Parallel Computing 19 (1993) 925–937
North-Holland

PARCO 772

Short communication

Algorithms for LU decomposition on a shared memory multiprocessor

John J. Buoni [a], Paul A. Farrell [b] and Arden Ruttan [b]

[a] *Dept. of Mathematics, Youngstown State University, Youngstown, OH 44242, USA*
[b] *Dept. of Mathematics & Computer Science, Kent State University, Kent, OH 44242, USA*

Received 31 October 1990
Revised 18 July 1992, 21 October 1992

Abstract

Buoni, J.J., Paul A. Farrell and Arden Ruttan, Algorithms for LU decomposition on a shared memory multiprocessor, Parallel Computing 19 (1993) 925–937.

In this paper we propose an improved algorithm for the parallel LU decomposition of an $(m+1)$-banded upper Hessenberg matrix on a shared memory multi-processor, which requires $O(2nm^2/p)$ parallel operations, where n is the dimension of the matrix and p is the number of processors. We show that for the special case of tridiagonal matrices this algorithms has a lower operation count than those in the literature and yields the best existing algorithm for the solution of tridiagonal systems of equations.

Keywords. Shared memory multiprocessor; LU decomposition; Hessenberg matrices; tridiagonal matrices; parallel algorithm.

1. Introduction

A number of authors over the last two decades have written on parallel algorithms for solving tridiagonal systems (See, e.g. [3,5,6,8–12]). We present a tridiagonal solver, which has a lower operation count than those in the literature for solving a single system. In addition, there are a number of common numerical situations, such as in the ADI method, where one needs to solve tridiagonal systems of the form $Ax = b_i$, $1 \le i \le k$, where A is known *ab initio* but the b_i's are not all known at the start of the computation, but rather arise as a result of an iteration process. The method presented here, since it produces an explicit LU decomposition, has added advantages in these cases.

2. LU decomposition algorithm

We shall, in fact, consider the LU decomposition of an $n \times n$ upper Hessenberg matrix, since the analysis is not significantly more difficult and further the additional generality leads

Correspondence to: A. Ruttan, Dept. of Mathematics & Computer Science, Kent State University, Kent, OH 44242, USA.

to insights, which produce a more efficient algorithm. Let $A = (a_{ij})$ be a banded $n \times n$ upper Hessenberg matrix with band width $m+1$, i.e. $a_{ij} \neq 0$ only when $\max\{1, i-1\} \leq j \leq \min\{n, m+i-1\}$, $1 \leq i \leq n$. We will assume that the LU decomposition is required for use in an iterative method of the type described in section 1. Of course, the LU decomposition could also be used for the solution of any linear systems or to find the eigenvalues of A using the LR method [7]. In any case, it suffices to consider the case where $a_{i+1,i} \neq 0$, $1 \leq i \leq n-1$, since otherwise the matrix is reducible, and we may consider the LU decomposition of the subproblems resulting from the reduction. Throughout this paper we will use the convention that any element with a nonpositive index has value zero.

As in most algorithms for shared memory multiprocessors, the object here is to partition the problem into a number of subproblems suitable for solution by tasks running on the available processors. We shall consider the general case of p processors, where $p \leq n$, but is not otherwise related to the size of the matrix problem. That is, in particular, we do not require that p be much less than n, or of order n, or much greater than n. We restrict consideration, in the paper, to the shared memory case since we do not wish to introduce considerations of communication complexity. On shared memory multiprocessors, these do not exist and analysis is performed solely in terms of computational complexity, that is the number of arithmetic operations. For simplicity and compatibility with the analysis of other similar algorithms in the literature, we shall consider all floating point operations as taking the same time.

Assuming that A has an LU factorization

$$A = LU,$$

where L is a unit lower triangular $n \times n$ matrix and $U = (u_{ij})$ is $n \times n$ upper triangular, it is readily verified that L is actually a bi-diagonal matrix, with unit diagonal, and U is an upper triangular banded matrix of bandwidth m. The special form of L allows one to readily determine L^{-1}. One finds that $L^{-1} = (\hat{l}_{ij})$ is an $n \times n$ lower triangular matrix given by

$$\hat{l}_{ij} := \begin{cases} \prod_{t=j+1}^{i} (-l_t) & i \geq j \\ 0 & i < j. \end{cases} \tag{1}$$

Using (1) and forming $U = L^{-1}A$, one obtains, for $1 \leq i, j \leq n$

$$u_{ij} = \sum_{s=j-m+1}^{\min\{i,j+1\}} \hat{l}_{is} a_{sj} = \sum_{s=j-m+1}^{\min\{i,j+1\}} \prod_{t=s+1}^{i} (-l_t) a_{sj}. \tag{2}$$

