DELFT UNIVERSITY OF TECHNOLOGY

REPORT 00-01

A PARALLEL BLOCK-PRECONDITIONED GCR METHOD FOR INCOMPRESSIBLE FLOW PROBLEMS

C. VUIK, J. FRANK AND A. SEGAL

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2000

A parallel block-preconditioned GCR method for incompressible flow problems

C. Vuik [*] J. Frank[*] [†] A. Segal[*]

Abstract

Solution of large linear systems encountered in computational fluid dynamics often naturally leads to some form of domain decomposition, especially when it is desired to use parallel machines. It has been proposed to use approximate solvers to obtain fast but rough solutions on the separate subdomains. In this paper approximate solutions via an inner preconditioned GMRES iteration to fixed tolerance and incomplete factorization (RILU, restricted to the diagonal) are considered. Numerical experiments for a Boussinesq flow problem are included which show speedups obtained on a cluster of workstations as well as on a distributed memory parallel computer. Additionally, the parallel implementation of GCR is addressed, with particular focus on communication costs associated with orthogonalization processes. It appears that the reorthogonalized classical Gram-Schmidt process has favorable properties with respect to rounding errors and efficiency.

Keywords: Domain decomposition; approximate subdomain solution; parallel Krylov subspace methods; orthogonalization methods; incompressible Navier-Stokes

1 Introduction

Efficient parallel algorithms are required to simulate incompressible turbulent flows in complex two- and three-dimensional domains. We consider the incompressible Navier-Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} - \frac{1}{R_e}\Delta \mathbf{u} + \mathbf{u}\nabla \cdot \mathbf{u} + \nabla p = \mathbf{f},$$

$$\nabla \cdot \mathbf{u} = 0,$$

[*]Delft University of Technology, Faculty of Information Technology and Systems, Department of Technical Mathematics and Informatics, P.O. Box 5031 2600 GA Delft, The Netherlands, **e-mail:**f**c.vuik j.frank a.segal** g**@math.tudelft.nl**

[†]Center for Mathematics and Computer Science (CWI), P.O. Box 94079, 1090 GB Amsterdam, The Netherlands,

where R_e is the Reynolds number. These equations are discretized on a structured grid. For a flow on a general domain we use an unstructured decomposition of the domain into subdomains of simpler shape, with a structured grid inside each subdomain. We have developed a parallel block-preconditioned GCR method to solve the resulting systems of linear equations.

For the spatial discretization of the Navier-Stokes equations, a finite volume method employing a staggered grid is used. The normal velocities are located at the centers of the faces of the cells and the other scalar unknowns (pressure, temperature etc.) are located in the center of the cells (for details see [22, 35, 3, 34]). The staggered grid is used to avoid pressure oscillations.

For the time discretization, backward Euler is used. With V^n and P^n representing the algebraic vectors of velocity and pressure unknowns at time t^n, respectively, we get

$$\frac{V^{n+1}-V^n}{\Delta t} = M(V^n)V^{n+1} - GP^{n+1}, \tag{1}$$

$$DV^{n+1} = 0, \tag{2}$$

where (1) represents the discretized momentum equation and (2) represents the discretized incompressibility condition. The matrix M is the linearized spatial discretization of the convection and stress in the Navier-Stokes equations, G is the discretized gradient operator, and D is the discretized divergence operator. To solve (1) and (2) with the time-accurate pressure correction method [27], these equations are approximated by:
Prediction

$$\frac{V^*-V^n}{\Delta t} = M(V^n)V^* - GP^n \tag{3}$$

and

$$\frac{V^{n+1}-V^n}{\Delta t} = M(V^n)V^* - GP^{n+1}, \tag{4}$$

$$DV^{n+1} = 0. \tag{5}$$

Subtraction of (3) from (4) gives

$$\frac{V^{n+1}-V^*}{\Delta t} = -G(P^{n+1}-P^n). \tag{6}$$

Taking the discretized divergence of both sides of (6) and using (5) results in the pressure correction equation:
Projection

$$DG\Delta P = \frac{DV^*}{\Delta t}, \tag{7}$$

where $\Delta P = P^{n+1} - P^n$. After the pressure correction ΔP has been computed from (7), it is substituted into (6), which leads to:

Correction

$$V^{n+1} = V^* - \Delta t G \Delta P. \tag{8}$$

In summary, the pressure correction method consists of three steps: (i) computation of V^* from (3), (ii) computation of ΔP from (7) and computation of V^{n+1} from (8). The linear systems are solved by a Krylov subspace method with an ILU [30, 31] or a multigrid [38] preconditioner.

