Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Reordering strategies and LU-decomposition of block tridiagonal matrices for parallel processing

A. van der Ploeg

Department of Numerical Mathematics

NM-R9618 1996

Report NM-R9618
ISSN 0169-0388

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Reordering Strategies and LU-decomposition of Block Tridiagonal Matrices for Parallel Processing

A. van der Ploeg

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

Solution of large sparse systems of linear equations continues to be a major research area with widespread application. In many applications, the unknowns appear in groups, and the coefficient matrix has a block structure corresponding to these groups. For example, in discretised incompressible Navier-Stokes equations the velocities and the pressure belonging to the same grid point form such a group.

Another example arises in molecular hydrodynamics computations when a combination of a finite element discretisation in a given spatial direction and a Fourier decomposition in other directions is used. In many cases, it appears to be important to be able to solve block tridiagonal systems efficiently. A possible strategy for obtaining parallelism is to apply a reordering of the unknowns before a decomposition of the coefficient matrix is constructed. In this paper, we compare reordering strategies based on block cyclic reduction and on domain decomposition. Estimates of the number of floating point operations and the amount of data transport will be discussed, and results on a CRAY C90 and CRAY T3D will be presented.

AMS Subject Classification (1991): 65F05, 65F10, 65F50, 65Y05
CR Subject Classification (1991): G.1.3, J.2.
Keywords & Phrases: CRAY C90, CRAY T3D, block tridiagonal, distributed memory, domain decomposition, nested dissection, parallel linear solver, reordering strategies.
Note: This research is sponsored by the NWO Priority Program "Massaal Parallel Rekenen" (MPR) and carried out within MPR-cluster 95MPR04: "Parallel computational magneto-fluid dynamics: nonlinear dynamics of thermonuclear, astrophysical, and geophysical plasmas and fluids".

1. Introduction

Numerical simulations are of increasing importance due to the demand for insight in the behaviour of physical phenomena on the one hand, and the advances in computational hardware and numerical algorithms on the other hand. The CPU-time is often dominated by the time needed for solving large systems of equations and, in many situations, the quality of the numerical simulations is determined by the limited amount of computer power and memory. This induces an unceasing quest for efficient implementations of advanced solution methods on vector/parallel supercomputers and massively parallel distributed-memory systems.

In many applications, the unknowns appear in groups, and the coefficient matrix has a block structure corresponding to these groups. An efficient solver for block tridiagonal systems of

linear equations is often very useful in such cases. For example, a block form of an ADI-method frequently requires the solution of such linear systems. In this paper we consider parallelisable methods to solve a block tridiagonal system

$$Mx = y, \qquad M \in \mathbb{C}^{N\times n} \times \mathbb{C}^{N\times n}, \qquad x \text{ and } y \in \mathbb{C}^{N\times n}, \tag{1.1}$$

in which n is the size of the subblocks, and N is the number of diagonal blocks.

This paper was inspired by the block tridiagonal systems of linear equations generated by the CASTOR finite-element spectral code which is applied intensively at FOM Rijnhuizen for the stability investigation of tokamak plasmas [5]. The model describing the behaviour of plasmas contains eight partial differential equations for eight unknowns: the magnetic field and velocity field, the plasma pressure, and the density. When it is assumed that the perturbed quantities are of the form $e^{\lambda t}$, after linearisation of the PDE's and after discretisation one obtains a generalised eigenvalue problem $Ax = \lambda Bx$. The discretisation consists of a combination of a truncated Fourier series in the poloidal direction, and of a finite element discretisation in the radial direction. As a result of this discretisation, the matrices A and B are block tridiagonal, A is non-Hermitian, and B is Hermitian positive definite. In general, the non-zero entries of A and B are not real, and we have to use complex arithmetic.

We want to solve the eigenvalue problem by a Jacobi-Davidson method [7]. This method frequently requires the approximate solution of a linear system $(A - \sigma B)x = y$ for given right-hand side vector y. Since σ will be some approximation of an eigenvalue we are looking for, $A - \sigma B$ can have a large condition number.

1.1 Incomplete LU-decompositions

If M is sparse, one can often solve (1.1) with an iterative method such as Bi-CGSTAB [10] or GMRES [6]. In order to improve the convergence speed, the iterative method is usually combined with a preconditioner. As an example, we consider a coefficient matrix $M = A - \sigma B$ coming from CASTOR, in which $n = 16$, and $N = 26$. A characteristic part of M is shown in Fig. 1. The dots indicate nonzero entries, and the size of a dot corresponds with the absolute value of a matrix entry as indicated at the right.

Table 1 shows the results of Bi-CGSTAB combined with an incomplete LU-decomposition as a preconditioner. As a stopping criterion for the iterative method we demand that the 2-norm of the preconditioned residual has to decrease with at least a factor 10^8. As a first step, M is scaled in such a way that the sum of the moduli of the entries in each row is equal to one. Next, a splitting $LU + R$ is constructed in which the factors L and U are lower- and upper-triangular matrices respectively, and the residual matrix R is small in some sense. The construction of L and U is based on a threshold parameter ε, e.g. all entries of the residual matrix are in modulus smaller than ε. Hence when $\varepsilon = 0$ we construct a complete decomposition of the matrix. The incomplete decomposition is combined with partial pivoting. In order not to disturb the block structure of the matrix, the search for pivot elements is restricted to the blocks on the main diagonal. From the results in Table 1 it appears that when more elements in the factors L and U are dropped, the number of iterations increases dramatically, even when those elements are very small in modulus. The subblocks of the coefficient matrices in CASTOR appear to be relatively full (more than 50

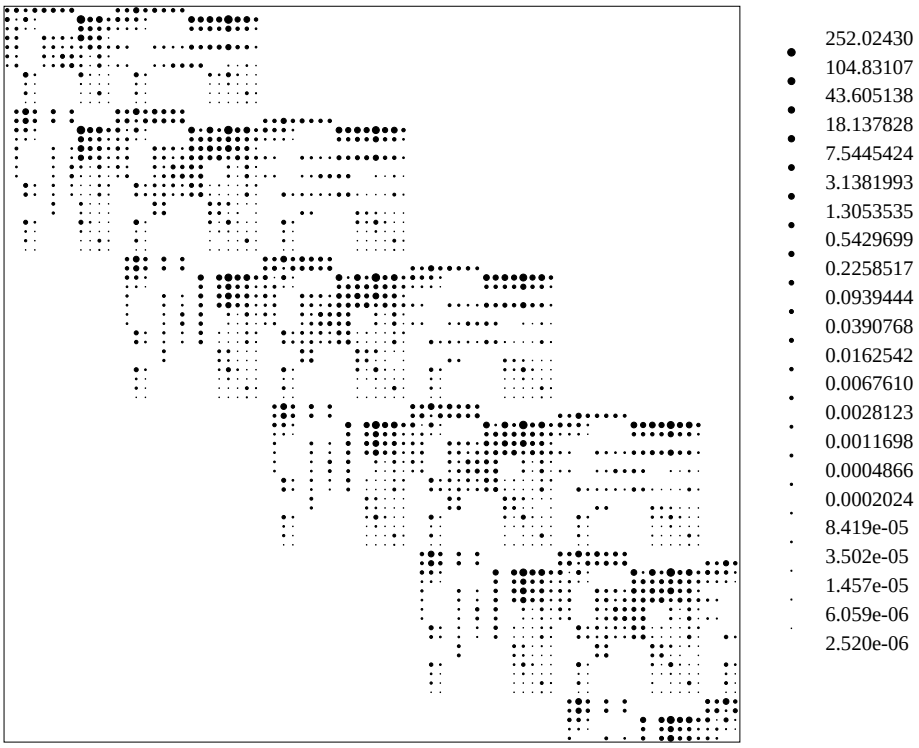

Figure 1: Characterictic part of the testmatrix $M = A - \sigma B$ coming from CASTOR.

percent of the entries is not zero). Hence when a complete LU-decomposition is constructed, the total number of non-zero entries will never be more than twice the number of non-zero entries in the coefficient matrix. Therefore, in the sequel of this paper we will focus on complete LU-decompositions.

