

Large tridiagonal and block tridiagonal linear systems on vector and parallel computers

Henk A. van der VORST

Department of Mathematics and Informatics, Delft University of Technology, 2600 AJ Delft, Netherlands

Abstract. In this paper an overview is given of a number of techniques for solving tridiagonal linear systems on vector and parallel computers. A generalization for block tridiagonal systems leads to a class of incomplete parallel (and vectorizable) decompositions that can be used as preconditionings for, e.g., the conjugate gradients method. Numerical experiments indicate that these parallel preconditionings may even be preferable over the standard ones on conventional serial computers.

Keywords. Linear algebra, tridiagonal linear systems, vector and parallel computers, overview.

1. Introduction

The discretisation of p.d.e.'s in 2D or 3D, by finite difference or finite element approximations, leads often to large block tridiagonal linear systems. Many of the more powerful iterative solution methods, like, e.g., Block SOR, (M)ICCG, line-ICCG, SIP, require the direct solution for the diagonal blocks, or factors of those. These diagonal blocks are often again tridiagonal matrices themselves. This part of the algorithm then leads to recurrence relations and this is considered to be a serious problem for execution on vector and parallel computers.

In Section 2 we will briefly discuss a number of techniques that help to make the solving of tridiagonal linear systems more suitable for modern architectures. It should be mentioned that these techniques are also applicable to more general banded systems.

These basic techniques may be easily generalized for use in connection with block tridiagonal linear systems. In particular we will discuss an application on incomplete decompositions, to be used as preconditionings for, e.g., the conjugate gradients method.

In Section 3 we will first discuss some of the problems, with respect to vectorization and parallelism, in standard incomplete decompositions.

In Section 4 a parallel variant of the incomplete decomposition for a model problem is proposed and considered in more detail. The effectiveness of this variant, as well as of vectorizable variants, is illustrated by numerical examples in Section 5.

2. Tridiagonal linear systems

It is assumed here that a given tridiagonal linear system $Tx = y$ has to be solved for many right-hand sides y , as is commonly the case in the problems of interest. It is then preferable to

factor T as $T = LU$, with L and U lower and upperbidiagonal, respectively. Hence the problem reduces to the solution of two bidiagonal systems.

We will therefore further restrict ourselves to parallel and vector algorithms for the solution of bidiagonal linear systems, though the discussed techniques are also applicable directly to tridiagonal systems (often they have been proposed originally for tridiagonal systems).

Further we will assume that T has been scaled such that $\text{diag}(L) = \text{diag}(U) = I$. This makes the formulas less complicated and it also helps to reduce computer time in actual computation.

The order of the linear systems is denoted by n , while n_p denotes the number of parallel processors.

2.1. Recursive doubling

For a detailed discussion, see [3,18]. The idea is the following. Write L as $L = I - B$. Then a recursive doubling step consists of premultiplication of $Lx = y$ by $I + B$, which yields

$$(I - B^2)x = (I + B)y. \quad (2.1)$$

The premultiplication by $I + B$ can be easily vectorized as well as carried out in parallel.

The system (2.1) can be rewritten as two independent systems, one bidiagonal linear system for the odd unknowns, the other one for the even unknowns. These systems can, of course, be solved in parallel. The recursive doubling technique can be used repeatedly, until a desired degree of parallelism is achieved.

Since the amount of scalar operations (i.e., non-vectorizable operations) is not significantly reduced by one step recursive doubling, this technique is not very helpful for use on vector computers. For a discussion on the effectiveness of this technique on parallel computers, when $n_p \ll n$, see [13].

2.2. Cyclic reduction

For details we refer to [8]. When applied to $Lx = y$, one step of cyclic reduction consists of eliminating the odd-numbered unknowns from the even-numbered equations. This reduction, which can be done in vector mode as well as in parallel, reduces the system to a lower bidiagonal system of half the size (for the even unknowns). Having solved this $\frac{1}{2}n$ th order system, the odd unknowns can be determined easily either in vector mode or in parallel.