Domain decomposition is used: (1) as a means of dealing with geometric complexity, (2) to deal with problems so large as to exceed workstation memory resources, or (3) as a source of parallelism. Concerning (1) and (2) the speedup obtained by parallelization of the method may be very significant, since the domain decomposition method is used even in the serial computation. On the other hand if exploitation of parallel computing resources is itself the reason for implementing domain decomposition, the results may be less pleasing, see [28]. A good speedup may be achieved by a constant overlap in physical space [7, 24, 36] and/or a coarse grid correction [20].

The domain decomposition method considered in this paper has the following features:

- non-overlapping [10, 6, 37],
- approximate subdomain solution [5, 6],
- accelerated by GCR [11, 26].

Theoretical results on approximate solution of subdomain problems for Schur complement domain decomposition methods are given by Haase, Langer and Meyer [16, 17].

Our method can be smoothly varied between a coarse grain parallel method when the subdomain problems are solved accurately [4], to a fine grain parallel method when only one subdomain iteration is done in every domain decomposition iteration (compare [32]). The other parts of the accelerated domain decomposition algorithm are vector updates, matrix vector products, and inner products. For a matrix vector product only nearest neighbor communications are required which is efficient on most parallel computers. Inner products, on the other hand, require global communications; therefore, the focus has been on reducing the number of inner products [12, 23], overlapping inner product communications with computation [9], or increasing the number of inner products that can be computed with a single communication [2, 21].

In this paper we investigate the speedup for a nonoverlapping, one-level additive Schwarz method with inaccurate subdomain solution applied to flow problems. The implementation of the method is based on MPI subroutines [14]. The details of our domain decomposition algorithm are given in Section 2.1. The GCR method is summarized in Section 2.2, whereas various orthogonalization methods are discussed in Section 2.3. Speedup results are presented in Section 3. The timings were made on a network of workstations (NOW) and a Cray T3E. This research is a continuation of work published in [13].

2 The block-preconditioned GCR method

We start with a detailed description of our iterative domain decomposition method. To accelerate its convergence a Krylov subspace method (GCR) is used. Finally we summarize some results about parallel orthogonalization methods, which are used in GCR.

2.1 The block Gauss-Jacobi preconditioner

The pressure correction algorithm, (3)-(8), is used for the solution of the Navier-Stokes equations on the global domain Ω. Let the domain be the union of N nonoverlapping subdomains Ω_m, $m = 1,\ldots,N$. The equations (3) and (7) are solved using domain decomposition. The correction of V^* is independently carried out in all blocks. In our implementation the normal component of the velocity has two values on each interface. To make its value unique the normal component is sent to neighboring blocks with a higher number and copied. In this paper, we assume that the subdomains intersect regularly, i.e. the grid lines are continuous across block-interfaces. For the description of the domain decomposition algorithm we start from a discretization of the momentum and pressure equations on the global grid.

Both the momentum equation (3) and the pressure equation (7) can be written as

$$Av = f, \tag{9}$$

with either $A = S(V^n, P^n) := \frac{I}{\Delta t} - M(V^n, P^n)$ and $v = V^*$, for the momentum equation or $A = DG$ and $v = \Delta P$, for the pressure equation. If we decompose A into blocks such that each block corresponds to all unknowns in a single subdomain, with a small modification for the momentum equation (see further on), then we get the block system

$$\begin{bmatrix} A_{11} & \ldots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \ldots & A_{NN} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix}. \tag{10}$$

In this system, one observes that the diagonal blocks A_{mm} express coupling among the unknowns defined on a common subdomain (Ω_m), whereas the off-diagonal blocks A_{mn}, $m \neq n$ represent coupling across subdomain boundaries. The only nonzero off-diagonal blocks are those corresponding to neighboring subdomains.

The unaccelerated domain decomposition iteration for Equation (9) is of the following form

$$v^{m+1} = v^m + K^{-1}(f - Av^m). \tag{11}$$

For the block Gauss-Jacobi the method matrix K is defined as

$$K = \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{NN} \end{bmatrix}.$$

When (11) is used, systems of the form $Kv = r$ have to be solved. Since there is no overlap the diagonal blocks $A_{mm}v_m = r_m$, $m = 1,\ldots,N$ can be solved in parallel. In our method these systems are solved by an iterative method. An important point is the required tolerance of these inner iterations (see [5, 13]). Since the number of inner iterations may vary from one subdomain to another, and in each outer iteration, the effective operator $\hat{K}^{-1}$ is nonlinear and varies in each outer iteration.

Our choice of approximate solution methods is motivated by the results obtained in [5]. In that paper, GMRES was used as to approximately solve subdomain problems to within fixed tolerances of 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1}. Additionally, a blockwise application of the RILU(D) preconditioner has been used [31].

