ELSEVIER

Future Generation Computer Systems 18 (2001) 31–40

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/future

A parallel block-preconditioned GCR method for incompressible flow problems

C. Vuik*, J. Frank[1], A. Segal

Faculty of Information Technology and Systems, Department of Applied Mathematical Analysis, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, Netherlands

Abstract

Efficient parallel algorithms are required to simulate incompressible turbulent flows in complex two- and three-dimensional domains. The incompressible Navier–Stokes equations are discretized in general coordinates on a structured grid. For a flow on a general domain we use an unstructured decomposition of the domain into subdomains of simple 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. The method can be smoothly varied between a coarse grain parallel method in which the subdomain problems are solved accurately with an inner iteration process and a fine grain parallel method when only a preconditioner is used to approximate the solution on the blocks. Parallel performance results for Boussinesq flow in a cavity are included.

Keywords: Domain decomposition; Approximate subdomain solution; Parallel Krylov subspace methods; Orthogonalization methods; Incompressible Navier–Stokes equations

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:

$$\frac{\partial \mathbf{u}}{\partial t} - \frac{1}{Re}\Delta \mathbf{u} + \mathbf{u}\nabla \cdot \mathbf{u} + \nabla p = \mathbf{f}, \quad \nabla \cdot \mathbf{u} = 0,$$

where Re is the Reynolds number. These equations are discretized in general coordinates using a staggered finite volume method on a structured grid (see [3,17,27,28]). For a flow on a general domain we use an unstructured decomposition of the domain into subdomains of simple 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.

Let V^n and P^n represent the algebraic vectors containing velocity and pressure unknowns at time t^n, respectively. Application of the backward Euler method in time yields

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

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

where (1) represents the discretized momentum equation and (2) the discretized incompressibility condition. The matrix F is the linearized spatial discretization of the convection and stress in the Navier–Stokes equations, G the discretized gradient operator, and D

* Corresponding author.
E-mail address: c.vuik@math.tudelft.nl (C. Vuik).
[1] Present address: CWI, P.O. Box 94079, 1090 GB Amsterdam, Netherlands.

0167-739X/01/$ – see front matter
PII: S0167-739X(00)00073-X

the discretized divergence operator. To solve (1) and (2) with the time-accurate pressure correction method [21], these equations are approximated by

$$(Prediction) \quad \frac{V^* - V^n}{\Delta t} = F(V^n)V^* - GP^n, \tag{3}$$

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

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

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:

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

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:

$$(Correction) \quad V^{n+1} = V^* - \Delta t\, G\Delta P. \tag{8}$$

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 [23,24] or a multigrid [30] preconditioner.

The linear systems (3) and (7) are solved with GCR [10,20] using a block-diagonal preconditioner based on a non-overlapping domain decomposition [6,9,29]. This allows us to handle more general domains with a structured discretization, by decomposing the domain into regions which are topologically similar to a square or cube, within each of which the grid is structured. The block-diagonal structure additionally facilitates parallelization, allowing us to handle very large domains in which memory limitations come into play.

Additionally, the independent blocks of the preconditioner can be solved as precisely as desired using an inner iteration procedure [5,6]. Thus, 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 [25]).

Efficient parallel implementation of GCR method requires, in addition to the preconditioner, a proper handling of the matrix vector multiplication 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 [11,18], overlapping inner product communications with computation [8], or increasing the number of inner products that can be computed with a single communication [2,16].

The details of our domain decomposition algorithm are given in Section 2.1. Our motivation for using GCR as the acceleration method is given in Section 2.2. Various orthogonalization methods, which can be used in GCR are discussed in Section 2.3. Speedup results are presented in Section 3.

2. The block-preconditioned GCR method

We begin by describing in detail our block-diagonal preconditioner, the Krylov subspace method used to accelerate the iterations, and its parallelization aspects, especially the question of an orthogonalization procedure.

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 M non-overlapping subdomains Ω_m, $m = 1, \ldots, M$. Eqs. (3) and (7) are solved using domain decomposition. We require that the subdomains intersect regularly, i.e. the grid lines are continuous across block interfaces. The correction of V^* in Eq. (8) is independently carried out in all blocks.

