PARALLELIZATION OF ROBUST MULTI-GRID METHODS: ILU FACTORIZATION AND FREQUENCY DECOMPOSITION METHOD*

PETER BASTIAN† AND GRAHAM HORTON†

Abstract. The parallelization of two multi-grid methods that are robust for systems of linear equations arising from the discretization of certain singularly perturbed elliptic partial differential equations is presented. The multi-grid method with ILU smoother and the Frequency Decomposition Method, based on a multiple coarse grid correction, were implemented on a MIMD computer with distributed shared memory using a ring configuration for the first and a hypercube configuration for the second method. The speedups were determined for various grid sizes and numbers of processors. An objective comparison of both methods against the multi-grid method with highly parallelizable red-black Gauß-Seidel smoother is made for the anisotropic equation and shows the superiority of both methods over the standard approach. The speedup of the ILU smoother is modelled and the influence of computer architecture on the speedup is discussed.

Key words. singularly perturbed partial differential equation, robust multi-grid method, incomplete factorization, MIMD computer, distributed shared memory

AMS(MOS) subject classifications. 65F10

1. Introduction. In this paper we study the parallel solution of systems of linear equations arising from the discretization of partial differential equations which depend on one or more parameters. The equation is assumed to be basically of elliptic type, except for certain extremal values of the parameters. As model equations for this class of singularly perturbed problems we consider the anisotropic equation

$$-\alpha \frac{\partial^2 u}{\partial x^2} - \beta \frac{\partial^2 u}{\partial y^2} = f(x,y) \tag{1}$$

on $\Omega = [0,1] \times [0,1]$, with

$$u(x,y) = b(x,y) \text{ or } \frac{\partial u}{\partial n} = b(x,y) \text{ on } \partial\Omega \tag{2}$$

and $\alpha, \beta \geq 0$, and n the exterior normal. Further we consider the convection-diffusion equation

$$-\epsilon \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) + c_1 \frac{\partial u}{\partial x} + c_2 \frac{\partial u}{\partial y} = f(x,y) \tag{3}$$

on $\Omega = [0,1] \times [0,1]$, $u(x,y) = b(x,y)$ on $\partial\Omega$, $\epsilon \geq 0$, $c_1, c_2 \in \mathbf{R}$.

The anisotropic equation is of elliptic type for $\alpha, \beta > 0$ and of parabolic type for $\alpha = 0$ or $\beta = 0$. The convection-diffusion equation is of elliptic type for $\epsilon > 0$ and of hyperbolic type and first order if $\epsilon = 0$. These two equations can serve as model equations for the numerical solution of the Navier-Stokes equations, where the influence of grids with high aspect ratio or a flow with high Reynolds number on the rate of convergence can be studied.

* This research was supported in part by the Stiftung Volkswagenwerk within its program 'Entwicklung von Berechnungsverfahren für Probleme der Strömungstechnik'.

† Institut für Mathematische Maschinen und Datenverarbeitung III, Universität Erlangen-Nürnberg, Erlangen, Federal Republic of Germany

A standard finite difference discretization of Eqn. (1) with Dirichlet boundary conditions on an equidistant grid $\Omega_l = \{(ih_l, jh_l) | 0 \leq i,j \leq N_l, h_l = 1/N_l, N_l = 2^{l+1}\}$ yields a system of linear equations

$$A_l u_l = f_l \tag{4}$$

where the index $l \in \mathbf{N}$ indicates the grid level within the multi-grid process. A_l is a block tridiagonal matrix if the grid points are ordered lexicographically first in x then in y direction:

$$A_l = h_l^{-2} \begin{pmatrix} T_l & -\beta I & & & \\ -\beta I & T_l & -\beta I & & \\ & \ddots & \ddots & \ddots & \\ & & -\beta I & T_l & -\beta I \\ & & & -\beta I & T_l \end{pmatrix} \tag{5}$$

The diagonal blocks T_l have the form

$$T_l = \begin{pmatrix} 2(\alpha+\beta) & -\alpha & & & \\ -\alpha & 2(\alpha+\beta) & -\alpha & & \\ & \ddots & \ddots & \ddots & \\ & & -\alpha & 2(\alpha+\beta) & -\alpha \\ & & & -\alpha & 2(\alpha+\beta) \end{pmatrix} \tag{6}$$

Eqn. (4) will serve as model equation throughout the paper.

If $K_l(\alpha,\beta)$ is the iteration matrix of an iterative method for the solution of the discrete problem Eqn. (4), then the method $K_l(\alpha,\beta)$ is called robust for the anisotropic equation iff

$$\|K_l(\alpha,\beta)\| \leq \zeta < 1 \quad \forall \alpha,\beta \geq 0,\ l \in \mathbf{N} \tag{7}$$

with $\|.\|$ the spectral norm.

We assume a central difference discretization of the diffusive terms of the convection-diffusion equation (3), and stable upwind differences for the convective terms. The robustness of a method for a discretization of Eqn. (3) can be defined similarly.

