1992

M. Louter-Nool

Numerical multigrid software: MGD5M, a parallel multigrid code with a twisted ILLU-relaxation

Department of Numerical Mathematics Report NM-R9226 December

Numerical Multigrid Software:

MGD5M, A Parallel Multigrid Code with a Twisted ILLU-Relaxation

Margreet Louter-Nool <greta@cwi.nl>
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Abstract.
In this paper we consider the twisted LDU-decomposition for tridiagonal matrices. This technique enables us to partly parallelize the Incomplete Line LU-relaxation(ILLU). This relaxation method is applied as smoothing process for the multigrid method MGD5V[3,10,12]. As a successor of MGD5V we present MGD5M which is a multitasked code based on the twisted ILLU-relaxation. We compare the performance results of MGD5V and MGD5M as measured on a dedicated four processor Cray Y-MP4/464.

1991 Mathematics subject classification: 65N55, 65F10, 65Y05.
1991 CR Categories: G.1.8, G.1.3, G.1.0.

Keywords & Phrases: Elliptic PDEs, ILLU-relaxation, multigrid methods, numerical software, parallel computers, sparse linear systems, twisted LDU-decomposition, vector computers.

Note: The implementation is available in ANSI Fortran 77. A variant, using macro- and autotasking techniques tuned for the Cray Y-MP, is available too.

1. Introduction

The Incomplete Line LU-relaxation(ILLU)[9] is often applied as preconditioning for CG-methods and as smoothing process for multigrid (see [3,5,10]). The ILLU as used within MGD5V[12] is a powerful relaxation method, since only a limited number of multigrid iterations is needed to achieve high accuracy. However, the time per iteration is considerable, partly because this method hardly permits vectorization and parallelization. A new technique to decrease the wall-clock time is to perform the underlying tridiagonal decompositions in a twisted form allowing parallelism. This technique has been proposed by Joubert et al.[4]. In this paper we concentrate on the twisted tridiagonal LDU-decomposition and its influence on the ILLU.

In Section 2, we describe the twisted LDU-factorization. The ILLU-relaxation based on the twisted factorization is treated in Section 3. Next, in Section 4, we describe how the ILLU-relaxation process can be carried out in parallel. In Hemker and De Zeeuw[3], an outline is given of the cycling process of MGD5V and MGD1V, two codes with the same global structure (see also Wesseling[11]). These codes solve linear systems that arise from the discretization of linear elliptic PDEs and share the same prolongation, restriction and Galerkin-approximation of coarse grid systems. The vectorization and parallelization of MGD1M, the multitasked variant of MGD1V, are described in Louter-Nool[7]. In Section 5, we focus on the implementation and performance of those routines, which are relevant for MGD5V and its successor MGD5M and which are not yet discussed in [7]. The execution times on the Cray Y-MP4 for MGD5M solving a simple Poisson equation are listed.

Report NM-R9226
ISSN 0169-0388
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Report NM-R9226
ISSN 0169-0388
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

2. The twisted LDU-factorization

Let A be a tridiagonal matrix of order n. Assume that the linear system $A\,x = y$ has to be solved. A frequently used technique is to factor A as $A = L\,D\,U$, with L lower and U upper bidiagonal, respectively, and D diagonal. If the decomposition exists then it is uniquely defined (and illustrated by Figure 1) by choosing

$$\text{diag}\,(L) = \text{diag}\,(U) = I. \tag{2.1}$$

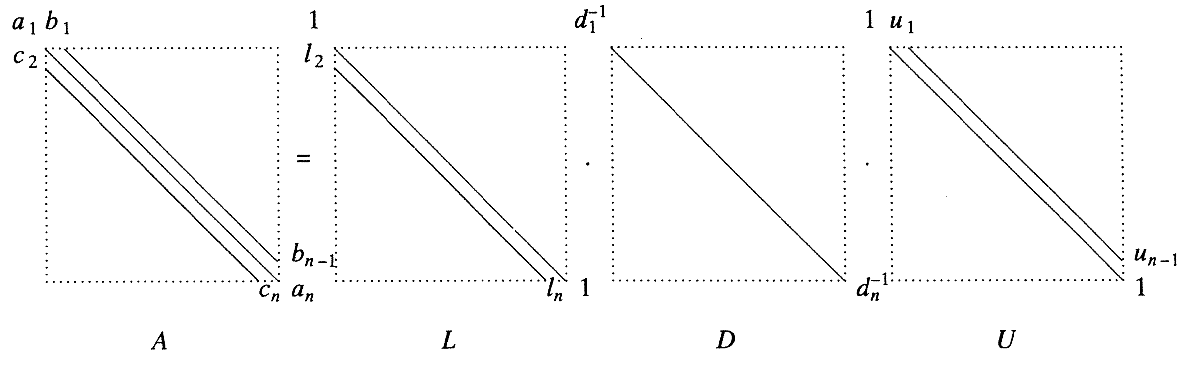

FIGURE 1. The standard LDU-decomposition.