When discretized on the global grid, both the momentum equation (3) and the pressure equation (7) can be written as a linear system

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

with either $A = S(V^n) := (1/\Delta t)I - F(V^n)$ and $v = V^*$ for the momentum equation or $A = DG$

and $v = \Delta P$ for the pressure correction 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

$$\begin{bmatrix} A_{11} & \cdots & A_{1M} \\ \vdots & \ddots & \vdots \\ A_{M1} & \cdots & A_{MM} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_M \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_M \end{pmatrix}. \tag{10}$$

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 Eq. (9) is analogous to a classical iteration of the form:

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

with the block Gauss–Jacobi method matrix K 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 approach these systems are solved by an iterative method. An important point is the required tolerance of these inner iterations (see [5,12]). Since the number of inner iterations may vary from one subdomain to another, and in each outer iteration, the effective operator $\hat{K}^{-1} \approx K^{-1}$ is nonlinear and varies in each outer iteration.

Our choice of approximate solution methods is motivated by the results obtained in [5,12], where GMRES was used 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 [24].

We cannot apply the above described block Gauss–Jacobi algorithm directly to the momentum matrix S because the normal velocity components on the block interfaces belong to two blocks. Instead, we first augment the matrix S in the following way. It is sufficient to consider a decomposition into two blocks ($N = 2$). Let the velocity unknowns be divided into three sets:

1. the set consisting of velocities belonging to block 1, excluding the normal velocities at the block interface,
2. the set consisting of the normal velocities at the interface,
3. the set consisting of velocities belonging to block 2, excluding the normal velocities at the block interface.

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

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

The system of equations $S(V^n)V^* = f$ can be augmented, doubling the interface unknowns to arrive at

$$\bar{S}(V^n)\bar{V}^* = \begin{bmatrix} S_{11} & S_{12} & 0 & S_{13} \\ S_{21} & S_{22} & 0 & S_{23} \\ S_{21} & 0 & S_{22} & S_{23} \\ S_{31} & 0 & S_{32} & S_{33} \end{bmatrix} \begin{bmatrix} \bar{V}_1^* \\ \bar{V}_2^* \\ \bar{V}_2'^* \\ \bar{V}_3^* \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_2 \\ f_3 \end{bmatrix}. \tag{13}$$

The solution of Eq. (13) satisfies $\bar{V}_2^* = \bar{V}_2'^*$ whenever S_{22} is invertible (see [19]) and in this case, Eq. (13) is equivalent to the original system of equations $S(V^n)V^* = f$. In view of Eq. (10), we have

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

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

2.2. The GCR method

The block Gauss–Jacobi iteration (11) described in Section 2.1 can be accelerated by a Krylov subspace method. To do so, we choose K^{-1} as the preconditioner. Due to the approximate solution of the subdomain problems the effective preconditioner is nonlinear and varies in each outer iteration. We choose the GCR method [10,20] because it can be used with a variable preconditioner, and furthermore is quite flexible for use with restart/truncation strategies when the number of iterations exceeds the prescribed limit 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 [22].

The preconditioned GCR method is given by the following algorithm.

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)
$\tilde{q} = A\tilde{v}$
$[q_k, v_k] = \textbf{orthonorm}(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$
$\gamma = q_k^{\mathrm{T}} r_{k-1}$
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 $\tilde{v}$.

Apart from the preconditioner, the main challenges to parallelization of GCR is parallel computation of the inner products, which require global communication and therefore do not scale. Much of the literature on parallel Krylov subspace methods and parallel orthogonalization methods has focused on performing several iterations before orthogonalizing, so that a number of vectors can be orthogonalized simultaneously with a single communication. However, this is not possible using a preconditioner which varies in each iteration. For this reason, we need a method for orthogonalizing a new vector against an orthonormal basis of vectors.

2.3. Orthogonalization methods

A disadvantage of the modified Gram–Schmidt method in parallel is 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 [14] 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 [26]. Below we reformulate this method for GCR (see also [12]).

In the Householder orthogonalization we use the notation 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 $P_i = I - 2(w_i w_i^{\mathrm{T}}/w_i^{\mathrm{T}} w_i)$, 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 orthonormal basis vectors. To compute w_{k+1} one must first apply the previous reflections to a_{k+1} as described in [26]: $\tilde{a} = P_k \cdots P_1 a_{k+1} = (I - 2W_k L_k^{-1} W_k^{\mathrm{T}})a_{k+1}$, where W_k is the matrix whose columns are $w_1, \ldots, w_k$, and where