The first method we consider is the multi-grid method with ILU smoother. This method is suitable for a finite difference or finite volume discretization of a wide range of two-dimensional partial differential equations (see [11]) including Eqns. (1) and (3). Several variants of the method have been published, for example the semi-implicit procedure (SIP) of Stone [10], 7- and 9-point ILU, or Wittum's ILU$_\beta$. Wittum showed in [12] that the multi-grid method with his ILU$_\beta$ smoother is robust for the anisotropic equation. The ILU$_\beta$ and the SIP method have both been succesfully used in Navier-Stokes solvers (see [3], [13]). The parallelization of the ILU method presented in section three is applicable to all of the above-mentioned variants and is described only for the basic 5-point version.

The second method we consider is the frequency decomposition multi-grid algorithm introduced by Hackbusch in [5]. The range of applicability of the method in this form is not as large as for the first method: it is able to solve Eqn. (1) but not Eqn. (3). The motivation behind the method is as follows: Let $K_l(\alpha,\beta)$ denote the iteration matrix of an iterative algorithm for the solution of Eqn. (4), then the error

in the n-th iterate $v_l^n = u_l - u_l^n$ is given by $v_l^n = K_l(\alpha, \beta)v_l^{n-1}$. With the eigenvectors $e_l^{\nu\mu}$ of A_l given by

$$(e_l^{\nu\mu})_{i,j} = \sin(\nu\pi i h_l)\sin(\mu\pi j h_l), \quad 0 \le \mu, \nu \le N_l - 1 \tag{8}$$

one defines the low frequencies $V_{00} = \text{span}\{e_l^{\nu\mu} | \nu, \mu \in \mathcal{L}\}$, the high frequencies in x-direction $V_{10} = \text{span}\{e_l^{\nu\mu} | \nu \in \mathcal{H}, \mu \in \mathcal{L}\}$, the high frequencies in y-direction $V_{01} = \text{span}\{e_l^{\nu\mu} | \nu \in \mathcal{L}, \mu \in \mathcal{H}\}$ and the high frequencies in both directions $V_{11} = \text{span}\{e_l^{\nu\mu} | \nu, \mu \in \mathcal{H}\}$ with $\mathcal{L} = \{1, \ldots, N_{l-1} - 1\}$ and $\mathcal{H} = \{N_{l-1}, \ldots, N_l - 1\}$. Standard multi-grid theory assumes the smoother to reduce the high frequency errors $v \in V_{10} \cup V_{01} \cup V_{11}$ and the coarse grid correction to reduce the low frequency errors $v \in V_{00}$. Fourier analysis of the two-grid method with a damped Jacobi smoother or a red-black Gauß-Seidel smoother applied to Eqn. (4) yields that the smoother fails to reduce the errors in V_{10} (V_{01}) if $\alpha \to 0$ ($\beta \to 0$). A first way to achieve robustness is to incorporate a more complex smoothing algorithm that works for all high frequencies, like the ILU method. Another approach is the frequency decomposition method, where four different coarse grid corrections are constructed to reduce the error in each of the spaces V_ι, $\iota \in \{00, 10, 01, 11\}$ on coarser grids. As smoother a simple damped Jacobi is sufficient.

In the next section the principles of parallel multi-grid algorithms will be shortly reviewed. Sections three and four describe the ILU iteration and the frequency decomposition method, respectively, together with the parallelization and speedup results for each method. In the fifth section the two methods will be compared with a parallel multi-grid method using a highly parallelizable red-black Gauß-Seidel smoother.

2. Parallelization of Multi-Grid Methods. When we assume that a two-dimensional partial differential equation is discretized on a rectangular grid with $(N_l - 1) \times (N_l - 1)$ unknowns then each processor is assigned to a subset of the unknowns (data partitioning). In a one-dimensional arrangement of n processors called a ring configuration of length n, processor p, $p \in \{0, \ldots, n-1\}$, is assigned to the grid points $\{(i,j) | max(1, pN_l/n) \le i < (p+1)N_l/n, 1 \le j < N_l\}$. In a $n \times m$ array configuration each processor is assigned to $N_l/n \times N_l/m$ unknowns. However if the sidelength of the grid is not divisible by the number of processors in the appropriate direction then some processors will be assigned more unknowns than others, which results in an unequal load balance. This is one source for losses of efficiency.

All components of the multi-grid method require only local operations, i.e. computations at grid point (i,j) need only values at the grid points $(i \pm 1, j \pm 1)$. In addition most operations (defect calculation, restriction, ...) can be carried out in parallel at each point. The number of points that can be computed simultaneously varies however between different smoothing procedures: This number is $(N_l - 1)^2$ for Jacobi smoothing, $(N_l - 1)^2/2$ for red-black smoothing and at most $N_l - 1$ for ILU and lexicographic Gauß-Seidel smoothers.

The speedup $S(n)$ of a parallel algorithm is defined as

$$S(n) = \frac{T_{Mono}}{T_{Multi}(n)} \tag{9}$$

with T_{Mono} the time needed for the execution of the program on a single computer and $T_{Multi}(n)$ the time taken on n processors. This is the factor by which computation time is reduced through the use of a parallel processor. The efficiency $E(n)$ of a parallel implementation is then defined as

$$E(n) = \frac{S(n)}{n} \tag{10}$$

The speedup for all operations carried out on a fixed level l within the multi-grid method depends heavily on the number of unknowns per processor, because a larger proportion of computing time is spent on communication and the effects of unequal load distribution are more pronounced if the number of grid points per processor is small. This means that a high speedup can be achieved on the fine grids (assuming a large number of grid points per processor on the fine grids), whereas the speedup deteriorates on the coarser grids. Nevertheless the overall speedup can be expected to be very high because most of the work is spent on the finest levels.

