INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 22, 195–210 (1996)

FAST ITERATIVE SOLVERS FOR THE DISCRETIZED INCOMPRESSIBLE NAVIER–STOKES EQUATIONS

C. VUIK

Faculty of Technical Mathematics and Informatics, Delft University of Technology, PO Box 5031, NL-2600 GA Delft, The Netherlands

SUMMARY

In this paper some iterative solution methods of the GMRES type for the discretized Navier–Stokes equations are treated. The discretization combined with a pressure correction scheme leads to two different types of systems of linear equations: the momentum system and the pressure system. These systems may be coupled to one or more transport equations. For every system we specify a particular ILU-type preconditioner and show how to vectorize these preconditions. Finally, some numerical experiments to show the efficiency of the proposed methods are presented.

KEY WORDS: incompressible Navier–Stokes equations; non-symmetric linear system; preconditioning; vector computer; iterative solver; GMRESR

1. INTRODUCTION

In this paper we treat the solution of the discretized incompressible Navier–Stokes equations. The discretization of these equations in general curvilinear co-ordinates is described in References 1–5. As space discretization a finite volume technique on a boundary-fitted structured grid is used. In Reference 6 iterative methods of the Krylov type to solve the discretized equations have been presented. Reference 6 also contains a short survey of other iterative methods. In this paper we shall give improvements of the iterative methods described in Reference 6 and apply them to a wider range of problems. The improvements with respect to Reference 6 are that (i) new preconditioners are given with a better rate of convergence, (ii) different vectorization techniques for the preconditioners are given and compared and (iii) the GMRESR method with reuse of search directions is introduced and appears to be twice as fast as the original GMRESR method for the solution of the pressure system (which is in general the most time-consuming part). The methods given will be applied to problems with large grid size (up to 160×320 cells) and problems which include transport equations. For other problems where the given methods are used successfully we refer to References 2, 7 and 8.

The discretized equations given in Reference 2 have also been solved by multigrid methods. For stationary problems we refer to References 9 and 10 and for non-stationary problems to References 11 and 12. For a stationary problem it is not easy to compare the various methods, since the multigrid method given in Reference 9 solves the momentum equations simultaneously with the pressure equation, whereas in our software we use a time-stepping method combined with pressure correction. For non-stationary problems the Krylov subspace methods described in this paper are more efficient than the multigrid methods described in References 11 and 12. Recently we have combined the Krylov subspace method with multigrid as a preconditioner. This combination gives promising results.[13]

CCC 0271–2091/96/030195–16
© 1996 by John Wiley & Sons, Ltd.

Received 19 November 1993
Revised 3 May 1995

Since the discretized equations contain non-symmetric matrices,[6] we are not able to use the conjugate gradient or conjugate residual method. This motivates us to use GMRES-like methods, which are robust and have an optimal rate of convergence.[14–16] The incompressible Navier–Stokes equations in general co-ordinates are given by[2] the continuity equation

$$U^{\alpha}_{,\alpha} = 0 \tag{1}$$

and the momentum equations

$$\frac{\partial}{\partial t}(\rho U^{\alpha}) + (\rho U^{\alpha} U^{\beta})_{,\beta} + (g^{\alpha\beta} p)_{,\beta} - \tau^{\alpha\beta}_{,\beta} = \rho f^{\alpha}, \tag{2}$$

where $\tau^{\alpha\beta}$ represents the deviatoric stress tensor

$$\tau^{\alpha\beta} = \mu(g^{\alpha\gamma} U^{\beta}_{,\gamma} + g^{\gamma\beta} U^{\alpha}_{,\gamma}),$$

with $g^{\alpha\beta}$ the contravariant metric tensor, μ the viscosity, p the pressure, U^{α} the contravariant velocity component, ρ the density of the fluid and f^{α} the contravariant component of a body force. The transport equation for a scalar C is given by

$$k_1 \frac{\partial C}{\partial t} + (U^{\alpha} C)_{,\alpha} - (K^{\alpha\beta} C_{,\beta})_{,\alpha} + k_2 C = k_3, \tag{3}$$

where k_1, k_2, k_3 and $K^{\alpha\beta}$ are given functions.

Before discretization the physical domain is mapped onto a computational domain consisting of a number of rectangular blocks. In this paper we restrict ourselves to the one-block case. In order to avoid possible pressure oscillations, a staggered grid arrangement is used. The pressure is computed in the cell centres and the normal velocity components are calculated at the centres of the cell faces. In the remainder of this paper n_i is the number of finite volumes in the x_i-direction. For further details and the discretization of the boundary conditions we refer to Reference 2.

Finally, the spatial discretization is combined with finite differences for the time derivative. We use the Euler backward scheme together with pressure correction. The time step is denoted by Δt. For a given function v and $n \in \mathbb{N}$, v^n is an approximation of $v(n\Delta t)$. After Newton linearization we obtain two systems of equations,[2,6] namely the momentum equation

$$M^{n+1} u^{n+1} = f^{n+1}, \quad \text{where } u^{n+1} = \begin{pmatrix} U_1^{n+1} \\ U_2^{n+1} \end{pmatrix}, \tag{4}$$

and the pressure equation

$$P \Delta p^{n+1} = g^{n+1}, \quad \text{where } \Delta p^{n+1} = p^{n+1} - p^n. \tag{5}$$

A discretization of (3) will be called a transport equation and denoted by

$$C^{n+1} c^{n+1} = k_3^{n+1}. \tag{6}$$

The iterative methods are applied to two test problems: the flow through a curved channel and a Boussinesq problem. The curved channel problem makes it possible to compare the results of Reference 6 with the results in this paper. The Boussinesq problem is chosen in order to illustrate the performance of the solution methods for Navier–Stokes equations coupled with a transport equation. In this problem a stretched grid is used. We note that in many other problems the behaviour of the iterative method is comparable with that in the aforementioned test problems.

Curved channel

The curved channel is displayed in Figure 1. As initial condition we take the velocities equal to zero. The boundary conditions are a parabolic velocity profile at inflow (boundary 1), a no-slip condition at

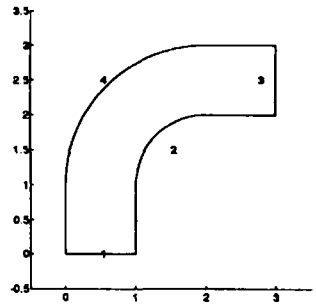

Figure 1. The physical domain of the curved channel problem

boundaries 2 and 4 and the normal stress and tangential velocity given at outflow (boundary 3). We take $\rho = 250$ and $\mu = 0{\cdot}5$.