Since the amount of scalar (= recursive) work is reduced by a factor of 2, by means of (fast) vector operations, this technique is a good candidate for vector computers like the CRAY-1. The use on a CYBER 205 is more cumbersome due to stride-problems. For a discussion on implementations for these computers see [26].

2.3. Divide and conquer

This technique, though not by the above name, has been proposed, for tridiagonal linear systems by Wang [27], and has been generalized for banded systems by Meier [10]. When applied to $Lx = y$, the idea is to divide L in blocks and to eliminate the lower bidiagonal elements within each block. This produces fill-in in a way as shown in Fig. 1.

Next the elements of the $\tilde{L}_{j+1,j}$ can be eliminated almost in parallel: each block $\tilde{L}_{j+1,j}$ can be eliminated as soon as the lowest element in $\tilde{L}_{j,j-1}$ has been eliminated. The elimination within each $\tilde{L}_{j+1,j}$ can be done by a linked triad operation which is very fast on vector processors, hence Wang's algorithm is very attractive for parallel vector processors (e.g., CRAY X-MP, Alliant-ELX8). We then may expect a speedup by a factor of $\frac{1}{2}n_p$. The algorithm is also

$$L : \begin{bmatrix} 1 & & & & \\ * & 1 & & & \\ & * & 1 & & \\ & & * & 1 & \\ & & & * & 1 \\ & & & & * & 1 \\ & & & & & * & 1 \\ & & & & & & * & 1 \\ & & & & & & & * & 1 \\ & & & & & & & & * & 1 \end{bmatrix} \Rightarrow \tilde{L} : \begin{bmatrix} 1 & & & & \\ & \tilde{L}_{11} & & & \\ & * & 1 & & \\ & & * & 1 & \\ & & & * & 1 \\ & & & & * & 1 \\ & & & & & * & 1 \\ & & & & & & * & 1 \\ & & & & & & & * & 1 \\ & & & & & & & & * & 1 \end{bmatrix}$$

Fig. 1.

very attractive for so-called message passing systems, i.e. systems of parallel processing elements with only local memories. For more details, see [14].

2.4. A twisted factorization

Instead of the standard LU -factorization, the matrix A may also be factored as $A = PQ$, in the following way:

$$\begin{bmatrix} \diagdown & & \\ & \diagdown & \\ & & \diagdown \end{bmatrix} = \begin{bmatrix} \diagdown & & \\ & \diagdown & \\ & & \diagdown \end{bmatrix} \begin{bmatrix} \diagdown & & \\ & \diagdown & \\ & & \diagdown \end{bmatrix} \quad (2.2)$$

$A \qquad \qquad P \qquad \qquad Q$

It can be proven that such a decomposition exists, when A is symmetric positive definite, or when A is an M -matrix, or when A is diagonally dominant.

The decomposition of A , as well as the back substitutions, can be carried out almost entirely in parallel. This technique has been proposed, in a slightly different form, in [9]. An analysis, which includes stability aspects is given in [24]. The computational complexity for the PQ -splitting is the same as for the LU -splitting. More parallelism may be introduced by previously discussed techniques.

3. Incomplete decompositions for block tridiagonal matrices

The 5-point finite difference discretisation of an elliptic p.d.e., over a rectangular grid in 2D, leads to a linear system $Ax = y$, with the following structure for the matrix A :

$$A = \begin{bmatrix} A_{11} & A_{12} & & & \\ A_{21} & A_{22} & A_{23} & & 0 \\ & A_{32} & A_{33} & A_{34} & \\ 0 & & & & A_{m-1,m} \\ & & & A_{m,m-1} & A_{m,m} \end{bmatrix} \quad (3.1)$$

The blocks $A_{i,j}$ are diagonal matrices, for $|i-j|=1$, and the $A_{i,i}$ are tridiagonal matrices. We will further assume this simple form for A (many ideas carry over to more complicated structures).