The solution of $A\,x = y$ is replaced by

$$L\,w = y \tag{2.2a}$$

$$D\,v = w \tag{2.2b}$$

$$U\,x = v. \tag{2.2c}$$

Two bidiagonal systems ((2.2a) and (2.2c)) have to be solved. Unfortunately, the recurrent relations for solving from the bidiagonal systems do not vectorize well. Moreover, they can not be performed in parallel.

In Joubert et al.[4] a twisted LU-factorization is proposed. Here we discuss the twisted LDU-factorization. On a vector machine the LDU-decomposition for a tridiagonal matrix leads to a faster solution scheme than the LU-decomposition. We prefer to store the matrix D as prescribed by Figure 1. This implies that the inverse matrix D^{-1} contains the values d_i, $i = 1, \cdots, n$, and therefore solving equation(2.2b) is just a plain vector-multiplication, which can be performed at high vector speed.

Assume n to be odd and let h be defined by

$$h = \frac{n+1}{2},$$

then the twisted LDU-factorization of A can be illustrated by

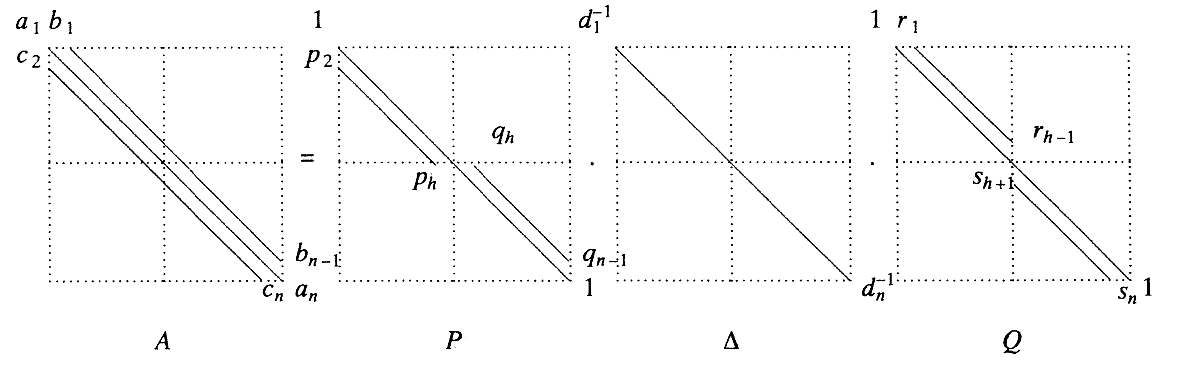

FIGURE 2. The Twisted LDU-decomposition.

According to the definition of D in Figure 1 the matrix Δ of Figure 2 is defined by the reciprocal values d_i^{-1}, $i = 1, \cdots, n$. For $i \geq h$ the elements of D and Δ are not equal.

Algorithm A1 describes the decomposition of the upper part of A, and A2 describes the lower part:

Algorithm A1:

```
$d_1 = \frac{1}{a_1}$
for $i = 2$ to $h-1$
    $p_i = c_i d_{i-1}$
    $d_i = \frac{1}{a_i - p_i b_{i-1}}$
$p_h = c_h d_{h-1}$
for $i = 1$ to $h-1$
    $r_i = b_i d_i$
```

Algorithm A2:

```
$d_n = \frac{1}{a_n}$
for $j = n-1$ to $h+1$ by $-1$
    $q_j = b_j d_{j+1}$
    $d_j = \frac{1}{a_j - q_j c_{j+1}}$
$q_h = b_h d_{h+1}$
for $j = n$ to $h+1$ by $-1$
    $s_j = c_j d_j$
```

Both parts are completely independent and can therefore be computed in parallel. At the end we compute d_h, which depends on values from the upper and lower part:

$$d_h = \frac{1}{a_h - p_h b_{h-1} - q_h c_{h+1}}. \tag{2.3}$$

Since d_h can not be calculated before the computation of the upper and lower part has been accomplished, this point is called the *synchronization* point in the computation.

Algorithm A1 contains a recursion and therefore the computation of p_i and d_i can not be vectorized. At the cost of one extra vector loop of length $h-1$, p_i and d_i can be computed separately, leaving one single recursive loop to compute d_i. This approach, which is described in Algorithm B1, delivers a better performance on vector computers than the A1 approach. A similar scheme to decouple the computation of q_j and d_j is applied in algorithm B2.