Table 1: Results of incomplete LU-decomposition.

ε	# entries $L+U$	# Bi-CGSTAB it.
0	17887	1
10^{-8}	17863	3
10^{-4}	14741	82

1.2 Complete block LU-decompositions

We do not use sparse matrix storage techniques, but we store all elements of the subblocks, including zero elements. This has the advantage that the construction of a complete block LU-decomposition can be implemented by using mainly level-3 BLAS routines. Once the factors L and U have been constructed, (1.1) can be solved with a pair of triangular solves, which can be implemented by using level-2 BLAS routines. The construction of the factors L and U has to be done only once if several systems with different right-hand sides but with

the same coefficient matrix have to be solved. These computations are referred to as the preprocessing. The computation of the solution for given right-hand side is referred to as the solution process. In the Jacobi-Davidson method, several systems with the same coefficient matrix have to be solved. Hence it is very important that the solution process is implemented efficiently.

A block LU-decomposition of M in the given ordering generates no fill-in blocks outside the blocks already present in the matrix. Therefore, on a sequential machine, this technique is efficient. In the sequel of this paper, this approach is referred to as the direct LU-approach. The construction of one block-row of the factors L and U requires two matrix-matrix multiplications of two subblocks of size n, and one complete LU-decomposition of a subblock on the main diagonal. Hence the number of multiplications for the preprocessing is approximately $\frac{7}{3}n^3N$. The number of multiplications for the solution process is approximately $3n^2N$ [1].

A drawback of the direct LU-approach is that the algorithm is strongly recursive: there is no parallelism between the subblocks, so that on parallel computers this technique seems only interesting when the size of the subblocks is large. In that case, a possible approach to obtain parallelism is to distribute each subblock over the processors, and to use PBLAS- and SCALAPACK routines for both the construction of the block LU-decomposition and the triangular solves [3].

In order to obtain more parallelism, one can combine the LU-decomposition with a reordering technique based on a block form of some parallelisable method to solve tridiagonal systems of linear equations, for example recursive doubling [8], cyclic reduction [4], a divide and conquer technique [11], or a domain decomposition method. In this paper, we study two reordering techniques. In Section 2 the complete LU-decomposition is combined with a reordering based on a domain decomposition technique, and in Section 3 we study an approach based on cyclic reduction. Section 4 describes a combination of both approaches. Estimates of the number of floating point operations and the amount of data transport will be discussed. The results of some numerical experiments are presented in Section 5, and in Section 6 some conclusions are presented.

2. Reordering based on domain decomposition

Let p_a be the number of available processors, and suppose that the integer $N_p = \lceil \frac{N}{p_a} \rceil$ [2] represents the number of diagonal blocks to be treated on one processor. The number of processors actually used is equal to $\lceil \frac{N}{N_p} \rceil$. In the sequel of this paper this number is denoted by p. In this section we describe a reordering technique based on a domain decomposition strategy with p non-overlapping subdomains. First we number the unknowns of all blocks, except those of blocks $N_p + 1$, $2N_p + 1$, etc. Those unknowns are numbered last.

After reordering, the block tridiagonal matrix of the resulting system of linear equations

[1] Suppose that one floating point operation (flop) denotes either a multiplication, a division, a subtraction, or an addition of two real variables. Since we use complex arithmetic, and since both in the preprocessing and in the solution process almost every multiplication can be combined with a subtraction or an addition, the number of flops is approximately equal to the number of multiplications multiplied by 8.

[2] By $\lceil x \rceil$ we denote the smallest integer $\geq x$

can be partitioned as

$$\begin{bmatrix} A_1 & 0 & \dots & 0 & A_{1,p+1} \\ 0 & A_2 & \dots & 0 & A_{2,p+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & A_p & A_{p,p+1} \\ A_{p+1,1} & A_{p+1,2} & \dots & A_{p+1,p} & A_{p+1} \end{bmatrix}, \tag{2.1}$$

in which the matrices A_j are block tridiagonal with N_p diagonal blocks for $j = 1$ and with $N_p - 1$ diagonal blocks for $j = 2, \dots, p-1$. The number of diagonal blocks in A_p is equal to $N - 1 - (p-1)N_p$. Note that A_{p+1} is block diagonal with $p-1$ blocks on the main diagonal. The corresponding system of equations is denoted by

$$\begin{bmatrix} C_1 & C_{12} \\ C_{21} & A_{p+1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{2.2}$$

where C_1 is a block diagonal matrix which has the block tridiagonal matrices A_j, $j = 1, \dots, p$ on the main diagonal. One possible strategy is to make a block decomposition of the coefficient matrix and write (2.2) as

$$\begin{bmatrix} I & 0 \\ C_{21}C_1^{-1} & I \end{bmatrix} \begin{bmatrix} C_1 & C_{12} \\ 0 & S \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{2.3}$$

in which S is the Schur-complement $A_{p+1} - C_{21}C_1^{-1}C_{12}$. Note that the action of C_1^{-1} can be computed in parallel, since one can exploit the structure of C_1 mentioned above.

In general, the Schur-complement is dense and its explicit computation should be avoided. However, if one starts with a block tridiagonal matrix, it can be shown that the Schur-complement is also block tridiagonal, and it is relatively easy to compute the action of S^{-1}. A drawback of applying the block decomposition (2.3) directly is that it requires the matrix-vector product $C_1^{-1}y$ twice. Therefore, we consider the approach of constructing an LU-decomposition of the coefficient matrix (2.1).

Assume that we want to construct a block lower-triangular factor L and a block upper-triangular factor U in such a way that $M = LU$ and the subblocks on the main diagonal of U are all unity blocks. The first step is to copy the block lower- and upper-triangular part of M to L and U respectively. Hence after this first step, $L + (U - I)$ has the block structure shown in (2.4)

$$L + (U - I) = \begin{bmatrix} L_1 & 0 & \dots & 0 & U_{1,p+1} \\ 0 & L_2 & \dots & 0 & U_{2,p+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & L_p & U_{p,p+1} \\ L_{p+1,1} & L_{p+1,2} & \dots & L_{p+1,p} & L_{p+1} + (U_{p+1} - I) \end{bmatrix}. \tag{2.4}$$

During the construction of the factors L and U, fill-in blocks of size n can only be generated in L_{p+1} and in U_{p+1}, and in $L_{p+1,j}$ and $U_{j,p+1}$ for $1 < j \leq p$.

