Fast iterative solvers for the discretized incompressible Navier-Stokes equations

Report 93-98

C. Vuik

TU Delft

Faculteit der Technische Wiskunde en Informatica
Faculty of Technical Mathematics and Informatics

Technische Universiteit Delft
Delft University of Technology

ISSN 0922-5641

Copies of these reports may be obtained from the bureau of the Faculty of Technical Mathematics and Informatics, Julianalaan 132, 2628 BL Delft, phone +3115784568.
A selection of these reports is available in PostScript form at the Faculty's anonymous ftp-site. They are located in the directory /pub/publications/tech-reports at ftp.twi.tudelft.nl

Fast iterative solvers for the discretized incompressible Navier-Stokes equations

C. Vuik
Faculty of Technical Mathematics and Informatics
Delft University of Technology
P.O. Box 5031
2600 GA Delft
The Netherlands

Abstract

In this paper some iterative solution methods of GMRES type to solve 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 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 preconditioners. Finally some numerical experiments to show the efficiency of the proposed methods are presented.

1 Introduction

In this paper we treat the solution of the discretized incompressible Navier-Stokes equations. The discretization of these equations in general curvilinear coordinates is described in [8], [15] [9], [13] and [23]. As space discretization a finite volume technique on a boundary fitted structured grid is used. In [22] iterative methods of Krylov type to solve the discretized equations have been presented. Reference [22] also contains a short survey of other iterative methods. In this paper we shall give improvements of the iterative methods described in [22] and apply them to a wider range of problems. The improvements with respect to [22] are: preconditioners with a better rate of convergence and vectorization of the preconditioners. The methods given will be applied to problems with large gridsize (up to 160×320 cells) and problems which include transport equations.

The discretized equations given in [15] have also been solved by multi-grid methods. For stationary problems we refer to [10], [12] and for instationary problems to [11], [25]. For a stationary problem it is not easy to compare the various methods, since the multigrid method given in [10] solves the momentum equations simultaneously with the pressure equation, whereas in our software we use a time stepping method combined with pressure correction. For instationary problems the Krylov subspace methods described in this paper are more efficient than the multigrid methods described in [11], [25]. Recently we have combined the Krylov subspace method with multigrid as a preconditioner. This combination gives promising results ([24]).

Since the discretized equations contain nonsymmetric matrices [22], 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], [19], [21]. The incompressible Navier-Stokes equations in general coordinates are given by ([15]): 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 centers and the normal velocity components are calculated at the centers 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 [15].

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 $n \in I\!N$, v^n is an approximation of $v(n\Delta t)$. After Newton linearization we obtain two systems of equations ([15], [22]), namely the momentum equation:

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

and the pressure equation:

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

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. We have found that in a number of problems the behaviour of the iterative methods is comparable to that in the aforementioned test problems.

Curved channel

The curved channel is displayed in Figure 1.

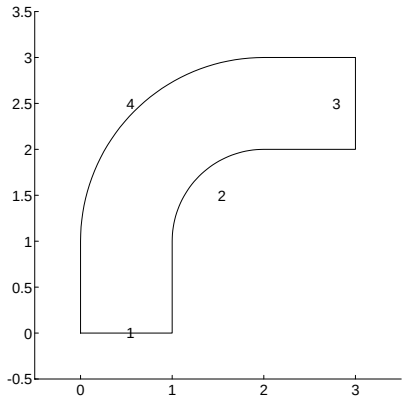

Figure 1: The physical domain of the curved channel problem.

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 boundaries 2 and 4 and the normal stress and tangential velocity given at outflow (boundary 3). We take $\rho = 250$, and $\mu = 0.5$.

