Comp. & Maths. with Appls. Vol. 11, No. 6, pp. 573–579, 1985
Printed in Great Britain.

0097-4943/85 $3.00 + .00
© 1985 Pergamon Press Ltd.

LU FACTORIZATION ON PARALLEL COMPUTERS

BENY NETA
Texas Tech University, Department of Mathematics, Lubbock, TX 79409, U.S.A.

and

HENG-MING TAI
Texas Tech University, Department of Electrical Engineering, Lubbock, TX 79409, U.S.A.

(*Received December* 1983)

Communicated by Ervin Y. Rodin

Abstract—A new parallel algorithm for the *LU* factorization of a given dense matrix *A* is described. The case of banded matrices is also considered. This algorithm can be combined with Sameh and Brent's [*SIAM J. Numer. Anal.* **14**, 1101–1113, (1977)] to obtain the solution of a linear system of algebraic equations. The arithmetic complexity for the dense case is $\frac{1}{3}n^2$ ($\frac{1}{2}bn$ in the banded case), using $3(n - 1)$ processors and no square roots.

1. INTRODUCTION

The parallel computers to be considered belong to a class of computers called the single instruction stream-multiple data stream (SIMD) machines. The more general multiple instruction stream-multiple data stream (MIMD) machines capable of executing different instructions simultaneously are not considered here. Wing and Huang[15], Wallach and Konrad[13], Jess and Kees[5], and Huang and Wing[3] discuss the solution of linear systems and *LU* factorization on MIMD machines. SIMD machines are best suited for algorithms requiring the same operations on large arrays of independent data.

The solution of a linear system of equations lies at the heart of many programs for scientific computation. With the recent development and availability of various parallel computers, new algorithms have appeared for solving tridiagonal systems of equations suitable for these machines. Notable among these methods are the recursive doubling method (Stone[12]), the cyclic reduction method (Lambiotte and Voigt[7]), and the partition method (Sameh and Kuck[11], Wang[14]). The solution of triangular systems was considered by Chen, Kuck and Sameh[1], Orcutt[8], Sameh and Brent[9] and Chen and Sameh[2]. See also [6, 10], and references there. Here we are interested in developing an algorithm for *LU* factorization of a matrix. This algorithm, combined, for example, with the one due to Sameh and Brent[9], yields the solution of linear systems on a parallel computer.

The arithmetic complexity for the dense (banded) case is $\frac{1}{3}n^2$ ($\frac{1}{2}bn$), using $3(n - 1)$ processors and no square roots.

In the next section the algorithm is developed. The main theorem will be proved in Section 3. Section 4 will be devoted to numerical experiments with the algorithm.

2. DEVELOPMENT OF THE ALGORITHM

Let A be an $n \times n$ matrix possessing an *LU* factorization (with $l_{ii} = 1, i = 1, 2, \ldots, n$). Then

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, \quad i \leq j, \tag{1}$$

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}}{u_{jj}}, \quad i > j. \tag{2}$$

Note that (1) expresses the entries of the ith row of U in terms of the first $i - 1$ rows of U and the first $i - 1$ columns of L. Again, (2) expresses the entries of the jth column of L in terms of the first $j - 1$ rows of U, the first $j - 1$ columns of L, and the jth diagonal entry of U. In this way we can build up U row by row and L column by column (alternately).

It is clear that in order to obtain the entries of L and U via (1)–(2), one cannot utilize the parallelism of the computer. Let us construct the following n matrices

$$T^{(m)} = (t_{ij}^{(m)}), \quad m = 0, 1, \ldots, n - 1.$$

The entries of the matrices are defined as follows:

$$t_{ij}^{(0)} = a_{ij}, \quad i \le i,j \le n, \tag{3.0}$$

$$t_{ij}^{(1)} = a_{11}a_{ij} - a_{i1}a_{1j}, \quad 2 \le i,j \le n, \tag{3.1}$$

$$t_{11}^{(1)} = t_{11}^{(0)},$$

$$t_{ij}^{(2)} = t_{22}^{(1)}t_{ij}^{(1)} - t_{i2}^{(1)}t_{2j}^{(1)}, \quad 3 \le i,j \le n, \tag{3.2}$$