To illustrate this, suppose that we have started with a block tridiagonal matrix with $N = 16$ and the matrix has been reordered according to the domain decomposition strategy as described above, where $N_p = 4$. Hence the blocks are numbered in the order

$$1 \quad 2 \quad 3 \quad 4 \quad \textit{14} \quad 5 \quad 6 \quad 7 \quad \textit{15} \quad 8 \quad 9 \quad 10 \quad \textit{16} \quad 11 \quad 12 \quad 13.$$

After calculating the LU-decomposition, the sparsity pattern of $L+U$ can be represented by

$$\begin{bmatrix}
a_1 & a_1 & & & & & & & & & & & & & & \\
a_1 & a_1 & a_1 & & & & & & & & & & & & & \\
 & a_1 & a_1 & a_1 & & & & & & & & & & & & \\
 & & a_1 & a_1 & & & & & & & & & \mathbf{a_1} & & & \\
 & & & & a_2 & a_2 & & & & & & & a_2 & & & \\
 & & & & a_2 & a_2 & a_2 & & & & & & f_2 & & & \\
 & & & & & a_2 & a_2 & & & & & & f_2 & \mathbf{a_2} & & \\
 & & & & & & & a_3 & a_3 & & & & & a_3 & & \\
 & & & & & & & a_3 & a_3 & a_3 & & & & f_3 & & \\
 & & & & & & & & a_3 & a_3 & & & & f_3 & \mathbf{a_3} & \\
 & & & & & & & & & & a_4 & a_4 & & & a_4 & \\
 & & & & & & & & & & a_4 & a_4 & a_4 & & f_4 & \\
 & & & & & & & & & & & a_4 & a_4 & & f_4 & \\
 & & & \mathbf{a_1} & a_2 & f_2 & f_2 & & & & & & & a_2 & f_2 & \\
 & & & & & & \mathbf{a_2} & a_3 & f_3 & f_3 & & & & f_2 & a_3 & f_3 \\
 & & & & & & & & & \mathbf{a_3} & a_4 & f_4 & f_4 & & f_3 & a_4
\end{bmatrix}. \tag{2.5}$$

Herein a_j denotes a block of size n that is already present in (2.4), and f_j denotes a block of size n that can contain fill-in. A fat block corresponds with the interaction of two domains. The indices denote on which processor a block is stored. This small example also illustrates that the fill-in pattern in the blocks $L_{p+1,j}$ and $U_{j,p+1}$ is very regular, and that the fill-in pattern in $L_{p+1} + (U_{p+1} - I)$ is block tridiagonal.

Since no fill-in blocks are generated on processor 1, it pays to make the first domain larger, e.g. to store more blocks on processor 1, in order to improve the load-balancing. However, when the number of processors p is large, this will only have a minor effect on the wall-clock time, and for ease of presentation, we will assume that the blocks are distributed as is illustrated by the example shown in (2.5). When both p and N_p are large, the number of fill-in blocks is approximately $\frac{2}{3}$ times the number of blocks that is already present. In that case, the calculation of one characteristic block-column of the factor L requires two matrix-matrix multiplications with two subblocks, and the construction of a complete LU-decomposition of a subblock on the main diagonal. The calculation of one characteristic block-row of U requires three matrix-matrix multiplications with two subblocks. The interaction of one such a block-column of L and block-row of U generates a contribution to the main diagonal of the Schur-complement. The calculation of this contribution requires another matrix-matrix multiplication with two subblocks. Hence the number of multiplications for the preprocessing is approximately $\frac{19}{3}n^3N$, and the number of multiplications for the solution process is approximately $5n^2N$.

2.1 Speed-up when communication does not dominate

Suppose that $N_p > 2$ and the system with the Schur-complement is solved by a direct LU-approach, e.g. by constructing a complete block LU-decomposition of the block tridiagonal matrix. The part of the matrix that corresponds with the interior of the domains has $N-p+1$ diagonal blocks. Only in the part that corresponds with the interior of domains $2, \ldots, p$ fill-in blocks are generated. The construction of an LU-decomposition of the Schur-complement, which contains $p-1$ diagonal blocks, is a sequential process. Hence the speed-up that can be obtained compared to the direct LU-approach is approximately equal to

$$\frac{N}{\frac{N-p+1-\frac{N}{p}+loss\frac{N}{p}}{loss\times p}+p-1} \tag{2.6}$$

where *loss* is equal to the CPU-time required for the direct LU-approach performed on one processor divided by the CPU-time of the domain decomposition method also performed on one processor. The multiplier *loss* expresses the overhead caused by the extra fill-in blocks, which are not present in a complete block LU-decomposition of the coefficient matrix in the original ordering. For the preprocessing, $loss \approx \frac{7}{19}$, and for the solution process, $loss \approx \frac{3}{5}$.

When the number of processors is sufficiently large, the speed-up is approximately

$$\frac{N}{\frac{N-p+1}{loss\times p}+p-1}. \tag{2.7}$$

When N is large compared to p, it follows from (2.7) that the speed-up compared to the direct LU-approach is approximately $loss \times p$. The ideal number of processors is in the neighbourhood of $\sqrt{\frac{N+1}{loss}}$, leading to a speed-up of approximately

$$\frac{N}{2\sqrt{\frac{N+1}{loss}}-1-\frac{1}{loss}}.$$

Of course, for large N this may not be realistic. The best choice for p is the minimum of $\sqrt{\frac{N+1}{loss}}$ and the number of available processors.

When p is of the same order as N, it makes sense to use the same domain decomposition approach for the system with the Schur-complement. The approach described in Section 3 can be regarded as a special case of a recursive domain decomposition approach.

2.2 Communication costs for the preprocessing

Suppose that the subblocks are divided over the processors as illustrated by the example shown in (2.5). Computation of an LU-decomposition of the block tridiagonal matrices L_j in (2.4) and of the fill-in in $L_{p+1,j}$ and $U_{j,p+1}$ for $j = 1, \ldots, p$ can be done in parallel and requires no communication. One also needs the product $L_{p+1,j}L_j^{-1}U_{j,p+1}$ since this is a part of the Schur-complement S. Most contributions of $L_{p+1,j}L_j^{-1}U_{j,p+1}$ to S are located on the proper processor. Only the contributions to the diagonal blocks of S coming from the fat blocks in (2.5) have to be transported to neighbouring processors. This can be combined with

the calculation of the block LU-decomposition of the block tridiagonal matrix S. This last part cannot be performed in parallel. The number of subblocks that has to be transported from one processor to another is equal to $p-1$.