We cannot apply the above described block Gauss-Jacobi algorithm directly to the momentum matrix S because the normal velocity components at the block interfaces belong to two blocks. First we augment the matrix S in the following way. For the sake of argument, consider a decomposition into two blocks ($N = 2$). Suppose the velocity unknowns are divided into three sets:

- The first set consists of velocities belonging to Block 1, excluding the normal velocities at the block interface.
- The second set consists of the normal velocities at the interface.
- The third set consists of velocities belonging to Block 2, excluding the normal velocities at the block interface

With respect to these sets of unknowns, the matrix $S(V^n,P^n)$ has the block form

$$S(V^n,P^n) = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix}. \tag{12}$$

The system of equations $S(V^n,P^n)V^* = f$ can be transformed to the equivalent system

$$\bar{S}(V^n,P^n)\bar{V}^* = \begin{bmatrix} S_{11} & S_{12} & 0 & S_{13} \\ S_{21} & S_{22} & 0 & S_{23} \\ S_{21} & 0 & S_{22} & S_{23} \\ S_{31} & 0 & S_{32} & S_{33} \end{bmatrix} \begin{bmatrix} \bar{V}_1^* \\ V_2^* \\ \bar{V}_2'^* \\ V_3^* \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_2 \\ f_3 \end{bmatrix} \tag{13}$$

The solution of Equation (13) always satisfies $\bar{V}_2^* = \bar{V}_2'^*$ if S_{22} is invertible (see [25]) and thus, Equation (13) is equivalent to the original system of equations $S(V^n,P^n)V^* = f$. In view of Equation (10) we have

$$A_{11} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \text{ and } A_{22} = \begin{bmatrix} S_{22} & S_{23} \\ S_{32} & S_{33} \end{bmatrix}, \tag{14}$$

so that the domain decomposition for the momentum equation has been described.

2.2 The GCR method

The block Gauss-Seidel method described in Section 2.1 can be accelerated by a Krylov subspace method. In such a method we choose K^{-1} as preconditioner. Due to our approximate solution of the subdomain problems the effective preconditioner is nonlinear and varies in each outer iteration. GCR [11, 26] is a Krylov method which can be used with a variable preconditioner. The GCR method can be restarted or truncated when the number of outer iterations exceeds the bound n_{trunc}. In practice truncated GCR converges faster than restarted GCR. If the number of outer iterations is less than n_{trunc} an optimized version of the GCR method is used [29].

Algorithm: GCR
Given: initial guess x_0
$r_0 = b - Ax_0$
for $k = 1, \ldots,$ convergence
  Solve $K\tilde{v} = r_{k-1}$ (approximately)
  $\tilde{q} = A\tilde{v}$
  $[q_k, v_k] =$ **orthonorm** $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$
  $\gamma = q_k^T r_{k-1}$
  Update: $x_k = x_{k-1} + \gamma v_k$
  Update: $r_k = r_{k-1} - \gamma q_k$
end

The vectors q_k and v_k are distributed over the processors in the same way as the solution vector x_k. All vectors $q_i, v_i, i \leq k$ are stored in memory. The function **orthonorm()** takes input vectors $\tilde{q}$ and $\tilde{v}$, orthogonalizes $\tilde{q}$ with respect to the q_i, $i < k$, and returns the modified vectors q_k such that $\|q_k\|_2 = 1$. In order to preserve the relation $\tilde{q} = A\tilde{v}$ equivalent operations are done with $\tilde{v}$.

The primary challenges to parallelization of GCR are parallelization of the preconditioning and parallel computation of the inner products. Inner products require global communication and therefore do not scale. Much of the literature on parallel Krylov subspace methods and parallel orthogonalization methods is focused on orthogonalizing a number of vectors simultaneously. However, this is not possible using a preconditioner which varies in each iteration. For this reason, we need a method for orthogonalizing one new vector against an orthonormal basis of vectors.

2.3 Orthogonalization methods

The modified Gram-Schmidt method suffers from the fact that the number of inner products increases proportionally to the iteration number and these inner products must be computed using successive communications. This is not the case if one uses the classical Gram-Schmidt method. In this algorithm all necessary inner products can be computed with a single global communication. Unfortunately, the classical Gram-Schmidt method is unstable with respect to rounding errors, so this method is rarely used. On the other hand, Hoffmann [18] gives experimental evidence indicating that a two-fold application of the classical Gram-Schmidt method is stable.

Another method which has been suggested is the parallel implementation of Householder transformations, introduced by Walker [33]. Below we reformulate this method for GCR (see also [13]).

In the Householder orthogonalization we use the notion a_k to represent the kth column of a matrix A and $a^{(i)}$ to represent the ith component of a vector a. Let a matrix $A \in \mathbb{R}^{n\times m}$, $m \leq n$ with linearly independent columns be factored as QZ, where Q is orthogonal and Z is upper triangular. Then the kth column of A is given by $a_k = Qz_k$ and the columns of Q form an orthonormal basis for the span of the columns of A.

