Some parallelizable preconditioners for GMRES

Report 96-50

R.R.P. van Nooyen
C. Vuik
P. Wesseling

TU Delft
Technische Universiteit Delft
Delft University of Technology

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

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 +31152784568.
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

Some parallelizable preconditioners for GMRES *

R. R. P. van Nooyen, C. Vuik, P. Wesseling

Contents

*This work was sponsored by the Stichting Nationale Computer Faciliteiten (National Computer Facilities Foundation, NCF) for the use of supercomputer facilities with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO). The Training and Research in Advanced Computer Systems (TRACS) program of the Edinburgh Parallel Computing Centre contributed supercomputer time and financial support for a 3 month stay in Edinburgh for the first author.

Abstract

Some preconditioners that show potential for parallelization are examined to judge their efficiency relative to the current ISNaS incompressible flow solver preconditioners. The experiments show that in 2D parallel versions of the current preconditioners outperform there rivals. In 3D the approximate inverse preconditioner shows promise as preconditioner for the pressure equation.

1 Introduction

To compute incompressible turbulent flows in complicated two- and three-dimensional domains we use a numerical method with the following properties. Boundary fitted coordinates and domain-decomposition are used to handle geometrically complicated domains. The finite volume method and a staggered grid are used for discretization in space. A combination of the Euler backward scheme and the pressure correction method is used to advance the solution in time. The computer program is referred to as the ISNaS (Information System for Navier-Stokes equations) incompressible flow solver [11].

Benchmark solutions for the discretization chosen can be found in [9]. Research into parallelism through multi-block techniques is described in [3]. Some results concerning the use of multigrid solvers in this code can be found in [19, 7, 8] . Periodic and anti-periodic boundary conditions are discussed in [12] . Some aspects of the discretization are discussed in [17, 13, 16, 18]. The treatment of turbulence is discussed in [20, 21]. In this paper we consider a two dimensional single block laminar flow as a test case for implementation on a Cray Research T3D.

The ISNaS solver uses pressure correction, it determines an intermediate velocity field and then calculates a correction to obtain a velocity field with zero divergence. The intermediate velocity field is obtained by applying a linear solver to the Newton linearization of the momentum equations. The pressure correction is obtained as the solution of a second linear system of equations.

The equations are discretized on a staggered grid and with curvilinear coordinates. The curvilinear coordinates result in some extra non-zero entries in the matrix. In 2D we initially find a 17 point stencil for the momentum vector component equations and a 9 point stencil for the pressure equation. With divergence freedom, the number of points needed in the momentum stencil reduces to 13.

It is impossible to treat, in one paper, all aspects of the parallelization of an incompressible Navier-Stokes code. As an initial analysis showed that matrix construction is embarrassingly parallel, we concentrate on the linear solvers for the momentum and pressure equations. Within the solver the preconditioner is probably the main problem. This paper studies the effectiveness and potential for parallelization for several preconditioners when compared with the preconditioners described in [15].

For later reference we include a short description of the GMRES(m) method as given in [10], but with a left-preconditioner M_1 and a right-preconditioner M_2,.

1. *Start*: Choose an initial estimate x_0, compute the initial residual $r_0 = M_1(f - Ax_0)$ and determine $v_1 = r_0/\|r_0\|$.

2. *Iterate*: For $j = 1, 2, \ldots, m$ do:

$$\begin{aligned} h_{i,j} &= (M_1 A M_2 v_j, v_i), i = 1, 2, \ldots, j, \\ \hat{v}_{j+1} &= M_1 A M_2 v_j - \sum_{i=1}^{j} h_{i,j} v_i, \end{aligned}$$

$$\begin{aligned} h_{j+1,j} &= \|\hat{v}_{j+1}\|, \\ v_{j+1} &= \hat{v}_{j+1}/h_{j+1,j} \end{aligned}$$

3. *Form the approximate solution*: $x_m = x_0 + M_2 \sum_{k=1}^{m} y_k v_k$ where the y_k minimize $\|M_1(r_0 - A\sum_{k=1}^{m} y_k M_2 v_k)\|$.

4. *Restart*: Compute $r_m = M_1(f - Ax_m)$, if satisfied then stop else compute $x_0 = x_m$, $v_1 = r_m/\|r_m\|$ and go to 2.

Where in step 2 the new search direction is made perpendicular to all previous search directions with the standard Gram-Schmidt method. For a full description of the GMRES method we refer to [10].