Boussinesq problem

In the Boussinesq problem the Navier-Stokes equations are coupled with a temperature (transport) equation. We use a standard benchmark problem, published by [3]. The physical domain and the 20×10 grid is displayed in Figure 2. Due to buoyancy we have a body force given by

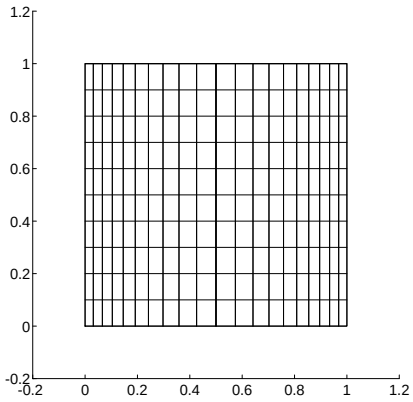

Figure 2: The 20×20 grid used in the Boussinesq problem.

$$\begin{aligned} f^1 &= 0 \ , \\ f^2 &= \tilde{g}\beta(T - T_0) \ , \end{aligned}$$

where $\tilde{g}$ is the acceleration of gravity, β a volume expansion coefficient, and T_0 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 $\rho = 1, \mu = 1, Pr = 0.71$ and $Ra = 10^6$.

In Section 2 we discuss the optimization of the RILUD preconditioner on vector computers. Furthermore an RILU preconditioner is given for the pressure equation. GMRES-like methods combined with RILU have a better rate of convergence than with RILUD, but requires 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 the RILUD preconditioner.

In Section 3 the RILUD preconditioner is adapted for the momentum equation. 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 search di-

rections leads to a faster rate of convergence for the pressure equation. Section 5 contains numerical experiments for two test problems on different gridsizes.

2 The pressure equation

In this section we consider the pressure equation. In Subsection 2.1 the preconditioner is optimized for vector computers. Next we derive a new preconditioner which has a better rate of convergence, and can be vectorized in exactly the same way.

2.1 Vectorization of the preconditioner

The discretization of the Navier-Stokes equations leads to a pressure equation, with a matrix P with nine nonzero diagonals (see [22]). In this subsection 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 [22] is denoted by ILUD.

The ILUD preconditioner is implicitly defined by the following rules [7], [17]:

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

If the last rule is replaced by

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

the MILUD preconditioner of [6] is obtained.
We have also used an averaged RILUD(α) preconditioner (see [2]). To define the RILUD preconditioner we note that if the diagonal elements d_i^{ILUD} and d_i^{MILUD} are calculated by the following expressions:

$$\begin{aligned} d_i^{ILUD} &= \varphi_{1,i}(d_{i-1}^{ILUD}, ..., d_1^{ILUD})\,; \\ d_i^{MILUD} &= \varphi_{2,i}(d_{i-1}^{MILUD}, ..., d_1^{MILUD})\,, \end{aligned}$$

then the diagonal elements of the RILUD(α) preconditioner are calculated by:

$$d_i^{RILUD} = \varphi_{3,i}(d_{i-1}^{RILUD}, ..., d_1^{RILUD})\,,$$

where $\varphi_{3,i} = \alpha\varphi_{2,i} + (1-\alpha)\varphi_{1,i}\,, \quad \text{and} \quad 0 \le \alpha \le 1\,.$

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. Due to recurrences a straightforward algorithm for this part runs in scalar speed on a vector machine. We first give an

optimization of such scalar code, according to the lines set out in [18]. Then we specify a vectorized version of the preconditioner.

Row scaling

In this paragraph it is shown that a row scaling of the pressure system leads to less work per iteration.

For the RILUD decomposition there exists a matrix R such that

$$P = LD^{-1}U - R \ .$$

Multiplication by D^{-1} leads to

$$\tilde{P} = D^{-1}P = D^{-1}LD^{-1}U - D^{-1}R = \tilde{L}\tilde{U} - \tilde{R} \ .$$

The matrices $\tilde{P}, \tilde{L}$ and $\tilde{U}$ have the following properties:

- $diag\,(\tilde{L}) = diag\,(\tilde{U}) = I$,
- the off-diagonal parts of $\tilde{L}$ and $\tilde{U}$ are equal to the corresponding parts of $\tilde{P}$;
- $diag\,(\tilde{L}\tilde{U}) = diag\,(\tilde{P})$.