The obtained linear system $Ax = y$ may be solved by conjugate gradients, when A is positive definite symmetric, or by bi-cg [5], cg-squared [17], or bidiagonalisation [15] in certain non-symmetric cases. For many systems, the convergence behaviour of these methods is considerably improved when an incomplete decomposition of A is used as a preconditioning matrix [2,6,11,12,21].

In the standard incomplete decomposition $A = LD^{-1}U - R$ we simply neglect all fill-in during the factorization of A . The product $LD^{-1}U$ is called the incomplete decomposition of A . This decomposition exists when A is an M -matrix and it is uniquely defined by the following rules:

(1) the non-zero elements of A are equal to the elements of $LD^{-1}U$ on corresponding places;

(2) $\text{diag}(L) = \text{diag}(U) = D$;

(3) the off-diagonal elements of the lower triangular matrix L and the upper triangular matrix U are equal to the corresponding elements of A .

The use of $(LD^{-1}U)^{-1}$ as a preconditioning matrix leads to recurrence relations, e.g., when solving for x from $Lx = y$.

For a vector supercomputer, part of the recurrency (that avoids vectorization) can be vectorized at the cost of a smaller loop length. This leaves us with a series of bidiagonal systems that have to be solved. In [20] it is suggested to approximate the solution of these systems by truncated Neumann series. This leads to rather high performances on vector computers like CRAY-1 or CYBER 205 [22], see also Section 5.

For computation on a parallel computer, parallelism in the preconditioning part may be introduced by techniques as described in Section 2.

Another idea is to neglect some of the off-diagonal blocks $L_{i+1,i}$ or $U_{i,i+1}$ in the incomplete decomposition, so that the backsubstitution algorithms can be split into independent tasks for different processors. This has been suggested by Seager [16], who discusses an implementation for CRAY X-MP computers.

In Section 4 we will propose a different technique that leads quite naturally to an incomplete parallel decomposition, which satisfies all the above-mentioned requirements (1)–(3), except that the factors are no longer triangular matrices.

Note. For this special structure of A , the recurrency can also be removed by computing the unknowns in a diagonal-wise order, i.e., compute $x_{k-i+1,i}$, $i = 1, \dots, k$, in parallel, for $k = 1, 2, \dots$ (the term diagonal-wise refers to the underlying grid), see, e.g. [22].

4. Incomplete parallel decomposition

4.1. The basic idea

Similarly to the 1D-case, in Section 2.4, we may try to split A in the twisted form $A = PQ$, where P and Q have the same structure as (2.2), except that their elements are now block-matrices. An obvious variant on the standard incomplete factorization is obtained when (part of) the fill-in in P and Q is neglected.

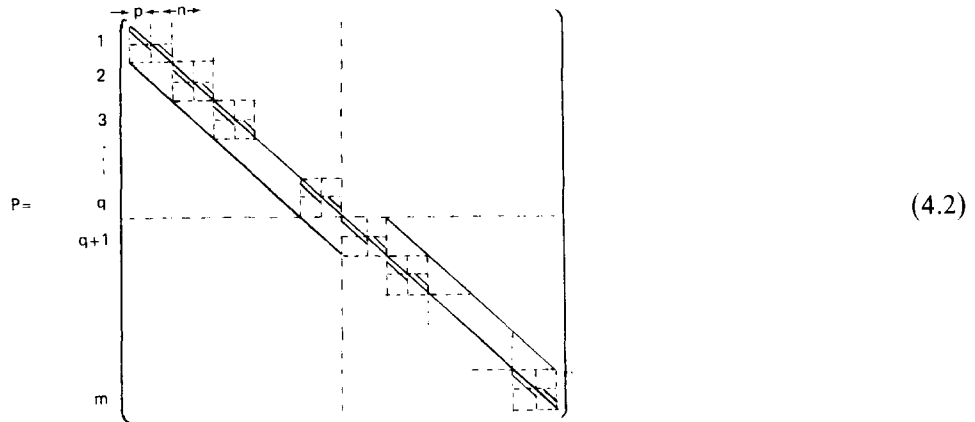
When all fill-in is neglected, the diagonal blocks of P are lower bidiagonal matrices and the non-zero off-diagonal blocks are diagonal matrices (Q has the structure of P^T).