As in the tridiagonal case, the well known substitution (cf. [4], pp. 473–474)

$$\begin{aligned} &y_1 = 1 \\ &l_i = y_{i-1}/y_i \quad i = 2, 3, \ldots, n \end{aligned} \tag{3}$$

can be used to simplify (2). Equation (2) reduces to

$$u_{ij} = \sum_{s=j-m+1}^{\min\{i,j+1\}} (-1)^{i-s} y_s a_{sj}/y_i, \quad 1 \leq i, j \leq n. \tag{4}$$

But $u_{j+1,j} = 0$, for $1 \leq j \leq n-1$, and therefore (4) yields the following linear systems in the unknowns y_i:

$$\begin{aligned} &y_1 = 1 \\ &y_j = 0, \ j \leq 0, \\ &y_{j+1} a_{j+1,j} = \sum_{s=j-m+1}^{j} (-1)^{j-s} y_s a_{sj}, \quad 1 \leq j \leq n-1. \end{aligned} \tag{5}$$

Actually, (5) defines an $m+1$ banded triangular linear system

$$Ty = e \tag{6}$$

where $y = (y_1, y_2, \ldots, y_n)^T$, $e = (1, 0, \ldots, 0)^T$, and $T = (t_{ij})_{i,j=1}^{n}$, is given by:

$$t_{ij} = \begin{cases} 1 & \text{if} \quad i = j = 1, \\ (-1)^{i-j} a_{j,i-1} & j \le i,\ i > 1, \\ 0 & j > i. \end{cases}$$

Thus the problem of finding an LU factorization of an upper Hessenberg matrix reduces to solving the banded triangular system described in (6) to obtain the y_i's and then using the solution of that system to evaluate L and U. In practice, L may be determined from Eq. (3). To determine U, note first that the elements of the $(m-1)$st super-diagonal, $u_{i,i+m-1}$ satisfy $u_{i,i+m-1} = a_{i,i+m-1}$, for $i = 1, \ldots, n-m+1$. Also $u_{1,j} = a_{1,j}$ for $j = 1, \ldots, m$. Thus these elements do not require any calculation. The $(m-2)$nd super-diagonal may then be calculated from the $(m-1)$st using

$$l_i u_{i-1,m+i-2} + u_{i,m+i-2} = a_{i,m+i-2}, \quad i = 2, \ldots, n-m+2.$$

Similarly, the jth super-diagonal is given in terms of the $(j+1)$st by

$$l_i u_{i-1,j+i} + u_{i,j+i} = a_{i,j+i}, \quad i = 2, \ldots, n-j. \tag{7}$$

Note that each element depends only on a single element of the next super-diagonal and on known values from L and A. Thus, the calculation of each super-diagonal of U is perfectly parallizable. In fact, the main diagonal may be obtained using 1 division rather than the multiplication and subtraction in (7) by

$$u_{i,i} = a_{i+1,i}/l_{i+1}, \quad i = 2, \ldots, n. \tag{8}$$

Further, the calculation of the super-diagonals can be chained, since the dependency graph for U, using (7) and (8), is

$$U := \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \cdots & \bullet & 0 & 0 & 0 & \cdots & 0 \\ 0 & \bullet & \downarrow & \downarrow & \cdots & \downarrow & \bullet & 0 & 0 & \cdots & 0 \\ 0 & 0 & \bullet & \downarrow & \cdots & \downarrow & \downarrow & \bullet & 0 & \cdots & 0 \\ & & \ddots & & & & & & & \vdots & \\ \vdots & & & \ddots & & & & \ddots & & & \bullet \\ 0 & & & & & & & \ddots & & & \downarrow \\ \vdots & & & & & & & & & & \downarrow \\ & & & & & & & & & \ddots & \ddots \\ 0 & \cdots & \cdots & \cdots & . & . & \cdots & \cdots & \cdots & 0 & \bullet \end{bmatrix} \tag{9}$$

where $\bullet$ indicates that the value is already known or can be calculated from L and A, and $\downarrow$ indicates that the value at the tip of the arrow requires that at the end, in addition to values from L and A.