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

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

so that $\tilde{L}, \tilde{U}$ also satisfy the MILUD rule. This is in contrast with a symmetric scaling (a row and column scaling ([18])) where this property may be get 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 $v_{j+1} = U^{-1}L^{-1}Pv_j$. So the amount of work per iteration is approximately two times as much as for the unpreconditioned system. In [5] 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 depends on the eigenvalue distribution of the matrix ([14], [20]). Since the spectrum of $U^{-1}L^{-1}P$ is equal to the spectrum of $L^{-1}PU^{-1}$ we expect 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 following equations:

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

the work to calculate v_{j+1} is reduced to two vector updates and the solution of an upper and lower triangular system. So one iteration of the preconditioned system costs approximately the same amount of flops as the unpreconditioned system. A disadvantage, however, is that the decrease of 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 [1] and [18]. The vector of unknowns will be denoted by $x(i,j)$ where i refers to the index of the corresponding finite volume in x_1-direction, and j the similarly in the x_2-direction. Straightforward solution of $Lx = y$ leads to the following expression:

for $j = 1, n_2$
  for $i = 1, n_1$

$$\begin{aligned} x(i,j) = y(i,j) \quad &- \quad L(i,j,5)x(i-1,j) - L(i,j,4)x(i+1,j-1) \\ &- \quad L(i,j,3)x(i,j-1) - L(i,j,2)x(i-1,j-1) \end{aligned}$$

  endfor
endfor

Note that recurrences prohibit vectorization.

In Figure 3 a diagonal ordering of the calculation is shown. In this figure the values of x in

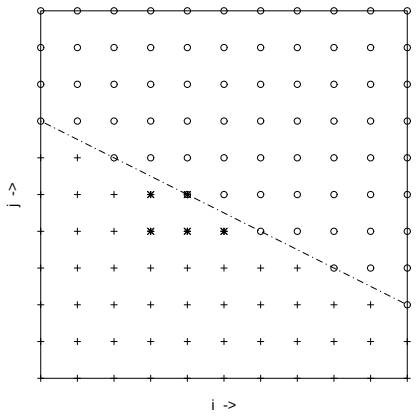

Figure 3: Ordering used for the vectorization of the solution of the system $Lx = y$.

the points denoted by a + sign have already been calculated. The points denoted by a $*$ sign display the stencil of L. Using this figure it is easily seen that all the points on the dashed diagonal ($i + 2j = c$) can be calculated independently. So this ordering leads to vectorizable code (compare [1]). 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 reordening, 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. Advantages are: all vectorlengths are equal to n_1, easy implementation, and no indirect addressing. A disadvantage is that one recurrence remains so $\frac{3}{4}$ of the work is done in vector speed and $\frac{1}{4}$ in scalar speed.

On the Convex C3840 the diagonal ordering leads to somewhat smaller computing times than the line ordering. On other machines the line ordering may be faster.

Single precision
One of the methods to solve the pressure equation is the GMRESR method [19]. This method consists of an inner and outer loop. In the inner loop a good search direction for the outer loop is calculated. Since this search direction is an approximate solution of a linear equation, the inner loop can be calculated with a low accuracy. On current computers single precision may be much faster than double precision arithmetic (compare [16] and [26]). Implementation of a single precision inner loop on the Convex C3840 leads to a 25% reduction of CPU time ([21]).

2.2 Better preconditioners

In this subsection we specify ILU preconditioners, which combined with a GMRES-like method have a faster rate of convergence, but require more memory. Vectorization of these preconditioners is possible along the same lines as in Subsection 2.1. Finally, some remarks about a preconditioner for a singular pressure matrix are given.

The first preconditioner considered in this subsection consists of the classical incomplete LU decomposition of P (all fill-in is neglected). This preconditioner is denoted by ILU:
ILU
The matrices L and U satisfy the following rules:

- $diag\,(L) = I$;
- the nonzero structure of the matrix $L + U$ is identical to the nonzero structure of P;
- if $P_{ij} \neq 0$ then $(LU)_{ij} = P_{ij}$.

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

$$\text{rowsum}\ \ (LU) = \ \text{rowsum}\ \ (P)\,, \tag{10}$$

which leads to the MILU preconditioner. Also for this preconditioner we always use an averaged method: RILU(α) which is defined in the same way as the RILUD(α) preconditioner,

but now (M)ILUD is replaced by (M)ILU.