The (incomplete) parallel splitting of A can be computed almost entirely in two parallel parts and also the back substitution for the factors can be done largely in parallel. This will be shown in detail in Section 4.2.

From the requirement that the non-zero elements of A are equal to the corresponding elements in PQ , we find that

$$A_{i,i} = P_{i,i}Q_{i,i} + \text{product of diagonal matrices.} \quad (4.1)$$

Hence, $P_{i,i}$ and $Q_{i,i}$ represent the factors of a tridiagonal matrix and thus they may also be replaced by the twisted factors as in (2.2). This leads to the following non-zero structure for P :



The factor Q has the same non-zero structure as P^T . The precise algorithm for the computation of P is given in Section 4.2.

So far we have silently assumed that the above-sketched incomplete decomposition exists. We only mention here that it can be proven to exist when A is an M -matrix [25], similarly as for the standard incomplete decomposition.

In this presentation we have chosen to start with an incomplete parallel decomposition for 2D systems and we have incorporated in it the parallel splitting for 1D systems. In some situations such a splitting for the 1D blocks may be undesirable, e.g., possibly when one uses some kind of block factorization. On the other hand it is now obvious how to handle the 3D case.

Another and very elegant approach of reducing the higher-dimensional case to lower-dimensional problems has been suggested in [1] by the so-called nested (incomplete) factorization. The idea of nesting can be easily combined by the idea of parallel decomposition. This may lead to hybrid decompositions in which, e.g., the 3D and 2D blocktridiagonal structures are decomposed in parallel form and the 1D blocks are solved by either parallel or vectorizable techniques. Such a choice highly depends on the specific architecture of the computer.

Except for the obtained parallelism in the decomposition, with factors as in (4.2), there is, quite surprisingly, another bonus. Our experiments indicate that the parallel incomplete decompositions, when used as preconditionings for the cg-method, often lead to a noticeable *decrease* in the number of iteration steps. Since their computational complexity is exactly equal to that of the standard incomplete decomposition, this suggests that they may also be considered seriously for use on conventional sequential computers.

4.2. Algorithms for the model problem

For the model problem, given by (3.1), we will present the algorithm for the construction of the incomplete parallel decomposition as well as for the back substitutions required for the preconditioning.

In order to keep the formulas simple we will assume A to be symmetric. The diagonal elements of $A_{j,j}$ are denoted by $a_{i,j}$, the superdiagonal elements of $A_{j,j}$ by $b_{i,j}$ and the diagonal elements of $A_{j,j+1}$ by $c_{i,j}$. The values for p and q , denoting where the factors change from lower to upper bidiagonal, are given in (4.2).

It turns out to be convenient to write the incomplete decomposition as $PD^{-1}P^T$, where D is a diagonal matrix with elements $d_{i,j}$ and $\text{diag}(P) = D$. One may verify that, when we neglect all fill-in during the construction of the decomposition, the non-zero off-diagonal elements of P are equal to the elements of A with corresponding indices. This implies that we need only to compute the elements $d_{i,j}$.

Convention. Throughout this section we will use the convention that parts of expressions in which indices $i < 1$, $i > n$, $j < 1$, $j > m$ occur, should be skipped, e.g., $\tilde{a}_{n,j} - b_{n,j}^2/d_{n+1,j}$ is the same as $\tilde{a}_{n,j}$.

For j fixed, the elements $\tilde{a}_{i,j}$ and $b_{i,j}$ define a symmetric tridiagonal matrix for which the $d_{i,j}$, of its decomposition, are computed by Algorithm S1D.

Algorithm S1D. Keep j fixed.

S1D(a): for $i = 1, 2, \dots, p$ do

$$d_{i,j} = \tilde{a}_{i,j} - b_{i-1,j}^2/d_{i-1,j}.$$

S1D(b): for $i = n, n-1, \dots, p+2$ do

$$d_{i,j} = \tilde{a}_{i,j} - b_{i,j}^2/d_{i+1,j}.$$

S1D(c): for $i = p+1$ do

$$d_{i,j} = \tilde{a}_{i,j} - b_{i,j}^2 - b_{i,j}^2/d_{i+1,j} - b_{i-1,j}^2/d_{i-1,j}.$$

Note that S1D(a) and S1D(b) can be executed in parallel.

