REPRINTED FROM:

Computational and Applied Mathematics, I

Algorithms and Theory

Selected and revised papers from the
IMACS 13th World Congress,
Dublin, Ireland, July 1991

edited by

C. BREZINSKI
Laboratoire d'Analyse Numérique
et d'Optimisation
Villeneuve d'Ascq, France

U. KULISCH
Institut für Angewandte Mathematik
Universität Karlsruhe
Karlsruhe, Germany

1992

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

Computational and Applied Mathematics, I
C. Brezinski and U. Kulish (Editors)
Elsevier Science Publishers B.V. (North-Holland)

Parallel LU decomposition of upper hessenberg matrices on a shared memory multiprocessor

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

[a]Department of Mathematics, Youngstown State University, Youngstown, OH 44555, U.S.A.

[b]Department of Mathematics & Computer Science, Kent State University, Kent, OH 44242, U.S.A.

Abstract

We propose an algorithm for the parallel LU decomposition of an upper Hessenberg matrix on a shared memory multi-processor. We consider the general case of p processors, where p is not related to the size of the matrix problem. We show that the LU decomposition of an $(m+1)$-banded Hessenberg matrix can be achieved in $O(\frac{3nm^2}{p})$ operations, where n is the dimension of the matrix and p is the number of processors. For tridiagonal matrices this algorithm has a lower operation count than those in the literature and yields the best existing algorithm for the solution of tridiagonal systems of equations.

1. INTRODUCTION

A number of authors over the last two decades have written on parallel algorithms for solving tridiagonal systems. (See, for instance [1],[5],[4],[8],[9],[11],[12], [13],[2]). These articles have considered the problem of solving tridiagonal systems for the form $Ax = b_i$, $1 \leq i \leq k$ where A and all of the b_i, are known at the start of the process. In such cases, the computations can be arranged to produce highly efficient parallel solutions to all k systems simultaneously. It should be noted, however, that there are a number of common numerical situations, for example the ADI method, where one needs to solve tridiagonal systems 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. For instance, consider the classical situation for the application of the ADI method [10]:
A five-point centered difference approximation of an elliptic partial differential equation of the form

$$-(K_1(x)u_x(x,y))_x - (K_2(y)u_y(x,y))_y + \sigma u(x,y) = S(x,y), \text{ where } (x,y) \in R,$$

R is the unit rectangle, $0 \leq x, y \leq 1$, $u(x,y) = \gamma(x,y)$ on the boundary of R, and K_1 and K_2 are continuous and positive on R.

If the mesh spacing for the five-point centered difference approximation is taken to be $h = \frac{1}{n+1}$, where n is a positive integer, then the ADI method becomes the following matrix iteration:

$$\begin{aligned}(H_1 + r_{m+1}I)u_{m+\frac{1}{2}} &= (r_{m+1}I - V_1)u_m + k \\ (V_1 + r_{m+1}I)Pu_{m+1} &= (r_{m+1}I - H_1)Pu_{m+\frac{1}{2}} + k\end{aligned}$$

where the matrices H_1 and V_1 are $n^2 \times n^2$ block diagonal matrices of the form

$$\begin{aligned}H_1 &= diag(E, E, \ldots, E) \\ V_1 &= diag(F, F, \ldots, F),\end{aligned}$$

with E and F known $n \times n$ tridiagonal matrices, I is identity matrix, P a known permutation matrix and $\{r_m\}$ a sequence of known constants. Because of the special structure of the problem, each ADI iteration may be written in the form

$$(E + r_m I)(u_{m+\frac{1}{2}})^{(j)} = (r_m I - F)(u_m)^{(j)} + (k)^{(j)} \quad (1)$$

$$(F + r_m I)P(u_{m+1})^{(j)} = (r_m I - E)P(u_{m+\frac{1}{2}})^{(j)} + (k)^{(j)} \quad (2)$$

for $1 \leq j \leq n$, $m = 1, 2, \ldots$ where $(u_m)^{(j)}$ and $(k)^{(j)}$ are the j^{th} components of u and k, respectively, relative to the blocking of H_1 and K_1 given above. Thus, for example, $(k)^{(j)}$ consists of components $(j-1)n + 1$ to jn of the vector k, and similarly for $(u_m)^{(j)}$.