The calculation of L using (3) requires $n-1$ divisions. From (7), the elements of the jth super-diagonal, for $j > 0$ can be calculated using 1 multiplication and 1 subtraction, and from (8) those of the main diagonal can be calculated in 1 division. It is clear that the number of elements in the jth super-diagonal of U is $n-j$ for $j = 0, \ldots, m-1$. Note that for all

elements of the $(m-1)$st super-diagonal and for the first element of each of the other super-diagonals no calculations are necessary. Hence the total computations for U is

$$(n-1)+2\sum_{j=1}^{m-2}(n-j-1).$$

This evaluates to

$$n(2m-3)-(m^2-m-1). \tag{10}$$

The latter term is negative for $m \geq 1$. Hence we get the following upper bound for the complexity of calculating U

$$n(2m-3).$$

Note that in the case of a tridiagonal system, $m=2$, (10) reduces exactly to $n-1$. Hence using p processors L and U can be calculated in the general case in

$$\left\lceil \frac{2n(m-1)}{p} \right\rceil \tag{11}$$

parallel operations, and in the special tridiagonal case in

$$\lceil 2(n-1)/p \rceil \tag{12}$$

parallel operations.

There remains only the solution of the triangular system $Ty=e$.

3. Algorithms for the triangular system

It should be noted that a significant portion of the time in the LU decomposition algorithm is spent in solving the triangular system (6). The major drawback to the algorithm given in [6] and the generalization of that algorithm given in [1] is that it requires a large number of $m \times m$ matrix products. In [2], Chen et al. describe an algorithm for solving a banded triangular system which requires substantially fewer $m \times m$ matrix products. The special nature of the triangular system given in (6) allows for certain simplifications of the original algorithm, which improve the computational complexity. We shall give here the general form of the algorithm for p processors. If we restrict the values of p, as in [1], a slightly better time complexity results.

We begin by partitioning T into g blocks, each of which is $s \times s$, where $g \geq 2$ and g is a power of 2. Thus $s = n/g \geq m$ and we may write

$$T=\begin{bmatrix} L_1 & & & & & & \\ U_2 & L_2 & & & & & \\ & U_3 & L_3 & & & & \\ & & U_4 & L_4 & & & \\ & & & \ddots & \ddots & & \\ & & & & U_g & L_g \end{bmatrix}$$

where L_i, $i=1,2,\ldots,g$ is an $s \times s$ lower triangular matrix with bandwidth m and U_i, $2 \leq i \leq g$ is an $s \times s$ upper triangular matrix, with nonzero entries only in the last m columns.

The algorithm begins by multiplying equation (6) by $D = \operatorname{diag}(L_1^{-1}, L_2^{-1}, \ldots, L_g^{-1})$ giving $DTy = De$ where

$$DT = \begin{bmatrix} I_s & & & & & & \\ L_2^{-1}U_2 & I_s & & & & & \\ & L_3^{-1}U_3 & I_s & & & & \\ & & L_4^{-1}U_4 & I_s & & & \\ & & & \ddots & \ddots & & \\ & & & & & L_g^{-1}U_g & I_s \end{bmatrix},$$

$$De = \begin{bmatrix} L_1^{-1} & & & & & \\ & L_2^{-1} & & & & \\ & & L_3^{-1} & & & \\ & & & \ddots & & \\ & & & & \ddots & \\ & & & & & L_g^{-1} \end{bmatrix} \begin{bmatrix} v_s \\ 0 \\ \vdots \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

and v_s is the s-vector $(1, 0, \ldots, 0)^{\mathrm{T}}$.

To compute this, one needs to determine $L_1^{-1}v_s$, $L_2^{-1}U_2, \cdots, L_g^{-1}U_g$. Since each diagonal element is involved in one division and each off diagonal element in one multiplication and one addition/subtraction, a careful analysis of the standard sequential algorithm [7, p. 108] shows that $L_1^{-1}v_s$ can be computed in $(2m-1)s - m^2$ operations using one processor. Since each U_j, $2 \le j \le g$ has only m nonzero columns, to compute $L_j^{-1}U_j$ one must solve m independent systems of the form $L_j x = c$. Therefore there are a total of $(g-1)m + 1$ such systems to be solved. Using p processors this gives $\lceil (mg - m + 1) \rceil$ systems per processor. This can be accomplished in

$$\lceil (mg - m + 1)/p \rceil \lceil (2m-1)n/g - m^2 \rceil \tag{13}$$

operations. Note that each processor is performing individual sequential operations here and hence the only loss of efficiency is the failure to employ up to $p-1$ of the processors at the end.

Since the first $s-m$ columns of U_j, $2 \le j \le k$ are zero, so are the first $s-m$ columns of $L_j^{-1}U_j$, $2 \le j \le g$. If we reblock the system $DTy = De$ into alternating $(s-m) \times (s-m)$ and $m \times m$ blocks we obtain

$$DT = \begin{bmatrix} I_{s-m} & & & & & & & & \\ 0 & I_m & & & & & & & \\ 0 & P_2 & I_{s-m} & & & & & & \\ 0 & Q_2 & 0 & I_m & & & & & \\ & & 0 & P_3 & \ddots & & & & \\ & & 0 & Q_3 & \ddots & I_{s-m} & & & \\ & & & & \ddots & 0 & I_m & & \\ & & & & & 0 & P_g & I_{s-m} & \\ & & & & & 0 & Q_g & 0 & I_m \end{bmatrix}$$

where $Is - m$ and I_m are $(s-m)\times(s-m)$ and $m\times m$ identity matrices respectively. Blocking the vectors y and De in the same fashion yields $y=(e_1, f_1, e_2, f_2, \cdots, e_g, f_g)^{\mathrm{T}}$, and $De=(e_1, f_1, 0, 0, \cdots 0)^{\mathrm{T}}$ where e_i and f_i are $(s-m)$ and m vectors respectively.