The elements $d_{i,j}$ that define the incomplete parallel decomposition $PD^{-1}P^T$ of A , with a non-zero structure of P as in (4.2) can be computed by Algorithm S2D, which uses S1D as a kernel.

Algorithm S2D. Computation of the $d_{i,j}$ for D in $PD^{-1}P^T$.

S2D(a): for $j = 1, 2, \dots, q$ do

$$\langle \text{Algorithm S1D, with } \tilde{a}_{i,j} \equiv a_{i,j} - c_{i,j-1}^2/d_{i,j-1} \rangle.$$

S2D(b): for $j = m, m-1, \dots, q+2$ do

$$\langle \text{Algorithm S1D, with } \tilde{a}_{i,j} \equiv a_{i,j} - c_{i,j}^2/d_{i,j+1} \rangle.$$

S2D(c): for $j = q+1$ do

$$\langle \text{Algorithm S1D, with } \tilde{a}_{i,j} \equiv a_{i,j} - c_{i,j-1}^2/d_{i,j-1} - c_{i,j}^2/d_{i,j+1} \rangle.$$

The parts S2D(a) and S2D(b) can be executed in parallel.

Be given the decomposition for a simple (1D) block, i.e., j fixed, then Algorithm P1D solves $x_{\cdot,j}$ from $Px_{\cdot,j} = \tilde{y}_{\cdot,j}$.

Algorithm P1D. Keep j fixed.

P1D(a): for $i = 1, 2, \dots, p$ do

$$x_{i,j} = (\tilde{y}_{i,j} - b_{i-1,j}x_{i-1,j})/d_{i,j}.$$

P1D(b): for $i = n, n-1, \dots, p+2$ do

$$x_{i,j} = (\tilde{y}_{i,j} - b_{i,j}x_{i+1,j})/d_{i,j}.$$

P1D(c): for $i = p+1$ do

$$x_{i,j} = (\tilde{y}_{i,j} - b_{i-1,j}x_{i-1,j} - b_{i,j}x_{i+1,j})/d_{i,j}.$$

The parts P1D(a) and P1D(b) can be executed in parallel.

The computation of x from $Px = y$ (i.e., $x = P^{-1}y$) is done by Algorithm P2D.

Algorithm P2D. Computation of $x = P^{-1}y$.

P2D(a): for $j = 1, 2, \dots, q$ do

$$\langle \text{Algorithm P1D, with } \tilde{y}_{i,j} \equiv y_{i,j} - c_{i,j-1}x_{i,j-1} \rangle.$$

P2D(b): for $i = m, m-1, \dots, q+2$ do

$$\langle \text{Algorithm P1D, with } \tilde{y}_{i,j} \equiv y_{i,j} - c_{i,j}x_{i,j+1} \rangle.$$

P2D(c): for $j = q+1$ do

$$\langle \text{Algorithm P1D, with } \tilde{y}_{i,j} \equiv y_{i,j} - c_{i,j-1}x_{i,j-1} - c_{i,j}x_{i,j+1} \rangle.$$

The parts P2D(a) and P2D(b) can be executed in parallel.

Since the building blocks P1D can also largely be executed in parallel, the computation of $P^{-1}y$, for a 2D system, can be done almost entirely in parallel on 4 processors. Of course, the degree of parallelism may even be increased by using techniques, as discussed in Section 2, for P1D.

Finally the Algorithm Q2D for solving z from $D^{-1}P^T z = x$ (and hence $PD^{-1}P^T z = y$) is given for completeness. It has Q1D as its basic element and has the same properties with respect to parallelism as P2D.

Algorithm Q1D. Keep j fixed.