From the form of (1) and (2), it is easily seen that for both equations one must solve n tridiagonal systems of the form $Ax = b_i$, $1 \leq i \leq n$, where the n righthand sides are known. This n–fold parallelism prompts one to consider attacking (1) and (2) using a parallel algorithm. In a sequential environment, it is most efficient to factor the matrices $(E + r_m I)$ and $(F + r_m I)$ into LU form and then to solve the system using the factored form. We shall demonstrate an algorithm for LU decomposition in the case of shared memory MIMD computers, which is also more efficient than the existing direct methods of solution on such architectures.

Moreover, it appears at first glance that there are an infinite number of matrices $(E + r_m I)$ and $(F + r_m I)$ for $m = 0, 1, \ldots$. In practice, however, the constants r_m, $m = 0, 1, \ldots$, are chosen to repeat after a short cycle of iterates – usually either 2, 4, or 8. From that perspective one usually has at most 16 $n \times n$ tridiagonal matrices, $(E + r_m I)$, $m = 0, 1, \ldots, 7$ and $(F + r_m I)$, $m = 0, 1, \ldots, 7$ which will be used repeatedly to solve (1) and (2).

Since the cost of storing 16 $n \times n$ factored tridiagonal matrices is small compared to the storing of the n^2 dimensional vector u (provided n is moderately large), in this context, it makes sense to factor and store those matrices at the start of the iteration. Thus the approach based on LU decomposition has the added advantage, over direct solution, that the factorization need only be performed once. Each subsequent iteration, with the same value of r_m, requires only the forward and back solve, thus reducing the cost further.

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 the additional generality leads 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]. 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 is not 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 this 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.

We assume that A is known to have an LU factorization, $A = LU$, where L is a unit lower bi-diagonal $n \times n$ matrix and $U = (u_{ij})$ is a banded $n \times n$ upper triangular, with m non-zero diagonals, including the main diagonal. The special form of L allows one to readily determine L^{-1}. One finds that $L^{-1} = (\hat{\ell}_{ij})$ is an $n \times n$ lower triangular matrix given by

$$\hat{\ell}_{ij} := \begin{cases} \prod_{t=j+1}^{i} (-\ell_t) & i \geq j \\ 0 & i < j. \end{cases} \tag{3}$$

Thus the elements of $U = L^{-1}A$, satisfy $1 \leq i, j \leq n$

$$u_{ij} = \sum_{s=j-m+1}^{\min\{i,j+1\}} \hat{\ell}_{is} a_{sj} = \sum_{s=j-m+1}^{\min\{i,j+1\}} \prod_{t=s+1}^{i} (-\ell_t) a_{sj}. \tag{4}$$

As in the tridiagonal case, the well known substitution (*cf.* [3], pp. 473 – 474)

$$\begin{aligned} y_1 &= 1 \\ \ell_i &= y_{i-1}/y_i \qquad i = 2, 3, \cdots, n \end{aligned} \tag{5}$$

can be used to simplify (4), giving

$$u_{ij} = \sum_{s=j-m+1}^{\min\{i,j+1\}} (-1)^{i-s} y_s a_{sj}/y_i, \quad 1 \leq i, j \leq n. \tag{6}$$

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

$$\begin{aligned} &y_j = 0, \ \ j \leq 0 \quad , \quad y_1 = 1 \\ &y_{j+1} a_{j+1,j} = \sum_{s=j-m+1}^{j} (-1)^{j-s} y_s a_{sj}, \quad 1 \leq j \leq n-1. \end{aligned} \tag{7}$$

It is clear that (7) defines an $m+1$ banded triangular linear system

$$Ty = e. \tag{8}$$

where

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

Thus the problem of finding an LU factorization of an upper Hessenberg matrix reduces to solving the banded triangular system described in (8) 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 equation (5). To determine U, note first that the elements of the $(m-1)^{\text{st}}$ 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 $(m-1)^{\text{st}}$ using

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

Similarly, the j^{th} super-diagonal is given in terms of the $(j+1)^{\text{st}}$ by

$$\ell_i u_{i-1,j+i} + u_{i,j+i} = a_{i,j+i}, i = 2, \ldots, n-j. \tag{9}$$

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 parallelizable. In fact, the main diagonal may be obtained using 1 division rather than the multiplication and subtraction in (9) by

$$u_{i,i} = a_{i+1,i}/\ell_{i+1}, i = 2, \ldots, n. \tag{10}$$

Further, the calculation of the super-diagonals can be chained, since the dependency graph for U, using (9) and (10), is