2.1. The DIRMU System. DIRMU is a multiprocessor consisting of 26 stand-alone micro-computers (8086/87) developed at the University of Erlangen-Nürnberg [6]. The processors communicate via distributed shared memories whereby each processor can be connected to the memories of up to seven other processors. In addition to this communication memory, called multiport memory, each processor has a private memory where the program and local data are stored. All variables shared with neighbouring processors are stored in the multiport memories in order to avoid explicit transportation of data. The distributed operating system DIRMOS supplies information about various possible connection structures like ring, array, tree or hypercube to a parallel program. In this way it is possible to use a connection structure that best fits the problem. DIRMOS also allows multiple users to run parallel programs on different parts of the system simultaneously and independently of each other.

2.2. Transputers. In order to compare the speedups on different computer architectures the ILU method was also implemented on an INMOS transputer system consisting of 7 T800 transputers connected in a ring. In contrast to DIRMU, the transputer system is a message passing architecture, where the time for sending a message consists of a fixed message setup time and a portion depending on the length of the message.

3. The Incomplete LU Factorization. In general we allow a 5-point finite difference star with variable coefficients denoted by

$$A_l(i,j) = \begin{bmatrix} & A^N_{i,j} & \\ A^W_{i,j} & A^C_{i,j} & A^E_{i,j} \\ & A^S_{i,j} & \end{bmatrix} \quad . \tag{11}$$

The discretization on a rectangular equidistant grid then yields a matrix with a sparsity pattern as for matrix A_l in Eqn. (4). As long as we regard ILU as a single grid method we will omit the index l. ILU is an iterative method which uses an approximate factorization of A into a product of lower and upper triangular matrices L and U with prescribed sparsity pattern. In the 5-point version the entries of L and U are allowed to be non-zero on the five non-zero diagonals of A. L and U are then defined in star notation as follows:

$$L(i,j) = \begin{bmatrix} & 0 & \\ L^W_{i,j} & L^C_{i,j} & 0 \\ & L^S_{i,j} & \end{bmatrix}, \; U(i,j) = \begin{bmatrix} & U^N_{i,j} & \\ 0 & U^C_{i,j} & U^E_{i,j} \\ & 0 & \end{bmatrix} \quad . \tag{12}$$

The matrix A is replaced by $LU - C$, where C is the error in the approximation. The iterative algorithm can be formulated without explicit calculation of C using the defect $d^n = f - Au^n$:

$$u^{n+1} = u^n + U^{-1}L^{-1}d^n = u^n + v^n \quad , \tag{13}$$

where v^n is obtained from solving $LUv^n = d^n$ in two steps:

$$L\delta = d^n \tag{14}$$

$$Uv^n = \delta \quad . \tag{15}$$

This approach is preferable to the use of C for three reasons; storage space for the diagonals of C is saved, the defect is available, since it must be calculated at every iteration for convergence testing, and the iteration is more numerically stable. The matrices L and U are triangular and therefore easily invertible once computed. If A is a tridiagonal matrix the incomplete LU factorization is equal to the exact LU factorization and this explains why the method can be applied with great success to discretizations of the singularly perturbed problems (1) and (3), since the discretization matrices of these equations tend towards tridiagonal matrices for the limit values of the parameters in the singularly perturbed sense. Existence and stability aspects of the ILU decomposition process are covered in [9].

The entries of L and U are computed recursively from the entries of A in the following manner:

$$L^S_{i,j} = A^S_{i,j}, \quad L^W_{i,j} = A^W_{i,j} \tag{16}$$

$$L^C_{i,j} = A^C_{i,j} - \frac{A^W_{i,j}A^E_{i-1,j}}{L^C_{i-1,j}} - \frac{A^S_{i,j}A^N_{i,j-1}}{L^C_{i,j-1}} \tag{17}$$

$$U^C_{i,j} = 1, \quad U^E_{i,j} = \frac{A^E_{i,j}}{L^C_{i,j}}, \quad U^N_{i,j} = \frac{A^N_{i,j}}{L^C_{i,j}} \quad . \tag{18}$$

The computation of the vectors δ and v^n is then given by forward and back substitution:

$$\delta_{i,j} = \frac{d_{i,j} - L^W_{i,j}\delta_{i-1,j} - L^S_{i,j}\delta_{i,j-1}}{L^C_{i,j}} \quad i = 1(1)N-1;\ j = 1(1)N-1 \tag{19}$$

$$v^n_{i,j} = \delta_{i,j} - U^E_{i,j}v_{i+1,j} - U^N_{i,j}v_{i,j+1} \quad i = N-1(-1)1;\ j = N-1(-1)1 \quad . \tag{20}$$