$$t_{22}^{(2)} = t_{11}^{(1)}t_{22}^{(1)},$$

$$\vdots$$

$$t_{ij}^{(m)} = t_{mm}^{(m-1)}t_{ij}^{(m-1)} - t_{im}^{(m-1)}t_{mj}^{(m-1)}, \quad m + 1 \le i,j, \le n,$$

$$t_{mm}^{(m)} = t_{m-1\,m-1}^{(m-1)}t_{mm}^{(m-1)}. \tag{3.m}$$

Remarks

1. The entries of $T^{(m)}$ can be computed simultaneously. The waiting time can be minimized if one works on more than one column at a time. For example, suppose $n = 10, p = 3 \times 9$; then $T^{(1)}$ is of order 9, and one can construct one column at a time. $T^{(2)}$ is of order 8, and one constructs one column and an entry from the next column. Therefore one needs seven steps to obtain all entries of $T^{(2)}$ except the last one. At this point one can start constructing $T^{(3)}$. The first time we have idle processors will be when $T^{(8)}$ is constructed, since we cannot construct all $T^{(8)}$ with the only entry of $T^{(9)}$.

2. All $T^{(m)}$ can be saved on A if they are computed by columns. Let $\mathbf{a}_i$, $\mathbf{t}_i^{(m)}$ be the ith columns of A, $T^{(m)}$ respectively. Then Eqs. (3) can be written as follows:

$$\mathbf{t}_i^{(0)} = \mathbf{a}_i, \tag*{(3.0)$'$}$$

$$\mathbf{t}_i^{(1)} = t_{11}^{(0)}\mathbf{t}_i^{(0)} - t_{i1}^{(0)}\mathbf{t}_1^{(0)}, \tag*{(3.1)$'$}$$

$$\mathbf{t}_i^{(2)} = t_{22}^{(1)}\mathbf{t}_i^{(1)} - t_{i2}^{(1)}\mathbf{t}_2^{(1)}, \tag*{(3.2)$'$}$$

$$\vdots$$

$$\mathbf{t}_i^{(m)} = t_{mm}^{(m-1)}\mathbf{t}_i^{(m-1)} - t_{im}^{(m-1)}\mathbf{t}_m^{(m-1)}. \tag*{(3.m)$'$}$$

The diagonal entries $t_{ii}^{(i)}$ are given as before.

3. If A is a banded matrix with bandwidth b and the number of processors $p = 3Mb$, then one can construct M columns of $T^{(m)}$ at a time. This is possible because in this case both L and U have a smaller bandwidth $((b + 1)/2)$.

4. It turned out that for moderate n, formulae (3) give rise to overflow. In order to avoid this problem one normalizes these formulae as follows:

$$\tau_{ij}^{(m)} = \frac{t_{ij}^{(m)}}{t_{mm}^{(m)}}.$$

Once all the $T^{(m)}$ are ready, one can construct L and U in parallel, using

$$u_{ij} = \frac{t_{ij}^{(i-1)}}{t_{i-1\,i-1}^{(i-1)}}, \quad 1 \le i \le j \le n \quad (t_{00}^{(0)} \equiv 1) \tag{4}$$

and

$$l_{ij} = \frac{t_{ij}^{(j-1)}}{t_{jj}^{(j-1)}}, \quad 1 \le j \le i \le n. \tag{5}$$

In terms of $\tau_{ij}^{(m)}$ formulae (4)–(5) are

$$u_{ij} = \tau_{ij}^{(i-1)}, \tag{4$'$}$$

$$l_{ij} = \frac{\tau_{ij}^{(j-1)}}{\tau_{jj}^{(j-1)}}. \tag{5$'$}$$

$$u_{N+1\,j} = a_{N+1\,j} - \sum_{k=1}^{N} l_{N+1\,k} u_{kj}.$$

Note that the entries of U are the normalized entries of $T^{(m)}$.

In the next section we prove that (4)–(5) are equivalent (1)–(2).