2.3 Communication costs for the solution process

The solution of $Mx = LUx = b$ for given right-hand side vector b can be done in two steps: first solve z from $Lz = b$, and then solve $Ux = z$. Consider the solution of $Lz = b$:

$$\begin{bmatrix} L_1 & 0 & \dots & 0 & 0 \\ 0 & L_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & L_p & 0 \\ L_{p+1,1} & L_{p+1,2} & \dots & L_{p+1,p} & L_{p+1} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_p \\ z_{p+1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \\ b_{p+1} \end{bmatrix}. \tag{2.8}$$

Note that in a practical implementation the right-hand side vector b can be overwritten by the solution z. Assume that the vectors z_j are located on processor j, for $j = 1, \dots, p$, and z_{p+1} is distributed over the processors according to the way in which L_{p+1} is distributed over the processors (as is illustrated by the example shown in (2.5)). The lower-triangular system (2.8) can be solved in three steps: The solution of $z_1, \dots, z_p$ can be done in parallel and requires no communication. Next, the vectors $L_{p+1,1}z_1, \dots, L_{p+1,p}z_p$ have to be calculated.Again this can be done in parallel and requires no communication. Finally, the vector z_{p+1} has to be calculated. This step cannot be performed in parallel and requires communication: parts of the vectors $L_{p+1,1}z_1, \dots, L_{p+1,p}z_p$ have to be transported to neighbouring processors. To illustrate this, consider the system $Lz = b$ where L is the matrix corresponding with (2.5)

$$\begin{bmatrix}
a & & & & & & & & & & & & & \\
a & a & & & & & & & & & & & & \\
 & a & a & & & & & & & & & & & \\
 & & a & a & & & & & & & & & & \\
 & & & & a & & & & & & & & & \\
 & & & & a & a & & & & & & & & \\
 & & & & & a & a & & & & & & & \\
 & & & & & & & a & & & & & & \\
 & & & & & & & a & a & & & & & \\
 & & & & & & & & a & a & & & & \\
 & & & & & & & & & & a & & & \\
 & & & & & & & & & & a & a & & \\
 & & & & & & & & & & & a & a & \\
 & & & & & & & & & & & & a & a \\
 & & \mathbf{a_1} & a & f & f & & & & & & a & & \\
 & & & & & \mathbf{a_2} & a & f & f & & & \mathbf{f_2} & a & \\
 & & & & & & & & \mathbf{a_3} & a & f & f & \mathbf{f_3} & a
\end{bmatrix}
\begin{bmatrix} z_{1,1} \\ z_{1,2} \\ z_{1,3} \\ z_{1,4} \\ z_{2,2} \\ z_{2,3} \\ z_{2,4} \\ z_{3,2} \\ z_{3,3} \\ z_{3,4} \\ z_{4,2} \\ z_{4,3} \\ z_{4,4} \\ z_{2,1} \\ z_{3,1} \\ z_{4,1} \end{bmatrix} = \begin{bmatrix} b_{1,1} \\ b_{1,2} \\ b_{1,3} \\ b_{1,4} \\ b_{2,2} \\ b_{2,3} \\ b_{2,4} \\ b_{3,2} \\ b_{3,3} \\ b_{3,4} \\ b_{4,2} \\ b_{4,3} \\ b_{4,4} \\ b_{2,1} \\ b_{3,1} \\ b_{4,1} \end{bmatrix}.$$

The first index of the vectors z and b indicates the processor on which it is stored. To solve, for example, the subvector $z_{3,1}$, the matrix-vector product of the subblock $\mathbf{a_2}$ and the subvector

$z_{2,4}$ is added to the matrix-vector product $\mathbf{f_2} z_{2,1}$. Next, the resulting vector is transported from processor 2 to processor 3, and the vector $z_{3,1}$ can be calculated on processor 3.

The number of subvectors of length n that has to be transported from one processor to another is equal to $p-1$. The costs for solving $Ux = z$ are the same. Hence the total number of vectors that has to be transported is $2p-2$.

A drawback of the approach described in this section is that the steps requiring communication cannot be performed in parallel. Hence the domain decomposition approach seems only interesting when large systems of equations have to be solved, in which N is large compared to p. In that case, the sequential part, which is also the part that requires communication, is small compared to the total amount of work.

3. Cyclic reduction

This is the approach described in [4]. For completeness, we include a short description of block cyclic reduction. Suppose that the block tridiagonal linear system is given by

$$s_j x_{j-1} + d_j x_j + t_j x_{j+1} = b_j \qquad j = 1, \ldots, N,$$

in which s_1 and t_N are zero. Another block tridiagonal system, approximately twice as small, can be obtained by multiplying equation $2j-1$ by $-s_{2j} d_{2j-1}^{-1}$, equation $2j+1$ by $-t_{2j} d_{2j+1}^{-1}$ and adding the results to equation $2j$. The new system of equations can be represented by

$$\tilde{s}_{2j} x_{2j-2} + \tilde{d}_{2j} x_{2j} + \tilde{t}_{2j} x_{2j+2} = \tilde{b}_{2j} \qquad j = 1, \ldots, \lfloor \frac{N}{2} \rfloor. \tag{3.1}$$

Herein

$$\tilde{s}_{2j} = -s_{2j} d_{2j-1}^{-1} s_{2j-1}, \tag{3.2}$$

$$\tilde{t}_{2j} = -t_{2j} d_{2j+1}^{-1} t_{2j+1}, \tag{3.3}$$

$$\tilde{d}_{2j} = d_{2j} - s_{2j} d_{2j-1}^{-1} t_{2j-1} - t_{2j} d_{2j+1}^{-1} s_{2j+1}, \tag{3.4}$$

$$\tilde{b}_{2j} = b_{2j} - s_{2j} d_{2j-1}^{-1} b_{2j-1} - t_{2j} d_{2j+1}^{-1} b_{2j+1}. \tag{3.5}$$

This strategy can be repeated until only one block equation with one unknown vector of length n is left. Once x_{2j-2} and x_{2j} are computed, they may be substituted in equation $2j-1$ to compute x_{2j-1}.

Block cyclic reduction can also be described by a complete block LU-decomposition: suppose that the blocks are ordered in a sequence based on cyclic reduction. First number the odd multiples of 2^0, then the odd multiples of 2^1, the odd multiples of 2^2, etc. This appears to be the one-dimensional equivalent of the reordering used in NGILU [2], [9]. For example, if $N = 14$, the blocks are numbered in the order

1 *8* 2 **12** 3 *9* 4 **14** 5 *10* 6 **13** 7 *11*.

After calculating the LU-decomposition, the sparsity pattern of $L+U$ can be represented by

$$\begin{bmatrix}
a & & & & & & a & & & & & & \\
 & a & & & & & a & & & & a & & \\
 & & a & & & & & a & & & a & & \\
 & & & a & & & & a & & & & & a \\
 & & & & a & & & & a & & & & a \\
 & & & & & a & & & a & & & a & \\
 & & & & & & a & & & a & & a & \\
a & a & & & & & & a & & & f_1 & & \\
 & & a & a & & & & & a & & f_1 & & f_1 \\
 & & & & a & a & & & & a & & f_1 & f_1 \\
 & & & & & & a & & & & a & f_1 & \\
 & a & a & & & & & f_1 & f_1 & & & a & & f_2 \\
 & & & & & a & a & & & f_1 & f_1 & & a & f_2 \\
 & & & a & a & & & f_1 & f_1 & & f_2 & f_2 & a
\end{bmatrix}.$$

Herein a represents a subblock already present in the original matrix, and f_j represents a block that can contain fill-in. In the first step of cyclic reduction, the fill-in blocks denoted by f_1 are generated. In the second step of cyclic reduction, the fill-in blocks denoted by f_2 are generated. The fact that there are no fill-in blocks in the neighbourhood of the main diagonal enables an efficient implementation of both the preprocessing and the solution process on supercomputers.