$$U := \begin{bmatrix} \bullet & \bullet & \bullet & \bullet & \cdots & \bullet & 0 & 0 & 0 & \cdots & 0 \\ 0 & \bullet & \downarrow & \downarrow & \cdots & \downarrow & \bullet & 0 & 0 & \cdots & 0 \\ 0 & 0 & \bullet & \downarrow & \cdots & \downarrow & \downarrow & \bullet & 0 & \cdots & 0 \\ & & \ddots & & & & & & & \vdots & \\ \vdots & & & \ddots & & & & \ddots & & & \bullet \\ 0 & & & & & & & \ddots & & & \downarrow \\ \vdots & & & & & & & & & & \downarrow \\ & & & & & & & & & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & . & . & \cdots & \cdots & \cdots & 0 & \bullet \end{bmatrix} \tag{11}$$

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.

Thus the calculation of L using (5) requires $n-1$ divisions. From (9), the elements of the j^{th} super-diagonal, for $j > 0$ can be calculated using 1 multiplication and 1 subtraction, and from (10) those of the main diagonal can be calculated in 1 division. It is clear that the number of elements in the j^{th} super-diagonal of U is $n-j$ for $j = 0, \ldots, m-1$. Note that for all elements of the $(m-1)^{\text{st}}$ super-diagonal and for the first element of each of the other superdiagonals no calculations are necessary. Hence, the total computations for U is

$$(n-1) + 2\sum_{j=1}^{m-2}(n-j-1) = n(2m-3) - (m^2-m-1). \tag{12}$$

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, $m = 2$, (12) reduces exactly to $n-1$. Hence using p processors L and U can be calculated in the general case in

$$\lceil \frac{2n(m-1)}{p} \rceil \tag{13}$$

parallel operations, and in the special tridiagonal case in

$$2\lceil (n-1)/p \rceil \tag{14}$$

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

3. ALGORITHM FOR THE TRIANGULAR SYSTEM

In [5], Lakshmivarahan and Dhall present an algorithm for calculating the LU factorization of a tridiagonal matrix. Their algorithm used the substitution given in (5) to produce a linear system, which is equivalent to that described by (8) with $m = 2$. Our algorithm is a generalization to the upper Hessenberg case of the algorithm presented in [5]. We remark that the improvement produced in the generalization leads also to a more efficient algorithm for the tridiagonal case.

In order to partition the problem we set

$$z_j = (y_{j-m+1}, y_{j-m+2}, \cdots, y_j)^T, \quad 1 \leq j \leq n,$$

and let $B_j, 1 \leq j \leq n-1$, be the $m \times m$ matrix

$$B_j := \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & 0 & 1 \\ b_1^j & \cdots & & b_{m-1}^j & b_m^j \end{bmatrix} \tag{15}$$

where $b_i^j := (-1)^{m-i} a_{j-m+i,j}/a_{j+1,j} \quad j = 1, 2, \cdots, n-1$. We obtain from (7) the m-vector iteration

$$z_{j+1} = B_j z_j \quad j = 1, 2, \cdots, n-1. \tag{16}$$

In order to evaluate $y_1, y_2, \cdots, y_n$, one must compute

$$z_{km+1} = \left(\prod_{i=1}^{k} C_i\right) z_1, \qquad k = 1,2,3,\cdots, N := \lceil (n-1)/m \rceil. \tag{17}$$

where $C_i := \prod_{j=(i-1)m+1}^{im} B_j$, $1 \le i \le N-1$ and $C_N := \prod_{j=(N-2)m+1}^{n-1} B_j$. To calculate the required products $\prod_{i=1}^{k} C_i$, $k = 1,2,\cdots,N$, one uses a variant of recursive doubling. Let $Z_1 = C_1 z_1$ and $Z_i = C_i$, $i = 2, \ldots, N$, then, assuming we have g processor groups, each group first calculates

$$D_{l,k} = \prod_{i=(k-1)M+1}^{(k-1)M+l} Z_i, \; k = 1, \ldots, g, \; l = 1, \ldots, M = N/g.$$

Then the g processor groups execute the following algorithm:

for $i := 0$ thru $log(g) - 1$ do
{distribute the g/2 independent calculations found in the}
{ j and k loops below among the g/2 groups of processors }
for $j := 2^i$ thru $g - 2^i$ step 2^{i+1} do
for $k := j + 1$ thru $j + 2^i$ do
{using 2 groups calculate }
for $l := 1$ thru M do
$D_{l,k} := D_{l,k} D_{M,j}$

It is easily seen that after the execution of the above algorithm $D_{l,k} = (\prod_{j=1}^{(k-1)M+l} C_j) z_1$. Note also that, for fixed i, the products for $j = 2^i$ are each a product of the form $D_{l,k} := D_{l,k} D_{M,j}$, which is a matrix–vector product, while for $j > 2^i$ the product is a matrix–matrix product. For the purpose of simplifying the complexity analysis, assume that $n = Nm + 1$, $p = g(2m-1)$ where g is a power of 2, and $N = gM$. Let us now analyse the stages separately.