3. MAIN RESULT

THEOREM

Let A be a $n \times n$ matrix possessing an LU factorization with $l_{ii} = 1$, $i = 1, 2, \ldots, n$. Then eqns (4)–(5) are equivalent to (1)–(2).

Proof. By induction. Using (1) for $i = 1$, (3.0), and the fact that $t_{00}^{(0)} \equiv 1$, one can immediately see that

$$u_{1j} = a_{1j} = \frac{t_{1j}^{(0)}}{t_{00}^{(0)}}, \quad 1 \le j \le n.$$

From (2) for $j = 1$ and (3.0), it is clear that

$$l_{i1} = \frac{a_{i1}}{a_{11}} = \frac{t_{i1}^{(0)}}{t_{11}^{(0)}}, \quad 1 \le i \le n.$$

Suppose that (4) is true for $1 \le i \le j \le N$ and that (5) is true for $1 \le j \le i \le N$. Then one needs to show that

$$u_{N+1\,j} = \frac{t_{N+1\,j}^{(N)}}{t_{NN}^{(N)}}, \quad j \ge N+1, \tag{6}$$

and

$$l_{i\,N+1} = \frac{t_{i\,N+1}^{(N)}}{t_{N+1\,N+1}^{(N)}}, \quad i \ge N+1. \tag{7}$$

To show (6) we start with (1) for $i = N + 1$:

Using the induction hypothesis and (3.0), we get

$$u_{N+1\,j} = t_{N+1\,j}^{(0)} - \frac{t_{N+1\,1}^{(0)} t_{1j}^{(0)}}{t_{11}^{(0)} t_{00}^{(0)}} - \frac{t_{N+1\,2}^{(1)} t_{2j}^{(1)}}{t_{22}^{(1)} t_{11}^{(1)}} - \cdots - \frac{t_{N+1\,N}^{(N-1)} t_{Nj}^{(N-1)}}{t_{NN}^{(N-1)} t_{N-1\,N-1}^{(N-1)}}.$$

The least common denominator for the first two terms, together with (3.1), yields

$$u_{N+1\,j} = \frac{t_{N+1\,j}^{(1)}}{t_{11}^{(1)}} - \frac{t_{N+1\,2}^{(1)} t_{2j}^{(1)}}{t_{22}^{(1)} t_{11}^{(1)}} - \cdots - \frac{t_{N+1\,N}^{(N-1)} t_{Nj}^{(N-1)}}{t_{NN}^{(N-1)} t_{N-1\,N-1}^{(N-1)}}.$$

Again we take a common denominator for the first two terms and use (3.2) to get

$$u_{N+1\,j} = \frac{t^{(2)}_{N+1\,j}}{t^{(2)}_{22}} - \cdots - \frac{t^{(N-1)}_{N+1\,N} t^{(N-1)}_{Nj}}{t^{(N-1)}_{NN} t^{(N-1)}_{N-1\,N-1}}.$$

Continuing this process, combining two terms at a time, one has

$$u_{N+1\,j} = \frac{t^{(N-1)}_{N+1\,j}}{t^{(N-1)}_{N-1\,N-1}} - \frac{t^{(N-1)}_{N+1\,N} t^{(N-1)}_{Nj}}{t^{(N-1)}_{NN} t^{(N-1)}_{N-1\,N-1}} = \frac{t^{(N)}_{N+1\,j}}{t^{(N)}_{NN}}.$$

In a similar fashion one proves (7).

The arithmetic complexity of the algorithm in the dense case is $\frac{1}{3}n^2$, using $3(n - 1)$ processors. In the banded case the complexity is $\frac{1}{2}bn$, using $3(n - 1)$ processors. In both cases square roots are not needed. Increasing the number of processors will reduce the complexity. For the dense case the complexity is reduced to $\frac{1}{2}n$ by using $3(n - 1)^2$ processors.