2 Preconditioners

We discuss two main types of preconditioner. The Incomplete LU preconditioners are based on an approximation of the matrix through the use of sparse approximations to the matrices L and U that form the LU decomposition of the matrix. The approximate inverse preconditioner is based on the approximation of the inverse through the minimization in a certain norm of the difference between the identity matrix and the product of the matrix with a sparse approximate inverse.

2.1 Some variations on Incomplete LU preconditioning

The well-known Incomplete LU preconditioners [6] are based on the following idea. If the LU decomposition of A is known, then to solve the system we only need to perform one backward and one forward substitution. It seems reasonable to assume that, if we have $M = LU$ and M approximates A, then an iterative solver might converge more quickly for $M^{-1}Ax = M^{-1}b$ than for $Ax = b$.

Now we consider the ILUD [6] preconditioner. Let $A = L_A + D_A + U_A$, where L_A is strictly lower triangular, D_A is a diagonal matrix and U_A is strictly upper triangular. In this case we take $M = (D + L_A)D^{-1}(D + U_A)$, with D such that $A_{ii} = M_{ii}$, i.e. A and M have identical main diagonals,

$$D_{ii} = A_{ii} - \sum_{j<i} A_{ij} D_{jj}^{-1} A_{ji} \ . \tag{1}$$

A variation on ILUD, called MILUD [5] uses a slightly different criterion to determine the elements of D. We still use $M = (D + L_A)D^{-1}(D + U_A)$, but now D is such that $\sum_j A_{ij} = \sum_j M_{ij}$, i.e. A and M have identical row sums,

$$D_{ii} = A_{ii} - \sum_{j<i} \sum_{k>j} A_{ij} D_{jj}^{-1} A_{jk} \tag{2}$$

2.1.1 Momentum equations

For the momentum equations we use a row by row combination of 0.05 times the D from ILUD and 0.95 times the D taken from $MILUD_2$, as described in [15], this is similar to RILUD as described in [1].

To describe the $MILUD_2$ preconditioner we use the following splitting

$$u = \begin{pmatrix} u^{(1)} \\ u^{(2)} \end{pmatrix} \tag{3}$$

where $u^{(1)}$ contains a vector of the first momentum vector component in all cells and $u^{(2)}$ contains a vector of the second momentum vector component in all cells. The corresponding partitions of A, M, L and U are

$$A = \begin{pmatrix} A^{(11)} & A^{(12)} \\ A^{(21)} & A^{(22)} \end{pmatrix} \tag{4}$$

$$M = \begin{pmatrix} M^{(11)} & M^{(12)} \\ M^{(21)} & M^{(22)} \end{pmatrix} \tag{5}$$

$$L = \begin{pmatrix} L^{(11)} & 0 \\ L^{(21)} & L^{(22)} \end{pmatrix} \tag{6}$$

$$U = \begin{pmatrix} U^{(11)} & U^{(12)} \\ 0 & U^{(22)} \end{pmatrix} \tag{7}$$

where $L = D + L_A$ and $U = D + U_A$. And we take $M = (D + L_A)D^{-1}(D + U_A)$ with D such that $\sum_j A_{ij}^{(qq)} = \sum_j M_{ij}^{(qq)}$, i.e. A and M have identical row sums for the sub-matrices $A^{(qq)}, M^{(qq)}$ found when we split A and M into parts according to velocity components.

The standard MILUD preconditioning would determine D using the conditions

$$\text{rowsum}(L^{(11)}{D^{(11)}}^{-1}U^{(11)}) + \text{rowsum}(L^{(11)}{D^{(11)}}^{-1}U^{(12)}) = \text{rowsum}(M^{(11)} + M^{(12)}) \tag{8}$$

and

$$\text{rowsum}(L^{(21)}{D^{(11)}}^{-1}U^{(11)}) + \text{rowsum}(L^{(21)}{D^{(11)}}^{-1}U^{(12)}) + \text{rowsum}(L^{(22)}{D^{(22)}}^{-1}U^{(22)}) = \tag{9}$$

$$\text{rowsum}(M^{(21)} + M^{(22)}) \ . \tag{10}$$

This is called $MILUD_1$ in [15].