The system $DTy = De$ can be decomposed into

$$-P_i f_{i-1} = e_i \quad i = 2, \ldots, g$$
$$Q_i f_{i-1} = -f_i \quad i = 2, \ldots, g.$$

Note that $f_i = (\prod_{j=2}^{i}(-Q_j))f_1$. Using a variation of the algorithm in [1], one calculates $f_2, f_3, \cdots f_g$. This requires $k = \log g$ steps each involving products of the form Q_jQ_k or Q_jf_j. To be precise, in the ith stage, $0 \le i \le k-1$, there are 2^i products of the form Q_jf_j, that is matrix-vector products, and $2^i(g/2^{i+1}-1)$ products of the form Q_jQ_k, that is matrix-matrix products. Thus, there are the equivalent of $m[2^i + 2^i(g/2^{i+1}-1)m]$ inner products, each of size m. These are assigned uniformly across the p processors. Each takes $(2m-1)$ operations, using 1 processor. Hence the time for the ith stage is:

$$(2m-1)\left\lceil \frac{m(mg/2 - 2^i(m-1))}{p} \right\rceil$$

Thus, summing over all the stage, the total operations to obtain the f_i's is

$$(2m-1)\sum_{i=0}^{k-1}\left\lceil \frac{m(mg/2 - 2^i(m-1))}{p} \right\rceil \tag{14}$$

The final stage of this algorithm involves calculating the e_i's $i = 2, \ldots, g$. Since P_i is an $(s-m)\times m$ matrix, the calculation of P_if_{i-1} is equivalent to $s-m$ inner products of length m. Since there are $(g-1)$ products, P_if_{i-1}, there are $(g-1)(n/g-m)$ inner products of length m, which will require a maximum of

$$(2m-1)\left\lceil \frac{n - m(g-1) - n/g}{p} \right\rceil \tag{15}$$

operations per processor. Combining (13), (14) and (15) one finds that using this algorithm, one can solve the banded triangular system (6) in

$$\left\lceil \frac{mg-m+1}{p} \right\rceil \left[(2m-1)n/g - m^2\right] + (2m-1)\sum_{i=0}^{k-1}\left\lceil \frac{m(0.5mg - 2^i(m-1))}{p} \right\rceil + (2m-1)\left\lceil \frac{n - m(g-1) - n/g}{p} \right\rceil \tag{16}$$

operations. Combining (11) and (17) we find that one can calculate the LU decomposition of an $(m+1)$ banded upper Hessenberg $n\times n$ matrix using p processors in

$$\left\lceil \frac{mg-m+1}{p} \right\rceil \left[(2m-1)n/g - m^2\right] + (2m-1)\sum_{i=0}^{k-1}\left\lceil \frac{m(0.5mg - 2^i(m-1))}{p} \right\rceil + (2m-1)\left\lceil \frac{n - m(g-1) - n/g}{p} \right\rceil + \left\lceil \frac{2n(m-1)}{p} \right\rceil. \tag{17}$$

It is clear that the time complexity here depends not only on the size of the matrix n, the number of bands m, and the number of processors p, but also on the size and number of blocks g used in the algorithm. If one chooses n, m, and p to be powers of 2 and sets $g \equiv p/m$ as in [2], then an upper bound for the time complexity can be readily determined. Indeed, since $mg - m + 1 < mg \equiv p$, (13) reduces to

$$(2m-1)n/g - m^2 = (2m-1)nm/p - m^2. \tag{18}$$

Likewise, since

$$\frac{m(mg/2 - 2^i(m-1))}{p} < \frac{m(mg/2)}{p} \equiv \frac{m}{2}, \tag{19}$$

Eq. (14) becomes

$$(2m-1)\frac{m}{2}\log(g) \equiv (2m-1)\frac{m}{2}\log(p/m). \tag{20}$$

Similarly for (15), since $(g-1)(n/g - m) < n - mg$, we have

$$(2m-1)\left(\frac{n}{p} - 1\right). \tag{21}$$

Therefore an upper bound for the time complexity is

$$(2m^2 + 3m - 3)\frac{n}{p} + \left(m^2 - \frac{m}{2}\right)\log(p/m) - m^2 - 2m + 1. \tag{22}$$

For the tridiagonal case (i.e. $m = 2$), the time complexity to produce an LU factorization is

$$11\frac{n}{p} + 3\log(p/2) - 9. \tag{23}$$

When the number of processors $p \ll n$, the first term dominates. For a tridiagonal matrix this becomes

$$11\frac{n}{p} + O(1). \tag{24}$$

Thus we find that the above algorithm is always preferable to that in [1].