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. So 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 to the off-diagonal part of $L + U$. With respect to vectorization we note that the nonzero structures of L and U are the same as in Subsection 2.1. So this preconditioner can be vectorized in the same way as the RILUD preconditioner.

The optimal choice of α is an open question. Results in [2] indicates that for symmetric matrices α close to 1 is a good choice. Furthermore, for increasing grid size the optimal value of α approaches 1. These insights are confirmed by our experiments (see Section 5). The combination of a GMRES-like method with RILU and $\alpha = 1$ has a fast convergence behaviour as well. However this choice is not used for the following reason: the stopping criterion is based on the norm of the preconditioned residual. Since $U^{-1}L^{-1}P$ has a large condition number for the choice $\alpha = 1$, this stopping criterion leads to inaccurate results in our experiments.

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

$$null\,(P) = \left\{ \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\} . \tag{11}$$

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 (10) can be written as:

$$LU \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = P \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} . \tag{12}$$

(12) together with (11) implies that $P \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = 0$, so LU is singular. Due 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 makes the iterative method to converge, but in our experiments $\alpha < 1$ leads to a much better rate of convergence.

We have also tried an incomplete decomposition of P where the stencil of P, L and U are given in Figure 4.
The matrices L and U are such that:

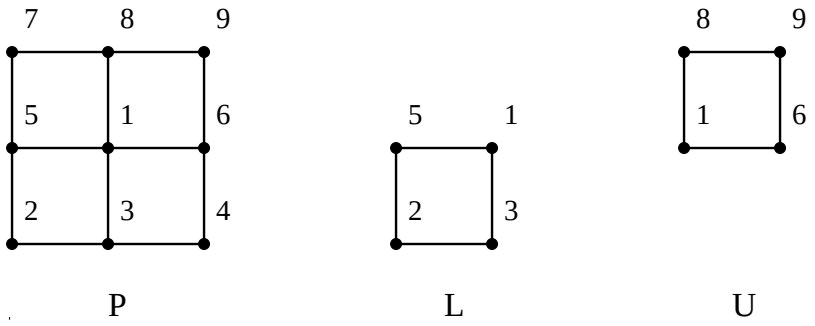

Figure 4: The stencils of P, L and U.

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

The amount of extra memory required is equal to 7 vectors. In our experiments the rate of convergence of this preconditioner is better than with RILUD but worse than with RILU. Due to the reduced stencils this preconditioning can be vectorized along the diagonals $i + j = c$ (see [1], [18]). This leads to a better vectorized code because the vector length of the loops is longer than for the diagonal ordering given in Subsection 2.1. However we conclude from our experiments that it is better to use the RILU preconditioner or the RILUD preconditioner.

3 The momentum and transport equations

In this section we consider the momentum and transport equations. The preconditioners specified in Section 2 will be adapted to the momentum equations. Furthermore, we consider a system resulting from a discretization of a transport equation. The insight obtained from the solution of the pressure and momentum equations will be used to solve the transport equations in an efficient way.

3.1 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 two times the dimension of the pressure matrix P. The matrix M^{n+1} has 13 nonzero elements per row. For the structure of M^{n+1} we refer to [22]. 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.

Due to the extra memory needed for the RILU preconditioner (13 extra vectors of length $2.n_1.n_2$) we restrict ourselves to the RILUD preconditioner for the momentum equation. The same optimization techniques as in Subsection 2.1 will be used.
So the preconditioner is combined with row scaling and the Eisenstat implementation. For

the vectorization we use the following block structure of M, L and U:

$$\begin{aligned} Mu &= \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} , \\ L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} &\quad , \quad U = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} . \end{aligned} \tag{13}$$

In [22] it has been shown that the nonzero 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 first part

$$L_{11}x_1 = y_1 \tag{14}$$

can be vectorized as indicated in Subsection 2.1. In the second part x_2 is calculated from

$$L_{22}x_2 = y_2 - L_{21}x_1 . \tag{15}$$

Since x_1 is already known, the right-hand side of (15) can be calculated in vector speed. Finally, the computation of x_2 from (15) can be done in the same way as the solution of (14).