3.1. Parallelization. As described the method consists of two parts: decomposition of A into L and U and the inversion of L and U by forward and backward substitution. First we will consider the decomposition process. The key to the parallelization lies in the observation that when Dirichlet or von Neumann boundary conditions are given, the 'west' coefficients of the leftmost variables $A^W_{1,j}$ and the 'east' coefficients of the rightmost variables $A^E_{N-1,j}$ are equal to 0. This means that the term $A^W_{i,j}A^E_{i-1,j}(L^C_{i-1,j})^{-1}$ drops out of Eqn. (17) and that therefore $L^C_{1,j}$ depends only on $L^C_{1,j-1}$. We therefore have the following data dependencies: $L^C_{i,j}$ can be computed as soon as $L^C_{i-1,j}$ and $L^C_{i,j-1}$ have been computed except on the west boundary, where $L^C_{1,j}$ depends only on $L^C_{1,j-1}$ and on the south boundary where $L^C_{i,1}$ depends only on $L^C_{i-1,1}$. Regarding the grid it follows that L^C at all locations (i,j) with

$i + j = k$, k fixed, can be computed independently of each other from L^C at the locations $i + j = k - 1$.

The two-dimensional grid is mapped onto a ring of processors as described in section two. The parallel algorithm then proceeds as follows: Processor 0 starts with its segment of the first row of grid points. After completion, a signal is sent to processor 1 which then starts to process its section of the first row. Meanwhile processor 0 has started with the computation of its part of the second row. All processors are running when processor $n-1$ begins with the computation of its part of the first row. Algorithm 3.1, which is executed simultaneously by each processor, shows the parallel computation of the coefficients $L^C_{i,j}$, where the variables $xmin(p)$ and $xmax(p)$ contain the leftmost and rightmost grid point column stored by processor p. On DIRMU, the execution of routine signal_row(j) by processor p indicates the completion of its segment of row j by incrementing a counter in the multiport memory of its right neighbour. The procedure wait_row(j) waits for this signal in order to be able to proceed with the computation of row j. Note that in the DIRMU implementation grids which are to be accessed by neighbouring processors are located in the multiport memory, the others in private memory.

ALGORITHM 3.1. *Parallel computation of the L^C entries.*

```
PROCEDURE computeLC (A, L);
BEGIN
    FOR j := 1 TO N − 1 DO
        wait_row(j);
        FOR i := xmin(p) TO xmax(p) DO
            L^C_{i,j} = A^C_{i,j} − A^W_{i,j}(L^C_{i−1,j})^{−1} A^E_{i−1,j} − A^S_{i,j}(L^C_{i,j−1})^{−1} A^N_{i,j−1};
        END;
        signal_row(j);
    END;
END computeLC ;
```

Zero entries in A^W and A^E are reproduced in L^W and U^E according to Eqns. (16) and (18). Therefore the forward substitution to compute δ leads to the same data dependencies as the computation of L^C and therefore can be parallelized using the same ideas. The data dependencies for the backward substitution are different. Here again the values of v^n at locations $i + j = k$ can be computed simultaneously but are dependent on values of v^n with $i + j = k + 1$, which means that the parallel algorithm starts in the upper right corner of the domain instead of the lower left.

The extension of the ILU method to 9-point difference stars is straightforward. For the computation of the matrices L, U and the vector δ we have the following data dependencies: computations at grid point (i, j) need values at grid points $(i-1, j)$, $(i-1, j-1)$, $(i, j-1)$ and $(i+1, j-1)$. We can use the same basic algorithm with the exception that at processor boundaries the values at grid point $(i+1, j-1)$, located in the right neighbour are needed. In the memory coupled system this poses no problem, but in the message passing system an additional data transfer is necessary. In the backward substitution process for computing v^n the computations at grid point (i, j) need values at locations $(i + 1, j)$, $(i + 1, j + 1)$, $(i, j + 1)$ and $(i - 1, j + 1)$.

3.2. Speedup Results. In order to better understand the behaviour of the parallel method we have developed a model for the speedup. Having an accurate model of the speedup has several advantages:

TABLE 1
Predicted (P) and Measured (M) ILU Efficiencies for the DIRMU Multiprocessor

h	Processors						
	2	4	6	8	12	16	20
1/32 P	0.92	0.84	0.73	0.71			
1/32 M	0.93	0.83	0.73	0.69			
1/64 P		0.93	0.89	0.86	0.77	0.75	0.63
1/64 M		0.95	0.91	0.86	0.77	0.73	0.62
1/128 P					0.90	0.87	0.81
1/128 M					0.93	0.88	0.82

TABLE 2
Predicted ILU efficiencies for large DIRMU configurations

h	Processors					
	4	8	16	32	48	64
1/256	0.98	0.97	0.94	0.88	0.79	0.78
1/512	0.99	0.98	0.97	0.94	0.91	0.89
1/1024	0.99	0.99	0.98	0.97	0.94	0.93

- The model can be used to predict speedups for large problems whose memory requirements exceed the capacity of the DIRMU system.
- Estimated speedups for larger processor configurations can be obtained
- The influence of the multiprocessor architecture can be examined by variation of the model parameters

Here a simplified version of the model is given:

$$S(n, N_l) = \frac{(N_l - 1)^2 * T_{OP}}{T_{startup} + T_{parallel}} \tag{21}$$

$$T_{startup} = (n-1) * T_{SY} + (N_l - 1 - \frac{N_l - 1}{n}) * T_{OP} \tag{22}$$

$$T_{parallel} = (N_l - 2) * T_{SY} + (N_l - 1) * \frac{N_l - 1}{n} * T_{OP} \tag{23}$$

$T_{startup}$ is the time the rightmost processor must wait until it can begin its computation. $T_{parallel}$ is then the time taken until the parallel computations are complete. T_{SY} (Synchronization time) is the time needed for a signal_row and wait_row pair, and T_{OP} (Operation time) is the time needed for the arithmetical computations at a single grid point. We can obtain the following estimate for the resulting efficiency:

$$E(n, N_l) \geq \frac{1}{\frac{T_{SY}}{T_{OP}}\phi(\phi - 1) + \phi} \tag{24}$$

where $\phi = \frac{n}{N_l - 1} + 1$.