We construct Q as the product of a series of Householder reflections, $Q = P_1 \cdots P_m$, used to transform A into Z. The matrices $P_i = I - 2\frac{w_i w_i^T}{w_i^T w_i}$, with $w_i^{(j)} = 0$ for $j < i$ have the property: $P_i(P_{i-1}\cdots P_1)a_i = z_i$.

Suppose one has already produced k orthogonal basis vectors. To compute w_{k+1} one must first apply the previous reflections to a_{k+1} as described in [33]: $\tilde{a} = P_k \cdots P_1 a_{k+1} = (I - 2W_k L_k^{-1} W_k^T)a_{k+1}$, where W_k is the matrix whose columns are $w_1, \ldots, w_k$, and where

$$L_k = \begin{bmatrix} 1 & & & \\ 2w_2^T w_1 & 1 & & \\ \vdots & & \ddots & \\ 2w_k^T w_1 & \ldots & 2w_k^T w_{k-1} & 1 \end{bmatrix}.$$

Note especially that in the $(k+1)$th iteration one must compute the last row of L_k, which is the vector $(2w_k^T W_{k-1}, 1)$, as well as the vector $W_k^T a_{k+1}$. This requires $2k-1$ inner products, but they may all be computed using only a single global communication.

Let $\hat{a}$ be the vector obtained by setting the first k elements of $\tilde{a}$ to zero. The vector w_{k+1} is chosen as: $w_{k+1} = \hat{a} + \text{sign}(\hat{a}^{(k+1)})\|\hat{a}\|_2 e_{k+1}$. In practice, the vectors w_k are normalized to length one. The length of w_{k+1} can be expressed as $\|w_{k+1}\|_2 = \sqrt{2\alpha^2 - 2\alpha\hat{a}^{(k+1)}}$ where $\alpha = \text{sign}(\hat{a}^{(k+1)})\|\hat{a}\|_2$. The $(k+1)$th column of Q is the new orthonormal basis vector:

$$q_{k+1} = \frac{1}{\alpha}\left[a_{k+1} - \sum_{i=1}^{k} \tilde{a}^{(i)} q_i\right].$$

Within the GCR algorithm, the same linear combination must be applied to the v_i to obtain v_{k+1}.

In Table 1 we summarize the round-off properties and the amount of work and communication for the following orthogonalization methods (for details see [13]):

- Classical Gram-Schmidt (CGS)
- Reorthogonalized Classical Gram-Schmidt (RCGS)
- Modified Gram-Schmidt (MGS)

- Householder (HH)

Comparing the costs we expect that the wall-clock time for RCGS and HH are comparable. When communication is slow (large latency) with respect to computation one expects that these

	round-off	daxpy	ddot	communications
CGS	bad	2k	k	1
MGS	good	2k	k	k
RCGS	good	3k	2k	2
HH	good	3k	2k	3

Table 1: Properties of the various orthogonalization methods

methods are faster than MGS, with of course a preference for RCGS. Otherwise MGS may be the fastest method because MGS needs fewer floating point operations.

3 Numerical experiments

In this section we illustrate the parallel performance of the block Gauss-Jacobi preconditioned GCR method when implemented within the Navier-Stokes software DeFT [34]. Numerical experiments were performed on a network of workstations (NOW) consisting of Hewlett-Packard 700-series machines connected by a 10 Mb Ethernet and on a Cray T3E.

The test problem considered was a two-dimensional Boussinesq flow [8] on $(0,1)\times(0,1)$. The governing equations are given by

$$\frac{\partial \mathbf{u}}{\partial t} - \frac{1}{R_e}\Delta \mathbf{u} + \mathbf{u}\nabla\cdot\mathbf{u} + \nabla p = \mathbf{g}\frac{G_r}{R_e^2}T, \tag{15}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{16}$$

$$\frac{dT}{dt} - \frac{1}{R_e P_r}\Delta T + \mathbf{u}\cdot\nabla T = 0, \tag{17}$$

with $\mathbf{g} = (0,-1)$ and boundary conditions $\mathbf{u}(0,y) = \mathbf{u}(1,y) = \mathbf{u}(x,0) = \mathbf{u}(x,1) = \mathbf{0}$, $T(0,y) = 1$, $T(1,y) = 0$, and $\partial T/\partial y(x,0) = \partial T/\partial y(x,1) = 0$. The Reynolds, Prandtl and Grashof numbers were taken to be $R_e = 1$, $P_r = 0.71$ (air) and $G_r = 1500$, respectively. The simulation was carried out for 10 timesteps of size $\Delta t = 0.05$ to diffuse the influence of start-up latencies. It is known that due to the temperature difference a circulating flow arises, with the number of vortices depending on the Grashof number. This recirculation makes the momentum equation (15) and heat transport equation (17) relatively difficult to solve. Since the domain is rectangular, it is easily decomposed into various block configurations.