Algorithm B1:

```
$d_1 = \frac{1}{a_1}$
$t_i = c_i b_{i-1}$                      $i = 2(1)h-1$
$d_i = \frac{1}{a_i - t_i d_{i-1}}$      $i = 2(1)h-1$
$p_i = c_i d_{i-1}$                      $i = 2(1)h$
$r_i = b_i d_i$                          $i = 1(1)h-1$
```

Algorithm B2:

```
$d_n = \frac{1}{a_n}$
$t_j = b_j c_{j+1}$                      $j = n-1(-1)h+1$
$d_j = \frac{1}{a_j - t_j d_{j+1}}$      $j = n-1(-1)h+1$
$q_j = b_j d_{j+1}$                      $j = n-1(-1)h$
$s_j = c_j d_j$                          $j = n(-1)h+1$
```

The twisted factorization leads to a twisted back substitution, too. In a similar way as in the non-twisted case (2.2a-c), the solution of $A\ x = y$ can be divided into three steps

$$P\,w = y \tag{2.4a}$$

$$\Delta\, v = w \tag{2.4b}$$

$$Q\,x = v. \tag{2.4c}$$

The first two steps can be combined and deliver the following independent systems

Algorithm C1:

```
$v_1 = y_1$
$v_i = y_i - p_i v_{i-1}$                $i = 2(1)h-1$
$v_i = v_i d_i$                          $i = 1(1)h-1$
```

Algorithm C2:

```
$v_n = y_n$
$v_j = y_j - q_j v_{j+1}$                $j = n-1(-1)h+1$
$v_j = v_j d_j$                          $j = n(-1)h+1$
```

For the center point we get

$$v_h = (\ y_h - p_h v_{h-1} - q_h v_{h+1}\ )\, d_h. \tag{2.5}$$

Relation (2.4c) leads to

$$x_h = v_h \tag{2.6}$$

for the center point, and the other elements of x can be computed by Algorithms D1 and D2

Algorithm D1:		**Algorithm D2:**	
$x_i = v_i - r_i\, x_{i+1}$	$i = h-1(-1)1$	$x_j = v_j - s_j\, x_{j-1}$	$j = h+1(1)n$

Note that the computation of the center point ((2.5) plus (2.6)) forms the bottleneck in the parallel process. The upper and lower part can entirely be carried out concurrently on two processors, except for the computation of x_h. As a consequence, the computation is split twice, doubling the parallel overhead.

3. The Incomplete twisted Line LU-relaxation

In Louter-Nool[7], we described how 7-point finite difference discretizations are obtained and how the discretization over a two-dimensional rectangular grid of $n_x \times n_y$ points leads to a linear system $A\,x = y$, where A has the following block-tridiagonal structure

$$A = \begin{bmatrix} B_1 & U_1 & & & & \\ L_2 & B_2 & U_2 & & & \\ & L_3 & B_3 & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & L_{n_y-1} & B_{n_y-1} & U_{n_y-1} \\ & & & & L_{n_y} & B_{n_y} \end{bmatrix}. \tag{3.1}$$

The block matrices B_i are tridiagonal matrices of order n_x, corresponding to the number of points on the horizontal grid lines. We have exactly n_y diagonal-blocks corresponding to the number of horizontal grid lines. The block matrices L_i and U_i are upper and lower bidiagonal matrices, respectively. The matrix A (3.1) has exactly the same structure as employed in MGD1V. The decomposition used by MGD1V, the Incomplete LU (ILU), neglects the block structure and focuses on the seven diagonals. The incomplete line-relaxation of MGD5V is, on the contrary, especially based on the block structure. Following the description in Hemker and De Zeeuw[3], we factorize the matrix A, such that

$$A = (L + D)\,D^{-1}\,(D + U) \tag{3.2}$$

with L, D and U given by, respectively

$$\begin{bmatrix} 0 & & & & & \\ L_2 & 0 & & & & \\ & L_3 & \cdot & & & \\ & & \cdot & \cdot & & \\ & & & \cdot & 0 & \\ & & & & L_{n_y-1} & 0 \\ & & & & & L_{n_y} \; 0 \end{bmatrix}, \quad \begin{bmatrix} D_1 & & & & & \\ & D_2 & & & & \\ & & D_3 & & & \\ & & & \cdot & & \\ & & & & \cdot & \\ & & & & D_{n_y-1} & \\ & & & & & D_{n_y} \end{bmatrix} \text{ and } \begin{bmatrix} 0 & U_1 & & & & \\ & 0 & U_2 & & & \\ & & 0 & \cdot & & \\ & & & \cdot & \cdot & \\ & & & & \cdot & U_{n_y-2} \\ & & & & 0 & U_{n_y-1} \\ & & & & & 0 \end{bmatrix}. \tag{3.3}$$