TABLE 3
Multi-grid efficiency on the DIRMU. Multi-grid data : ILU smoother, $\nu_1 = 1,\ \nu_2 = 0,\ \gamma = 1,\ h = 1/64$

Processors	Efficiency
5	0.88
6	0.82
7	0.76
8	0.73
9	0.67
10	0.64
11	0.65
12	0.58
13	0.54
14	0.50
15	0.46
16	0.44

This estimate, which was pointed out to the authors by G. Wittum, shows how the ratio T_{SY}/T_{OP} will be an important factor in the resulting speedup. The smaller this ratio the higher will be the speedup for given values of n and N_l. Alternatively, it can be seen that a value of T_{SY}/T_{OP} which is not small will require a large number of grid points per processor to achieve a high efficiency. In a message passing system T_{SY} will include the message setup time and the time required to transport the value of a boundary point $L^C_{xmax(p),j}$ from one processor to the next, while in a memory coupled system T_{SY} is the time that passes from incrementing a variable in shared memory until the neighbouring processor recognizes that the variable has changed its value.

Tables 1 through 3 show the results for the DIRMU implementation. Table 1 shows a comparison of the measured efficiency E of the ILU method and the predictions for various configuration lengths and grid sizes obtained by an enhanced speedup model which includes specific hardware parameters of the DIRMU multiprocessor. Table 2 shows the predictions obtained for larger problems and configurations. Finally, Table 3 shows the efficiency of the multigrid method with ILU smoothing and $h = 1/64$ on the finest grid.

The model is found to predict the measured efficiency very well for the grid and configuration sizes available on DIRMU. It is seen that $h < 1/64$ gives 'reasonable' grid sizes for DIRMU, with efficiencies over 75% for up to 14 processors for $h = 1/64$, and up to 23 processors for $h = 1/128$. The results are typical for a parallel algorithm with data-distribution, where the efficiency decreases when the number of processors is increased, owing to the growing communication and synchronization requirements relative to the number of parallelizable operations. For the same reason large grids can be more efficiently processed than smaller ones.

The model was considered to be accurate enough to enable predictions for larger problems. Here efficiencies better than 75% can be expected with up to 64 processors for a grid of side length 256, and the efficiency is better than 90% for grids with side length 512 or larger.

Table 4 shows the efficiencies measured for the transputer implementation. Note that T_{OP} on the transputer system is about sixty times shorter than that of the

TABLE 4
Efficiency for the ILU method on the transputer system.

h	Processors					
	2	3	4	5	6	7
1/32	0.89	0.79	0.76	0.67	0.62	0.58
1/64	0.94	0.88	0.87	0.83	0.79	0.73
1/128	0.97	0.94	0.93	0.90	0.87	0.85

DIRMU system, and that T_{SY} is about three times as small. The difference in the ratio of T_{SY} and T_{OP} means that the transputer system needs grids with twice the side length in order to achieve efficiencies comparable to DIRMU.

Note that it is possible to reduce the number of messages by grouping the grid lines [7]. With this modification, each processor computes k line segments, for some small k, before passing data on to the next processor. In this way, the efficiency for message-passing parallel computers with a high message setup overhead can be greatly enhanced. This has been shown both by an appropriate model and by direct simulation [8].

4. The Frequency Decomposition Method. The reduction of high frequency errors on coarser grids is made possible by the introduction of new prolongation operators p_{00}, p_{10}, p_{01} and p_{11}, which are constructed such that an error function v on the coarse grid is mapped to the space V_ι by p_ι. p_{00} is identical to the usual bilinear interpolation operator. The restriction operators r_ι are constructed from p_ι by transposition: $r_\iota = \frac{1}{4} p_\iota^T$, which results in the usual full weighting operator for r_{00}. Now we can describe the two grid algorithm. Assuming that we enter the cycle with the n-th iterate u_l^n, first a smoothing algorithm is applied ν_1 times on the fine grid. Then the defect $d_l = f - Au^n$ is calculated. Now the four restricted defects $d_{l-1}^\iota = r_\iota d_l$ can be computed and the four defect equations $A_{l-1}^\iota v_{l-1}^\iota = d_{l-1}^\iota$ have to be solved on the coarse grid. The coarse grid matrices are $A_{l-1}^\iota = r_\iota A_l p_\iota$ (Galerkin product). After the solution of the coarse grid equations the fine grid solution is corrected by $u_l^{n+1} = u_l^n + \sum_\iota p_\iota v_{l-1}^\iota$. Note that the four coarse grid corrections (restriction of the defect, solution of the coarse grid equations, prolongation of the corrections) can be computed independently of each other. As in the standard multi-grid method the multi-grid cycle is obtained by an approximate solution of the coarse grid equations with a recursive call (γ times) of the method.