The preconditioner RILUD is a combination of the ILUD and MILUD preconditioner. Our experiments showed that for the curved channel problem the choice α close to zero is optimal, whereas in the Boussinesq problem α close to one is optimal. 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 following equation:

$$\text{rowsum } (LD^{-1}U) = \text{ rowsum } (M) , \text{ so} \tag{16}$$

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

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. 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 the u_1 and u_2 velocities. As a consequence, α close to one can lead to a bad rate of convergence. This insight motivates us to propose a slightly adapted preconditioner which is called MILUD_2.
MILUD_2
The matrices L, D and U satisfy the same rules as for MILUD_1 except the 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 works 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 one. Hence the MILUD_2 preconditioner is more robust than MILUD_1.

The transport equation

In this subsection we describe the iterative method that we use to solve a 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, 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 a 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 consist 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 the 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 the iterative methods applied to a transport equation is comparable to the momentum equation. The matrix C^{n+1} depends on the geometry, 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 the 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. 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 is an approximation of Δp^{n+1}):

GMRESR algorithm

$r_0 = b - Px_0$, $k = -1$;
while $\|r_{k+1}\|_2 >$tol do

$k := k+1$, compute $u_k^{(0)}$ and $c_k^{(0)} = Pu_k^{(0)}$;
for $i = 0, 1, ..., k-1$ do
$\alpha_i = c_i^T c_k^{(i)}$; $c_k^{(i+1)} = c_k^{(i)} - \alpha_i c_i$; $u_k^{(i+1)} = u_k^{(i)} - \alpha_i u_i$;
$c_k = c_k^{(k)} / \|c_k^{(k)}\|_2$; $u_k = u_k^{(k)} / \|c_k^{(k)}\|_2$;
$x_{k+1} = x_k + u_k c_k^T r_k$;
$r_{k+1} = r_k - c_k c_k^T r_k$.

In the original GMRESR algorithm $u_k^{(0)}$ is computed by one iteration of GMRES(m) applied to $Py_k^{(0)} = r_k$. Other variants are proposed in [4] and [24]. In this paper the original GMRESR algorithm is used combined with the "min alfa" truncation strategy (for the details of "min alfa" see [21]).

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, $k = 0, 1, ..., n_s$ 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 $span\ \{c_0, ..., c_{n_s}\}$ as follows:

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

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 timestep is equal to n_2. The vectors u_k and c_k, $k = 0, 1, ..., n_s = \min(n_s + n_2, n_t)$ are stored in memory. These directions are reused in the third timestep etc. Note that n_t is an upperbound of the number of direction vector, 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 "min alfa" 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. So the search directions remain the same for every time step $n \geq 2$.

To illustrate this adaptation of the GMRESR algorithm we give results for the first test problem on a 16×64 grid with $\rho = 250$, $\mu = 0.5$, implying a Reynolds number of 500. We use GMRESR with GMRES(8) as inner loop, and no preconditioning. In Table 1 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 speedup when the search directions are reused. The convergence behaviour is given in Figure 5. For the original GMRESR algorithm the superlinear convergence sets in after 13 outer iterations. The GMRESR algorithm with reuse of search directions leads to fast convergence from the

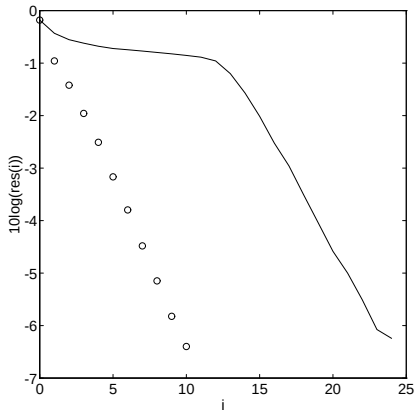

Figure 5: Convergence behaviour of GMRESR (——), and GMRESR with reuse of search directions (0) (grid 16×64).

beginning. So the gain in iterations and CPU time is not a consequence of the decrease of the norm of the initial residual due to (19), but a consequence of the fact that the components in slowly converging eigenvectors are absent, due to the expanded orthogonalization. Compare the description of the superlinear convergence behaviour of GMRES as given in [20]. The results in Table 1 show that the number of iterations decreases when the value of n_t increases. This agrees with our explanation 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.

	original GMRESR(8)		GMRESR(8) with reuse	
n_t	outer iterations	CPU	outer iterations	CPU
30	24	0.40	8	0.16
20	24	0.39	10	0.19
10	30	0.46	18	0.30

Table 1: Number of iterations and CPU time for different GMRESR variants.