Q1D(a): for $i = p+1$ do

$$z_{i,j} = \tilde{x}_{i,j}.$$

Q1D(b): for $i = p, p-1, \dots, 1$ do

$$z_{i,j} = \tilde{x}_{i,j} - b_{i,j}x_{i+1,j}/d_{i,j}.$$

Q1D(c): for $i = p+2, p+3, \dots, n$ do

$$z_{i,j} = \tilde{x}_{i,j} - b_{i-1,j}/d_{i,j}.$$

After Q1D(a) has been executed, the parts Q1D(b) and Q1D(c) can be executed in parallel.

Algorithm Q2D. Computation of $z = (D^{-1}P^T)^{-1}x$.

Q2D(a): for $j = q+1$ do

$$\langle \text{Algorithm Q1D, with } \tilde{x}_{i,j} \equiv x_{i,j} \rangle.$$

Q2D(b): for $j = q, q-1, \dots, 1$ do

$$\langle \text{Algorithm Q1D, with } \tilde{x}_{i,j} \equiv x_{i,j} - c_{i,j}z_{i,j+1}/d_{i,j} \rangle.$$

Q2D(c): for $j = q+2, q+3, \dots, m$ do

$$\langle \text{Algorithm Q1D, with } \tilde{x}_{i,j} \equiv x_{i,j} - c_{i,j-1}z_{i,j-1}/d_{i,j} \rangle.$$

After Q2D(a) has been executed, the parts Q2D(b) and Q2D(c) can be executed in parallel.

4.3. Additional notes

(1) Note the similarity between S1D and S2D, P1D and P2D, and Q1D and Q2D, respectively. From the presentation of this algorithms it is now clear how to proceed to the 3D-case. For these 3D systems one obtains a parallel algorithm suitable for 8 processors.

(2) It is easily verified that the computational complexity of S2D, P2D and Q2D is exactly the same as for the corresponding parts in the standard incomplete Choleski decomposition (with no fill-in). Hence, with respect to the amount of work per iteration step, one obtains the parallelism for free.

(3) It is often advantageously to scale the matrix A , after S2D has been executed, in such a way that $D = I$. This is done in the following way:

- (a) Execute S2D for the matrix A .
- (b) Scale A and y (for the linear system $Ax = y$):

```

for  $j = 1, \dots, m$  do
  for  $i = 1, \dots, n$  do
     $\bar{a}_{i,j} = a_{i,j}/d_{i,j}$ 
     $\bar{y}_{i,j} = y_{i,j}/\sqrt{d_{i,j}}$ 
     $\bar{b}_{i-1,j} = b_{i-1,j}/\sqrt{d_{i-1,j}d_{i,j}}$ 
     $\bar{c}_{i,j-1} = c_{i,j-1}/\sqrt{d_{i,j-1}d_{i,j}}$ 

```

(c) Solve $\bar{A}\bar{x} = \bar{y}$, with \bar{A} represented by $\bar{a}_{i,j}$, $\bar{b}_{i,j}$ and $\bar{c}_{i,j}$. For the preconditioning part use P2D and Q2D with the above values for $\bar{b}_{i,j}$ and $\bar{c}_{i,j}$ and replace the $d_{i,j}$ in P2D and Q2D by the value 1.

- (d) Finally scale \bar{x} back to x :

```

for  $j = 1, \dots, m$  do
  for  $i = 1, \dots, n$  do
     $x_{i,j} = \bar{x}_{i,j}/\sqrt{d_{i,j}}$ 

```

(4) Since the non-zero elements of A (or, after scaling, \bar{A}) are equal to the corresponding elements of $PD^{-1}P^T$, one may implement the preconditioned cg-algorithm in the economic way as suggested by Eisenstat [4]. This saves essentially half of the work for the matrix-vector products.

(5) The algorithms P1D and Q1D may also be replaced by algorithms that compute the solution of the occurring bidiagonal systems by an approximate solution, obtained by a truncated Neumann series for the inverses of these bidiagonal systems (as in [20]).