We construct a complete LU-decomposition in such a way that the upper-triangular factor U has unity subblocks on the main diagonal. The blocks $d_{2j-1}^{-1}s_{2j-1}$ and $d_{2j+1}^{-1}t_{2j+1}$ are stored in U. Note that these blocks can be calculated using level 3 BLAS-routines and that the inverses of d_{2j-1} and d_{2j+1} do not have to be constructed: only a complete decomposition of the blocks on the main diagonal is required.

When N is large, the number of fill-in blocks of size n is again approximately $\frac{2}{3}$ times the number of blocks that is already present. Hence the storage requirements are approximately the same as for the method described in Section 2. The calculation of the new coefficient matrices (3.2)-(3.4) requires the calculation of $d_{2j-1}^{-1}s_{2j-1}$ and $d_{2j+1}^{-1}t_{2j+1}$, 4 matrix-matrix multiplications with two subblocks, and the construction of a complete LU-decomposition of the subblock on the main diagonal. Hence the total number of multiplications required for the preprocessing is approximately

$$\tfrac{19}{3}n^3N(\tfrac{1}{2}+\tfrac{1}{4}+\ldots+\tfrac{1}{2}^{2\log N}) = \tfrac{19}{3}n^3N(1-1/N) \approx \tfrac{19}{3}n^3N.$$

The number of multiplications required for the solution process is approximately $5n^2N$.

3.1 Speed-up when communication does not dominate

Eliminating the unknowns belonging to odd blocks can be done in parallel. Both the preprocessing and the solution process parallelise. Suppose that N is larger than p and both N and p are large. At the first step of cyclic reduction p processors can be used to eliminate $\frac{N}{2}$

block equations. Hence the time required for the first step is proportional to $\frac{N}{2p}$. At the first $^2\log N - {}^2\log p$ steps p processors can be used. Hence the time for the first $^2\log N - {}^2\log p$ steps of cyclic reduction is proportional to

$$\frac{N}{p}(\tfrac{1}{2} + \tfrac{1}{4} + \ldots + \tfrac{1}{2}^{{}^2\log N - {}^2\log p}) \approx \frac{N}{p}(1 - \frac{p}{N}).$$

After these $^2\log N - {}^2\log p$ steps have been performed, approximately $^2\log p$ additional steps of cyclic reduction are required to solve the system of linear equations. Those remaining steps are performed on less than p processors: at step k the number of processors is approximately $N \times 2^{-k}$. Hence it follows that the speed-up compared to the direct LU-approach that can be obtained with cyclic reduction is approximately

$$loss\frac{N}{\frac{N-p}{p} + {}^2\log p} = loss\frac{p}{1 + \frac{p}{N}({}^2\log p - 1)}, \tag{3.6}$$

where *loss* is defined in a similar way as in (2.6). Equation (3.6) expresses that when N is large compared to p, the speed-up is almost linear.

3.2 Communication costs for the preprocessing

Suppose that the blocks $s_1, \ldots, s_{N_p}$ are located on processor one, $s_{N_p+1}, \ldots, s_{2N_p}$ are located on processor two etc, and the same holds for the blocks t_j, d_j, $d_{2j+1}^{-1}t_{2j+1}$, and $d_{2j-1}^{-1}s_{2j-1}$. The preprocessing consists of $^2\log N$ steps. At step k the unknowns belonging to blocks with numbers equal to an odd multiple of 2^{k-1} are eliminated, which requires the following operations.

- The computation of a complete LU-decomposition of the blocks on the main diagonal. This requires no communication.
- The computation of the blocks $d_{2j+1}^{-1}t_{2j+1}$, and $d_{2j-1}^{-1}s_{2j-1}$. This requires no communication.
- The computation of the new coefficient blocks $\tilde{s}_{2j}$ and $\tilde{t}_{2j}$. This only requires communication when the blocks with numbers $2j-1$, $2j$, and $2j+1$ are not located on the same processor. When $k \leq {}^2\log N - {}^2\log p$ this happens $p-1$ times, and the number of blocks that has to be transported from one processor to a neighbouring processor is $p-1$. When $k > {}^2\log N - {}^2\log p$, the subblocks are divided over less than p processors: the number of processors p_k is approximately $N \times 2^{-k}$, and the number of blocks to be transported is approximately $p_k - 1$.
- The computation of the new diagonal blocks. The number of subblocks to be transported is approximately $p-1$ when $k > {}^2\log N - {}^2\log p$, and $p_k - 1$ otherwise.

Summarising, the total number of subblocks that has to be transported from one processor to another is approximately

$$2(p-1)({}^2\log N - {}^2\log p) + 2p(\tfrac{1}{2} + \tfrac{1}{4} + \ldots + \tfrac{1}{2}^{{}^2\log p}) - 2\,{}^2\log p$$

$$= 2(p-1)({}^2\log N - {}^2\log p) + 2(p-1) - 2\,{}^2\log p$$

$$= 2(p-1)(1 + {}^2\log N - {}^2\log p) - 2\,{}^2\log p.$$

3.3 Communication costs for the solution process
The solution process consists of two parts: first the right-hand side vectors $\tilde{b}_{2j}$ (3.5) have to be calculated, and the vectors x_j have to be calculated in a back-substitution process. The calculation of $\tilde{b}_{2j}$ only requires communication when the blocks with numbers $2j$, $2j-1$ and $2j+1$ are not located on the same processor. Using the same arguments as above, it can be shown that the number of vectors of length n that has to be transported for the calculation of the new right-hand side vectors is approximately

$$(p-1)(1 + {}^2\log N - {}^2\log p) - {}^2\log p.$$

The back-substitution process requires the same amount of data transport. Hence the total number of vectors that has to be transported is approximately twice this amount.

The estimates for the amount of data transport mentioned in this section are much higher than the communication costs required in the domain decomposition approach. However, most of the work including communication can be performed in parallel.

4. A COMBINED APPROACH

A drawback of the domain decomposition strategy is that the smaller system with the Schur-complement is solved by the direct LU-approach which does not parallelise. Hence when p exceeds a critical value, performance degrades. Block cyclic reduction has the disadvantage that it requires more data transport. A winning strategy might be a combination of both: first reduce the size of the system of linear equations with the domain decomposition strategy, and solve the resulting system with the Schur-complement (which is again block tridiagonal and has $p-1$ diagonal blocks) with block cyclic reduction.

4.1 Speed-up when communication does not dominate
When p is a power of two, ${}^2\log p - 1$ steps of cyclic reduction are required in order to reduce the system with the Schur-complement to only one block-equation. Hence when, in addition, both p and N are large and communication does not dominate (for example, on shared memory computers) the approximate speed-up of the combined approach compared to the direct LU-approach can be expressed as

$$loss\frac{N}{\frac{N-p+1}{p} + {}^2\log p - 1} = loss\frac{p}{1 + \frac{1}{N} + \frac{p}{N}({}^2\log p - 2)}, \tag{4.1}$$

where again *loss* is defined in a similar way as in (2.6). For large p, this is approximately the same as the speed-up that can be obtained with cyclic reduction (3.6). However, the amount of data transport required for the combined approach is considerably less than in the cyclic reduction approach.