Finally, one can complete the solution of the tridiagonal problem $Ax = b$, using the algorithm given in [2], which is valid for all $p \leq n$. This algorithm solves a unit triangular system of bandwidth 2 in $5(n/p) + 2\log(p) - 5$ operations (cf. [2], p. 274). Thus, including the normalization of U and b, we can solve $Uz = b$ and $Lx = z$ in at most

$$12\frac{n}{p} + 4\log(p) - 10. \tag{25}$$

Note that, whenever $p > 3 \times 2^{1/3} \approx 3.78$, this is better than the method in [6], which also involves a large number of matrix products.

The total time to solve a tridiagonal system is at most

$$23\frac{n}{p} + 7\log(p) - 19. \tag{26}$$

While the choice of $g = pm$ simplifies the complexity analysis it is doubtful that one should make that choice in practice. In fact, the numerical experiments described in the next section indicate that one should probably choose $g \approx n/m$.

4. Numerical results

We shall consider the implementation of this algorithm on a shared memory multiprocessor. Most current machines of this type, which are in widespread use, are of the MIMD type with local caching to reduce bus contention. Among these are the Encore Multimax, the Sequent Balance and Symmetry, which have specially designed bus architectures. Such architectures are now widely believed to be limited to approximately 30 processors before bus contention becomes such as to cancel the performance enhancement of adding more processors.

We implemented the algorithm on an Encore Multimax 320 and a Sequent Balance B21000. These processors both have efficient caching, which reduces the effect of possible bus contention.

The algorithms described above was first run on the Argonne National Laboratories Advanced Research Computing Facility's Encore Multimax 320 with 20 processors. Each processor is a National Semiconductor 32332 chip running at 15 MHz. The Encore operating system only permits the allocation of processes, which may or may not run on separate processors. Thus care was taken to undertake the timings when the load on the system was very low (less than 0.10) and no more than 16 processes were requested. In such circumstances the processes can normally be guaranteed to run on separate processors. Even then, it is clear that at 16 processors, in a multiprocessor environment of only 20 processors we are in severe danger of suffering contention for one or more of the processors. This leads to slight degradation of performance over that predicted, when using a high proportion of the processors. The code was written in Encore Parallel Fortran (epf). This compiler provides an auto parallelization feature which will attempt to run code using multiple processors without explicit directives from the user. The implementation of the algorithm did not use this facility but explicitly scheduled tasks using the *parallel, doall* and *barrier* constructs. The timings were undertaken for a matrix of dimension 250000. Because of the limited number os processors available, the algorithm was tested only for the tridiagonal case ($m = 2$). We also include the *efficiency* of the algorithm which is defined as:

$$\text{efficiency} = \frac{T_s}{p \times T_p},$$

where T_s is the time for a sequential version of the algorithm and T_p is the time for the algorithm on p processors.

The timings given in *Table 1* indicate that the algorithm does in practice require time inversely proportional to the number of processors used and hence shows a linear speedup. The efficiency figures reflect the fact that the algorithm was run in a multiprocessor environment with swapping and bus contention.

More extensive tests were then undertaken, on the Sequent Balance B21000, in the Department of Mathematics and Computer Science at Kent State. This has 26 processors, each with a 32-bit National Semiconductor NS32032 capable of 0.75 MIPS, and 32 Mb of

Table 1
Execution time for tridiagonal case, with $T_s = 50.9$

Processors	User time	Efficiency
4	15.8	80.5
8	8.0	79.6
16	4.3	74.0

shared memory. The Balance operating system, DYNIX, provides the ability to bind processes to processors, using the processor affinity facility, and also a utility `team`, which modifies system parameters to permit more accurate timings. The latter gives the highest priority to the program and disables swapping, page fault frequency adjustments and process aging. These privileges allow a user program to execute with a minimum of system overhead to distort benchmark times. This facility was employed in all the timings given below, and effectively eliminated contention for the processors, although bus contention was still noticeable for more than 16 processors. The coding used the Sequent parallel directives to parallelize do loops in the Fortran code, library calls to the microsecond clock for the timings and to the parallel processing library to set and manage the number of processors. The overhead of these operations was negligible. The timings do not include the initial time to spawn the processes, since in a practical situation these would normally be spawned in the code preceding the call to the LU decomposition routine.