Remark concerning error analysis

Let $*$ denote any of the four arithmetic operations. Then a floating-point operation (on a computer having a double-precision accumulator) satisifes

$$fl(x * y) = x * y(1 + \delta)$$

and $|\delta| \leq \varepsilon$, where ε is a unit roundoff (see[11]). It can be shown (see[11]) that

$$fl(t^{(1)}_{ij}) = t^{(1)}_{ij}(1 + \delta_1\varepsilon + \theta_1\varepsilon),$$

and, by induction,

$$fl(t^{(m)}_{ij}) = t^{(m)}_{ij}(1 + \delta_m\varepsilon + \theta_m\varepsilon), \tag{8}$$

where $\theta_m = 2^m - 1$ and $\delta_m = \delta_{m-1} + \theta_{m-1} + 2$. Thus $\delta_m = 2^m + m - 1$. Using (4)–(5) one obtains

$$fl(u_{ij}) = u_{ij}(1 + \delta_{i+1}\varepsilon), \tag{9}$$

$$fl(l_{ij}) = l_{ij}(1 + \delta_{j+1}\varepsilon). \tag{10}$$

4. NUMERICAL EXPERIMENTS

In order to gain more insight into the performance of the algorithm presented, several examples have been selected for illustration. Since we do not have a parallel processor, we simulated it on a serial computer (VAX 11/780). In this simulation each processor is brought into the VAX 11/780 one at a time, and its program is executed. When this is completed, it is then rolled out, and the processor that is furthest back in its execution time is brought in next. The time spent by a processor consists of the program execution time and the data communication time. Here the synchronization time was negligible.

The solution times of various examples were obtained by using a time scale based on the VAX 11/780 computer. To investigate the behavior and advantages for the concurrent execution of the system simulation problems, the performance evaluation parameters chosen are (1) speed-up ratio, (2) waiting time, and (3) efficiency. The speed-up ratio is defined as the ratio of the serial solution time (TS) to the parallel solution time (TP). In any algorithm for parallel processing, the speed-up ratio should ideally be proportional to the number of processors used (p). In practice it is difficult to keep all processors busy all the time. Thus, when a processor has executed all the ready tasks, it must wait for new data from other processors for further execution. Hence, the waiting time (TW) occurs. The efficiency (EFF) represents the ratio of the difference between the total parallel solution time and the waiting time to the total parallel solution time; i.e.

$$\mathrm{EFF} = \frac{p \cdot \mathrm{TP} - \mathrm{TW}}{p \cdot \mathrm{TP}}.$$

In our first experiment we have taken a 5 × 5 matrix using a various number of processors. The parallel solution time TP, the waiting time TW, the efficiency EFF, and the speed-up ratio are given in Tables 1, 2, 3 for dense, tridiagonal, and pentadiagonal matrices, respectively. Note that time is measured in μsec.

Table 1. Dense 5 × 5 matrix; TS = 34.6

p	TP	TW	EFF (%)	Speed-up ratio
3	15.2	3	93.4	2.27
5	8.4	7.4	82.4	4.12
8	6.4	16.6	67.6	5.41

Table 2. Tridiagonal 5 × 5 matrix; TS = 10.4

p	TP	TW	EFF (%)	Speed-up ratio
3	4.0	1.6	86.7	2.60
5	2.8	3.6	74.3	3.71
8	2.4	8.8	54.2	4.33

Table 3. Pentadiagonal 5 × 5 matrix; TS = 14.0

p	TP	TW	EFF (%)	Speed-up ratio
3	5.2	1.6	89.7	2.69
5	3.6	3.6	80.0	3.83
8	2.4	5.2	72.9	5.83

In the second experiment we have taken a 10 × 10 matrix. The results are summarized in the following tables.

Table 4. Dense 10 × 10 matrix; TS = 303

p	TP	TW	EFF (%)	Speed-up ratio
3	102	3	99.0	2.97
5	61.6	5	98.4	4.92
8	38.4	12.2	96.0	7.89

Table 5. Tridiagonal 10 × 10 matrix; TS = 50.4

p	TP	TW	EFF (%)	Speed-up ratio
3	17.2	1.2	97.7	2.93
5	10.8	3.6	93.3	4.67
8	6.8	7.2	86.8	7.41

Table 6. Pentadiagonal 10 × 10 matrix; TS = 76

p	TP	TW	EFF (%)	Speed-up ratio
3	26.0	2.0	97.4	2.92
5	15.6	3.6	95.3	4.87
8	10.0	7.2	91.0	7.60

In the last experiment we have taken a 20 × 20 matrix. The results are given in the following tables.