The blocks L_i and U_i are identical to the ones in (3.1). For the block matrices D_i we obtain the requirement

$$D_1 = B_1 \tag{3.4}$$

$$D_i = B_i - L_i\, D_{i-1}^{-1}\, U_{i-1} \qquad i = 2, \cdots, n_y.$$

Note that the matrices D_i are of full order, which makes the method not very attractive. For that reason, the *Incomplete* Line LU-relaxation has been introduced by Underwood[9]. The term $L_i\, D_{i-1}^{-1}\, U_{i-1}$ is replaced by its truncated tridiagonal form. So (3.4) becomes

$$\tilde{D}_1 = B_1 \tag{3.5}$$
$$\tilde{D}_i = B_i - \textbf{tridiag}\,(\,L_i\, \tilde{D}_{i-1}^{-1}\, U_{i-1}\,) \qquad i = 2, \cdots, n_y.$$

The computation of the matrices $\tilde{D}_i$ must be carried out in sequential order. Hence, the desired parallelism must be introduced at a lower level, for instance, at the level of the computation of the inverse of $\tilde{D}_{i-1}$. This will result in an *Incomplete Twisted Line LU-relaxation.*

At this point our scheme is going to deviate from the original ILLU scheme. By selecting the twisted form we introduce parallelism at the level of grid lines. In the following, $\tilde{D}_{i-1}$ is briefly represented by T. The tri-diagonal matrix T can be factorized in a twisted way

$$T = P\, \Delta\, Q, \tag{3.6}$$

where P, Δ and Q have the same meaning as in Figure 2. As discussed in Section 2, the factorization can be split up into two independent parts. For the inverse of the factorized T we obtain

$$T^{-1} = Q^{-1}\, \Delta^{-1}\, P^{-1}. \tag{3.7}$$

This implies that the inverse of Q as well as the inverse of P are required. The inverse of Q is explicitly given by

$$Q^{-1} = \begin{bmatrix} 1 & \rho_{1,2} & \rho_{1,3} & \cdot & \rho_{1,h} & & & & \\ & 1 & \rho_{2,3} & \cdot & \rho_{2,h} & & & & \\ & & 1 & \cdot & \cdot & & & & \\ & & & \cdot & \rho_{h-1,h} & & & & \\ & & & & 1 & & & & \\ & & & & \sigma_{h+1,h} & \cdot & & & \\ & & & & \cdot & \cdot & 1 & & \\ & & & & \sigma_{n_x-1,h} & \cdot & \sigma_{n_x-1,n_x-2} & 1 & \\ & & & & \sigma_{n_x,h} & \cdot & \sigma_{n_x,n_x-2} & \sigma_{n_x,n_x-1} & 1 \end{bmatrix} \tag{3.8}$$

with

$$\rho_{i,j} = (-1)^{j-i} \prod_{k=i}^{j-1} r_k \qquad i = 1, \cdots, h-1; j = i+1, \cdots, h \tag{3.9}$$

and

$$\sigma_{i,j} = (-1)^{i-j} \prod_{k=j+1}^{i} s_k \qquad i = h+1, \cdots, n_x; j = h, \cdots, i-1. \tag{3.10}$$

The inverse of P is given by

$$P^{-1} = \begin{bmatrix} 1 & & & & & & & & \\ \phi_{2,1} & 1 & & & & & & & \\ \phi_{3,1} & \phi_{3,2} & \cdot & & & & & & \\ \cdot & \cdot & \cdot & 1 & & & & & \\ \phi_{h,1} & \phi_{h,2} & \cdot & \phi_{h,h-1} & 1 & \psi_{h,h+1} & \cdot & \psi_{h,n_x-1} & \psi_{h,n_x} \\ & & & & & 1 & \cdot & \cdot & \cdot \\ & & & & & & \cdot & \psi_{n_x-2,n_x-1} & \psi_{n_x-2,n_x} \\ & & & & & & & 1 & \psi_{n_x-1,n_x} \\ & & & & & & & & 1 \end{bmatrix} \tag{3.11}$$

with

$$\phi_{i,j} = (-1)^{i-j} \prod_{k=j+1}^{i} p_k \qquad i = 2, \cdots, h; j = 1, \cdots, i-1 \qquad (3.12)$$

and

$$\psi_{i,j} = (-1)^{j-i} \prod_{k=i}^{j-1} q_k \qquad i = h, \cdots, n-1; j = i+1, \cdots, n. \qquad (3.13)$$