As described above, the algorithm produces four matrices A_{l-1}^ι on level $l-1$. These produce 16 coarse grid matrices $A_{l-1}^{\iota\kappa}$ on level $l-2$, and in general 4^k coarse grid equations result at level $l-k$. Hackbusch showed that not all coarse grid equations are necessary and gave a criterion that results in the necessary coarse grid tree of Figure 1. This criterion states that a coarse grid matrix $A_{l-k}^{\iota_1\iota_2\cdots\iota_k}$ can be omitted if

$$\exists \iota, \kappa \in \{\iota_1, \iota_2, \ldots, \iota_k\} : \ \iota \neq \kappa \wedge \iota, \kappa \neq 00 \tag{25}$$

Note that most of the nodes have two descendants (type II nodes in the following) and only some nodes have four descendants (type I nodes). The number of coarse grid equations on level $l-k$ is thus reduced to $3 \cdot 2^k - 2$. For further details of the method we refer to [5]. This gives us the following algorithm. The parameter κ is used to omit the unnecessary coarse grids.

ALGORITHM 4.1. *The frequency decomposition multi-grid algorithm using the necessary coarse grids only.*

```
PROCEDURE fdm (l, A_l, u_l, f_l, κ);
BEGIN
    IF l = 0 THEN u_l := A_l^{-1} f_l
    ELSE
        u_l := S_l^{ν1}(u_l, f_l);                          (* pre-smoothing *)
        d_l := f_l − A_l u_l; v_l := 0;                     (* coarse grid correction *)
        IF κ = 00 THEN
            I := {00, 10, 01, 11}
        ELSE
            I := {00, κ}
        END;
        FOR ι ∈ I DO
            d^ι_{l−1} := r_ι d_l;
            v^ι_{l−1} := 0;
            IF κ = 00 THEN λ := ι ELSE λ := κ END;
            FOR k := 1 TO γ DO fdm(l − 1, r_ι A_l p_ι, v^ι_{l−1}, d^ι_{l−1}, λ) END;
            v_l := v_l + p_ι v^ι_{l−1};
        END;
        u_l := u_l + v_l;
        u_l := S_l^{ν2}(u_l, f_l);                          (* post-smoothing *)
    END;
END fdm ;
```

It can be shown that the complexity of one application of algorithm 2 is $O(n_l)$ if $\gamma = 1$ and $O(n_l \log n_l)$ if $\gamma = 2$ (n_l denotes the number of unknowns on the finest grid). For further details we refer to [5].

4.1. Parallelization. The frequency decomposition multi-grid algorithm allows parallelism to be exploited in two ways [2]. Firstly each processor can be assigned to a subset of the unknowns as described in section 2. A high speedup can be expected because simple Jacobi smoothing is sufficient. Secondly task partitioning can be used on the coarser grids, where the speedup normally deteriorates (one task is one coarse grid correction).

Therefore we use a two-level strategy. The fine grids, where the number of grid points per processor is assumed to be large, are distributed amongst all processors of an array configuration as described in section 2. On the coarser grids, the configuration is split into halves and each subconfiguration is assigned to a different coarse grid correction, thus avoiding the problem of the decreasing number of unknowns per processor usually associated with parallel multigrid methods. The level where the subdivision process starts can be prescribed by the parameter l_{subdiv}.

The subdivision strategy is as follows: If a $2n \times m$ array configuration is used on level l and a type II node is encountered, then the configuration is subdivided into two equal halves of size $n \times m$. Note that the whole defect grid function is needed in each of the two halves, which seems to require an explicit data transport, and that the situation is reversed when the two corrections have been computed. If a type I node is encountered, the configuration is also split into two equal halves, where first the 00 and 10 corrections are computed in parallel followed by the parallel computation of the 01 and 11 corrections. Now the strategy is applied recursively, whereby subconfigurations can be further divided.

FIG. 1. *Necessary coarse grid tree. The dashed and dotted subtrees are computed in parallel.*

The parallel computation of the 00 and 10 corrections, each on one half of the configuration, results in an unequal load balance, as can be seen from Figure 1, where the dashed subtree (00 correction) has more nodes than the dotted subtree (10 correction). This can be partially balanced by using a different parameter γ in the multi-grid cycle for type I and type II nodes. If $\gamma = 1$ is used when calling a type I node, and $\gamma = 2$ is used when calling a type II node, the work spent in the two subtrees is almost equal.

In order to make the exact solver on level 0 as efficient as possible the level 0 grids should be located in a single processor. To achieve this, a configuration of the form $2^{n'} \times 2^{m'}$ has to be used on the finest level. Now, after $n' + m'$ subdividing steps, $2^{n'+m'}$ configurations of size 1×1 are reached. This restriction results in a lower bound for the parameter l_{subdiv}: $l_{subdiv} \geq n' + m'$.