On the other hand, $MILUD_2$ would determine D using the conditions

$$\text{rowsum}(L^{(11)}{D^{(11)}}^{-1}U^{(11)}) = \text{rowsum}(M^{(11)}) \tag{11}$$

and

$$\text{rowsum}(L^{(21)}{D^{(11)}}^{-1}U^{(12)}) + \text{rowsum}(L^{(22)}{D^{(22)}}^{-1}U^{(22)}) = \text{rowsum}(M^{(22)}) \ . \tag{12}$$

2.1.2 Pressure equations

For the pressure we use $M = (D + L)D^{-1}(D + U)$. With the sparsity pattern of L and U equal to that of L_A and U_A, for each i we calculate: for all $l < i$ and $A_{il} \neq 0$:

$$L_{il} + \sum_{j < \min(i,l)} L_{ij} D_{jj}^{-1} U_{jl} = A_{il} \ , \tag{13}$$

for all $l > i$ and $A_{iL} \neq 0$:

$$U_{il} + \sum_{j < \min(i,l)} L_{ij} D_{jj}^{-1} U_{jl} = A_{il} \ , \tag{14}$$

$$D_{ii} + \sum_{j<i} L_{ij} D_{jj} U_{ji} = A_{ii} \ . \tag{15}$$

A variation on this is MILU, where again we calculate an alternative D by imposing the condition $\sum_j A_{ij} = \sum_j M_{ij}$ i.e.,

$$D_{ii} + \sum_{j<i} \sum_{k>j} L_{ij} D_{jj}^{-1} U_{jk} + \sum_{j<i} L_{ij} + \sum_{i<j} U_{ij} = \sum_j A_{ij} \ . \tag{16}$$

In the ISNaS solver we use RILU(0.975), which is a row by row combination of the two diagonal matrices, 0.025 times the ILU version and 0.975 times the MILU version [14], [1].

2.1.3 A parallel version

On a parallel machine ILU is not very efficient, but in [2] it is shown that we can use the fact that we have a 2D rectangular domain to obtain some parallelism as follows: make vertical strips, distribute the strips over the processors and overlap calculation of strips at different "heights" on different processors.

We will illustrate this by discussing a highly simplified version of this procedure for a five point stencil on a square $n \times n$ grid. Each vertical strip consists of one grid cell in horizontal direction, so n processors are used. This avoids the complications due to synchronisation needed for additional stencil elements along diagonals and notation needed to indicate which processor builds the stencil for a given grid cell.

Our preconditioner uses the lexicographical cell ordering, sorting first by y-coordinate and then by x-coordinate. As our starting cell we take the cell in the lower left corner of the domain

During the construction of the preconditioner we have two classes of cells, namely cells where the preconditioner has been constructed and cells where this is not the case.

During application of the preconditioner there are two back substitution steps, namely one for an upper triangular matrix and one for a lower triangular matrix. During a back substitution step there are again two classes of cells, namely cells where the solution is known and cells where we still need to calculate it. The preconditioner construction and the back substitution for a lower triangular matrix follow the same pattern. We start in the lower left corner and calculate new cells according to the lexicographical ordering. For the upper triangular matrix the back substitution starts in the upper right corner and calculates new cell values according to the reverse lexicographical ordering. We shall only describe the parallel version of the preconditioner construction. The parallel version of back substitution for lower triangular matrices follows the same pattern and the back substitution algorithm for upper triangular matrices follows almost immediately from that for a lower triangular matrix.

For both preconditioner construction and back substitution for a lower triangular matrix the new values (preconditioner elements or solution values) to be generated in a cell depend only on the values already generated (preconditioner elements or solution values) in the cell directly to the left of the current cell and the cell directly below the current cell, i.e. its neighbours to the west and to the south.

To describe the construction and application of the preconditioner we introduce virtual cells all around the 2D domain and treat these cells as cells where the values to be generated are already known.

Now assign the responsibility to generate the new values needed in the cell (i, j) with centre (x_i, y_j) to processor P_i. If we assume a regular grid and a coordinate indexing such that $x_1 < x_2 < \ldots < x_n$, $y_1 < y_2 < \ldots < y_n$, then we see immediately that all cells with indices (i', j') such that $i' + j' = k + 1$ can be calculated as soon as all calculations for cells with indices (i, j) with $i + j = k$ have been completed. Given our distribution of cells over processors, on average $n^2/(2n - 1)$ or approximately $n/2$ processors are at work at the same time.

Please note that on a rectangular $n \times m$ with n processors grid we find that on average $mn/(m + n - 1)$ or approximately $\max(m/(1 + m/n), n/(1 + n/m))$ processors are at work at the same time. This may allow us to reach efficiencies considerably better than the 50% possible on a $n \times n$ grid. We make use of this in the experiments.

2.1.4 Multi-colour version

An alternative is the use of a different ("coloured") point ordering. Again a parallelism versus efficiency problem occurs, because the number of communication steps needed is equal to the number of colours used but it appears from table 3 that the number of iterations needed is

inversely proportional to the number of colours used. Possible orderings for a nine point stencil in 2D: different colours for different rows of cells, different colours for different columns of cells, multicolour chessboard. The different colours decouple adjacent cells and remove the need for the bulk of the communication needed in the preconditioner.