For the incomplete decomposition (3.5) only *five* diagonals of T^{-1} (3.7) are of interest. We denote them by $[t^{(-2)}, t^{(-1)}, t^{(0)}, t^{(1)}, t^{(2)}]$. The main diagonal is represented by $t^{(0)}$, whereas $t^{(1)}$ and $t^{(2)}$ represent the super- and $t^{(-2)}$ and $t^{(-1)}$ the sub-diagonals. The upper and lower part of the diagonals can be computed concurrently, as is expressed by Algorithm E1 and E2.

$$t_h^{(0)} = d_h \; ; \; t_{h-1}^{(2)} = d_h \, r_{h-1} \, q_h \; ; \; t_{h+1}^{(-2)} = d_h \, s_{h+1} \, p_h \qquad (3.14)$$

Algorithm E1:		**Algorithm E2:**	
$t_i^{(0)} = t_{i+1}^{(0)} \, p_{i+1} \, r_i + d_i$	$i = h-1(-1)1$	$t_j^{(0)} = t_{j-1}^{(0)} \, q_{j-1} \, s_j + d_j$	$j = h+1(1)n_x$
$t_i^{(1)} = -t_{i+1}^{(0)} \, r_i$	$i = h-1(-1)1$	$t_{j-1}^{(1)} = -t_{j-1}^{(0)} \, q_{j-1}$	$j = h+1(1)n_x$
$t_{i+1}^{(-1)} = -t_{i+1}^{(0)} \, p_{i+1}$	$i = h-1(-1)1$	$t_j^{(-1)} = -t_{j-1}^{(0)} \, s_j$	$j = h+1(1)n_x$
$t_i^{(2)} = -t_{i+1}^{(1)} \, r_i$	$i = h-2(-1)1$	$t_{j-2}^{(2)} = -t_{j-2}^{(1)} \, q_{j-1}$	$j = h+2(1)n_x$
$t_{i+2}^{(-2)} = -t_{i+2}^{(-1)} \, p_{i+1}$	$i = h-2(-1)1$	$t_j^{(-2)} = -t_{j-1}^{(-1)} \, s_j$	$i = h+2(1)n_x$

Now, the description of the inverse of T has been completed, since we know how to compute the relevant diagonals of $\tilde{D}_{i-1}^{-1}$. The next step is the computation of the tridiagonal matrix in

$$\tilde{D}_i = B_i - \textbf{tridiag} \, (L_i \, \tilde{D}_{i-1}^{-1} \, U_{i-1}),$$

which can be carried out analogously to the original scheme (see [8], formula(2.27)).

4. THE SMOOTHING PROCESS

The ILLU-relaxation has been used as a smoothing process in MGD5V. At each level the vector x in equation

$$A \, x = y \qquad (4.1)$$

is overwritten by a better (smoothed) approximation. Rewriting the decomposition (3.2) as

$$A = (L \, D^{-1} + I) (D + U), \qquad (4.2)$$

the iterative method becomes:

$$r = y - A \, x \qquad (4.3)$$

$$(L \, D^{-1} + I) \, z = r \qquad (4.4a)$$

$$(D + U) \, c = z \qquad (4.4b)$$

$$x = x + c. \qquad (4.5)$$

We define x, r, c and z as

$$x = (x_1^{\mathrm{T}} \mid x_2^{\mathrm{T}} \mid \cdots \mid x_{n_y}^{\mathrm{T}})^{\mathrm{T}},$$

$$r = (r_1^{\mathrm{T}} \mid r_2^{\mathrm{T}} \mid \cdots \mid r_{n_y}^{\mathrm{T}})^{\mathrm{T}},$$

$$c = (c_1^{\mathrm{T}} \mid c_2^{\mathrm{T}} \mid \cdots \mid c_{n_y}^{\mathrm{T}})^{\mathrm{T}},$$

$$z = (z_1^{\mathrm{T}} \mid z_2^{\mathrm{T}} \mid \cdots \mid z_{n_y}^{\mathrm{T}})^{\mathrm{T}},$$

respectively, where x_j, r_j, c_j and z_j, $j = 1,...,n_y$ are vectors of length n_x. These definitions, together with the block structures of L, D and U as given by (3.3), enable us to rewrite (4.4a) into n_y smaller systems:

$$z_1 = r_1 \qquad (4.6a)$$

$$z_j = r_j - L_j \tilde{D}_{j-1}^{-1} z_{j-1} \qquad j = 2, \cdots, n_y$$

and equation (4.4b) becomes

$$c_{n_y} = \tilde{D}_{n_y}^{-1} z_{n_y} \qquad (4.6b)$$

$$c_j = \tilde{D}_j^{-1} (z_j - U_j c_{j+1}) \qquad j = n_y{-}1(-1)1.$$

Finally, the improved solution is given by

$$x_j = x_j + c_j \qquad j = 1, \cdots, n_y. \qquad (4.7)$$

The computation of z_j in (4.6a) corresponds to the solution of two bidiagonal systems of order n_x. The first one operates from the left and right boundary points to the middle of the horizontal grid line. After that point has been computed the second one starts at the center and operates outerwards, analogously to the solution process described by Algorithms C1, C2 and D1 and D2. Note, that first all vectors z_j have to be updated before the corrections c_j, $j = n_y, \cdots, 1$ as described by (4.6b) can be computed sequentially from the upper grid line down to the lower grid line.