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

Note especially that in the $(k+1)$th iteration one must compute the last row of L_k, which is the vector $(2w_k^{\mathrm{T}}W_{k-1}, 1)$, as well as the vector $W_k^{\mathrm{T}}a_{k+1}$. 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: $w_{k+1} = \hat{a} + \mathrm{sign}(\hat{a}^{(k+1)})\|\hat{a}\|_2 e_{k+1}$. In practice, the vectors w_k are normalized to length one. The length of w_{k+1} can be expressed as $\|w_{k+1}\|_2 = (2\alpha^2 - 2\alpha\hat{a}^{(k+1)})^{1/2}$, where $\alpha = \mathrm{sign}(\hat{a}^{(k+1)})\|\hat{a}\|_2$. 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 linear combination with the same coefficients 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 [12]):

- 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) compared to computation, one expects that these methods are faster than MGS, with of course a preference for RCGS. Otherwise, when computational costs dominate, MGS is the fastest method because it requires fewer floating point operations.

Table 1
Properties of the various orthogonalization methods

	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

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 [27]. 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 [7] on $(0,1)\times(0,1)$. The governing equations are given by

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

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

$$\frac{\mathrm{d}T}{\mathrm{d}t} - \frac{1}{Re\,Pr}\Delta^2 T + \mathbf{u}\cdot\nabla T = 0 \tag{17}$$

with $\mathbf{g} = (0,-1)$ and boundary conditions $\mathbf{u}(0,y) = \mathbf{u}(1,y) = \mathbf{u}(x,0) = \mathbf{u}(x,1) = 0, T(0,y) = 1, T(1,y) = 0$, and $\partial T/\partial y(x,0) = \partial T/\partial y(x,1) = 0$. The Reynolds, Prandtl and Grashof numbers were taken to be $Re = 1, Pr = 0.71$ (air) and $Gr = 1500$, 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 various 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 [15,22].

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

Table 2
Wall-clock time and iteration counts given in parentheses for a Gauss–Jacobi and block Gauss–Jacobi preconditioning (RILU)

Blocks	Subgrid	Gauss–Jacobi		Block Gauss–Jacobi	
		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)

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 following sections.

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.

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:

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

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 4
Attained speedups over sequential implementation (pressure 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

- 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.

For the pressure equation the speedups of GMR1 and GMR2 are less than expected when 25 blocks are used. The reason for this is a deterioration of the load balancing for GMR1 and GMR2 when the number of blocks increases. When four blocks are used the number of inner iterations is more or less the same for all blocks. However, when GMR2 is used on 25 blocks the ratio between the maximum and minimum number of iterations per block is 1.35. RILU has a perfect load balance because on each block the amount of work is the same.

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. The discretized Navier–Stokes equations

Table 5
Scalability study on 120 × 120 grid (momentum equation)

Blocks	Single block solution time: 21.4 (1)			
	GMR6	GMR2	GMR1	RILU(1)
2 × 2	204 (30)	74.4 (31)	56.5 (34)	62.9 (91)
3 × 3	73.8 (42)	34.4 (42)	28.2 (44)	31.4 (98)
4 × 4	42.9 (46)	22.7 (46)	19.4 (54)	20.1 (96)
5 × 5	30.4 (51)	17.9 (51)	17.7 (54)	17.3 (97)

are solved for a number of timesteps. Tables 5 and 6 give cumulative 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 drastically 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 four blocks.

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 [13]. 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.

Table 6
Scalability study on 120 × 120 grid (pressure equation)

Blocks	Single block solution time: 30.8 (1)			
	GMR6	GMR2	GMR1	RILU(0.95)
2 × 2	68.7 (31)	39.1 (39)	44.8 (55)	21.3 (67)
3 × 3	36.2 (46)	24.0 (58)	24.9 (70)	17.2 (95)
4 × 4	23.8 (55)	17.5 (67)	16.7 (76)	13.7 (108)
5 × 5	19.5 (63)	19.9 (78)	17.5 (87)	15.5 (119)

Table 7
Scalability study with fixed block size (momentum equation)

Blocks	GMR6	GMR2	GMR1	RILU(1)
2 × 2	10.6 (20)	5.8 (20)	5.0 (21)	4.8 (39)
3 × 3	16.3 (32)	9.3 (33)	8.3 (34)	8.8 (60)
4 × 4	23.1 (41)	12.8 (41)	11.5 (43)	11.8 (77)
5 × 5	30.4 (51)	17.9 (51)	17.7 (54)	17.3 (97)

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 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 Fig. 1 the parameters

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

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

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 four processors and less than 6400 on nine processors.