Extensive tests were undertaken for matrices with bandsize m ranging from 2 to 10, and for matrix sizes, n, from 2560 to 98304. Some representative results are presented below in *Fig. 1* which, on the left gives the speedup for a problem for a matrix of size 98304 and bandwidth 3, and on the right gives the speedup for a problem for a matrix of size 16384 and bandwidth 8. Speedup, S_p, is defined as

$$S_p = \frac{T_s}{T_p},$$

where T_p is the time on p processors and T_s is the time for the sequential version of the algorithm. This shows that, until bus contention becomes noticeable, the speedup is nearly linear, independent of the matrix size and bandwidth. This proved to be the case for all matrices tested, provided the blocksize s was chosen appropriately.

A further check was made to determine whether the degradation was due to the implementation of the algorithm or to unavoidable contention among memory references over the shared bus. To quantify the latter, the same parallel programming directives and library calls were used to parallelize a matrix-matrix multiply operation on matrices of dimension 100×100 and 200×200. These are perfectly parallelizable operations and should show the maximum possible speedup for the machine (without specific programming to maximize cache access operations). In theory this should be linear. In practice, they showed almost identical dropoff, as illustrated by the *matrix multiply* line in *Fig. 1*. We conclude therefore, that the implementation attains the maximum speedup possible on the Sequent, in the presence of bus contention.

It is also to be noted that the time for the algorithm depends not only on the size and bandwidth of the matrix, but also on the blocksize s chosen. In general, the larger the number of processors the smaller the blocksize should be, although this is not a linear relationship, as can be seen in *Fig. 2(a)*, which gives the optimum blocksize for a matrix with $n = 98304$ and $bands = 3$. The time varies considerably with the block size. This may be seen in *Fig. 2(b)*, which shows the time for a fixed matrix with $n = 16384$ and $bands = 8$, using different blocksizes.

5. Conclusion

The method described does in fact achieve linear speedup as indicated by the tables and figures in Section 4.