5. Numerical results

In this section we present a few examples that give an impression of the effects on the convergence behaviour of the cg-method, when the parallel or vectorised preconditionings are used instead of the standard one. All algorithms have been coded in FORTRAN and the observed MFLOPS-rates are given for execution on a CRAY X-MP48. The incomplete parallel preconditioning, as given in Section 4.2, has been coded for only one processor, just as the other preconditionings. For a discussion on FORTRAN implementations for the preconditioned cg-algorithm, see [23].

The following preconditioned cg-algorithms have been compared for systems $Ax = b$, A as in Section 4.

(1) ICCG: the standard incomplete decomposition. The recurrency in the x -direction has been optimised by the CRAY-subroutine FOLR [23, Section 4.4].

(2) DICCG: the standard incomplete decomposition. The recurrencies have been removed by computing the unknowns in diagonal-wise order [22, Section 4.2].

(3) VICCG-2T: based upon the standard incomplete decomposition. The recurrencies in the x -direction have been replaced by 2-term truncated Neumann series [20].

(4) MICCG: for comparison reasons Gustafsson's modified ICCG process [7] has been included in the test. The suggested $O(h^2)$ modification to the diagonal has been omitted. The implementation is similar to that of ICCG and hence the Mflops-rates are identical.

(5) PICCG: the incomplete parallel decomposition as proposed in Section 4.2. The recurrencies in P1D and Q1D have been optimised by FOLR (as in ICCG).

(6) VPICCG-2T: as PICCG, but now the recurrencies in P1D and Q1D have been replaced by 2-term truncated Neumann series (as in VICCG-2T).

The following Mflops-rates have been observed for problems with $n = m = 200$ (CFT 1.14, COS 1.14):

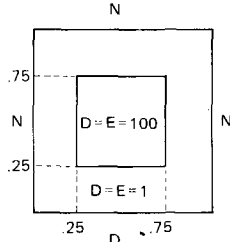
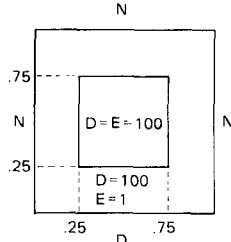
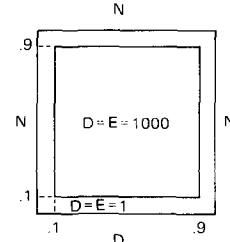
Method	1	2	3	4	5	6
Mflops-rate	61	110	123	61	61	109

The algorithms have been tested on linear systems that arise from 5-point finite difference discretisations of the p.d.e.:

$$-\frac{\partial}{\partial x} \left(D \frac{\partial}{\partial x} u \right) - \frac{\partial}{\partial y} \left(E \frac{\partial}{\partial x} u \right) = F \quad \text{on } [0, 1] \times [0, 1].$$

The boundary conditions are either Dirichlet (D: $u = 0$) or Neumann (N: $\partial u / \partial n = 0$).

The 3 test problems are characterised by:

Problem 1	Problem 2	Problem 3
		
$\Delta x = \Delta y = \frac{1}{39}$ ($n = 60, m = 59$)	$\Delta x = \frac{1}{149}, \Delta y = \frac{1}{150}$ ($n = m = 150$)	$\Delta x = \frac{1}{149}, \Delta y = \frac{1}{150}$ ($n = m = 150$)

The start vector x_0 has been chosen such that $\|Ax_0 - b\|_2 = O(1)$, and the iteration process has been terminated when $\|Ax_i - b\|_2 < 10^{-11}$. This led to the iteration results shown in Table 1 (for lower precisions about the same ratios between the iteration numbers could be observed).