Table 8
Scalability study with fixed block size (pressure equation)

Blocks	GMR6	GMR2	GMR1	RILU(0.95)
2 × 2	4.0 (20)	2.7 (24)	2.8 (29)	2.7 (38)
3 × 3	8.7 (35)	6.6 (42)	6.5 (46)	6.4 (64)
4 × 4	13.2 (49)	10.3 (58)	10.5 (64)	9.4 (92)
5 × 5	19.5 (63)	19.9 (78)	17.5 (87)	15.5 (119)

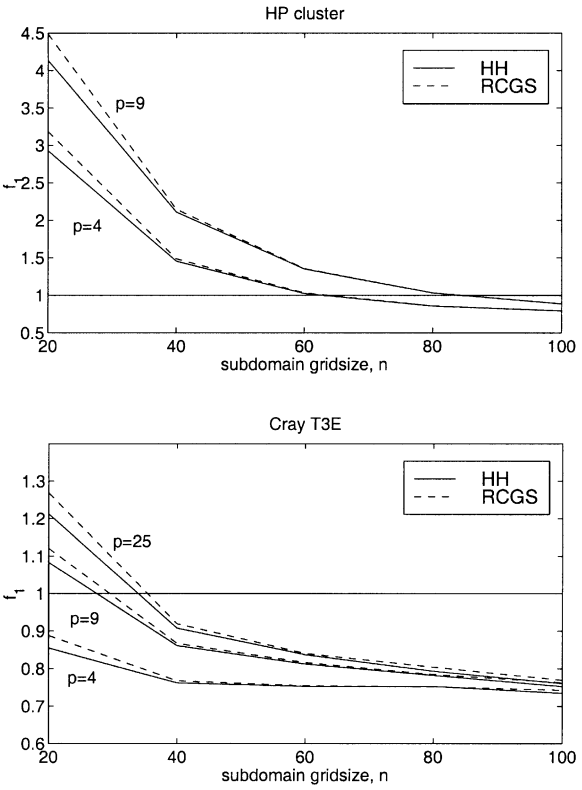

Fig. 1. Measured speedup with Householder (HH) orthogonalization and reorthogonalized classical Gram–Schmidt (RCGS) with respect to modified Gram–Schmidt (MGS).

On the Cray T3E, the number of unknowns per processor should be fewer than 1000 for nine 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
Comparison of computation times on a Cray T3E using MGS and RCGS orthogonalization processes (24×24 subgrid resolution, RILU subdomain approximation)

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

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

Table 9 compares times obtained on a Cray T3E. We see that the orthogonalization time is actually 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 four-block decomposition, the workstations were directly connected by ethernet, whereas for the nine-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.

4. Conclusions

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

- The block Gauss–Jacobi 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 one, 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.
- Parallelization of a multi-block problem leads to good speedups; however using this kind of domain decomposition simply for exploiting a parallel machine leads to only a modest decrease of wall-clock time.

Acknowledgements

The authors thank HPaC for providing computing facilities on the Cray T3E and Kees Dekker for carefully reading the manuscript.