Boussinesq problem

In the Boussinesq problem the Navier–Stokes equations are coupled with a temperature (transport) equation. We use a standard benchmark problem published in Reference 17. The physical domain and the 20×10 grid are displayed in Figure 2. Owing to buoyancy, we have a body force given by

$$f_1 = 0, \qquad f_2 = \tilde{g}\beta(T - T_0),$$

where $\tilde{g}$ is the acceleration due to gravity, β is a volume expansion coefficient and T_0 is a reference temperature. For the velocities we take no-slip boundary conditions. The temperature satisfies a transport equation. As temperature boundary conditions we take $T = 1$ at the left-hand wall and $T = 0$ at the right-hand wall. The lower and upper walls are isolated. We calculate the solution with $\rho = 1$, $\mu = 1$, $Pr = 0{\cdot}71$ and $Ra = 10^6$.

In Section 2 we discuss and compare different vectorization strategies for incomplete LU-type preconditioners. In Section 3 an RILU preconditioner is given for the pressure equation. We observe that GMRES-like methods combined with RILU have a better rate of convergence than with RILUD but require more memory than RILUD. For the pressure equation the memory is available (because for the momentum system much more storage is needed), so we always use the RILU preconditioner. In order to reduce storage, the momentum equation has been solved with a new variant of the RILUD preconditioner. The insights obtained from the solution of the pressure and momentum equations are used to solve the transport equations. A new variant of GMRESR is given in Section 4. Reuse of

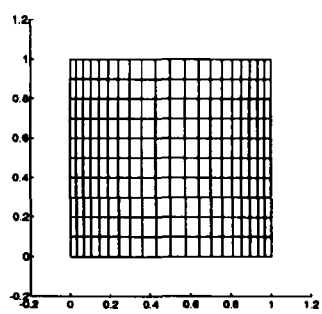

Figure 2. The 20×10 grid used in the Boussinesq problem

search directions leads to a faster rate of convergence for the pressure equation. Section 5 contains numerical experiments for two test problems on different grid sizes.

2. OPTIMIZATION AND VECTORIZATION OF THE PRECONDITIONERS

The discretization of the Navier–Stokes equations leads to a pressure equation with a matrix P with nine non-zero diagonals.[6] Here an incomplete $LD^{-1}U$ decomposition of P is used as preconditioner, so that the iterative method is applied to

$$U^{-1}DL^{-1}Px = U^{-1}DL^{-1}b \tag{7}$$

instead of to $Px = b$. In this paper the preconditioner given in Reference 6 is denoted by ILUD. The ILUD preconditioner is implicitly defined by the following rules:[18,19]

(a) $\mathrm{diag}(L) = \mathrm{diag}(U) = D$
(b) the off-diagonal parts of L and U are equal to the corresponding parts of P
(c) $\mathrm{diag}(LD^{-1}U) = \mathrm{diag}(P)$.

If the last rule is replaced by

$$\mathrm{rowsum}(LD^{-1}U) = \mathrm{rowsum}(P), \tag{8}$$

the MILUD preconditioner of Reference 20 is obtained. We also use an RILUD(α) preconditioner, which is an average of the ILUD and MILUD preconditioners.[21]

It is well known that using an ILUD-type preconditioner leads to the solution of systems of linear equations with an upper or lower triangular matrix. Owing to recurrences, a straightforward algorithm for this part runs at scalar speed on a vector machine. We first give an optimization of such a scalar code according to the lines set out in Reference 22. Then we specify a vectorized version of the preconditioner.

Row scaling

For the RILUD decomposition there exists a matrix R such that $P = LD^{-1}U - R$. Multiplication by D^{-1} leads to $\tilde{P} = D^{-1}P = D^{-1}LD^{-1}U - D^{-1}R = \tilde{L}\tilde{U} - \tilde{R}$. With $\tilde{b} = D^{-1}b$ we apply the iterative method to $\tilde{U}^{-1}\tilde{L}^{-1}\tilde{P}x = \tilde{U}^{-1}\tilde{L}^{-1}\tilde{b}$. Note that the multiplication by D in every iteration is no longer necessary. Furthermore, the solution of the triangular systems is cheaper, because the main diagonals of $\tilde{L}$ and $\tilde{U}$ are equal to the identity matrix. A nice property of this row scaling by D^{-1} is that if L, D and U satisfy the MILUD rule (8), then

$$\mathrm{rowsum}(D^{-1}P) = \mathrm{rowsum}(D^{-1}LD^{-1}U),$$

so that $\tilde{L}$ and $\tilde{U}$ also satisfy the MILUD rule. This is in contrast with a symmetric scaling (a row and column scaling,[22] where this property may be lost for the scaled system. In the remainder of this section the row-scaled quantities are denoted by P, L, U and b.

Eisenstat implementation

In every iteration step we have to compute $v_{j+1} = U^{-1}L^{-1}Pv_j$. Thus the amount of work per iteration is approximately twice as much as for the unpreconditioned system. In Reference 23 it is shown that much of the extra work can be avoided. To achieve this, it is necessary to apply the iterative method to

$$L^{-1}PU^{-1}y = L^{-1}b, \tag{9}$$

where the solution vector x is given by $x = U^{-1}y$. The rate of convergence of GMRES-like methods mainly depends on the eigenvalue distribution of the matrix.[14,24] Since the spectrum of $U^{-1}L^{-1}P$ is equal to the spectrum of $L^{-1}PU^{-1}$, we expect and observe the same convergence behaviour if we use (9) instead of (7). During the iterative solution of (9) we have to calculate $v_{j+1} = L^{-1}PU^{-1}v_j$. Using the equations

$$\begin{aligned} v_{j+1} &= L^{-1}PU^{-1}v_j = L^{-1}(L + P - L - U + U)U^{-1}v_j \\ &= U^{-1}v_j + L^{-1}\{v_j + [\mathrm{diag}(P) - I]U^{-1}v_j\}, \end{aligned}$$

the work to calculate v_{j+1} is reduced to two vector updates and the solution of an upper and lower triangular system. Thus one iteration of the preconditioned system costs approximately the same amount of flops as the unpreconditioned system. A disadvantage, however, is that the decrease in CPU time is small on vector computers, since a matrix–vector product is avoided, which is well vectorizable, whereas the hard-to-vectorize parts remain.