In the inner iteration process, blocks were solved to varying accuracies using GMRES with a restart of 40, preconditioned with the relaxed incomplete factorization, RILU(α), of [1] using a relaxation parameter $\alpha = 0.975$ for the pressure correction equation (7) and $\alpha = 1$ for the momentum and transport equations. In the extreme case, we perform no GMRES iterations and use only the RILU preconditioner on the blocks. For the outer iterations, GCR was used with a Krylov subspace of dimension 25 and employing the Jackson and Robinson truncation strategy [19, 29].

The timings listed in this section are wall-clock times obtained with MPI timing routines, and indicate the time spent in the linear solver part of the code. In particular, they do not include time required to construct the matrices.

In all of our tests, we observed very similar behavior for the transport equation (17) as for the pressure equation (7), so we will neglect the discussion of the transport equation in the ensuing.

3.1 Comparison with diagonal scaling

As a basis for comparison of the effectiveness of the block preconditioner, we ran a few tests using a simple diagonal scaling (Gauss-Jacobi) preconditioner. This preconditioner is very popular in a parallel computing environment. Table 2 gives wall-clock times and iteration counts using both preconditioners. The table indicates that the number of iterations required for convergence with diagonal preconditioning is quite large and increases drastically as the grid is refined. Furthermore the block Gauss-Jacobi preconditioner needs much less wall-clock time than the Gauss-Jacobi preconditioner.

		Gauss-Jacobi		block Gauss-Jacobi	
blocks	subgrid	Momentum	Pressure	Momentum	Pressure
2×2	24×24	13.8 (119)	9.0 (144)	4.8 (39)	2.6 (38)
	60×60	159 (301)	101 (390)	62.6 (91)	21.2 (69)
3×3	24×24	25.2 (180)	19.7 (226)	8.7 (60)	6.1 (64)

Table 2: Wall-clock time and iteration counts given in parentheses for a Gauss-Jacobi and block Gauss-Jacobi preconditioning

3.2 Comparison with serial block-preconditioner

To measure the cost of parallelization, we compare the parallel and sequential computation times using the block Gauss-Jacobi preconditioners. Tables 3 and 4 give the speedup factors on a Cray T3E for the momentum and pressure equations, respectively, using the approximate solvers or the RILU preconditioner on the blocks. The subdomain approximations will be denoted as follows:

- GMR6 = restarted GMRES with a tolerance of 10^{-6},
- GMR2 = restarted GMRES with a tolerance of 10^{-2},

- GMR1 = restarted GMRES with a tolerance of 10^{-1},
- RILU = one application of an RILU preconditioner.

The trends are as expected: when the blocks are solved very accurately, the relative cost of communication to computation is low, giving a high speedup in parallel; whereas for the less accurate approximations, the communications are relatively more expensive, and a lower speedup is observed. In general, the parallel efficiency is quite high for a small number of blocks but decreases as the number of blocks is increased. The speedups are higher for the momentum equation than for the pressure equation.

blocks	GMR6	GMR2	GMR1	RILU(1)
4	3.7	3.6	3.6	3.4
9	8.1	7.5	7.4	7.0
16	14.0	12.9	12.6	12.2
25	18.4	17.1	16.6	16.4

Table 3: Attained speedups over sequential implementation (Momentum equation, 24×24 subgrid resolution)

blocks	GMR6	GMR2	GMR1	RILU(0.95)
4	3.2	3.0	2.9	2.6
9	6.4	5.9	5.4	5.0
16	10.5	9.3	9.2	9.3
25	14.2	9.9	9.6	12.1

Table 4: Attained speedups over sequential implementation (Pressure equation, 24×24 subgrid resolution)

3.3 Scalability comparison

3.3.1 Fixed problem size

In this section we compare the parallel computation times for a fixed problem size on a 120×120 grid. The grid is decomposed into 2×2, 3×3, 4×4 and 5×5 subdomains. Tables 5 and 6 give timing results for the momentum and pressure equations. The number of outer iterations required in the final timestep is given in parentheses.

The single block solution times are listed in each table for reference. The number of necessary outer iterations increases severely in the multiblock case as compared to the single block case of only one iteration. This initial loss of convergence rate can only be offset in the case of the momentum equation by using very rough approximations on the blocks and many processors. For the pressure equation, some speedup can already be obtained with only 4 blocks.