4.2 Communication costs for the preprocessing
The calculation of the block tridiagonal Schur-complement with $p-1$ diagonal blocks again requires the transportation of $p-1$ subblocks from one processor to another. The calculation of a complete block LU-decomposition using the cyclic reduction approach requires approximately $2(p-1) - 2\,{}^2\log p$ subblocks to be transported. Hence the total number of subblocks to be transported is approximately

$$3(p-1) - 2\,{}^2\log p.$$

4.3 Communication costs for the solution process
Using the same arguments as in Section 2 and 3, it can be shown that the number of subvectors of length n that has to be transported is approximately

$$2(p-1) + 2(p-1) - 2\,{}^2\log p = 4(p-1) - 2\,{}^2\log p.$$

5. Numerical experiments

In this section, we present the CPU-times measured in seconds necessary for both the preprocessing and the solution process using a complete block LU-decomposition combined with three ordering strategies: no reordering of the blocks (NR), block cyclic reduction (CR), and domain decomposition (DD(N_p)). We will also present results of the combined method on a CRAY T3D. The times for the solution process are the times for applying the triangular solves with the factors L and U once. Hence no iterative refinement is used.

5.1 Results on a workstation
First we present the results of some numerical experiments performed on a Silicon Graphics workstation working on 100 Mhz with a MIPS R4010 floating point chip. We consider the example coming from CASTOR also used in Section 1. Table 2 shows the results of the methods that exploit the sparsity pattern of the subblocks by using the compressed row storage (CRS-)format [1], and Table 3 shows the methods in which all elements of the subblocks were stored, including entries that are zero. In the latter method we use BLAS-routines as much as possible. The results in Tables 2 and 3 show that even on a scalar machine it pays (for

Table 2: CPU-seconds on a workstation. Subblocks are stored in CRS-format.

	NR	CR	DD(7)
preprocessing	0.30	0.45	0.44
solution	0.01	0.02	0.02

the matrices coming from CASTOR) to store all elements of the subblocks, and to get rid of the indirect addressing necessary to exploit the non-zero pattern.

5.2 Results on a CRAY C90
We are interested in a parallel implementation of the algorithms, but on many computer architectures it is also important that the implementations vectorise per processor. Therefore,

Table 3: CPU-seconds on a workstation. Sparsity pattern of subblocks not exploited.

	NR	CR	DD(7)
preprocessing	0.11	0.23	0.20
solution	0.01	0.01	0.01

we performed some tests on one processor of a CRAY C90, which has a theoretical peak performance of 952 Mflop/sec.

In order to be able to experiment easily with the number of diagonal blocks and the size of these blocks, we performed some numerical experiments with a system of linear equations that can easily be generated, with a coefficient matrix that has the same block structure as the matrices coming from CASTOR, and with subblocks which are relatively full. The diagonal blocks of this block tridiagonal system are chosen to be equal to

$$\begin{bmatrix} 0 & 0 & \dots & 0 & 2n \\ 0 & 0 & \dots & 2n & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 2n & 0 & \dots & 0 \\ 2n & 0 & \dots & 0 & 0 \end{bmatrix},$$

and all entries of the sub- and super-diagonal blocks are equal to -1. In the numerical experiments we did of course not exploit the fact that all entries of the sub- and super-diagonal blocks are equal. Note that after performing some steps of a complete block LU-decomposition, the blocks on the main diagonal loose their sparsity. Since the resulting block tridiagonal matrix has, apart from a factor one or minus one, the same determinant as a weakly diagonally dominant M-matrix, it follows that it is non-singular. The right-hand side vector was chosen in such a way that the exact solution is known, and the accuracy of the calculated solution could be checked. In the first numerical experiments, we obtained a speed of approximately 200 Mflop/sec. which is not very good. It appears that when the block size is a multiple of a power of 2, the implementation suffers from memory bank conflicts. One way to overcome this difficulty is to extend each subblock with a dummy row and a dummy column. Tables 4 and 5 show the CPU-times of the methods that construct a complete block LU-decomposition combined with several ordering strategies. The speed in Mflop/sec. is shown in brackets.

Table 4: CPU-seconds and Megaflop rates on one processor of C90 for preprocessing.

	NR	CR	DD(7)
$N = 100,\ n = 65$	0.79 (640)	1.80 (760)	1.66 (760)
$N = 200,\ n = 65$	1.58 (650)	3.69 (750)	3.37 (750)
$N = 100,\ n = 129$	5.68 (700)	14.25 (760)	13.05 (760)

Table 5: CPU-seconds and Megaflop rates on one processor of C90 for solution process.

	NR	CR	DD(7)
$N = 100$, $n = 65$	0.020 (500)	0.028 (600)	0.027 (590)
$N = 200$, $n = 65$	0.041 (490)	0.057 (590)	0.055 (580)
$N = 100$, $n = 129$	0.059 (670)	0.090 (730)	0.085 (730)

The results show that the CPU-times scale linearly with N and that the implementations vectorise very well, which is due to the use of highly optimised BLAS-routines. When n doubles, the CPU-times for the solution process increase with a factor smaller than 4. This is due to the fact that increasing the size of the subblocks is favourable for vectorisation.

5.3 Results on a CRAY T3D

The CRAY T3D (Torus 3 Dimensional interconnecting topology) is a massively parallel machine with physically distributed but logically shared memory. It can contain a maximum of 2048 processor elements (PEs). Each PE consists of a DEC 21064 processor, associated logic, and a connection to the interprocessor communication network. The processors have a theoretical peak performance of 150 Mflop/sec. The local memory of each PE can be accessed by all other PEs across the communication network which has a high bandwidth: the data transfer rate is 300 Mbyte/sec. in each of the 6 directions.

We performed tests on a configuration in Eagan (MN, USA) with 16 PEs. We have implemented the methods described in the previous section by using the data and work sharing programming style. This implies distribution of data and loops that contain the computations on data using directives. We also used the following directives for optimisation:

1. PE_RESIDENT. Ensures the compiler that data access is local.
2. NO BARRIER inserted where allowed after DOSHARED loops. The compiler places an implicit barrier after each DOSHARED loop.

This programming style results in portable code since all changes needed are implemented through directives. Of course the parallelisability is not portable to other architectures without replacing the machine-specific directives.

In order to study the speed-up obtained when using several PEs compared to the direct LU-approach using one PE, we consider two examples which are so small that all data can be stored on one processor: the system of linear equations described in Section 5.2 with $N = 64$ and $N = 128$, and $n = 64$. The timings were obtained with the *rtc* intrinsic function which returns real-time clock values. With the direct LU-approach implemented on one PE, the CPU-times required for the preprocessing and the solution process are shown in Table 6. Again the speed in Mflop/sec. is shown in brackets. The high Megaflop rates are due to the use of optimised BLAS-routines.