Vectorization

In this subsection we discuss some ways to vectorize the solution of triangular systems. The ideas for these vectorizations come from References 22 and 25. The vector of unknowns will be denoted by $x(i, j)$, where i refers to the index of the corresponding finite volume in the x_1-direction and j to that in the x_2-direction. Straightforward solution of $Lx = y$ leads to recurrences which prohibit vectorization.

In Figure 3 a diagonal ordering of the calculation is shown. In this figure the values of x at the points denoted by a + sign have already been calculated. The points denoted by a * sign display the stencil of L. Using this figure for the nine-point preconditioner, it is easily seen that all the points on the chain line diagonal ($i + 2j = c$) can be calculated independently. Thus this ordering leads to vectorizable code (compare with Reference 25). This implementation has the following drawbacks: the initial and final diagonals have a small vector length and indirect addressing is used. Indirect addressing costs extra CPU time and may lead to memory bank conflicts. Indirect addressing can be avoided by an explicit reordering of the unknowns. After this reordering, the unknowns are stored in memory in the same way as they are accessed in the diagonal-wise calculation of x from $Lx = y$. Especially for large values of n_1, explicit reordering gives a faster code on the Convex C3840 that we used in our experiments.

Another way to vectorize the code is to change the order such that all the points on the lines parallel to the x_1-axis are calculated together (line ordering). First suppose that the values $x(i, k)$, $i = 1, \ldots, n_1$, $k = 1, \ldots, j$ have already been determined. To calculate $x(i, j+1)$, $i = 1 \ldots, n_1$, the values $x(i-1, j)$, $x(i, j)$, $x(i+1, j)$ and $x(i-1, j+1)$ are used. Since the first three values are already known for all i, this part of the calculation (75% of the work) can be

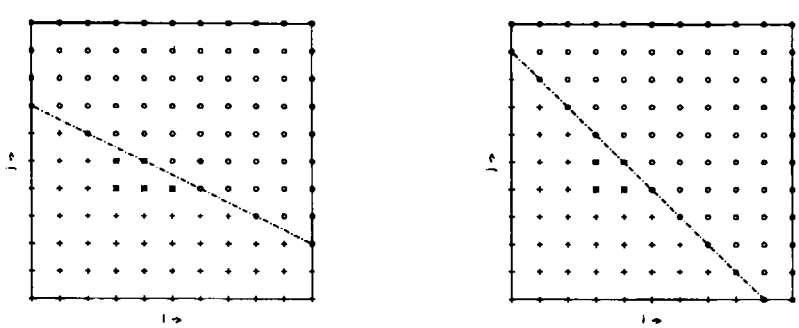

Figure 3. Ordering used for the vectorization of the solution of the system $Lx = y$ for a nine-point preconditioner (left) and a seven-point preconditioner (right)

done at vector speed. One recurrence remains due to the use of $x(i-1, j+1)$, so 25% of the work is done at scalar speed. Advantages of this vectorization technique are that all vector lengths are equal to n_1, easy implementation and no indirect addressing. On the Convex C3840 the diagonal ordering leads to somewhat smaller computing times than the line ordering.

In Section 3 we also define a seven-point preconditioner for the pressure equation. It follows from Figure 3 that this preconditioner can be vectorized along the diagonals $i+j=c$.[22,25] Since the vector length of the loops is twice the vector length of the diagonal ordering for the nine-point preconditioner, a seven-point preconditioner leads to better vectorized code.

To analyse the vectorization of the RILUD preconditioner for the momentum matrix M, we use the block structure

$$Mu = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \qquad L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}, \qquad U = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}. \tag{10}$$

In Reference 6 it has been shown that the non-zero structure of the diagonal blocks M_{11} and M_{22} is the same as for the matrix P. Let us now consider the computation of x from $Lx=y$. The vectorization techniques described for the nine-point preconditioner for the pressure matrix can be used to obtain x_1 from

$$L_{11}x_1 = y_1. \tag{11}$$

Thereafter x_2 is calculated from

$$L_{22}x_2 = y_2 - L_{21}x_1. \tag{12}$$

Since x_1 is already known, the right-hand side of (12) can be calculated at vector speed. Finally, the computation of x_2 from (12) can be done in the same way as the solution of (11).

3. PRECONDITIONERS FOR THE DIFFERENT LINEAR SYSTEMS

In this section, preconditioners are given for the pressure, momentum and transport equations. For the pressure equation a nine-point and a seven-point ILU preconditioner are described. Some remarks are given on preconditioning a singular pressure matrix. A new variant of the MILUD preconditioner is given for the momentum equation. In this preconditioner the difference between the velocities u_1 and u_2 is taken into account. Finally, the insights obtained from the solution of the pressure and momentum equations will be used to solve the transport equation in an efficient way.

The pressure equation

First we consider the classical incomplete LU decomposition of P (all fill-in is neglected). This preconditioner is denoted by ILU. For ILU the matrices L and U satisfy the following rules:

(a) $\mathrm{diag}(L)=I$
(b) the non-zero structure of the matrix $L+U$ is identical with the non-zero structure of P
(c) if $P_{ij} \neq 0$, then $(LU)_{ij}=P_{ij}$.

The last rule can for $i=j$ be replaced by

$$\mathrm{rowsum}(LU) = \mathrm{rowsum}(P), \tag{13}$$

which leads to the MILU preconditioner. Also, for this preconditioner we always use an averaged method, RILU(α).

It is known that for a five-point stencil the RILUD and the RILU preconditioner are the same. However, for a nine-point stencil it is easily seen that RILU leads to a preconditioner different from

RILUD. Note that for this preconditioner the matrices L, U and P should be kept in memory. Thus the amount of extra memory for this preconditioner is nine vectors (the same amount of memory as needed for P). The Eisenstat implementation cannot be used for this preconditioner, since the off-diagonal part of P is not identical with the off-diagonal part of $L+U$. This implies that an iteration using RILU is more expensive than an iteration using RILUD. Since the non-zero structures of L and U are the same for RILU and RILUD, the RILU preconditioner can be vectorized in the same way as the RILUD preconditioner.