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 nonlinear or time-dependent problems).

We also give results for GMRESR combined with an RILU preconditioner ($\alpha = 0.975$). We again use the first test problem but now on a 64×256 grid. The results given in Table 2 and Figure 6 are comparable with the results for the 16×64 grid.

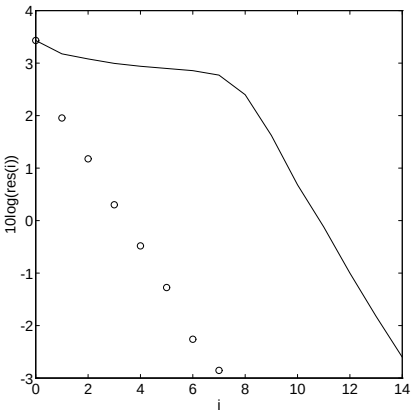

Figure 6: Convergence behaviour of GMRESR (—), and GMRESR with reuse of search directions (0), combined with an RILU preconditioner (grid 64×256).

	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

Table 2: Number of iterations and CPU time for different GMRESR variants combined with RILU ($\alpha = 0.975$).

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. Since $M^{n+1} \neq M^n$ the relation $M^{n+1}u_k = c_k$ does no longer hold. So only the vectors u_k are stored in memory. The adapted GMRESR algorithm is now started with the following loop:

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

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 direc-

tions are different or 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, $\rho = 250$, $\mu = 0.5$ and $\Delta t = 0.15$. The convergence behaviour of GMRESR, with GMRES(4) as inner loop, and without preconditioning is given in Figure 7 for the momentum equation in the third time step. From this figure it appears that the convergence behaviour of GMRESR

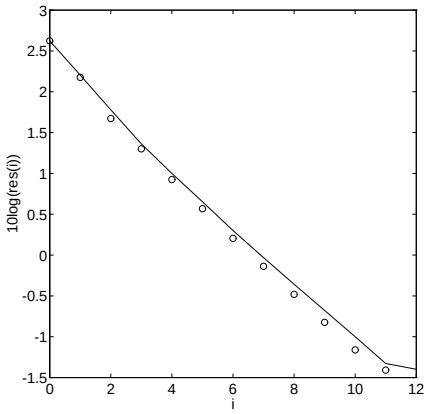

Figure 7: Convergence behaviour of GMRESR (——), and GMRESR with reuse of search directions (0).

is linear. Note that there is only a small gain in iterations, whereas the CPU time is larger. So for the momentum equation, GMRESR with reuse of search directions does not lead to a faster solution method.

For the transport equation in the second testproblem we obtain the same results as for the momentum equation.

5 Numerical results

In this section we present the results of some numerical experiments. We start with the curved channel problem. The efficiency 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 3 the megaflop rate is given for the diagonal-wise ordering given in Subsection 2.1.

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

Table 3: Megaflop rate of the vectorized RILU(D) preconditioner with diagonal ordering for the pressure equation.

Without vectorization the multiplication by L^{-1} or U^{-1} has a megaflop rate equal to 9. From Table 3 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 Subsection 2.2 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 with preconditioning it is based on $\|U^{-1}L^{-1}r_k\|_2$. Table 4 gives the number of iterations for GMRES (without restarting) and various choices of α. The iteration process is stopped if $\|r_k\|_2/\|r_0\|_2 \leq 10^{-6}$.

α / grid size	0.975	0.99	1
16×64	23	24	22
32×128	34	34	32
64×256	57	49	46
128×512	104	84	64

Table 4: Number of iterations of GMRES using the RILU(α) postconditioner for the pressure equation.