2.1.5 The Neumann series

To avoid back substitution we can try to approximate the inverse of the lower and upper triangular matrices by a series of matrix multiplications. One possible method is a truncated Neumann series. If M is non-singular matrix with norm smaller than one, then we can approximate M^{-1} by the first n terms in the Neumann series:

$$(I - M)^{-1} = (I + M)\prod_{i=1}^{\infty}(I + M^{2^i}). \tag{17}$$

The use of this series replaces back substitution (not very parallel) by matrix multiplication (highly parallel).

2.1.6 A multiplicative version

A variation on $ILUD$ that only uses matrix-multiplication and seems reasonably effective in 3D is the following method. Take L_A, D_A and U_A as in the previous section. Define

$$D_{ii} = 1 - \sum_{j<i}(A_{ij}A_{ji})/(A_{ii}A_{jj}) \ , \tag{18}$$

$$L_1^{-1} = (I - D_A^{-1}L_A) \ , \tag{19}$$

$$U_1^{-1} = (I - D^{-1}D_A^{-1}U_A) \ , \tag{20}$$

and

$$M = D_A L_1 D U_1 \ .$$

We will show that under certain conditions $M^{-1}A$ resembles the identity matrix. First we invert M.

$$M^{-1} = U_1^{-1}D^{-1}L_1^{-1}D_A^{-1} =$$
$$(I - D^{-1}D_A^{-1}U_A)D^{-1}(I - D_A^{-1}L_A)D_A^{-1}.$$

Multiplication of A by M^{-1} gives

$$M^{-1}A = M^{-1}(D_A + L_A + U_A)$$

or

$$M^{-1}A = (I - D^{-1}D_A^{-1}U_A)D^{-1}(I - D_A^{-1}L_A)(I + D_A^{-1}L_A + D_A^{-1}U_A)$$

Now write $L = D_A^{-1}L_A$ and $U = D_A^{-1}U_A$,

$$M^{-1}A = (I - D^{-1}U)D^{-1}(I - L)(I + L + U) =$$

$$(I - D^{-1}U)D^{-1}(I + U - L^2 - LU) \ .$$

We reorder the terms and add and subtract a term D,

$$(I - D^{-1}U)D^{-1}(I + U - L^2 - LU) = (I - D^{-1}U)D^{-1}(D + U - L^2 + I - LU - D) =$$

$$I - (D^{-1}U)^2 - (I - D^{-1}U)(D^{-1}L^2 - D^{-1}(I - LU - \mathrm{diag}(I - LU)))$$

Now, if we assume that A is scaled such that $|A_{ii}| > 1$ and $|A_{ii}| > |A_{ij}|$ for $i \neq j$ then the elements of L, U, $D^{-1}L$ and $D^{-1}U$ are of order $e < 1$. There are n_c non-zero elements per row in A, so

$$I - (D^{-1}U)^2 - (I - D^{-1}U)(D^{-1}L^2 - D^{-1}(I - LU - \mathrm{diag}(I - LU))) = I + (I - D^{-1}U)E \ ,$$

where E is a matrix with zero diagonal and off-diagonal elements of order $n_c e^2/2$. This may mean that the resulting matrix is diagonally dominant, but sufficient conditions have not been derived and the preconditioner fails for certain A.

2.2 Approximate inverse preconditioning

An approximate inverse P of A is determined as follows. Either $\|AP-I\|$ or $\|PA-I\|$ is minimized for a given norm and given bounds on the sparsity pattern for P [4]. In this report we shall use the Frobenius norm for this purpose. This norm is defined as

$$\|W\|_F^2 = Tr(W^T W) = \sum_{i=1,j=1}^{N} W_{ij}^2 \ . \tag{21}$$

We limit the search to matrices P with a given sparsity pattern, such that

$$\|AP - I\|_F^2 = \sum_{k=1}^{n} \|(AP - I)e_k\|_2^2 \tag{22}$$

is minimized. If P_k denotes the k-th column of P we have to minimize $\sum_{k=1}^{n} \|AP_k - e_k\|_2^2$. Since P_k has only non-zero elements in say m_k selected places the minimization leads to n independent least squares problems. If $m_k = n$ the vector P_k is the k-th column of A^{-1}. Otherwise we have to solve for every $k \in \{1, 2, ..., n\}$ an $n \times m_k$ least squares problem and P_k is the k-th column of an approximate inverse of A (for further details see [4]).