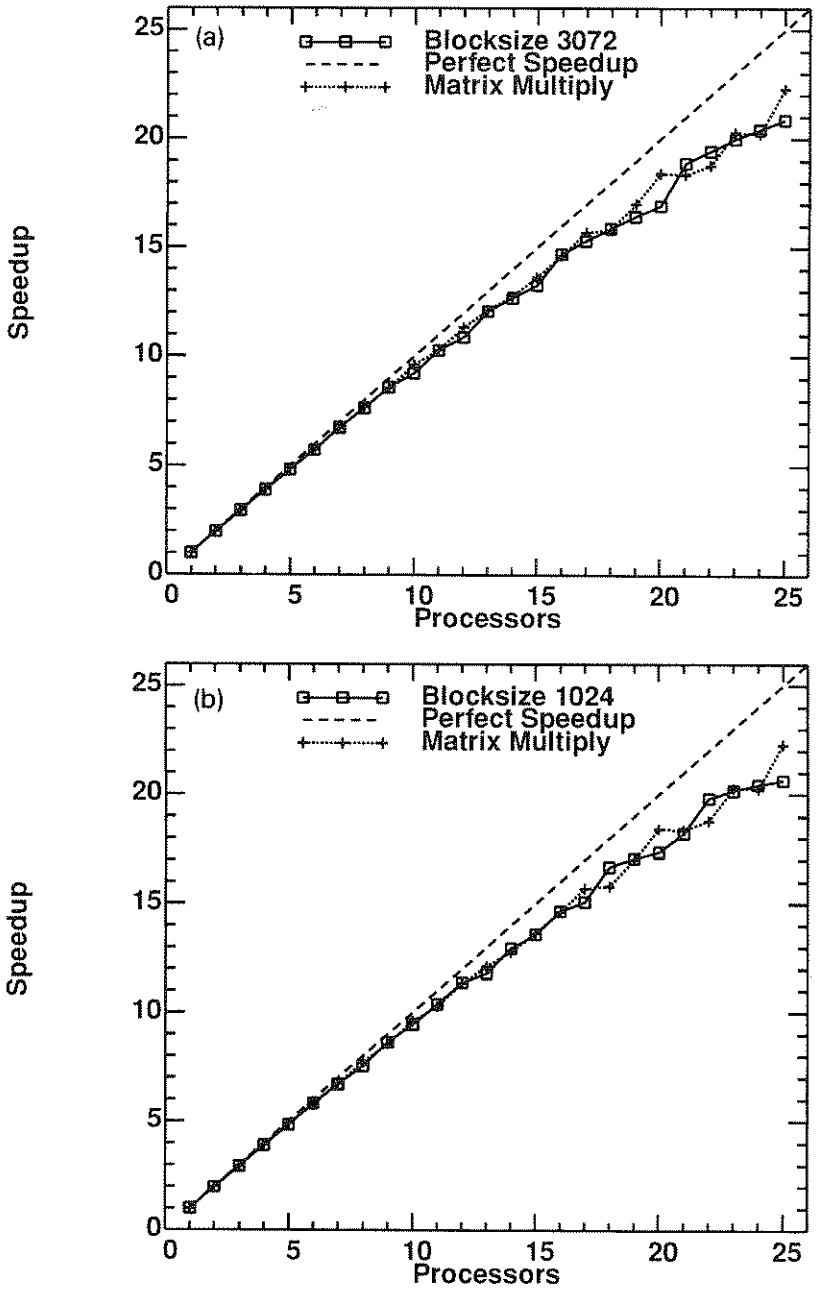

Fig. 1. Speedup for matrices with (a) $n = 98304$, *bands* = 3 and (b) $n = 16384$, *bands* = 8.

In the tridiagonal case, the algorithm is not only better than existing algorithms in the literature for LU decomposition, but also has better computational complexity for the solution of a single tridiagonal system, as indicated in *Table 2*, where $n' = n + 1 = 2^t$.

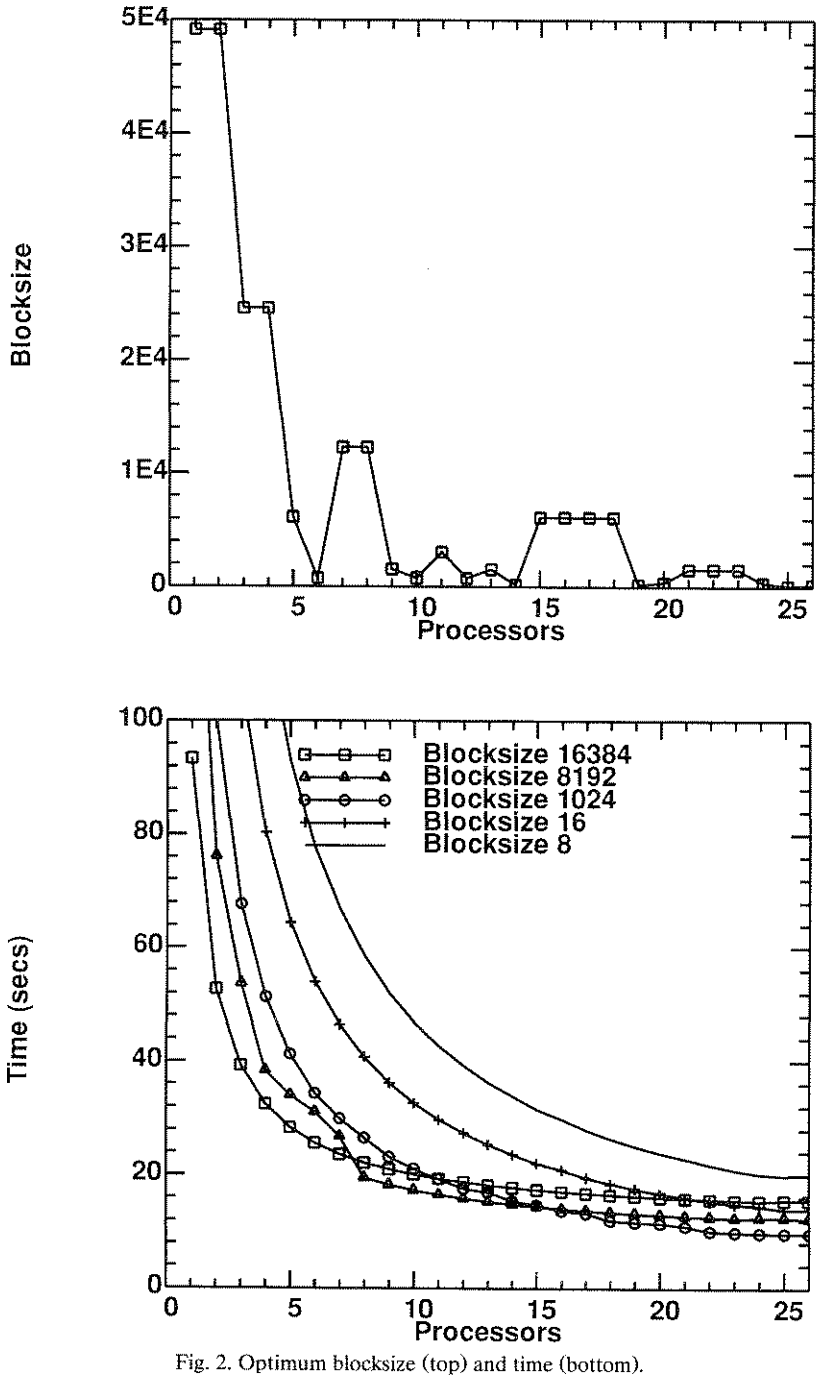

Fig. 2. Optimum blocksize (top) and time (bottom).