5. The Efficiency on the Cray Y-MP

5.1. *Definitions and test problem*

We refer to the new *multitasked* implementation of MGD5V as MGD5M. This variant is, just like MGD5V, written in ANSI Fortran. By means of compiler directives the code has been adapted for autotasking[2] on the Cray Y-MP. Also macrotasking has been applied[2]. Actually, there are two implementations, one vector and one parallel variant. A reason for this is that there are all kinds of parallel overhead, varying from threshold tests, load balancing to synchronization overhead. Besides, although the computational complexity of the twisted factorization and relaxation is exactly equal to the standard decomposition, the vector performance may be less for the twisted case. To minimize the parallel overhead, it is often preferable to split the computation into (large) equal parts that can be performed concurrently, rather than to switch constantly from two to four processors. We will return to this point later on in this section. Therefore, in the definition of the parallel speedup we may not neglect the parallel overhead. The purely vectorized version, not based on the twisted decomposition, has been compiled with the –Zv option, and so has the original code MGD5V. The parallel implementation has been compiled with –Zp.

The vector speedup is expressed by

$$S_{vector} = \frac{\textit{Execution time of } \text{MGD5V}}{\textit{Execution time of } \text{vector-MGD5M}}. \qquad (5.1)$$

The execution times have both been measured on a dedicated Cray Y-MP4. The parallel speedup is denoted by

$$S_p = \frac{\textit{Execution time of } \text{vector-MGD5M } \textit{on } 1 \textit{ processor}}{\textit{Execution time of } \text{MGD5M } \textit{on } \text{p } \textit{processors}}. \qquad (5.2)$$

Again the values have been obtained in dedicated mode. The total acceleration of MGD5M with respect to MGD5V is the product of S_{vector} and S_p.

As a test problem we solve the Poisson equation on the unit square with Dirichlet boundary conditions, zero initial estimates and the right-hand-side constructed according to the exact solution

$$x(1-x)+y(1-y). \tag{5.3}$$

This problem was also used to measure the acceleration of the MGD1M[7] compared with the vector code MGD1V by De Zeeuw[12]. The convergence of both methods is so rapid that only a few multigrid iterations are needed, viz., 5 multigrid iterations for MGD5V and 6 iterations for MGD1V. After that number of iterations the Euclidian norm of the residu is less than 10^{-9}. All the mentioned codes use a fixed multigrid strategy. Therefore the particular choice of the test problem is not important. Only the required number of multigrid iterations depends on the problem. Except for the twisted factorizations, MGD5M operates in a similar way as MGD5V. In [10] Van der Vorst remarks that parallel incomplete decompositions often lead to a decrease in the number of iteration steps. For more complicated problems such as the convection-diffusion or the anisotropic diffusion equation, this might be an extra reason to use MGD5M instead of MGD5V. We plan to pay attention to this phenomenon in the near future. For a detailed discussion on the robustness of MGD5M, we refer to [8].

5.2. Performance of the twisted factorization and relaxation

In MGD5M the twisted LDU-factorization on the level of horizontal grid lines has been introduced. As mentioned before this technique has only been applied at the finest grid. The ILLU is performed by the subroutines ILLUDC and ILLUDT. The latter performs the twisted LDU. In Table 1 the Megaflop rates of three different runs are listed. In the first two columns results of MGD5V by De Zeeuw[12] and the vector-MGD5M are shown. The third column shows the performance of ILLUDT executed on two processors. Actually, three processors were involved then. During the twisted decomposition one processor is computing the non-twisted decomposition on coarser grids. We will return to this way of parallelism later on. The results presented have been obtained by runs on seven and eight levels of discretization. For both cases the coarsest grid consists of 5x5 grid points. Dealing with seven levels, we have a finest grid of 257x257 grid points, on which the twisted factorization is carried out. For the eight level case the finest grid contains 513x513 points, and leads to a parallel speedup for the twisted form of S_2 = 1.85.

Number of grid points	MGD5V	MGD5M		
	Vector	Vector	$p=2$	S_2
257 x 257	54	59	91	1.54
513 x 513	56	61	113	1.85

TABLE 1. Mflops of ILLUDC and ILLUDT: Influence of parallel overhead.