Table 1

Problem	Method					
	1	2	3	4	5	6
1	100	100	104	96	92	102
2	191	191	191	315	159	174
3	210	210	224	225	178	212

References

- [1] J.R. Appleyard and I.M. Cheshire, Nested factorization, *Proc. Reservoir Simulation Symposium*, San Francisco (1983) paper SPE 12264.
- [2] P. Concus, G.H. Golub and D.P. O'Leary, A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in: J.R. Bunch and D.J. Rose, eds., *Sparse Matrix Computations* (Academic Press, New York, 1976).
- [3] P. Dubois and G. Rodrigue, An analysis of the recursive doubling algorithm, in: D. Kuck et al., eds., *High Speed Computer and Algorithm Organization* (Academic Press, New York, 1977).
- [4] S.C. Eisenstat, Efficient implementation of a class of preconditioned conjugate gradient methods, *SIAM J. Sci. Statist. Comput.* **2**(1) (1981) 1–4.
- [5] R. Fletcher, Conjugate gradient methods for indefinite systems, *Lecture Notes in Mathematics* **506** (Springer, Berlin, 1976) 73–89.
- [6] G.H. Golub and C.F. van Loan, *Matrix Computations* (North Oxford Academic, Oxford, 1983).
- [7] I. Gustafsson, A class of 1st order factorization methods, *BIT* **18** (1978) 142–156.
- [8] D. Heller, Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems, *SIAM J. Numer. Anal.* **13** (4) (1976) 484–496.
- [9] G.R. Joubert and E. Cloeth, The solution of tridiagonal linear systems with an MIMD parallel computer, *Proc. 1984 GAMM Conference, Z. Angew. Math. Mech.* (1984).
- [10] U. Meier, A parallel partition method for solving banded systems of linear equations, *Parallel Comput.* **2** (1985) 33–43.
- [11] J.A. Meijerink and H.A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M -matrix, *Math. Comput.* **31** (137) (1977) 148–162.
- [12] J.A. Meijerink and H.A. van der Vorst, Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems, *J. Comput. Phys.* **44** (1981) 134–155.
- [13] P.H. Michielse, Solution methods for bidiagonal and tridiagonal linear systems for parallel and vector computers, Report 87-04, Delft University of Technology, Delft, 1987.
- [14] P.H. Michielse and H.A. van der Vorst, Data transport in Wang's partition method, Report 86-32, Delft University of Technology, Delft, 1986.
- [15] C.C. Paige and M.A. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares, *ACM Trans. Math. Software* **8**(1) (1982) 43–71.
- [16] M.K. Seager, Parallelizing conjugate gradient for the CRAY X-MP, *Parallel Comput.* **3** (1) (1986) 35–47.
- [17] P. Sonneveld, CGS: a fast Lanczos type solver for nonsymmetric linear systems, Report 84-16, Delft University of Technology, Delft, 1984.
- [18] H.S. Stone, An efficient parallel algorithm for the solution of a tridiagonal linear system of equations, *J. ACM* **20**(1) (1973) 27–38.
- [19] H.A. van der Vorst, Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems, *J. Comput. Phys.* **44** (1981) 1–19.
- [20] H.A. van der Vorst, A vectorizable variant of some ICCG Methods, *SIAM. J. Sci. Statist. Comput.* **3** (3) (1982) 86–92.
- [21] H.A. van der Vorst, Preconditioning by incomplete decompositions, Thesis, Utrecht University, 1982.
- [22] H.A. van der Vorst, Comparative performance tests of Fortran codes on the CRAY-1 and CYBER 205, in: J. van Leeuwen and J.K. Lenstra, eds., *Parallel Computers and Computations*, CWI Syllabus **9** (CWI, Amsterdam, 1985).
- [23] H.A. van der Vorst, The performance of FORTRAN implementations for preconditioned conjugate gradients on vector computers, *Parallel Comput.* **3** (1986) 49–58.
- [24] H.A. van der Vorst, Analysis of a parallel solution method for tridiagonal linear systems, *Parallel Comput.*, to appear.
- [25] H.A. van der Vorst, Incomplete parallel decompositions, in progress.
- [26] H.A. van der Vorst and J.M. van Kats, The performance of some linear algebra algorithms in FORTRAN on CRAY-1 and CYBER 205 supercomputers, Report TR-17, Academisch Computer Centrum, Utrecht, 1984.
- [27] H.H. Wang, A parallel method for tridiagonal equations, *ACM Trans. math. Software* **7** (2) (1981) 170–183.