Initialization
One must first calculate the entries b_i^j of the B_j. There are $m(n-1)$ of these, half of which require only a division, and the other half, which also require a negation (or multiplication by -1). Thus the p processors must do $3m(n-1)/2$ operations which takes

$$\lceil 3m(n-1)/2p \rceil. \tag{18}$$

Stage I
Calculate the C_i, $1 \le i \le N$, using g processor groups. Each group calculates $M = N/g$ of these. Since

$$C_i = \prod_{j=(i-1)m+1}^{im} B_j,$$

the formation of C_i requires $m-1$ products of B_j by the partially formed C_i. If these product are formed by starting with B_{im} and working down, then each product of the form

$$\prod_{j=im-k+1}^{im} B_j, k = 1, 2, \ldots, m$$

has k rows, with m possibly nonzero elements, and $m-k$ rows, with one 1 and $m-1$ zeros. Moreover, any column of B_{im-k} contains at most 2 entries, one of which is a 1. Therefore, forming

$$\prod_{j=im-k}^{im} B_j$$

requires a multiply and an addition for each of the m non-zero values in each of the k rows (plus the time for copying elements which we neglect). Using m of the processors in a group, the total time required to calculate each C_i is $\sum_{k=1}^{m-1} 2k = m(m-1)$. With each group calculating N/g of the C_i, one finds that the total time for stage 1 is

$$N(m-1)m/g = (n-1)(m-1)/g. \tag{19}$$

Stage 2
The k^{th} processor group sequentially calculates all products of the form

$$D_{l,k} = \prod_{i=(k-1)M+1}^{(k-1)M+l} Z_i,\ k = 1, \ldots, g,\ l = 1, \ldots, M,\ \ M = N/g.$$

Each group does $N/g-1$ matrix-matrix products of size m. Using $2m-1$ processors, any m–inner product requires $1 + \lceil \log(m) \rceil$ operations. Using chaining the required $(N/g-1)m^2$ m–inner products can be calculated in

$$(N/g-1)m^2 + 2\lceil \log(m) \rceil = (n-1)m/g - m^2 + 2\lceil \log(m) \rceil. \tag{20}$$

time.

Stage 3
This consists of $q = \log g$ stages, in which recursive doubling is applied to the C_i's. For this stage we regroup the processors into g/2 groups each containing 4m-2 processors. From the algorithm above, at the i^{th} stage, $i = 0, \ldots, q-1$ there are $2^i M$ matrix–vector products and $(g/2 - 2^i)M$ matrix–matrix products to be calculated. Each processor group does either matrix–vector products or matrix–matrix products, but at the qth stage only matrix–vector products remain.

At each of the first $q-1$ sub–stages the multiplication of the matrix–matrix product predominates, and at least one group is always engaged in calculating matrix–matrix products. That group does $(q-1)Mm^2$ m–inner products during the first $q-1$ substages and Mm m–inner products at the last substage. The maximum number of inner products performed during the q substages by any group is, thus,

$$(q-1)Nm^2/g + Nm/g = (\log g - 1)(n-1)m/g + (n-1)/g. \tag{21}$$

Assuming that we chain between sub–stages of stage 3, any group can compute that number of inner products in

$$(\log g - 1)(n-1)m/(2g) + (n-1)/(2g) + 2\lceil \log(m) \rceil. \tag{22}$$

For a general upper Hessenberg matrix with band width $m+1$, combining (18)-(22), the time required to calculate its y_i's in this fashion is

$$\frac{n}{2p}\left[6m^2 - 2m + 1 + m(2m-1)\log\left(\frac{p}{2m-1}\right)\right] + O(m). \tag{23}$$