	Single block solution time = 21.4 (1)			
blocks	GMR6	GMR2	GMR1	RILU(1)
2×2	200. (36)	72.7 (38)	56.1 (58)	62.1 (78)
3×3	74.4 (50)	34.9 (52)	28.7 (62)	30.8 (77)
4×4	42.4 (56)	22.9 (39)	20.2 (69)	19.3 (80)
5×5	32.4 (51)	18.3 (51)	16.7 (54)	17.0 (97)

Table 5: Scalability study on 120×120 grid (Momentum equation)

	Single block solution time = 30.8 (1)			
blocks	GMR6	GMR2	GMR1	RILU(0.95)
2×2	71.5 (29)	40.4 (29)	44.1 (30)	20.5 (66)
3×3	35.9 (33)	22.8 (33)	23.7 (34)	15.8 (87)
4×4	22.9 (39)	21.3 (65)	17.5 (40)	13.0 (103)
5×5	19.5 (62)	19.0 (76)	21.3 (91)	14.5 (116)

Table 6: Scalability study on 120×120 grid (Pressure equation)

3.3.2 Fixed subdomain size

It is often argued that a better measure of the effectiveness of a parallel algorithm is obtained by fixing the per-processor problem size while increasing the number of processors [15]. In this section we therefore fix the subdomain grid at 24×24, and the domain decomposition is increased from a single block to a 5×5 block decomposition. In tables 7 and 8 we list the wall-clock times for the momentum and pressure equations, respectively. For perfect scaling, the wall-clock time would be constant, independent of the number of blocks. Given in parentheses are the number of outer iterations required in the final time step. For a fixed block size we observe for the momentum equation that the computation time scales roughly as the square root of the number of blocks. For the pressure equation the scaling is somewhat poorer, especially for the 5×5 block decomposition. For both equations there is a large increase in the number of outer iterations.

blocks	GMR6	GMR2	GMR1	RILU(1)
2×2	10.5 (22)	5.7 (22)	4.9 (24)	4.7 (36)
3×3	16.4 (39)	9.3 (39)	8.3 (43)	8.5 (50)
4×4	22.3 (50)	12.7 (51)	11.6 (57)	11.6 (66)
5×5	32.4 (51)	18.3 (51)	16.7 (54)	17.0 (97)

Table 7: Scalability study with fixed block size (Momentum equation)

3.4 Orthogonalization methods

In this section we compare parallel performances of the modified Gram-Schmidt (MGS), Householder (HH), and reorthogonalized classical Gram-Schmidt (RCGS) processes on a NOW and on

blocks	GMR6	GMR2	GMR1	RILU(0.95)
2×2	4.2 (20)	2.7 (23)	2.8 (20)	2.6 (37)
3×3	8.6 (27)	6.3 (27)	6.5 (28)	6.0 (62)
4×4	12.8 (36)	10.3 (36)	10.2 (37)	8.9 (90)
5×5	19.5 (62)	19.0 (76)	21.3 (91)	14.5 (116)

Table 8: Scalability study with fixed block size (Pressure equation)

a Cray T3E. First we compare these methods for an artificial test problem. Thereafter we make a comparison for the Boussinesq problem.

In our first experiment the wall-clock times in the orthogonalization part are measured when 60 GCR iterations are performed. In Figure 1 the parameters

$$\mathcal{F}_{\mathrm{HH}} = \frac{\text{orthog. time MGS}}{\text{orthog. time HH}} \text{ and } \mathcal{F}_{\mathrm{RCGS}} = \frac{\text{orthog. time MGS}}{\text{orthog. time RCGS}}$$

are plotted as functions of n. In each subdomain an $n \times n$ grid is used. The number of subdomains is equal to the number of processors. On the workstation cluster (HH) and (RCGS) are only advantageous when the number of unknowns is less than 3600 on 4 processors and less than 6400 on 9 processors. On the Cray T3E, the number of unknowns per processor should be fewer than 1000 for 9 or even 25 processors. For larger problems the smaller amount of work involved in modified Gram-Schmidt orthogonalization outweighs the increased communication cost. Furthermore we observe that RCGS is somewhat more efficient than HH. Therefore we have not implemented the Householder orthogonalization in our Navier-Stokes solver.

Finally we report the total wall-clock time spent solving the linear systems originating from the two-dimensional Boussinesq flow problem. We consider the case for which orthogonalization is most likely to be a factor, i.e. relatively small blocks approximated by the RILU preconditioner. Since the approximate block solver is cheaper in this case, the communication costs weigh more heavily.

Table 9 compares times obtained on the Cray T3E. We see that the orthogonalization time is really negligible on the Cray, so that neither of the strategies (MGS/RCGS) provides a significant advantage.

Table 10 presents analogous results on the NOW. For the 4-block decomposition, the workstations were directly connected by Ethernet, whereas for the 9-block decomposition, the workstations were located at different points on the local network, such that some messages had to pass through routers. Due to the relatively low communication bandwidth of the Ethernet, the inner product communications become an expensive part of the computation, and a good speedup can be achieved by using reorthogonalized classical Gram-Schmidt.