Figures 2 and 3 show both the speed-up *predicted* by (2.7), (3.6), and (4.1) (solid line), and the *measured* speed-up using 2,4,8, and 16 PEs ('o'). The speed-ups compared to the

Table 6: CPU-seconds and Megaflop rates on one processor of the T3D for the direct LU-approach.

	preprocessing	solution process
$N = n = 64$	3.80 (81)	0.088 (70)
$N = 128$, $n = 64$	7.65 (81)	0.18 (69)

direct LU-approach using one PE are shown. When two PEs are used this can be less than one, since the methods described in Section 2-4 have some overhead: there are fill-in blocks that are not present in the direct LU-approach. This overhead is expressed in the factor *loss* in (2.7), (3.6), and (4.1). Note that the equations giving the predicted speed-up are based on the assumption that communication does not dominate. Hence one would expect that the actually measured speed-up is less than the predicted speed-up. However, when the same amount of data is distributed over more processors, the performance per processor can increase by cache effects.

From the results it appears that for the domain decomposition strategy the predicted speed-up corresponds quite well with the actually measured speed-up. Since the smaller system with the Schur-complement is solved with the direct LU-approach, this part of the computations cannot be parallelised. When p increases, the size of the last Schur-complement increases also. Hence when the number of PEs exceeds the critical value $\sqrt{\frac{N+1}{loss}}$, performance degrades. In the combined method, the system with the Schur-complement is solved with cyclic reduction. Hence this method has better parallelisation aspects, especially for increasing number of PEs. The actually measured speed-up is quite close to the predicted speed-up, which indicates that the time is not dominated by communication. When cyclic reduction is used for the whole system, the predicted speed-up is slightly smaller than for the combined method. Moreover, with the combined method, the amount of data transport is significantly smaller than with cyclic reduction. Therefore, the measured speed-up using cyclic reduction is less than the measured speed-up obtained by the combined method. When $\frac{N}{p}$ increases, the effects of data transport become less important, and the predicted speed-up becomes more accurate.

6. Conclusions

The block tridiagonal systems of linear equations generated by CASTOR can efficiently be solved by a complete block LU-decomposition. Since the subblocks are relatively full, it is not advisable to try to exploit the sparsity pattern of the subblocks. To achieve good performance, it is better to store all elements of the subblocks, and to use BLAS-routines for the construction of the factors L and U and for the triangular solves. In order to obtain more possibilities for parallelism, the decomposition can be combined with a reordering technique. In this paper, we have considered reordering strategies based on block cyclic reduction, domain decomposition with non-overlapping subdomains, and a combination of these two methods. When communication times do not dominate, and when p is small compared to N, these reordering techniques give approximately the same speed-up compared to the direct LU-approach. However, in many cases the communication can certainly not be neglected. We have given estimates of the amount of data transport, both for the preprocessing and the

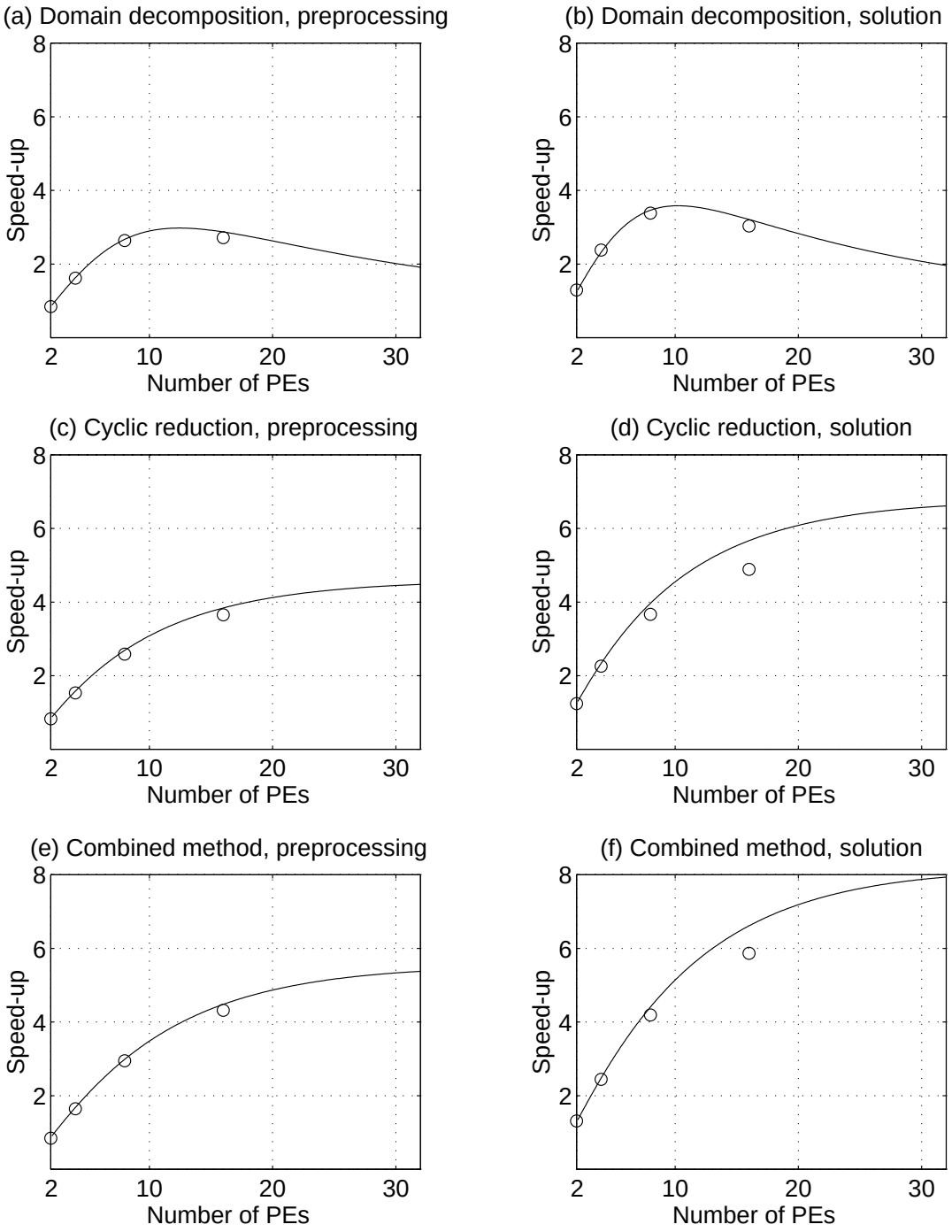

Figure 2: Predicted (solid line) and measured ('o') speed-up compared to the direct LU-approach on a CRAY T3D. $n = 64$ and $N = 64$. Measured speed-ups are shown for $p = 2$, 4, 8, and 16.