The optimal choice of α is an open question. Results in Reference 21 indicate that for symmetric matrices α close to unity is a good choice. Furthermore, for increasing grid size the optimal value of α approaches unity. These insights are confirmed by our experiments (see Section 5).

For a problem where all boundary conditions for the velocities are of the Dirichlet type, the pressure matrix P is singular. The null space of P is given by

$$\mathrm{null}(P) = \mathrm{span}\left\{\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}\right\}. \tag{14}$$

For such a problem RILU($\alpha=1$) gives a breakdown of the iterative method (the same for RILUD($\alpha=1$)). This can be explained as follows. Equation (13) can be written as

$$LU\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix} = P\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}. \tag{15}$$

Equation (15) together with (14) implies that

$$LU\begin{pmatrix}1\\ \\ 1\end{pmatrix} = 0,$$

so the matrix LU is singular. Owing to the definition of L, it follows that U is singular, which leads to a breakdown of the preconditioned GMRES method. A closer look at U shows that the last main diagonal element is equal to zero. Changing this element to a small number causes the iterative method to converge, but in our experiments $\alpha < 1$ leads to a much better rate of convergence.

Since a seven-point preconditioner leads to better vectorized code (compare Section 2), we also use a seven-point incomplete decomposition of P, where the stencils of P, L and U are given in Figure 4. The matrices L and U are such that

$$\text{if } L_{ij} \neq 0 \quad \text{or} \quad U_{ij} \neq 0, \quad \text{then } (LU)_{ij} = (P)_{ij}.$$

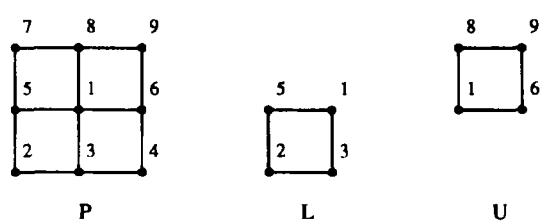

Figure 4. The stencils of P, L and U

The amount of extra memory required is equal to seven vectors. In our experiments the rate of convergence of this preconditioner is better than with RILUD but worse than with RILU. Although this preconditioner is better vectorizable, we conclude from our experiments that it is better to use the nine-point RILU preconditioner.

The momentum equation

The momentum equation is given by $M^{n+1}u^{n+1} = f^{n+1}$. The dimension of the matrix M^{n+1} is twice the dimension of the pressure matrix P. The matrix M^{n+1} has 13 non-zero elements per row. For the structure of M^{n+1} we refer to Reference 6. Note that the matrix P only depends on the geometry and boundary conditions, whereas M^{n+1} depends also on the time and the choice of the time step Δt, on ρ and on μ. In the following we delete the superscript $n+1$ for brevity. Owing to the extra memory needed for the RILU preconditioner (13 extra vectors of length $2n_1n_2$), we restrict ourselves to the RILUD preconditioner for the momentum equation.

We observe that the optimal choice of α used in RILUD(α) depends strongly on the test problem. Since our solver is mainly used as a black box solver, we prefer a preconditioner such that one choice of α is optimal for a wide range of problems. For that reason we consider the MILUD preconditioner more carefully. In the following we denote the MILUD preconditioner by MILUD_1. The matrices L, D and U of MILUD_1 satisfy the equation

$$\text{rowsum}(LD^{-1}U) = \text{rowsum}(M),$$

so using the block structure given in (10), the following equations are obtained:

$$\begin{aligned} \text{rowsum}(L_{11}D_{11}^{-1}U_{11} + L_{11}D_{11}^{-1}U_{12}) &= \text{rowsum}(M_{11} + M_{12}), \\ \text{rowsum}(L_{21}D_{11}^{-1}U_{11} + L_{21}D_{11}^{-1}U_{12} + L_{22}D_{22}^{-1}U_{22}) &= \text{rowsum}(M_{21} + M_{22}). \end{aligned} \tag{16}$$

It is well known that the MILUD_1 preconditioner is very effective if the solution is a slowly varying function. In the extreme case of no variation the multiplication by M and $LD^{-1}U$ leads to the same result and so the preconditioned GMRES method converges in one iteration. In the momentum equations (4) the vector u consists of two parts: u_1, the velocity component in the x_1-direction, and u_2, the velocity component in the x_2-direction. Since in our code we use contravariant fluxes, which implies that velocity components are scaled by the length of cell sides, it is possible that there is a large difference between u_1 and u_2. As a consequence, α close to unity can lead to a bad rate of convergence, because, although the components u_1 and u_2 may be slowly varying functions, the difference between them may be large. This insight motivates us to propose a slightly adapted preconditioner. MILUD_2, where the MILU approach is used in a decoupled way.

For MILUD_2 the matrices L, D and U satisfy the same rules as for MILUD_1, except rule (16) which is replaced by

$$\begin{aligned} \text{rowsum}(L_{11}D_{11}^{-1}U_{11}) &= \text{rowsum}(M_{11}), \\ \text{rowsum}(L_{21}D_{11}^{-1}U_{12} + L_{22}D_{22}^{-1}U_{22}) &= \text{rowsum}(M_{22}). \end{aligned} \tag{17}$$

We expect that this preconditioner will work well if u_1 and u_2 are slowly varying functions, whereas the difference between u_1 and u_2 may be large. In all our experiments this preconditioner has a nice convergence behaviour for α close to unity. Hence the MILUD_2 preconditioner is more robust than MILUD_1.

The transport equation

Transport equations of the type (3) can be used to describe the transport of temperature, certain quantities occurring in engineering models of turbulence,[8] the concentration of salt in an estuary, etc.

We distinguish between two classes of transport equations. The first class describes the transport of a passive scalar. In this case the Navier–Stokes equations can be solved independently of the transport equation. Thereafter the velocities u_1 and u_2 can be used in (3) to obtain a solution of the transport equation. The second class describes the transport of an active scalar. This class consists of applications where the Navier–Stokes equations are coupled with the transport equation (5), e.g. a Boussinesq problem or turbulence modelling. Since the transport equation has the same properties for both classes, the choice of iterative solution method is independent of the type of scalar.