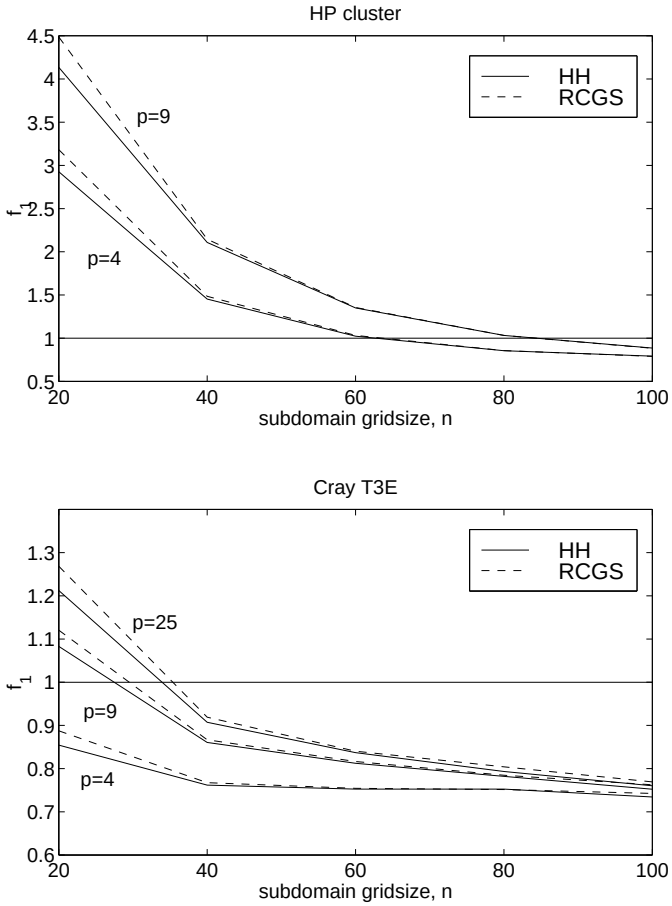

Figure 1: Measured speedup with Householder (HH) orthogonalization and reorthogonalized classical Gram-Schmidt (RCGS) with respect to modified Gram-Schmidt (MGS)

4 Conclusions

In this paper we have presented parallel performance results for a block Gauss-Seidel preconditioned GCR method for the Navier-Stokes equations. We summarize these results in the following remarks:

- The block Gauss-Seidel preconditioner is perfectly parallel and gives better performance than a simple diagonal scaling which is also perfectly parallel.
- The convergence rate degrades substantially as the number of blocks is increased from 1, but less appreciably thereafter. Current research into using overlap or multilevel techniques promises to improve this behavior.
- It is sometimes advantageous to use the reorthogonalized classical Gram-Schmidt process on workstation clusters, particularly when the number of workstations is large and the network is slow.

	MGS		RCGS	
blocks	Momentum	Pressure	Momentum	Pressure
2×2	4.7	2.6	4.7	2.6
3×3	8.5	6.0	8.5	5.5
4×4	11.6	8.9	11.7	8.4

Table 9: Comparison of computation times on a Cray T3E using MGS and RCGS orthogonalization processes (24×24 subgrid resolution, RILU subdomain approximation)

	MGS		RCGS	
blocks	Momentum	Pressure	Momentum	Pressure
2×2	38.6	36.5	23.3	17.0
3×3	563	799	164	236

Table 10: Comparison of computation times on a NOW using MGS and RCGS orthogonalization processes (24×24 subgrid resolution, RILU subdomain approximation)

- Parallelization of a multi-block problem leads to good speedups, however using domain decomposition only for exploiting a parallel machine leads to a modest decrease of wall-clock time.

Acknowledgment

The authors thank HPaC for providing computing facilities on the Cray T3E.

References

[1] O. Axelsson and G. Lindskog. On the rate of convergence of the preconditioned gradient method. *Num. Math.*, 48:499–523, 1986.

[2] Z. Bai, D. Hu, and L. Reichel. A Newton-basis GMRES implementation. *IMA J. Num. Anal.*, 14:563–581, 1994.

[3] H. Bijl and P. Wesseling. A unified method for computing incompressible and compressible flows in boundary-fitted coordinates. *J. Comp. Phys.*, 141:153–173, 1998.

[4] 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.

[5] E. Brakkee, C. Vuik, and P. Wesseling. Domain decomposition for the incompressible Navier-Stokes equations: solving subdomain problems accurately and inaccurately. *Int. J. for Num. Meth. Fluids*, 26:1217–1237, 1998.

[6] J.H. Bramble, J.E. Pasciak, and A.T. Vassilev. Analysis of non-overlapping domain decomposition algorithms with inexact solves. *Math. Comp.*, 67:1–19, 1998.