In our 2D experiments we use a sparsity pattern based on a 5×5 stencil (see Figure 1). In our 3D experiments we either use the sparsity pattern of the pressure matrix or we allow non-zero elements in P based on a $5 \times 5 \times 5$ stencil.

3 Numerical Experiments

In the first set of experiments we examine the behaviour of GMRES with the different preconditioners on grids with square and elongated cells. We are interested mainly in the number of iterations needed to obtain a factor 10^{-6} reduction of the momentum residual and a factor 10^{-4} reduction of the pressure residual. In the second set of experiments we check the dependence on the number of cells in the grid. We also give some preliminary results in 3D.

3.1 Iteration counts for various preconditioners in 2D

In most of our problems we use a 16x16 cell grid. When we use other number of grid cells, this number is explicitly stated. We consider four differently sized domains. A unit square, a $[0, 8] \times [0, 1]$ rectangle, a $[0, 8] \times [0, 1]$ rectangle, and an $[0, 8] \times [0, 8]$ square. On all domains we consider flow in all four directions. In this way we can examine possible dependencies of the iteration counts on flow direction, cell size and shape. We use $dt = 0.1$ and a parabolic inflow with a maximum value of one (furthermore we choose the density $\rho = 5$, and the dynamic viscosity $\mu = 0.5$). Unless otherwise noted, flow direction has no significant influence on the number of iterations. Modified Gram-Schmidt orthogonalization is used.

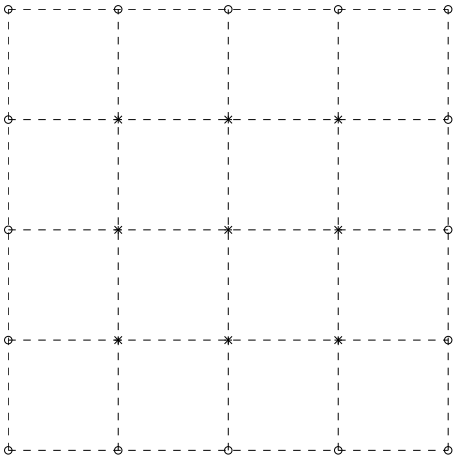

Figure 1: The original 9 point stencil is denoted by *, whereas the additional points of the 5×5 stencil used for the approximate inverse P are denoted by o

3.1.1 Momentum preconditioners

For the solution of the momentum equations we always use the GMRES(60) method, meaning that the GMRES method is restarted after 60 iterations. In Table 1 we give the number of iterations needed by GMRES when solving the momentum equations using the following preconditioners:

none No preconditioner.

diagonal Left multiplication of the matrix with the inverse of its diagonal.

ISNaS $MILUD_2$ The special version of MILUD described in section 2.1.1.

Multiplicative ILUD The variation on ILUD described in section 2.1.6.

In order to investigate the dependence of the convergence on the number of grid points we have done some experiments using the unit square domain with a 16x16, 32x32 and a 64x64 grid. The results of these experiments are given in Table 4.

3.1.2 Pressure preconditioners

The pressure equation is solved by the GMRES(200) method. In Table 2 and 3 we give iteration counts for some preconditioners for the pressure equation. The considered preconditioners are:

none No preconditioner.

diagonal Left multiplication of the matrix with the inverse of its diagonal.

ISNaS RILU The matrix M^{-1} generated by the version of RILU(0.975) described in section 2.1.2 is used as a right-preconditioner. The sparsity pattern of the L and U matrix in the preconditioner is taken from the original matrix.

RILU The matrix M from 2.1.2 is considered as the product of the lower and upper triangular matrices in the formula $M = LU$. L^{-1} is used as left-preconditioner and U^{-1} is used as right-

preconditioner.

Multiplicative ILUD The variation on ILUD described in section 2.1.6.

Approximate inverse An approximate inverse is generated as described in section 2.2. The prescribed sparsity pattern is based on a 5×5 stencil. In Table 2 the number of iterations is given for the matrix P resulting from minimization of $\|PA - I\|_F$ as a left-preconditioner. The results for the minimization strategy $\|AP - I\|_F$ are more or less the same.

Neumann series RILU; one-three terms The matrix M from 2.1.2 is considered as the product of the lower and upper triangular matrices in the formula $M = LU$. L^{-1} is used as left-preconditioner and U^{-1} is used as right-preconditioner. However, in stead of solving $Ly = f$ and $Ux = y$, we determine an approximation to x and y by matrix-vector multiplication using an approximation to L^{-1} and U^{-1}. We use a Neumann series 2.1.5 to determine this approximation.