We note that equation (3) resembles the equations given in (2). This explains why the convergence behaviour of an iterative method applied to a transport equation is comparable with its behaviour when it is applied to the momentum equation. The matrix C^{n+1} depends on the geometry, the boundary conditions, the velocities, the time step and the choice of the functions K_1, $K^{\alpha\beta}$ and K_2. An important difference is that the momentum equations describe a vector quantity whereas a transport equation describes a scalar quantity. As a consequence, the dimensions and structure of a transport matrix are the same as those of the pressure matrix. This motivates us to solve a transport equation with a GMRES-like method combined with an RILU preconditioner.

4. REUSE OF SEARCH DIRECTIONS FOR THE GMRESR METHOD

In this section we describe a new technique to save iterations and CPU time using the GMRESR method.[15] The key idea is the following: if a system of linear equations is solved with different right-hand sides, then the information obtained from the solution process for the first right-hand side vector is used for the following right-hand sides.

We describe the adapted GMRESR algorithm for the pressure equation

$$P\Delta p^{n+1} = g^{n+1}. \tag{18}$$

In this equation the matrix P is constant whereas the right-hand sides are different in every time step. The GMRESR algorithm is given by ($b = g^{n+1}$ and x_k approximates Δp^{n+1})[6,15,16]

$r_0 = b - P_{x_0}, k = -1;$
while $\|r_{k+1}\|_2 >$ tol do
  $k := k + 1$, compute $u_0^{(0)}$ and $c_k^{(0)} = Pu_k^{(0)}$;
  for $i = 0, 1, \ldots, k-1$ do
    $\alpha_i = c_i^{\mathrm{T}} c_k^{(i)}, c_k^{(i+1)} = c_k^{(i)} - \alpha_i c_i, u_k^{(i+1)} = u_k^{(i)} - \alpha_i u_i;$
  endfor
  $c_k = c_k^{(k)}/\|c_k^{(k)}\|_2, u_k = u_k^{(k)}/\|c_k^{(k)}\|_2;$
  $x_{k+1} = x_k + u_k c_k^{\mathrm{T}} r_k, r_{k+1} = r_k - c_k c_k^{\mathrm{T}} r_k;$
endwhile.

In this paper we use the original GMRESR algorithm as presented in Reference 15, where $u_k^{(0)}$ is computed by one iteration of GMRES(m) applied to $Py_k^{(0)} = r_k$. Other variants are proposed in References 13 and 26.

Note that the vectors u_k and c_k used in the GMRESR algorithm should be stored in memory. Owing to memory limitations, it is necessary to bound the number of search directions u_k (and c_k) to be kept in memory. Different techniques[15,16] can be used to select which search directions are stored. In this paper we use the minalfa truncation strategy,[16] which is defined in the following way. Suppose that the maximal number of search directions kept in memory is equal to n_t. As long as the number of outer iterations is less than n_t, all search directions are stored in memory. Thereafter the new search direction overwrites the old search direction with the smallest absolute value of α_i in the **for**-loop.

We now describe the GMRESR method with reuse of search directions for the pressure equation. In the first time step we solve $P\Delta p^1 = g^1$ with the GMRESR method. The number of outer iterations is equal to n_1, while GMRESR is truncated after n_t outer iterations. In the first time step the search directions u_k, $k = 0, 1, \ldots, n_s$, are used, where $n_s = \min(n_1, n_t)$. These vectors and the vectors $c_k = Pu_k$ are stored in memory. For the solution of $P\Delta p^2 = g^2$ we use the following adapted version of GMRESR. Before we start the iteration process, the residual is made perpendicular to $\mathrm{span}\{c_0, \ldots, c_{n_s}\}$ as follows:

$$\begin{aligned} &\textbf{for } k = 0, 1, \ldots, n_s \quad \text{do} \\ &\quad x_0 = x_0 + u_k c_k^{\mathrm{T}} r_0, \quad r_0 = r_0 - c_k c_k^{\mathrm{T}} r_0, \\ &\textbf{endfor}. \end{aligned} \tag{19}$$

Thereafter we start the iteration, where the orthogonalization process in the GMRESR algorithm now runs from $i = 0$ to $\min(n_s + k - 1, n_t)$. The number of outer iterations in the second time step is equal to n_2. The vectors u_k and c_k, $k = 0, 1, \ldots, n_s = \min(n_s + n_2, n_t)$, are stored in memory. These directions are reused in the third time step, etc. Note that n_t is an upper bound of the number of direction vectors which are reused.

Different strategies are possible for the selection of search directions which are kept in memory. In the experiments reported here, we start by storing all search directions. If $n_s + k - 1$ becomes equal to n_t, the minalfa truncation strategy is used to discard an old search direction. This implies that the search directions stored in memory may be different in every time step. Another strategy could be to obtain the n_s search directions in the first time step and reuse these in every following time step. Thus the search directions remain the same for every time step $n \geqslant 2$.

To illustrate this adaptation of the GMRESR algorithm, we give results for the first test problem on a 64×256 grid with $\rho = 250$ and $\mu = 0{\cdot}5$, implying a Reynolds number of 500. We use GMRESR with GMRES(4) as inner loop and the RILU($\alpha = 0{\cdot}975$) preconditioner. In Table I the results are given for the pressure equation at the second time step. The CPU time is measured in seconds on one processor of a Convex C3840. Note that there is a considerable speed-up when the search directions are reused. The convergence behaviour is shown in Figure 5.

For the original GMRESR algorithm the superlinear convergence sets in after seven outer iterations. The GMRESR algorithm with reuse of search directions leads to fast convergence from the beginning. Thus the gain in iterations and CPU time is not a consequence of the decrease in the norm of the initial residual due to (19), but a consequence of the fact that the components in slowly converging eigenvectors are absent owing to the expanded orthogonalization. Compare the description of the superlinear convergence behaviour of GMRES as given in Reference 24. The results in Table I show that the number of iterations decreases when the value of n_t increases. This agrees with our explanation that if more search directions are reused (n_t larger), then more components in slowly converging eigenvectors are absent, so a faster rate of convergence results. A drawback is that increasing n_t leads to larger memory requirements.