In addition, comparing it with methods for solving equations, such as those which arise in ADI [1, section 1], which do not perform the decomposition but rather solve the n subproblems anew at each iteration, a further improvement in computational efficiency results, since

Table 2
Complexity of the solution of a single linear system for tridiagonal matrices

Method	Processors	Time	Speedup	Efficiency
Serial Gaussian elimination	1	8n	–	–
Recursive doubling [4]	n	$24 \log n$	$\frac{n}{3 \log n}$	$\frac{1}{3 \log n}$
Odd-Even reduction [4]	$n'/2$	$19 \log n' - 14$	$\frac{8n}{19 \log n' - 14}$	$\frac{16}{19 \log n' - 14}$
Odd-Even elimination [4]	n'	$14 \log n' + 1$	$\frac{8n}{14 \log n' + 1}$	$\frac{14}{14 \log n' + 1}$
Lakshmivarahan Dhall [6]	$n/2$	$18 \log n$	$\frac{4n}{9 \log n}$	$\frac{8}{9 \log n}$
Lakshmivarahan Dhall [6]	p $3 \le p \le \frac{3n}{4}$	$(n/p)[25+9 \log p/3]-3$	$\frac{8p}{[25+9 \log p/3]-3p/n}$	$\frac{8}{[25+9 \log p/3]-3p/n}$
Buoni, Farrell, Ruttan [1]	p	$(n/p)[47/2+6 \log p/3]$	$\frac{8}{47/2+6 \log p/3}$	$\frac{8}{47/2+6 \log p/3}$
Improved algorithm	p $1 \le p \le n$	$23\frac{n}{p}+7 \log(p)-19$	$\frac{8p}{23+(7 \log(p)-19)\frac{p}{n}}$	$\frac{8}{23+(7 \log(p)-19)\frac{p}{n}}$

one need only perform the forward and back solves rather then performing the full elimination.

Although all the machines considered were of MIMD shared memory type the algorithm could, in fact, be executed, achieving the same linear speedup on a SIMD shared memory processor if such existed, since the algorithm can easily be reformulated so that each processor performs the same operations, with access to the shared data structure indexed by the group (partition) related to each processor.

References

[1] J. Buoni, P.A. Farrell and A. Ruttan, Parallel LU decomposition of Upper Hessenberg Matrices, in: C. Brezinski and U. Kulisch, eds., *Comput. & Appl. Math. I - Algor. & Theor* (1992) 61–70

[2] S.C. Chen, D.J. Kuck and A.H. Sameh, Practical parallel band triangular system solvers, *ACM TOMS* 4(3) (1978) 270–277.

[3] D.B. Gannon and J. van Rosendall, On the impact of communication complexity on the design of parallel numerical algorithms, *IEEE Trans. Comput.* 33 (1984) 1180–1194.

[4] R.W. Hockney and C.R. Jesshope, *Parallel Computers 2 - Architecture, Programming and Algorithms* (Hilger, Bristol, 1988).

[5] S.L. Johnsson, Solving tridiagonal systems on ensemble architectures, *SIAM J. Sci. Stat. Comput.* 8 (1987) 354–392.

[6] S. Lakshmivarahan and S.K. Dhall, A new class of parallel algorithms for solving tridiagonal systems, *IEEE Fall Joint Computer Conf.* (1986) 315–324.

[7] G.W. Stewart, *Introduction to Matrix Computation* (Academic Press, New York, 1973) 441.

[8] H.S. Stone, An efficient parallel algorithm for the solution of a tridiagonal system of equations, *J. ACM* 20 (1973) 27–38.

[9] H.S. Stone, Parallel tridiagonal equation solvers, *ACM Trans. Software* 1 (1975) 289–307.
[10] H.A. van de Vorst, Large tridiagonal and block tridiagonal linear systems on vector and parallel computers, *Parallel Comput.* 5 (1987) 45–54.
[11] H.A. van der Vorst, Analysis of a parallel solution method for tridiagonal systems, *Parallel Comput.* 5 (1987) 303–311.
[12] H.H. Wang, A parallel method for tridiagonal equations, *ACM Trans. Math. Software* 7 (1981) 170–183.