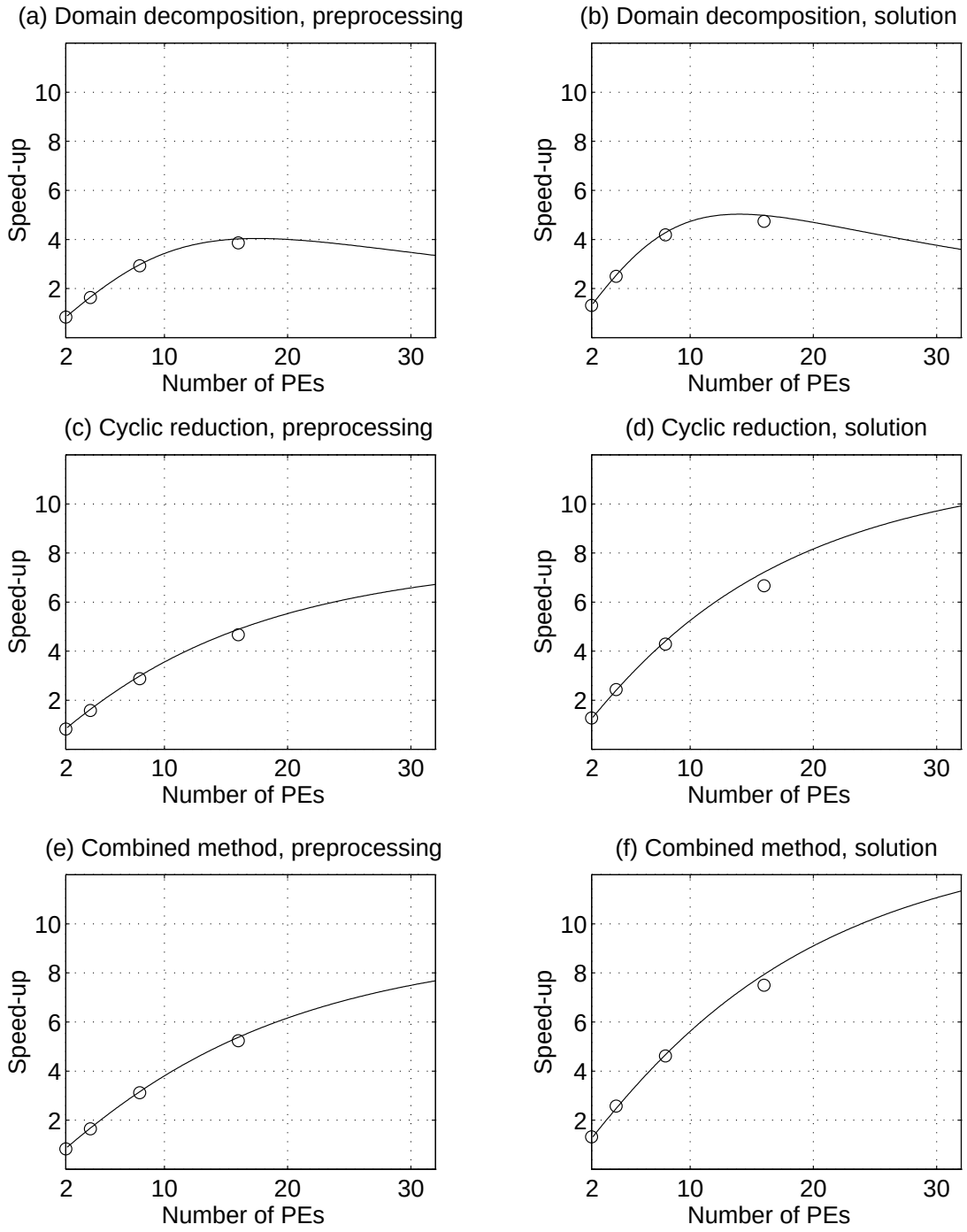

Figure 3: Predicted (solid line) and measured ('o') speed-up compared to the direct LU-approach on a CRAY T3D. $n = 64$ and $N = 128$. Measured speed-ups are shown for p =2, 4, 8, and 16.

solution process. For the cyclic reduction approach these estimates are significantly higher than for the domain decomposition approach. However, with the cyclic reduction approach most of the work including communication can be performed in parallel, whereas in the domain decomposition approach all work requiring communication is sequential in nature. A winning strategy is the combination of both: first reduce the size of the system of linear equations with the domain decomposition strategy, and solve the resulting system with the Schur-complement with block cyclic reduction. Table 7 gives an estimate of the time required for communication, both for the preprocessing and the solution process. Herein mat_s and

Table 7: Communication times approximately.

	Cyclic reduction	Domain decomp.	Combined approach.
Preproc.	$2mat_r\ ^2\log N$	$(p-1)\ mat_s$	$(1+\ 2\ ^2\log p)\ mat_r$
Solution	$2vect_r\ ^2\log N$	$2(p-1)\ vect_s$	$2(1+\ ^2\log p)\ vect_r$

$vect_s$ are the times required to transport a single subblock of size n and a single vector of size n respectively, and mat_r and $vect_r$ are the times required to transport at most $p-1$ blocks and $p-1$ vectors, respectively, in parallel. The choice of the method depends of course on the ratio between $vect_r$ and $vect_s$ and between mat_r and mat_s.

The results on the CRAY T3D confirm that the combined method is the most promising method. The actually measured speed-up is quite close to the predicted speed-up (4.1), which indicates that communication does not dominate.

Acknowledgements
The author wishes to thank drs. M. Nool and dr.ir. H.J.J. te Riele for many stimulating discussions and numerous suggestions for improving the presentation of the paper. He also thanks dr. M.A. Botchev and dr. G.L.G. Sleijpen for carefully reading the paper and suggesting several improvements. He gratefully acknowledges the Dutch National Computing Facilities Foundation NCF for the provision of computer time on the CRAY C90 and CRAY T3D.

References

1. R. Barrett, M. Berry, T.F. Chan, J. Demmel, , J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. Templates for the solution of linear systems: builing blocks for iterative methods. http://www.netlib.org/templates/.
2. E.F.F. Botta and A. van der Ploeg. Preconditioning techniques for matrices with arbitrary sparsity patterns. In *Proceedings of the IX International Conference on Finite Elements in Fluids*, pages 989–998. Università di Padova, October 1995. Venice. Editors: M.M Cechhi, K. Morgan, J. Periaux, and B.A. Schreffer.
3. J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R.C. Whaley. A proposal for a set of parallel basic linear agebra subprograms. LAPACK working note 100.

4. D. Heller. Some aspects of the cyclic reduction algorithm for block tridiagonal linear systems. *SIAM J. Numer. Anal.*, 13:484–496, 1978.

5. W. Kerner, S. Poedts, J.P. Goedbloed, G.T.A. Huysmans, B. Keegan, and E. Schwartz. -. In P. Bachman and D.C. Robinson, editors, *Proceedings of 18th Conference on Controlled Fusion and Plasma Physics*. EPS: Berlin, 1991. IV.89-IV.92.

6. Y. Saad and M.H. Schultz. A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

7. G.L.G. Sleijpen and H.A. van der Vorst. A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 17(2), April 1996.

8. H.S. Stone. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *JACM*, 20:27–38, 1973.

9. A. van der Ploeg, E.F.F. Botta, and F.W. Wubs. Nested grids ILU-decompostion (NGILU). *Journal of Comp. and Appl. Math.*, 66:515–526, January 1996. Proceedings of the Sixth International Congress on Computational and Applied Mathematics.

10. H.A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.

11. H.H. Wang. A parallel method for tridiagonal equations. *ACM Trans. Math. Softw.*, pages 170–183, 1981.