We formulate the result here in terms of p and m, since we wish to compare it with methods involving different partitions of the problem. In the tridiagonal case $m = 2$, and this reduces to

$$\frac{n}{p}\left[\frac{21}{2} + 3\log(\frac{p}{3})\right]. \tag{24}$$

Thus, from (13) and (23), the total cost of an LU factorization is

$$\frac{n}{2p}\left[6m^2 + 2m - 3 + (2m^2 - m)\log\left(\frac{p}{2m-1}\right)\right] + O(m). \tag{25}$$

In the tridiagonal case, by (14) and (24), the cost of producing an LU factorization is

$$\frac{n}{p}\left[\frac{25}{2} + 3\log\left(\frac{p}{3}\right)\right] + O(1). \tag{26}$$

If one's goal is to solve the linear system $Ax = b$, in addition to solving (8), one must also perform the backsolve by solving the banded triangular systems $Lz = b$ and $Ux = z$. In the tridiagonal case, this may be done by casting it as the solution of two linear recurrences, similar to (2.3) and (2.4) in [5]. The recurrences may then be cast in the form $z_i = \alpha_i z_{i-1} + \beta_i$ and solved using *Algorithm A* and *Algorithm Y* from [5]. The complexity involved, in the tridiagonal case, is $2n/p$, to cast the analogue of (2.3) in [5] in the appropriate form, and $3n(2 + \log(2p/3))/p$ to solve the two recurrences, using *Algorithm A* and *Algorithm Y*. Adding these gives a total of

$$\frac{n}{p}(8 + 3\log(\frac{2p}{3})). \tag{27}$$

The cost of solving for one righthand side, given by (26) and (27), is thus

$$\frac{n}{p}\left[\frac{47}{2} + 6\log\left(\frac{p}{3}\right)\right] + O(1). \tag{28}$$

4. CONCLUSIONS

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 1, where $n' = n + 1 = 2^t$.

Further let us consider again cases, such as the ADI method discussed in the introduction, where A is known in advance but the b_i are not. Comparing this algorithm with methods, such as Recursive-Doubling, which do not perform the LU decomposition, a further improvement in computational efficiency results, since one need only perform the forward and back solves, for each right-hand side, rather than performing the full elimination.

Table 1
Complexity of the solution of a single linear system for tridiagonal matrices.

	Processors	Time
Serial Gaussian Elimination	1	8n
Recursive Doubling [3]	n	$24 \log n$
Odd-Even Reduction [3]	$n'/2$	$19 \log n' - 14$
Odd-Even Elimination [3]	n'	$14 \log n' + 1$
Lakshmivarahan Dhall [5]	$n/2$	$18 \log n$
Lakshmivarahan Dhall [5]	$p,\ 3 \leq p \leq 3n/4$	$(n/p)[25 + 9 \log p/3] - 3$
Algorithm described here	p	$(n/p)[47/2 + 6 \log p/3]$

5. REFERENCES

[1] S.C. Chen, D.J. Kuck, A.H. Sameh, Practical Parallel Band Triangular System Solvers, *ACM TOMS* **4(3)** (1978) 270-277.

[2] D.B. Gannon and J. van Rosendall, On the Impact of Communication Complexity on the Design of Parallel Numerical Algorithms, *IEEE Trans. on Computers* **33** (1984) 1180-1194.

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

[4] S.L. Johnsson, Solving Tridiagonal Systems on Ensemble Architecture *SIAM J. Sci. Stat. Comput.* **8** (1987) 354-392.

[5] S. Lakshmivarahan, S.K. Dhall, A New Class of Parallel Algorithms for Solving Tridiagonal Systems, *IEEE Fall Joint Computer Conference* (1986) 315-324.

[6] A.H. Sameh, R.P. Brent, Solving Triangular Systems on a Parallel Computer, *SIAM JNA* **14(6)** (1977) 1101-1113.

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

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

[9] H.S. Stone, Parallel tridiagonal equation solvers, *ACM Trans. Math. Software* **1** (1975) 289-307.

[10] R.S. Varga, *Matrix Iterative Analysis*, (Prentice–Hall Englewood Cliffs, N.J., 1962, p322).

[11] H.A. van der Vorst, Large tridiagonal and black tridiagonal linear systems on vector and parallel computers, *Parallel Computing* **5** (1987) 45-54.

[12] H.A. van der Vorst, Analysis of a parallel solution method for tridiagonal systems, *Parallel Computing* **5** (1987) 303-311.

[13] H.H. Wang, A parallel method for tridiagonal equations, *ACM Trans. Math. Software* **7** (1981) 170-183.