Table I. Number of iterations and CPU time for different GMRESR variants combined with RILU ($\alpha = 0{\cdot}975$)

	Original GMRESR(4)		GMRESR(4) with reuse	
n_t	Outer iterations	CPU	Outer iterations	CPU
20	14	2·56	7	1·5
15	14	2·56	8	1·7
10	14	2·56	10	2·0

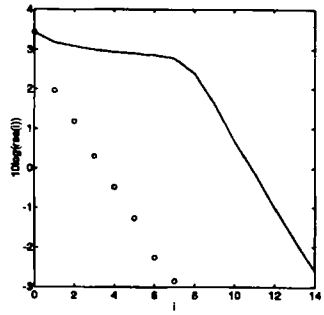

Figure 5. Convergence behaviour of GMRESR (———) and GMRESR with reuse of search directions (○) combined with an RILU preconditioner (grid 64 × 256)

We conclude that the reuse of search directions is a good idea if the original GMRESR algorithm applied to the linear system of equations has a superlinear convergence behaviour. If, furthermore, the required accuracy is low, the CPU time decreases considerably when we reuse the search directions (low accuracy is in general sufficient for non-linear or time-dependent problems). Note that this gain in CPU time is important, because the solution of the pressure equation is in general the most time-consuming part.

Reuse of search directions can also be used for the momentum equations $M^{n+1}u^{n+1} = f^{n+1}$. Although $M^{n+1} \neq M^n$, we expect that after some time steps the search directions for M^{n+1} and M^n are related. For the reused vectors u_k and c_k the relation $M^{n+1}u_k = c_k$ no longer holds, because $M^{n+1} \neq M^n$. Thus only the vectors u_k are stored in memory. The adapted GMRESR algorithm is now started with the loop

for $k = 0, \ldots, n_s$
  $u_k^{(0)} = u_k,\ c_k^{(0)} = M^{n+1}u_k^{(0)};$
  for $i = 0, \ldots, k-1$
    $\alpha_i = c_i^{\mathrm{T}} c_k^{(i)},\ c_k^{(i+1)} = c_k^{(i)} - \alpha_i c_i,\ u_k^{(i+1)} = u_k^{(i)} - \alpha_i u_i;$
  endfor
  $c_k = c_k^{(k)}/\|c_k^{(k)}\|_2,\ u_k = u_k^{(k)}/\|c_k^{(k)}\|_2;$
endfor.

Thereafter the GMRESR method continues with (19) and the expanded orthogonalization as for the pressure equation. For the momentum equation we see only a small gain in iterations and in general no gain in CPU time. There are two reasons for this: firstly the search directions are different for M^{n+1} and M^n and secondly the original GMRESR method converges linearly. The second reason implies that it is improbable to obtain a faster convergence by reusing search directions. This is illustrated by the first test problem with the 16 × 64 grid, $\rho = 250$, $\mu = 0{\cdot}5$ and $\Delta t = 0{\cdot}15$. The convergence behaviour of GMRESR with GMRES(4) as inner loop is given in Figure 6 for the momentum equation in the third time step. From this figure it appears that the convergence behaviour of GMRESR is linear. Note that there is only a small gain in iterations whereas the CPU time is larger. In other experiments (larger grid sizes and/or using preconditioners) we obtain comparable results. Thus for the momentum equation the GMRESR algorithm with reuse of search directions does not lead to a faster solution method. For the transport equation in the second test problem we obtain the same results as for the momentum equation.

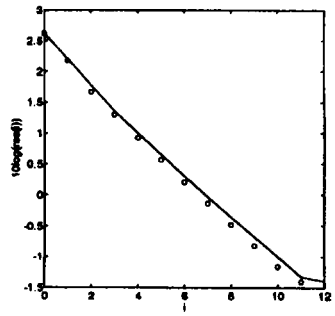

Figure 6. Convergence behaviour of GMRESR (———) and GMRESR with reuse of search directions (○)

5. NUMERICAL RESULTS

In this section we present the results of some numerical experiments. We start with the curved channel problem. The efficiencies of the solution methods for the momentum and pressure equations are given using vectorized ILU-type preconditioners. Thereafter we measure the CPU times required to solve the Navier–Stokes equations for various grid sizes. For the Boussinesq problem comparable experiments have been done. In all cases the CPU time has been measured in seconds on one processor of a Convex C3840.

Curved channel problem

Consider the curved channel problem described in Section 1. First we investigate the vectorization of the preconditioner. On the Convex, the megaflop rate for a vector update (which runs in vector speed) is 35 Mflop/s. In Table II the megaflop rate is given for the diagonal-wise ordering presented in Section 2 for the pressure equation. Without vectorization the multiplication by L^{-1} or U^{-1} has a megaflop rate equal to 9. From Table II it appears that the megaflop rate for the vectorized version becomes higher for increasing grid size. For large grid sizes it is equal to the megaflop rate of a vector update.

In Section 3 we have given some guidelines for the choice of α for the RILU preconditioner used in the solution of the pressure equation. We have performed experiments for various values of α. In general we prefer postconditioning instead of preconditioning. The reason for this is that using postconditioning, which means the solution of $PU^{-1}L^{-1}y = b$ and $x = U^{-1}L^{-1}y$, the termination criterion is based on $\|r_k\|_2$, whereas with preconditioning it is based on $\|U^{-1}L^{-1}r_k\|_2$. Table III gives the number of iterations for GMRES (without restarting) and various choices of α. The iteration process is stopped if $\|r_k\|_2/\|r_0\|_2 \leqslant 10^{-6}$. Note that for this problem $\alpha = 1$ leads to the minimal number of iterations. Furthermore, for small grid sizes $\alpha \in [0{\cdot}975, 1]$ leads to the same number of iterations, whereas for large grid sizes the optimal values of α are close to unity and the sensitivity of the number of iterations required to α increases. The results in Table III are obtained by using full GMRES. In order to reduce memory requirements and CPU time, we always use the GMRESR method in practical computations. When GMRESR is used, it appears that RILU(0·99) is the best choice.

Table II. Megaflop rate of the vectorized RILU(D) preconditioner with diagonal ordering for the pressure equation

Grid size	16 × 64	32 × 128	64 × 256	128 × 512
Mflop/s	15	22	32	35

Table III. Number of iterations of GMRES using the RILU(α) postconditioner for the pressure equation

Grid size	$\alpha = 0{\cdot}975$	$\alpha = 0{\cdot}99$	$\alpha = 1$
16 × 64	23	24	22
32 × 128	34	34	32
64 × 256	57	49	46
128 × 512	104	84	64

In Figure 7 the numbers of iterations for full GMRES combined with the MILUD or MILU postconditioner are given for the pressure equation. One iteration costs approximately the same amount of CPU time for both postconditioners. Thus this figure gives a good idea of the performance of the postconditioners. Note that especially for large grid sizes the MILU postconditioner becomes much better than MILUD. The results presented in Figure 7 motivate us to use an RILU preconditioner instead of an RILUD preconditioner.