The parallel overhead caused by autotasking plays an important role in the performance of the Cray Y-MP. We have timed three different parallel and one vector implementations (see Table 2) of the solution routine SOLVE performing (4.6a-b). The three parallel variants are illustrated in Figure 3.

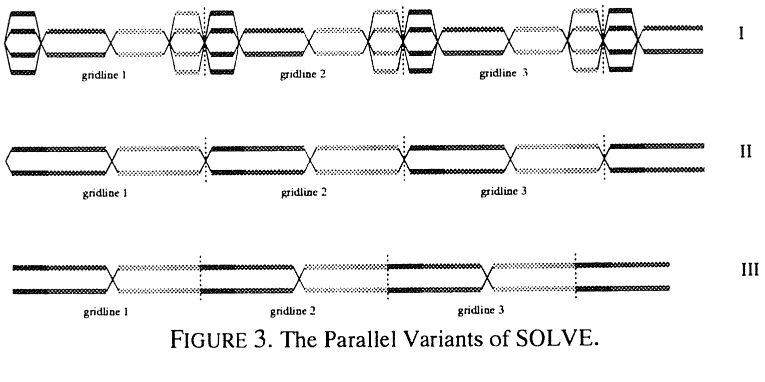

FIGURE 3. The Parallel Variants of SOLVE.

First we observe that apart from the bidiagonal systems which have to be solved, other operations must be carried out in this routine. Those operations can easily be distributed over more than two processors. However, we have not measured a reduction in the wall-clock time when four processors were used instead of two. The Megaflop rates we observed for this variant can be found in the second column (Parallel I). A second approach (Parallel II) divides the computational work per grid line into two equal parts using the parallel CASE-construction (see [1]). The best performance (Parallel III), however, is achieved when the backward sweep on grid line j is immediately followed by the forward sweep on grid line $j+1$ performing (4.6a). Thus, the synchronization point has been moved to the beginning of the grid line loop. This reduces the parallel overhead by a factor of 2. In a similar way (4.6b) has been implemented.

	Vector	Parallel I	Parallel II	Parallel III
257×257	39	49	52	59
513×513	41	59	65	70

TABLE 2. Mflops of SOLVE: Influence of parallel overhead.

5.3. Execution times

Finally, we review the total execution time of the main computational parts. In Table 3 the times achieved for computation on 7 levels are listed, whereas Table 4 gives results on eight levels. In the first column the results of original MGD5V by De Zeeuw[12] are given. The second and third column show the times of runs with the vector-MGD5M, and the vector speedup as defined by (5.1). The columns 4 till 7 show the performance of MGD5M on $p = 1,2,3,4$ processors and the parallel speedup defined by (5.2).

The highest speedup factor (vector as well as parallel) is achieved by Galerkin. In [6] Lioen describes how for the Galerkin approximations the amount of work per grid point can be reduced from 191 (cf. Wesseling[11]) to 74 floating point operations. Moreover, this strongly improved implementation appears to be highly parallel. On four processors we obtain a parallel speedup of 3.70. In Tables 3 and 4, the time needed to compute the decompositions on all grids is given. The decompositions on different grids are independent, so we may execute them concurrently. The time needed to compute the decompositions on all grids except the finest is about one third of the time needed to decompose the finest. This implies that the decomposition time for all grids together can be made approximately equal to the time for the twisted decomposition on the finest grid when using three or more processors.

From Tables 3 and 4 we may conclude that the main gain in performance for the multigrid iteration process denoted by MG-cycles arises from parallelization and not from vectorization. MGD5V, originally tuned for the Cyber 205, turns out to be efficient on a one-processor Cray Y-MP, too. We observe that the parallel speedup increases with the number of processors, although the most time-consuming part, the ILLU-relaxation is optimal for two processors. This is caused by the other well parallelized techniques applied in the multigrid smoothing process, such as prolongation and restriction. MGD1M uses exactly the same operators to transfer values from a fine grid to a next coarser grid and vice versa. For the performance on the Cray Y-MP4 of the routines PROLON and RESTRI, which perform the prolongation and restriction, respectively, we refer to Louter-Nool[7].

Finally, we want to make a small comparison between MGD1M and MGD5M. From Sonneveld et al.[8] we know that MGD5M is more robust than MGD1M: there exist problems that can be solved by MGD5M and not by MGD1M. Provided MGD1M can solve the same problem as MGD5M with similar convergence rate, which method is to be prefered at a Cray Y-MP4? For eight levels the minimum execution time for one iteration step of MGD1M is $.272/6 = .045$ and of MGD5M $.647/5 = .129$, or in other words a MGD1M iteration step is about three times faster than a MGD5M iteration step. The ratio for the original codes is 1.64. This indicates that for the Cray Y-MP MGD1M becomes more and more attractive compared to MGD5M, provided that MGD1M can solve the problem with the same accurency as MGD5M.