RILU (i-direction reordering, 2-8 colours) We use the RILU method described under the heading RILU(0.975), but in stead of lexicographical ordering of the grid cells, we divide the cells into columns of different colours (the colours alternate) and order the cells first by colour and within a set of a given colour by the standard lexicographical ordering.

RILU (j-direction reordering, 2-8 colours) We use the RILU method described under the heading RILU(0.975), but in stead of lexicographical ordering of the grid cells, we divide the cells into rows of different colours (the colours alternate) and order the cells first by colour and within a set of a given colour by the standard lexicographical ordering.

RILU (ij-direction reordering, 4-25 colours) We use the RILU method described under the heading RILU(0.975), but in stead of lexicographical ordering of the grid cells, we divide the cells into sets of different colours by creating a multi-colour chessboard structure and order the cells first by colour and within a set of a given colour by the standard lexicographical ordering.

We used the unit square domain with a 16x16, 32x32 and a 64x64 grid to determine the dependence of iteration counts on the number of cells in the grid. The results are presented in Table 5.

3.2 Iteration counts for various preconditioners in 3D

In this subsection we solve the Navier-Stokes equations on a unit cube. We use various numbers of grid cells to discretize the Navier-Stokes equations. For some preconditioners we investigate the convergence behaviour of preconditioned GMRES applied to the momentum and pressure equations.

3.2.1 Momentum preconditioners

In Table 6 results are presented using the following preconditioners:

none No preconditioner.

diagonal Left multiplication of the matrix with the inverse of its diagonal.

ISNaS $MILUD_2$ The special version of MILUD described in section 2.1.1.

multiplicative ILUD The variation on ILUD described in section 2.1.6.

3.2.2 Pressure preconditioners

Table 7 contains the results for the following preconditioners:

ISNaS RILUD The matrix M^{-1} generated by the version of RILUD as described in section 2.1.1 is used as a right-preconditioner.

Multiplicative ILUD The variation on ILUD described in section 2.1.6.

Approximate inverse (19) An approximate inverse is generated as described in 2.2. The prescribed sparsity pattern is taken from the pressure matrix.

Approximate inverse (125) An approximate inverse is generated as described above. The sparsity pattern is based on a $5 \times 5 \times 5$ stencil.

4 Analysis

4.1 Iteration counts for various preconditioners in 2D

The number of iterations exhibits a strong direction dependence for flows on grids with elongated grid cells. This is expected for the momentum equations. To explain this dependence for the pressure equation we notice that this equation also depends on the direction of the flow. On the outflow boundary the boundary condition for the pressure equation is of Dirichlet type and it is of Neumann type at the other boundaries. This implies that for the 8×1 rectangle the Dirichlet condition holds on the long side for the up and down flow direction, whereas it holds on the short side for the left and right flow direction.

In the remaining part of this subsection we only consider preconditioners for the 2D pressure equation (see Table 2 and 3).

4.1.1 Approximate inverse

Note that multiplication with the larger approximate inverse matrix takes 25/9 times the work needed for a normal matrix-vector multiplication. The construction of the preconditioner is also very calculation intensive. On the other hand, both construction (almost completely local) and application (standard matrix-vector product) should parallelize very well.

4.1.2 Neumann series MILU; one-three terms.

Note that a Neumann series of n terms needs approximately $n + (n-1)n/2$ lower or upper triangular matrix-vector multiplications whereas the work needed to do one back substitution is approximately that needed for one triangular matrix-vector multiplication. The results show that we need a relatively large number of terms in the sequence to get an efficient preconditioner. This method would only be attractive if the triangular matrix-vector multiplication was at least six times more efficient than the back substitution needed in standard MILU.

4.1.3 MILU multicolour

The use of multiple colours decouples neighbouring cells and allows for easy parallelization, but for each extra colour we would need an extra communication step and the work between communication steps halves as the total work to be done remains equal to the work in one back substitution

for the entire system. The results show that we need large numbers of colours to obtain reasonable efficiency. This can easily be explained by information transport information. In standard back substitution information from the lower left corner of a 2D domain filters upwards to the upper right corner in one step and influences all intermediate cells. With two colours only half the number of intermediate cells gets the full update. As the number of colours increases the fraction of cells that corresponds to the first colour, i.e. the cells that only get the information from their own colour, decreases as one over the number of colours.

4.2 Iteration counts using different number of grid points in 2D

From Table 4 and 5 it follows that halving cell size in both directions (i.e. doubling the number of grid cells in both directions) will double the number of GMRES iterations needed for most preconditioners. The exceptions are MILUD for the momentum equations and RILU or the three term RILU Neumann series for the pressure equations. For these preconditioners the number of iterations seems to increase by a factor of $\sqrt{2}$.