Using a hypercube configuration it turns out that the explicit transfer of data mentioned above can be omitted in the DIRMU implementation. An array configuration, where the number of processors in both directions is a power of 2 can be mapped onto a hypercube configuration using so-called gray codes. If we assume a $2^{n'+1} \times 2^{m'}$ configuration that is subdived into two halves of size $2^{n'} \times 2^{m'}$ then in each row of processors the same data transports would be necessary. Therefore it is sufficient to consider one row of processors (this is a ring of length $2^{n'}$). Instead of taking the processors $\{0, \ldots, 2^{n'} - 1\}$ and $\{2^{n'}, \ldots, 2^{n'+1} - 1\}$ as the two halves we take the processors

$$Left = \{p \mid 0 \leq p < 2^{n'+1},\ p \bmod 4 = 0 \ \vee \ p \bmod 4 = 3\}$$
$$Right = \{p \mid 0 \leq p < 2^{n'+1},\ p \bmod 4 = 1 \ \vee \ p \bmod 4 = 2\}$$

It can be shown by inspection of the gray codes, that

- the two halves form a ring configuration

TABLE 5
Speedups for different cycle forms

cycle	l	2 processors		4 processors		8 processors		16 processors	
		$S(2)$	$E(2)$	$S(4)$	$E(4)$	$S(8)$	$E(8)$	$S(16)$	$E(16)$
V	3	1.82	0.91	3.26	0.81	5.43	0.68		
	4	1.90	0.95	3.58	0.89	6.28	0.79	11.00	0.69
	5			3.59	0.90	6.80	0.85	12.48	0.78
	6							13.95	0.87
mixed	3	1.93	0.97	3.59	0.90	5.82	0.73		
	4	1.98	0.99	3.80	0.95	7.04	0.88	11.89	0.74
	5			3.92	0.98	7.63	0.95	14.39	0.90
	6							15.47	0.97
W	3	1.86	0.93	3.21	0.80	5.24	0.66		
	4	1.88	0.94	3.50	0.88	5.78	0.72	9.47	0.59
	5			3.56	0.89	6.49	0.81	11.04	0.69
	6							12.29	0.77

- all required data during the restriction and prolongation operations is located in neighbouring processors. This means that no data transports are necessary in the memory coupled DIRMU system
- this process can be applied recursively.

4.2. Speedup Results. The speedups and efficiencies have been determined for 2, 4, 8 and 16 processors (this is due to the hypercube configuration used) and various grid sizes. Table 5 shows the results for the V-cycle, the mixed cycle described above, and for the W-cycle. Index l indicates the grid fineness. The speedups obtained for the mixed cycle are very good even for coarse grids. However it should be noted that numerical tests have shown that the W-cycle is necessary to achieve a convergence rate independent of h_l.

5. Comparison. Speedup alone is not sufficient as an objective criterion for the comparison of iterative methods on a parallel computer. In addition we have to consider the convergence rate and the operation count, as these factors also play a key role in the overall efficiency of a solution procedure. We therefore define the normalized parallel computation time T_{np} to be

$$T_{np} = \frac{T_{cycle}}{-\ln \rho \cdot S(n)} \tag{26}$$

where T_{cycle} is the time needed for one iteration on a single computer, ρ is the average convergence rate and $S(n)$ is the speedup of the method on n processors. T_{np} is the time needed for n processors to reduce the error in the solution by a factor of $1/e$. Alternatively, the operation count for one Iteration can be used instead of T_{cycle}, in order to provide a measure that is independent of differences in implementation between the methods under comparison.

Table 6 shows the convergence rates and values of T_{np} obtained on DIRMU for 8 and 16 processors for the frequency decomposition method, and for standard multi-grid methods with ILU and Gauß-Seidel red-black smoothing applied to the

TABLE 6
Normalized Parallel Computation Times for the anisotropic equation, $h = 1/64$

α	β	GS − rb			FDM			ILU		
		ρ	$T_{np}(8)$	$T_{np}(16)$	ρ	$T_{np}(8)$	$T_{np}(16)$	ρ	$T_{np}(8)$	$T_{np}(16)$
1	1	0.108	0.9	0.5	0.086	5.9	3.4	0.121	0.86	0.73
$\frac{1}{2}$	2	0.393	2.23	1.1	0.196	8.8	5.2	0.150	0.96	0.81
$\frac{1}{10}$	10	0.938	32.5	16.2	0.295	11.8	6.9	0.135	0.91	0.77
10^{-2}	10^2	0.977	89.4	44.7	0.050	4.8	2.8	$8 \cdot 10^{-4}$	0.26	0.22
10^{-5}	10^5	0.977	89.4	44.7	0.051	4.8	2.8	$4 \cdot 10^{-15}$	0.05	0.05

TABLE 7
Normalized Parallel Computation Times for the convection-diffusion equation, $h = 1/64$

ϵ	GS − rb		ILU	
	$T_{np}(8)$	$T_{np}(16)$	$T_{np}(8)$	$T_{np}(16)$
1	3.71	1.85	4.23	3.57
0.1	5.02	2.51	5.35	4.51
0.01	7.97	3.99	3.97	3.35
0.001	14.1	7.05	2.27	1.92

anisotropic equation with $h = 1/64$ on the finest grid. Two pre-smoothing steps were used for the red-black and frequency decomposition methods, and one for the ILU method. No post-smoothing was done. In addition V-cycles were used for the red-black and ILU measurements, and W-cycles for the frequency decomposition tests. Full-Weighting was used as restriction operator in all cases.