Table IV gives CPU time for the solution of the pressure equation using the RILU(0·99) postconditioner combined with truncated GMRESR(m) and reuse of the search directions. The results are measured in the second time step. The best results given in Reference 6 for the pressure equation are obtained with the GMRES method and the RILUD(0·95) preconditioner. For the 16 × 64 grid this costs 31 iterations, 0·6 s of CPU time and 31 memory vectors. Comparing the results given in Table IV with the results given in Reference 6, we see a large gain in CPU time. Part of this gain comes from the fact that the Convex C3840 is 2·5 times faster than the Convex C240 used in Reference 6, but in addition the new method is approximately three times faster.

In Reference 6 the momentum equation has been solved with GMRES(5) combined with a diagonal preconditioner. For the 16 × 64 grid this costs 57 iterations and 0·6 s of CPU time. In this paragraph the results are produced by GMRES(20) combined with a diagonal or ILUD preconditioner. For the momentum equation the preconditioned system $L^{-1}M^{n+1}U^{-1}y = L^{-1}b$, $x = U^{-1}y$ has been solved. We observe that termination criteria based on $\|r_k\|_2$ and $\|L^{-1}r_k\|_2$ lead to the same results. The iteration process is stopped if $\|L^{-1}r_k\|_2/\|L^{-1}r_0\|_2 \leqslant 10^{-4}$. The experiments are done in the second time step. Table V demonstrates that ILUD saves many iterations and much CPU time. For this problem MILUD_1 leads to worse results, whereas the number of iterations and CPU time for MILUD_2 are comparable with ILUD. Comparing the results for the 16 × 64 grid with Reference 6, we see again a large gain in CPU time. Since the termination criterion in Reference 6 is slightly stronger than that used in this paper, there is a small difference in the number of iterations. The last column in Table V contains the CPU time to build the momentum and pressure equations. Note that

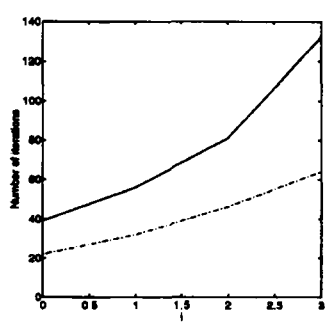

Figure 7. Number of iterations of full GMRES combined with MILUD (———) and MILU (–·–·). The grid size is equal to $(16 \times 2^i) \times (64 \times 2^i)$

Table IV. Amount of memory, CPU time and number of iterations for the pressure equation

Grid size	n_t	m	Iterations	CPU	Memory vectors
16 × 64	10	3	7	0·09	32
32 × 128	15	3	7	0·32	46
64 × 256	15	4	7	1·21	47
128 × 512	20	6	7	7·38	55

Table V. Number of iterations and CPU time using different preconditioners for the momentum equations

		Diagonal		ILUD		
Grid size	Time step	Iterations	CPU	Iterations	CPU	Building of systems
16 × 64	0·15000	41	0·24	7	0·075	0·07
32 × 128	0·07500	38	0·75	6	0·20	0·18
64 × 256	0·03750	36	2·74	6	0·73	0·60
128 × 512	0·01875	39	13·05	7	3·21	2·16

comparison of Tables IV and V shows that the solution of the pressure equation is the most time-consuming part, as has been the general experience on Cartesian grids in the past.

Boussinesq problem

For the Boussinesq problem we shall start with experiments for the pressure equation. We have used the RILU(0·975) postconditioner combined with GMRESR(m), where the search directions are reused. The results we present in Table VI are measured in the third time step, which was found to be typical. The termination criterion used is the same as for the curved channel problem.

Table VII gives results for the momentum equations. In these experiments GMRES(20) combined with ILUD, RILUD_1(0·95) and RILUD_2(0·95) as preconditioners has been used. In all cases a time step $dt = 4 \times 10^{-4}$ has been chosen independently of the grid size. We note that the convergence behaviour of RILUD_2 is much better than that of RILUD_1 and that for increasing grid size RILUD_2 becomes much better than ILUD and RILUD_1(0·95). Since RILUD_2 has at least the same performance as ILUD for other problems, e.g. the curved channel problem, it is recommended to use the RILUD_2(0·95) preconditioner in all cases.

Finally, Table VIII gives the results for the transport equation. In every time step first the momentum and pressure equations are solved and then the transport equation. The computed temperature is used on the right-hand side of the momentum equation in the next time step. The GMRES(20) method combined with the MILU postconditioner has been used. The iteration process is stopped if

Table VI. Amount of memory, CPU time and number of iterations for the pressure equation

Grid size	n_t	m	Iterations	CPU	Memory vectors
20 × 40	10	4	5	0·047	32
40 × 80	15	5	6	0·33	46
80 × 160	15	6	10	2·0	47
160 × 320	20	7	11	9·7	55

Table VII. Number of iterations and CPU time for the momentum equation

	ILUD		RILUD_1		RILUD_2	
Grid size	Iterations	CPU	Iterations	CPU	Iterations	CPU
20 × 40	5	0·031	5	0·033	5	0·029
40 × 80	8	0·15	10	0·17	8	0·15
80 × 160	13	0·86	27	2·1	11	0·75
160 × 320	23	6·8	68	2·3	13	3·2

Table VIII. Number of iterations and CPU time for the transport equation

Grid size	Iterations	CPU	Building of systems
20 × 40	6	0·016	0·05
40 × 80	10	0·11	0·15
80 × 160	14	0·48	0·49
160 × 320	16	2·3	1·8

$\|r_k\|_2/\|r_0\|_2 \leqslant 10^{-6}$. Note that comparison of Tables VI, VII and VIII again shows that the solution of the pressure erquation is the most time-consuming part.

6. CONCLUSIONS

In this paper we have described properties of GMRES-type iterative methods combined with ILU-type preconditioners to solve a discretization of the incompressible Navier–Stokes equations in general co-ordinates with the pressure correction method. Comparing the results of this paper with those of Reference 6, we note a considerable decrease in CPU time to solve the pressure and momentum equations owing to the novel idea of reuse of search directions for the pressure equation, improvements in pre- and postconditioning and vectorization.