Note that for this problem $\alpha = 1$ leads to the minimal number of iterations. Furthermore for small grid sizes $\alpha \in [0.975, 1]$ leads to the same number of iterations, whereas for large grid sizes the optimal values of α are close to one, and the sensitivity of the number of iterations required to α increases.

In Figure 8 the number 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. So this figure gives a good idea of the performance of the postconditioners. Note that especially for large grid sizes MILU becomes much better than MILUD. The results presented in Figure 8 motivate us to use an RILU preconditioner instead of a RILUD preconditioner. Whereas the results of Table 4 motivate us to choose RILU(0.99) for the pressure equation. This choice of α is a compromise between a fast convergence and a not too large condition number.

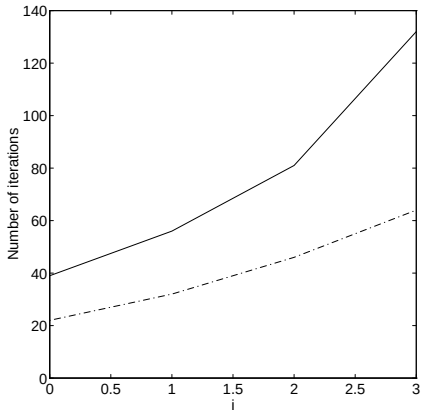

Figure 8: Number of iterations of full GMRES combined with MILUD (——) and MILU ($- \cdot - \cdot$). The grid size is equal to $(16.2^i) \times (64.2^i)$.

Table 5 shows CPU times for the solution of the pressure equation, using the RILU(0.99) postconditioner combined with truncated GMRESR(m) and reuse of the search directions.

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 5: The amount of memory, CPU time and the number of iterations for the pressure equation.

The results are measured in the second time step. Comparing these results with [22] 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 [22], but in addition the new method is approximately 3 times faster.

In [22] the momentum equation has been solved with a diagonal preconditioner. In this subsection 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. Termination criteria based on $\|r_k\|_2$ and $\|L^{-1}r_k\|_2$ lead to the same results. Furthermore, the Eisenstat implementation is used, which saves CPU time. The iteration process is stopped if $\|r_k\|_2/\|r_0\|_2 \leq 10^{-4}$. The experiments are done in the second time step. Table 6 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

		diagonal		ILUD		building
grid size	time step	iterations	CPU	iterations	CPU	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

Table 6: Number of iterations and CPU time using different preconditioners for the momentum equations.

with [22] we see again a large gain in CPU time. The last column in Table 6 contains the CPU time to build the momentum and pressure equation. Note that comparison of tables 5 and 6 show 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 show 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

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 7: The amount of memory, CPU time and the number of iterations for the pressure equation.

problem.

Table 8 gives results for the momentum equations. In these experiments GMRES(20) combined with ILUD and RILUD_2(0.95) as preconditioners have been used. In all cases a time step $dt = 4.10^{-4}$ has been chosen independently of the grid size. Note that for increasing grid size RILUD_2 becomes much better than ILUD. 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 9 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

	ILUD		RILUD_2	
grid size	iterations	CPU	iterations	CPU
20×40	5	0.034	5	0.034
40×80	8	0.22	8	0.22
80×160	13	0.94	10	0.73
160×320	23	8.1	12	4.1

Table 8: The number of iterations and CPU time for the momentum equation.

temperature is used in the right-hand side of the next time step.

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

Table 9: The number of iterations and CPU time for the transport equation.

The GMRES(20) method combined with the MILU postconditioner has been used. The iterations process is stopped if $\|r_k\|_2/\|r_0\|_2 \leq 10^{-6}$. Note that comparison of tables 7, 8, and 9 again shows that the solution of the pressure equation 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 coordinates with the pressure correction method. Comparing the results of this paper with [22] we note a considerable decrease of CPU time to solve the pressure and momentum equations, due to the novel idea of reuse of search directions for the pressure equation and 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 $\alpha = 0.99$ appears to be a good choice for the average parameter, whereas in the singular case $\alpha = 0.975$ 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 required memory is available because the memory required to store the momentum matrix can be re-used.