4.3 Iteration counts using different number of grid points in 3D

From Table 6 we conclude, that the MILUD preconditioner is again much better than that of the multiplicative version. The results in Table 7 show that in 3D the approximate inverse can compete with RILUD. Although the relation between the increase in iterations and the increase of the number of cells is more favourable for RILUD. Possibly further study will allow this to be rectified. It then remains to weigh parallelization problems for 3D RILUD versus the extra work (125 in stead of 19 non-zero entries per row in the preconditioner) needed for the best approximate inverse or the quicker increase of the number of iterations needed when the number of cells grows for the more economical version of the approximate inverse.

5 Conclusions

The preconditioners MILUD for the momentum equations and RILU for the pressure equation are fast and efficient both in terms of the number of GMRES iterations needed on a given grid and in terms of the additional iterations needed when the number of cells is increased. The three term Neumann series variation on RILU shares the properties of RILU to a certain extent, but needs to do six times as much work. The other preconditioners simply do not give acceptable performance on small grids and their performance degrades quickly on larger grids. It is expected that this disadvantage will not be compensated sufficiently by better parallel performance.

In 3D the approximate inverse preconditioner with fixed sparsity pattern may be a suitable and easily parallelized alternative to RILUD.

References

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

[2] P. Bastian and G. Horton. Parallization of robust multi-grid methods: ILU factorization and frequency decomposition method. *SIAM J. Stat. Comput.*, 12:1457–1470, 1991.

[3] E. Brakkee, A. Segal, and C.G.M. Kassels. A parallel domain decomposition algorithm for the incompressible Navier-Stokes equations. *Simulation Practice and Theory*, 3:185–205, 1995.

[4] J.D.F. Cosgrove, J.C. Diaz, and A. Griewank. Approximate inverse preconditionings for sparse linear systems. *Int. J. Computer Math.*, 44:91–110, 1992.

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

[6] 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.

[7] 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.

[8] C.W. Oosterlee and P. Wesseling. Steady incompressible flow around objects in general coordinates with a multigrid solution method. *Num. Methods for Part. Diff. Equations*, 10:295–308, 1994.

[9] 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.

[10] 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.

[11] 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.

[12] Guus Segal, Kees Vuik, and Kees Kassels. On the implementation of symmetric and anti-symmetric periodic boundary conditions for incompressible flow. *Int. J. Num. Meth. Fluids*, 18:1153–1165, 1994.

[13] P. van Beek, R.R.P. van Nooyen, and P. Wesseling. Accurate discretization on non-uniform curvilinear staggered grids. *J. Comp. Phys.*, 117:364–367, 1995.

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

[15] C. Vuik. Fast iterative solvers for the discretized incompressible Navier-Stokes equations. *Int. J. for Num. Meth. Fluids*, 22:195–210, 1996.

[16] P. Wesseling, C.G.M. Kassels, C.W. Oosterlee, A. Segal, C. Vuik, S. Zeng, and M. Zijlema. Computing incompressible flows in general domains. In F.-K. Hebeker, R. Rannacher, and G. Wittum, editors, *Numerical methods for the Navier-Stokes equations*, pages 298–314, Braunschweig, 1994. Vieweg.

[17] 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.

[18] P. Wesseling, P. van Beek, and R.R.P. van Nooyen. Aspects of non-smoothness in flow computations. In A. Peters, G. Wittum, B. Herrling, U. Meissner, C.A. Brebbia, W.G. Gray, and G.F. Pinder, editors, *Computational Methods in Water Resources X*, pages 1263 – 1271, Dordrecht, 1994. Kluwer.

[19] S. Zeng and P. Wesseling. Multigrid solution of the incompressible Navier-Stokes equations in general coordinates. *SIAM J. Num. Anal.*, 31:1764–1784, 1994.

[20] M. Zijlema, A. Segal, and P. Wesseling. Finite volume computation of incompressible turbulent flows in general coordinates on staggered grids. *Int. J. Numer. Meth. Fluids*, 20:621–640, 1995.

[21] 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.