[7] X. Cai, W.D. Gropp, and D.E. Keyes. A comparison of some domain decomposition and ILU preconditioned iterative methods for non-symmetric elliptic problems. *Num. Lin. Alg. Appl.*, 1:477–504, 1994.

[8] 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.

[9] E. de Sturler and H. A. van der Vorst. Reducing the effect of global communication in GMRES(*m*) and CG on parallel distributed memory computers. *Appl. Num. Math.*, 18:441–459, 1995.

[10] M. Dryja and O.B. Widlund. Domain decomposition algorithms with small overlap. *SIAM J. Sci. Comp.*, 15:604–620, 1994.

[11] S.C. Eisenstat, H.C. Elman, and M.H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J. Num. Anal.*, 20:345–357, 1983.

[12] J. Erhel. A parallel GMRES version for general sparse matrices. *Electronic Transactions on Numerical Analysis (*`http://etna.mcs.kent.edu`*)*, 3:160–176, 1995.

[13] J. Frank and C. Vuik. Parallel implementation of a multiblock method with approximate subdomain solution. *Appl. Num. Math.*, 30:403–423, 1999.

[14] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI, portable programming with the Message-Passing Interface*. Scientific and Engineering Computation Series. The MIT Press, Cambridge, 1994.

[15] J.L. Gustafson. Reevaluating Amdahl's law. *Comm. ACM*, 31:532–533, 1988.

[16] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet domain decomposition method. Part I: An algebraic approach. *Computing*, 47:137–151, 1991.

[17] G. Haase, U. Langer, and A. Meyer. The approximate Dirichlet domain decomposition method. Part II: Applications to 2nd-order elliptic B.V.P.s. *Computing*, 47:153–167, 1991.

[18] Walter Hoffman. Iterative algorithms for Gram-Schmidt orthogonalization. *Computing*, 41:335–348, 1989.

[19] J.P. Jackson and P.C. Robinson. A numerical study of various algorithms related to the preconditioned conjugate gradient method. *Int. Num. Meth. Engng*, 21:1315–1338, 1985.

[20] C.B. Jenssen and P.A. Weinerfelt. Coarse grid correction scheme for implicit multiblock Euler calculations. *AIAA Journal*, 33:1816–1821, 1995.

[21] G. Li. A block variant of the GMRES method on massively parallel processors. *Parallel Computing*, 23:1005–1019, 1997.

[22] A. Segal, P. Wesseling, J. Kan, C.W. Oosterlee, and K. Kassels. Invariant discretization of the incompressible nawier-stokes equations in boundary fitted co-ordinates. *Int. J. for Num. Meth. Fluids*, 15:411–426, 1992.

[23] R. B. Sidje. Alternatives for parallel Krylov subspace basis computation. *Num. Lin. Alg. Appl.*, 4(4):305–331, 1997.

[24] J.C. Strikwerda and C.D. Scarbnick. A domain decomposition method for incompressible flow. *SIAM J. Sci. Comput.*, 14:49–67, 1993.

[25] W. Tang. Generalized Schwarz splittings. *SIAM J. Sci. Stat. Comput*, 13:573–595, 1992.

[26] H.A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Num. Lin. Alg. Appl.*, 1:369–386, 1994.

[27] J. van Kan. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM J. Sci. Stat. Comput.*, 7:870–891, 1986.

[28] Eric F. van de Velde. Domain decomposition vs. concurrent multigrid. Technical Report Caltech 217-50, Center for Research on Parallel Computation, California Institute of Technology, 1994.

[29] C. Vuik. Further experiences with GMRESR. *Supercomputer*, 55:13–27, 1993.

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

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

[32] C. Vuik, R.R.P. van Nooyen, and P. Wesseling. Parallelism in ILU-preconditioned GMRES. *Parallel Computing*, 24:1927–1946, 1998.

[33] Homer F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.

[34] P. Wesseling, A. Segal, and C.G.M. Kassels. Computing flows on general tree-dimensional nonsmooth staggered grids. *J. Comp. Phys.*, 149:333–362, 1999.

[35] P. Wesseling, A. Segal, C.G.M. Kassels, and H. Bijl. Computing flows on general two-dimensional nonsmooth staggered grids. *J. Eng. Math.*, 34:21–44, 1998.

[36] J.A. Wright and W. Shyy. A pressure-based composite grid method for the Navier-Stokes equations. *J. Comput. Phys.*, 107:225–238, 1993.

[37] J. Xu and J. Zou. Some nonoverlapping domain decomposition methods. *SIAM Review*, 40:857–914, 1998.

[38] S. Zeng, C. Vuik, and P. Wesseling. Numerical solution of the incompressible Navier-Stokes equations by Krylov subspace and multigrid methods. *Adv. Comp. Math.*, 4:27–49, 1995.