Table 7. Dense 20 × 20 matrix; T_s = 1064

p	TP	TW	EFF (%)	Speed-up ratio
3	355.2	1.6	99.8	2.99
5	213.2	2.0	99.8	4.99
8	133.6	8.0	99.2	7.96

Table 8. Tridiagonal 20 × 20 matrix; T_s = 220.4

p	TP	TW	EFF (%)	Speed-up ratio
3	74.0	1.6	99.3	2.98
5	44.8	3.6	98.4	4.92
8	28.4	6.8	97.0	7.76

Table 9. Pentadiagonal 20 × 20 matrix; T_s = 350.0

p	TP	TW	EFF (%)	Speed-up ratio
3	117.2	1.6	99.5	2.99
5	70.8	3.6	99.0	4.94
8	44.8	9.2	97.4	7.81

Remarks

1. The efficiency goes down when the number of processors increases. The efficiency is the highest when the matrix is dense.

2. The speed-up ratio is an increasing function of the number of processors and the bandwidth.

3. The efficiency increases with the order; thus, this algorithm is good for large matrices.

Acknowledgements—The authors would like to thank the referee for his suggestions, which greatly improved the paper.

REFERENCES

1. S. C. Chen, D. J. Kuck and A. H. Sameh, Practical parallel band triangular system solvers. *ACM Trans. Math. Software* **4**, 270–277 (1980).
2. S. C. Chen and A. H. Sameh, On parallel triangular system solvers. *Proc.* 1975 *Sagamore Computer Conf. on Parallel Processing*, pp. 237–238, August (1975).
3. J. W. Huang and O. Wing, Optimal parallel triangulation of a sparse matrix. *IEEE Trans. Circuits Syst.* **CAS-26**, 726–732 (1979).
4. E. Isaacson and H. B. Keller, *Analysis of Numerical Methods*. John Wiley, New York (1966).
5. J. A. G. Jess and H. G. M. Kees, A data structure for parallel L/U decomposition. *IEEE Trans. Comput.* **C-31**, 231–239 (1982).

6. R. M. Kant and T. Kimura, Decentralized parallel algorithms for matrix computation. *Proc. Fifth Annual Symp. Computer Architecture*, pp. 96–100, Palo Alto (1978).
7. J. J. Lambiotte and R. G. Voigt, The solution of tridiagonal linear systems of the CDC STAR-100 computer. *ACM Trans. Math. Software* **1**, 308–329 (1975).
8. S. E. Orcutt, *Parallel Solution Methods for Triangular Linear System of Equations*, Technical Report No. 77, Digital Syst. Lab., Stanford Electronics Labs., Stanford, CA (1974).
9. A. H. Sameh and R. P. Brent, Solving triangular linear systems on a parallel computer. *SIAM J. Numer. Anal.* **14**, 1101–1113 (1977).
10. A. H. Sameh and D. J. Kuck, A parallel QR algorithm for symmetric tridiagonal matrices. *IEEE Trans. Comput.* **C-26**, 147–153 (1977).
11. A. H. Sameh and D. J. Kuck, On stable parallel linear system solvers. *J. Assoc. Comput. Mach.* **25**, 81–91 (1978).
12. H. S. Stone, Parallel tridiagonal equation solvers. *ACM Trans. Math. Software* **1**, 289–307 (1975).
13. Y. Wallach and V. Konrad, On block-parallel methods for solving linear equations. *IEEE Trans. Comput.* **C-29**, 354–359 (1980).
14. H. H. Wang, A parallel method for tridiagonal equations. *ACM Trans. Math. Software* **7**, 170–183 (1981).
15. O. Wing and J. W. Huang, A computation model of parallel solution of linear equations. *IEEE Trans. Comput.* **C-29**, 632–638 (1980).