Preconditioner type	Flow direction	Domain form			
		1×1	8×1	1×8	8×8
none	up, down	30	31	15	9
	left, right	30	15	31	9
diagonal	up, down	30	30	13	8
	left, right	30	13	30	8
ISNaS MILUD	up	6	4	4	3
	other	7	4	4	3
Multiplicative ILUD	up	13	19	6	4
	down	12	18	6	4
	left	13	6	18	4
	right	13	6	19	4

Table 1: Iteration counts with several momentum preconditioners in 2D

Preconditioner type	Flow Direction	Domain form			
		1×1	8×1	1×8	8×8
none	up	55	89	158	54
	down	54	89	160	54
	left	54	160	89	54
	right	54	158	89	54
diagonal	up	52	87	155	52
	down	52	87	155	52
	left	52	155	87	52
	right	52	155	87	52
ISNaS RILU	up	14	6	18	14
	down	13	5	19	13
	left	14	12	9	13
	right	15	12	12	14
RILU	up	13	5	17	13
	down	12	5	17	12
	left	13	11	9	12
	right	13	11	11	13
Multiplicative ILUD	up	37	51	100	38
	down	37	52	100	37
	left	37	100	52	37
	right	37	100	51	38
Approximate inverse	up	16	35	63	16
	down	16	35	62	16
	left	16	62	35	16
	right	16	62	35	16
Neumann series RILU; one term	up	38	59	152	39
	down	44	71	153	44
	left	42	161	75	43
	right	39	170	59	43
Neumann series RILU; two terms	up	26	33	107	26
	down	29	42	108	29
	left	29	123	43	28
	right	29	131	34	29
Neumann series RILU; three terms	up	20	17	63	20
	down	20	23	63	20
	left	21	72	23	21
	right	21	74	18	21

Table 2: Iteration counts with several pressure preconditioners in 2D

Preconditioner type	Flow Direction	Domain form			
		1×1	8×1	1×8	8×8
RILU (i-direction reordering, 2 colours)	up	35	13	110	35
	down	33	13	111	36
	left	35	22	59	36
	right	35	23	58	36
RILU (i-direction reordering, 4 colours)	up	30	13	71	30
	down	29	13	71	30
	left	30	22	38	30
	right	30	22	38	30
RILU (i-direction reordering, 8 colours)	up	27	13	50	27
	down	27	12	50	26
	left	27	22	24	26
	right	27	22	26	27
RILU (j-direction reordering, 2 colours)	up	35	59	22	35
	down	35	59	22	34
	left	35	108	12	34
	right	35	108	13	34
RILU (j-direction reordering, 4 colours)	up	27	37	23	27
	down	26	36	23	25
	left	25	68	12	26
	right	25	68	14	26
RILU (j-direction reordering, 8 colours)	up	21	20	20	20
	down	20	20	20	19
	left	20	40	11	20
	right	20	40	13	21
RILU (ij-direction reordering, 4 colours)	up	37	61	109	36
	down	37	61	109	36
	left	37	109	61	36
	right	37	108	61	36
RILU (ij-direction reordering, 9 colours)	up	41	58	101	42
	down	44	62	101	44
	left	44	101	62	44
	right	41	101	58	42
RILU (ij-direction reordering, 16 colours)	up	34	38	74	33
	down	34	38	74	33
	left	34	71	39	33
	right	34	71	39	33
RILU (ij-direction reordering, 25 colours)	up	33	43	79	34
	down	38	49	78	38
	left	38	76	49	38
	right	33	76	43	34

Table 3: Iteration counts with several pressure preconditioners in 2D

Preconditioner type	mesh		
	16x16	32x32	64x64
none	30	59	128
diag	30	59	129
ISNaS MILUD	7	11	16
Multiplicative ILUD	14	24	53

Table 4: Iteration counts for several momentum preconditioners in 2D, with a varying number of grid cells

Preconditioner type	mesh		
	16x16	32x32	64x64
none	54	109	208
diagonal	52	108	227
ISNaS RILU	13	18	26
Neumann series MILUD; one term	42	78	154
Neumann series MILUD; two term	29	44	84
Neumann series MILUD; three term	21	30	46
Multiplicative ILUD	37	75	152
Approximate inverse	16	33	67

Table 5: Iteration counts for several pressure preconditioners in 2D, with a varying number of grid cells

Preconditioner type	mesh				
	8x8x8	10x10x10	12x12x12	16x16x16	32x32x32
ISNaS MILUD	2	3	3	4	8
Multiplicative ILUD	3	4	4	5	12

Table 6: Momentum preconditioners in 3D

Preconditioner type	mesh				
	8x8x8	10x10x10	12x12x12	16x16x16	32x32x32
ISNaS RILUD	15	16	17	21	35
Multiplicative ILUD	23	29	34	45	102
Approximate inverse (19)	15	19	23	30	80
Approximate inverse (125)	9	12	14	18	37

Table 7: Pressure preconditioners in 3D