	MGD5V	MGD5M					
	Vector	Vector	S_{vector}	$p = 1$	$p = 2$	$p = 3$	$p = 4$
Galerkin	.037	.010		.010	.005	.005	.004
Speedup			3.70		2.00	2.00	2.50
Decompose	.064	.059		.064	.046	.034	.033
Speedup			1.08		1.28	1.73	1.79
MG-cycles	.305	.279		.303	.198	.188	.185
Speedup			1.09		1.41	1.48	1.51
Total	.406	.348		.377	.249	.227	.222
Speedup			1.17		1.40	1.53	1.57

TABLE 3. Wall-Clock time in seconds and speedups for 7 Levels.

	MGD5V	MGD5M					
	Vector	Vector	S_{vector}	$p = 1$	$p = 2$	$p = 3$	$p = 4$
Galerkin	.130	.037		.037	.019	.013	.010
Speedup			3.51		1.95	2.85	3.70
Decompose	.239	.221		.227	.153	.102	.101
Speedup			1.08		1.44	2.17	2.19
MG-cycles	1.133	1.045		1.099	.689	.660	.647
Speedup			1.08		1.52	1.58	1.62
Total	1.502	1.303		1.357	.861	.775	.758
Speedup			1.15		1.51	1.68	1.72

TABLE 4. Wall-Clock time in seconds and speedups for 8 Levels.

ACKNOWLEDGEMENT
The author would like to thank Paul de Zeeuw for his clear introduction in the multigrid software, especially the complicated ILLU. His explanation enabled her to optimize the multigrid code so efficiently, and to do this within a very short period. The author is also very grateful to the people of SARA for helping her to run the timing programs on a dedicated machine.

Financial support for this work was provided by Cray Research, Inc. under a 1991 University Research & Development Grant for the project 'Numerical Multigrid Software for the Cray Y-MP4'. This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF) for use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO).

REFERENCES
1. CRAY CF77 COMPILING SYSTEM (1990). Volume 4: Parallel Processing Guide, Publication SG-3074 4.0.
2. CRAY MULTITASKING PROGRAMMER'S MANUAL (1989). publication SR-0222 F.
3. P.W. HEMKER and P.M. DE ZEEUW (1985). Some implementations of multigrid linear systems solvers, in: D.J. Paddon and H. Holstein, Eds., *Multigrid Methods for Integral and Differential Equations*, Inst. Math. Appl. Conf. Ser. New Ser. (Oxford Univ. Press, New York), pp. 85-116.
4. G.R. JOUBERT and E. CLOETE (1985). The Solution of Tridiagonal Linear Systems with an MIMD

Parallel Computer, *ZAMM Z. Angew. Math. u. Mech.*, Vol. 65, No 4, pp. T383-385.

5. R. KETTLER (1982). Analysis and Comparison of Relaxation Schemes in Robust Multigrid and Preconditioned Conjugated Gradient Methods, in: W. Hackbusch and U. Trottenberg, Eds., *Lecture Notes in Mathematics*, 960 (Springer, Berlin), pp. 502-534.
6. W.M. LIOEN (to appear). *An optimally efficient algorithm for Galerkin Coarse-grid approximation*, CWI.
7. M. LOUTER-NOOL (1992). *MGD1M, a parallel multigrid code with a fast vectorized ILU-relaxation*, CWI Report NM-R9222.
8. P. SONNEVELD, P. WESSELING and P.M. DE ZEEUW (1985). Multigrid and Conjugate Gradient Methods as convergence acceleration techniques, in: D.J. Paddon and H. Holstein, Eds., *Multigrid Methods for Integral and Differential Equations* , Inst. Math. Appl. Conf. Ser. New Ser. (Oxford Univ. Press, New York), pp. 117-167.
9. R.R. UNDERWOOD (1976). *An approximate factorization procedure based on the block Cholesky decomposition and its use with the conjugate gradient method*, Report NEDO-11386, General Electric Co., Nuclear Energy Div., San Jose, California.
10. H.A. VAN DER VORST (1987). Large tridiagonal and block tridiagonal linear systems on vector and parallel computers, *Parallel Computing*, Vol 5, pp. 45-54.
11. P. WESSELING (1982). A robust and efficient multigrid method, in: W. Hackbusch and U. Trottenberg, Eds., *Lecture Notes in Mathematics*, 960 (Springer, Berlin), pp. 614-630.
12. P.M. DE ZEEUW (1986). NUMVEC FORTRAN library manual. Chapter: Elliptic PDEs, Routine: MGD1V and MGD5V; CWI Report NM-R8624.