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 not only depend on the geometry and boundary conditions, but also on the other parameters as there are: time, time step, ρ, μ etc. So 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] C.C. Ashcraft and R.G. Grimes. On vectorizing incomplete factorization and SSOR preconditioners. *SIAM J. Sci. Stat. Comput.*, 9:122–151, 1988.

[2] O. Axelsson and G. Lindskog. On the eigenvalue distribution of a class of preconditioning methods. *Numer. Math.*, 48:479–498, 1986.

[3] G. de Vahl Davis and I.P. Jones. Natural convection in a square cavity: a comparison exercise. *Int. J. Num. Meth. Fluids*, 3:227–248, 1983.

[4] E. De Sturler and D.R. Fokkema. Nested Krylov methods and preserving the orthogonality. In T.A. Manteuffel and S.F. McCormick, editors, *Proceedings of the Sixth Copper Mountain Multigrid Conference on Multigrid Methods*, VA, 1993. NASA Langley Research Center, Hampton.

[5] S.C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Stat. Comput.*, 2:1–4, 1981.

[6] I.A. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[7] 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. Comp.*, 31:148–162, 1977.

[8] A.E. Mynett, P. Wesseling, A. Segal, and C.G.M. Kassels. The ISNaS incompressible Navier-Stokes solver: invariant discretization. *Applied Scientific Research*, 48:175–191, 1991.

[9] C.W. Oosterlee. *Robust multigrid methods for the steady and unsteady incompressible Navier-Stokes equations in general coordinates.* PhD thesis, Delft University of Technology, The Netherlands, 1993.

[10] C.W. Oosterlee and P. Wesseling. A multigrid method for an invariant formulation of the incompressible Navier-Stokes equations in general co-ordinates. *Communications in Applied Numerical Methods*, 8:721–734, 1992.

[11] C.W. Oosterlee and P. Wesseling. Multigrid schemes for time-dependent incompressible Navier-Stokes equations. *Impact Comp. Science Engng*, 5:153–175, 1993.

[12] C.W. Oosterlee and P. Wesseling. A robust multigrid method for a discretization of the incompressible Navier-Stokes equations in general coordinates. *Impact. Comp. Science Engng.*, 5:128–151, 1993.

[13] 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. Num. Meth. Fluids*, 17:301–321, 1993.

[14] Y. Saad and M.H. Schultz. GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7:856–869, 1986.

[15] 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. Num. Meth. Fluids*, 15:411–426, 1992.

[16] K. Turner and H.F. Walker. Efficient high accuracy solutions with GMRES(m). *SIAM J. Sci. Stat. Comput.*, 13:815–825, 1992.

[17] 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.

[18] H.A. Van der Vorst. High performance preconditioning. *SIAM J. Sci. Stat. Comp.*, 10:1174–1185, 1989.

[19] H.A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Numer. L.A.A.*, 1993. to appear.

[20] H.A. van der Vorst and C. Vuik. The superlinear convergence behaviour of GMRES. *J. Comput. Appl. Math.*, 48:327–342, 1993.

[21] C. Vuik. Further experiences with GMRESR. *Supercomputer*. to appear.

[22] C. Vuik. Solution of the discretized incompressible Navier-Stokes equations with the GMRES method. *Int. J. for Num. Meth. Fluids*, 16:507–523, 1993.

[23] 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. *Comp. Fluid Dynamics Journal*, 1:27–33, 1992.

[24] S. Zeng, C. Vuik, and P. Wesseling. Solution of the incompressible Navier-Stokes equations in general coordinates by Krylov subspace and multigrid methods. Report 93-64, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, 1993.

[25] S. Zeng and P. Wesseling. Multigrid solution of the incompressible Navier-Stokes equations in general coordinates. *SIAM J. Num. Anal.*, 1993. to appear.

[26] M. Zubair, S.N. Gupta, and C.E. Grosch. A variable precision approach to speedup iterative schemes on fine grained parallel machines. *Parallel Comp.*, 18:1223–1232, 1992.