The pressure equation has been solved with GMRESR combined with an RILU postconditioner. In the case of a non-singular pressure matrix $\alpha = 0{\cdot}99$ appears to be a good choice for the average parameter, whereas in the singular case $\alpha = 0{\cdot}975$ should be preferred. Finally, reuse of the GMRESR search directions leads to a large reduction of CPU time in the solution of the pressure equation.

The momentum equation has been solved with GMRES(20) combined with RILUD_2. A good choice for α is 0·95. The properties of the momentum equation depend not only on the geometry and boundary conditions but also on other parameters such as time, time step, ρ, μ, etc. Thus the number of iterations and CPU time may be different for different values of these parameters.

The transport equation has been solved with GMRES(20) combined with MILU postconditioning. Solving for the pressure takes most of the time, as in the Cartesian case.

REFERENCES

1. A. E. Mynett, P. Wesseling, A. Segal and C. G. M. Kassels, 'The ISNaS incompressible Navier–Stokes solver: invariant discretization', *Appl. Sci. Res.*, **48**, 175–191 (1991).
2. A. Segal, P. Wesseling, J. Van Kan, C. W. Oosterlee and K. Kassels, 'Invariant discretization of the incompressible Navier–Stokes equations in boundary fitted co-ordinates', *Int. j. numer. methods fluids*, **15**, 411–426 (1992).
3. C. W. Oosterlee, 'Robust multigrid methods for the steady and unsteady incompressible Navier–Stokes equations in general coordinates', *Ph.D. Thesis*, Delft University of Technology, 1993.

4. C. W. Oosterlee, P. Wesseling, A. Segal and E. Brakkee, 'Benchmark solutions for the incompressible Navier–Stokes equations in general co-ordinates on staggered grids', *Int. j. numer. methods fluids*, **17**, 301–321 (1993).
5. P. Wesseling, A Segal, J. J. I. M. Van Kan, C. W. Oosterlee and C. G. M. Kassels, 'Finite volume discretization of the incompressible Navier–Stokes equations in general coordinates on staggered grids', *Comput. Fluid Dyn. J.*, **1**, 27–33 (1992).
6. C. Vuik, 'Solution of the discretized incompressible Navier–Stokes equations with the GMRES method', *Int. j. numer. methods fluids*, **16**, 507–523 (1993).
7. G. Segal, K. Vuik and K. Kassels, 'On the implementation of symmetric and antisymmetric periodic boundary conditions for incompressible flow', *Int. j. numer. methods fluids*, **18**, 1153–1165 (1994).
8. M. Zijlema, A. Segal and P. Wesseling, 'Invariant discretization of the k–ε model in general co-ordinates for prediction of turbulent flow in complicated geometries', *Comput. Fluids*, **24**, 209–225 (1995).
9. C. W. Oosterlee and P. Wesseling, 'A multigrid method for an invariant formulation of the incompressible Navier–Stokes equations in general co-ordinates', *Commun. Appl. Numer. Methods*, **8**, 721– 734 (1992).
10. C. W. Oosterlee and P. Wesseling, 'A robust multigrid method for a discretization of the incompressible Navier–Stokes equations in general coordinates', *Impact Comput. Sci. Eng.*, **5**, 128–151 (1993).
11. C. W. Oosterlee and P. Wesseling, 'Multigrid schemes for time-dependent incompressible Navier–Stokes equations', *Impact Comput. Sci. Eng.*, **5**, 153–175 (1993).
12. S. Zeng and P. Wesseling, 'Multigrid solution of the incompressible Navier–Stokes equations in general coordinates', *SIAM J. Numer. Anal.*, **31**, 1764–1784 (1994).
13. S. Zeng, C. Vuik and P. Wesseling, 'Solution of the incompressible Navier–Stokes equations in general coordinates by Krylov subspace and multigrid methods', *Adv. Comput. Math.*, **4**, 27–49 (1995).
14. Y. Saad and M. H. Schultz, 'GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems', *SIAM J. Sci. Stat. Comput.*, **7**, 856–869 (1986).
15. H. A. Van der Vorst and C. Vuik, 'GMRESR: a family of nested GMRES methods', *Numer. Lin. Alg. Appl.*, **1**, 369–386 (1994).
16. C. Vuik, 'Further experiences with GMRESR', *Supercomputer*, **55**, 13–27 (1993).
17. G. de Vahl Davis and I. P. Jones, 'Natural convection in a square cavity: a comparison exercise', *Int j. numer. methods fluids*, **3**, 227–248 (1983).
18. J. A. Meijerink and H. A. Van der Vorst, 'An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix', *Math. Comput.*, **31**, 148–162 (1977).
19. H. A. Van der Vorst, 'Iterative solution method for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems', *J. Comput. Phys.*, **44**, 1–19 (1981).
20. I. A. Gustafsson, 'A class of first order factorization methods', *BIT*, **18**, 142–156 (1978).
21. O. Axelsson and G. Lindskog, 'On the eigenvalue distribution of a class of preconditioning methods', *Numer. Math.*, **48**, 479–498 (1986).
22. H. A. Van der Vorst, 'High performance preconditioning', *SIAM J. Sci. Stat. Comput.*, **10**, 1174–1185 (1989).
23. S. C. Eisenstat, 'Efficient implementation of a class of preconditioned conjugate gradient methods', *SIAM J. Sci. Stat. Comput.*, **2**, 1–4 (1981).
24. H. A. Van der Vorst and C. Vuik, 'The superlinear convergence behaviour of GMRES', *J. Comput. Appl. Math.*, **48**, 327–341 (1993).
25. C. C. Ashcraft and R. G. Grimes, 'On vectorizing incomplete factorization and SSOR preconditioners', *SIAM J. Sci. Stat. Comput.*, **9**, 122–151 (1988).
26. E. De Sturler and D. R. Fokkema, 'Nested Krylov methods and preserving the orthogonality', in T. A. Manteuffel and S. F. McCormick (eds), *Proc. Sixth Copper Mountain Conf. on Multigrid Methods*, NASA Langley Research Center, Hampton, VA, pp. 111–126, 1993.