 & -0.3333 \end{bmatrix}.$$

We first compute C

$$C = F_1T_2 - T_2F_3 = \begin{bmatrix} 530 & -374 \\ 400 & -214 \end{bmatrix},$$

and, thus, $\mathcal{C}$ is found as

$$\mathcal{C} = \begin{bmatrix} 530 & -374 & 400 & -214 \end{bmatrix}^T.$$

Multiplying $\mathcal{C}$ with H^{-1}, we obtain $\mathcal{F}_2$ as

$$\mathcal{F}_2 = H^{-1}\mathcal{C} = \begin{bmatrix} -7 & -8 & -5 & -2 \end{bmatrix}^T,$$

which is the matrix F_2 in the stacked row format. Thus, we find F_2 as

$$F_2 = \begin{bmatrix} -7 & -8 \\ -5 & -2 \end{bmatrix}.$$

Computing, 57(1)85-92, 1996.

References

[1] R. H. Bartels and G. W. Stewart. Solution of the matrix equation $AX + XB = C$. *Communications of the ACM*, 15(9):820–826, 1972.

[2] R. E. Bellman. *Introduction to Matrix Analysis.* New York, NY: McGraw-Hill, 1970.

[3] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods.* Englewood Cliffs, NJ: Prentice-Hall, 1989.

[4] C. Davis. Explicit functional calculus. *Linear Algebra and its Applications*, 6:193–199, 1973.

[5] G. H. Golub and C. F. van Loan. *Matrix Computations.* Baltimore, MD: The Johns Hopkins University Press, 2nd edition, 1989.

[6] Ç. K. Koç. A divide-and-conquer algorithm for functions of triangular matrices. Unpublished Manuscript, June 1995.

[7] L. Lei and T. Nakamura. A fast algorithm for evaluating the matrix polynomial $I + A + \cdots + A^{N-1}$. *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, 39(4):299–300, April 1992.

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

[9] B. N. Parlett. A recurrence among the elements of functions of triangular matrices. *Linear Algebra and its Applications*, 14:117–121, 1976.

Computing, 57(1)85-92, 1996.