Table 6 shows that as soon as the two factors α and β differ by a factor of ten or more, the two robust methods are superior to the standard parallel algorithm with red-black smoothing. In the case of strongly anisotropic coefficients, the ILU method is several hundred times as fast and the frequency decomposition method is about sixteen times as fast. Even for the Poisson equation ($\alpha = \beta = 1$) the ILU method is comparable to red-black smoothing.

The multi-grid method with ILU smoother has been found to be superior to the frequency decomposition method also for other problems we tested. Table 7 shows values of T_{np} for ILU and red-black smoothing applied to a discretization of Eqn. (3), where upwind differencing has been used for the convective terms and central differences have been used for the diffusive terms. The flow direction was 45 degrees ($c_1 = c_2 = 1$). Again the ILU method is superior to red-black smoothing. The frequency decomposition method in the present form could not be applied to the convection-diffusion equation with strong convection because the stable upwind differences on the finest grid are gradually changed to unstable central differences on the coarser grids by the Galerkin products.

6. Concluding Remarks. The results presented here show that the multi-grid method with ILU smoother can be effectively parallelized and is greatly superior to the multi-grid method with red-black smoother often used on parallel machines. The efficient solution properties of the method in the case of vanishing ellipticity

are provided by the ILU relaxation, which in this case would also be appropriate as a preconditioner for a Conjugate Gradient scheme which is known to be highly parallelizable, see, for example [1]. In the elliptic case however, multi-grid is known to be an optimal order method, having a rate of convergence independent of the grid size. It has to be emphasized that the ILU algorithm was not changed to achieve the parallelization. The method is especially suited for the anisotropic equation and the convection-diffusion equation with strong convection, which are important problems in practice.

The second method considered also showed superiority over the standard approach for the anisotropic equation, but is applicable only to a smaller range of problems than the ILU method.

The parallelization of the ILU method can be easily extended to the three-dimensional case but it should be noted that the method is not robust for the three-dimensional anisotropic equation. Whether the frequency decomposition method remains robust in the sense of Eqn. 7, which requires a bounded rate of convergence also for $h \to 0$, for three dimensional problems is not known yet. Further work will include the incorporation of the ILU method into a parallel solver for the incompressible Navier-Stokes equations.

REFERENCES

[1] AXELSSON, O., EIJKHOUT, V.: *Robust Vectorizable preconditioners for Three-dimensional Elliptic Difference Equations with anisotropy*, in Algorithms and Applications on Vector and Parallel computers, pp. 279-306, North-Holland, 1987.

[2] BASTIAN, P.: *Die Frequenzzerlegungsmethode als robustes Mehrgitterverfahren: Implementierung und Parallelisierung*, Diplomarbeit, IMMD III, Universität Erlangen- Nürnberg 1989

[3] BECKER, C., FERZIGER, J. H., PERIC, M., SCHEUERER, G.: *Finite Volume Multi-Grid Solution of the Two-dimensional Incompressible Navier-Stokes Equations*, in Robust Multi-Grid Methods, Proceedings of the Fourth GAMM Seminar, Notes on Numerical Fluid Mechanics, Volume 23, Vieweg Verlag, Braunschweig, 1988.

[4] HACKBUSCH, W.: *Multi-Grid Methods and Applications*, Springer, Berlin, Heidelberg 1985.

[5] HACKBUSCH, W.: *The Frequency Decomposition Multi-Grid Method*, in Robust Multi-Grid Methods, Proceedings of the Fourth GAMM Seminar, Notes on Numerical Fluid Mechanics, Volume 23, Vieweg Verlag, Braunschweig, 1988.

[6] HÄNDLER, W., MAEHLE, E., WIRL, K.: *The DIRMU Testbed for High-Performance Multiprocessor Configurations* Proceedings of the first International Conference on Supercomputing Systems, St. Petersburg, 1985.

[7] HORTON, G.: *Parallelisierung eines Mehrgitterverfahrens mit ILU Glättung*, Diplomarbeit, IMMD III, Universität Erlangen-Nürnberg 1989

[8] KNIRSCH, R.: *Parallele Lösung eines nichtlinearen Gleichungssytemes mit dem SUPRENUM Simulator*, Studienarbeit, IMMD III, Universität Erlangen-Nürnberg 1989

[9] MEIJERINK, J. A., VAN DER VORST H. A.: *An Iterative Solution Method for Linear Systems of Which the Coefficient Matrix is a Symmetric M-Matrix*, Mathematics of Computation, 31(1977), pp. 148-162.

[10] STONE, H. L.: *Iterative Solution of Implicit Approximations of Multidimensional Partial Differential Equations*, SIAM J. Numer. Anal., (5)1968, pp. 530-558.

[11] WESSELING, P.: *Theoretical and Practical Aspects of a Multigrid Method*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 387-407.

[12] WITTUM, G.: *On the Robustness of ILU Smoothing*, SIAM J. Sci. Stat. Comput. 10 pp. 699-717 (1989).

[13] WITTUM, G.: *Multi-Grid Methods for Stokes- and Navier-Stokes-Equations*, Numer. Math., 54(1989).