References

[1] O. Axelsson, G. Lindskog, On the rate of convergence of the preconditioned gradient method, Numer. Math. 48 (1986) 499–523.
[2] Z. Bai, D. Hu, L. Reichel, A Newton-basis GMRES implementation, IMA J. Numer. Anal. 14 (1994) 563–581.
[3] H. Bijl, P. Wesseling, A unified method for computing incompressible and compressible flows in boundary-fitted coordinates, J. Comput. Phys. 141 (1998) 153–173.
[4] E. Brakkee, A. Segal, C.G.M. Kassels, A parallel domain decomposition algorithm for the incompressible Navier–Stokes equations, Simul. Pract. Theory 3 (1995) 185–205.
[5] E. Brakkee, C. Vuik, P. Wesseling, Domain decomposition for the incompressible Navier–Stokes equations: solving subdomain problems accurately and inaccurately, Int. J. Numer. Methods Fluids 26 (1998) 1217–1237.
[6] J.H. Bramble, J.E. Pasciak, A.T. Vassilev, Analysis of non-overlapping domain decomposition algorithms with inexact solves, Math. Comput. 61 (1998) 1–19.
[7] G. de Vahl Davis, I.P. Jones, Natural convection in a square cavity: a comparison exercise, Int. J. Numer. Methods Fluids 3 (1983) 227–248.
[8] E. de Sturler, H.A. van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, Appl. Numer. Math. 18 (1995) 441–459.
[9] M. Dryja, O.B. Widlund, Domain decomposition algorithms with small overlap, SIAM J. Sci. Comput. 15 (1994) 604–620.
[10] S.C. Eisenstat, H.C. Elman, M.H. Schultz, Variational iterative methods for non-symmetric systems of linear equations, SIAM J. Numer. Anal. 20 (1983) 345–357.
[11] J. Erhel, A parallel GMRES version for general sparse matrices, Electron. Trans. Numer. Anal. 3 (1995) 160–176 (http://etna.mcs.kent.edu).
[12] J. Frank, C. Vuik, Parallel implementation of a multiblock method with approximate subdomain solution, Appl. Numer. Math. 30 (1999) 403–423.
[13] J.L. Gustafson, Reevaluating Amdahl's law, Commun. ACM 31 (1988) 532–533.
[14] W. Hoffmann, Iterative algorithms for Gram–Schmidt orthogonalization, Computing 41 (1989) 335–348.
[15] J.P. Jackson, P.C. Robinson, A numerical study of various algorithms related to the preconditioned conjugate gradient method, Int. Numer. Methods Eng. 21 (1985) 1315–1338.
[16] G. Li, A block variant of the GMRES method on massively parallel processors, Parallel Computing 23 (1997) 1005–1019.
[17] A. Segal, P. Wesseling, J. Kan, C.W. Oosterlee, K. Kassels, Invariant discretization of the incompressible Navier–Stokes equations in boundary fitted co-ordinates, Int. J. Numer. Methods Fluids 15 (1992) 411–426.
[18] R.B. Sidje, Alternatives for parallel Krylov subspace basis computation, Numer. Linear Algebra Appl. 4 (4) (1997) 305–331.
[19] W. Tang, Generalized Schwarz splittings, SIAM J. Sci. Statist. Comput. 13 (1992) 573–595.
[20] H.A. van der Vorst, C. Vuik, GMRESR: a family of nested GMRES methods, Num. Linear Algebra Appl. 1 (1994) 369–386.
[21] J. van Kan, A second-order accurate pressure-correction scheme for viscous incompressible flow, SIAM J. Sci. Statist. Comput. 7 (1986) 870–891.
[22] C. Vuik, Further experiences with GMRESR, Supercomputer 55 (1993) 13–27.
[23] C. Vuik, Solution of the discretized incompressible Navier–Stokes equations with the GMRES method, Int. J. Numer. Methods Fluids 16 (1993) 507–523.
[24] C. Vuik, Fast iterative solvers for the discretized incompressible Navier–Stokes equations, Int. J. Numer. Methods Fluids 22 (1996) 195–210.
[25] C. Vuik, R.R.P. van Nooyen, P. Wesseling, Parallelism in ILU-preconditioned GMRES, Parallel Computing 24 (1998) 1927–1946.
[26] H.F. Walker, Implementation of the GMRES method using Householder transformations, SIAM J. Sci. Statist. Comput. 9 (1) (1988) 152–163.
[27] P. Wesseling, A. Segal, C.G.M. Kassels, Computing flows on general three-dimensional nonsmooth staggered grids, J. Comput. Phys. 149 (1999) 333–362.
[28] P. Wesseling, A. Segal, C.G.M. Kassels, H. Bijl, Computing flows on general two-dimensional nonsmooth staggered grids, J. Eng. Math. 34 (1998) 21–44.
[29] J. Xu, J. Zou, Some nonoverlapping domain decomposition methods, SIAM Rev. 40 (1998) 857–914.
[30] S. Zeng, C. Vuik, P. Wesseling, Numerical solution of the incompressible Navier–Stokes equations by Krylov subspace and multigrid methods, Adv. Comput. Math. 4 (1995) 27–49.

Kees Vuik received his Master's degree in Applied Mathematics from the Delft University of Technology in 1982. After a short stay at Philips Research Laboratories, he obtained his PhD on the solution of moving boundary problems from Utrecht University in 1988. In the same year, he joined the Numerical Analysis Group at the Delft University of Technology. His current interests are parallel iterative methods which are applied to linear systems originating from discretized partial differential equations.

Jason Frank was born in Hutchinson, Kansas, USA in 1970. He obtained his Bachelor and Master of Science degrees in aerospace engineering from the University of Kansas in 1992 and 1994, respectively. Thereafter he spent one half year as a visiting researcher at the CWI in Amsterdam, the Netherlands, before beginning his PhD research in numerical analysis at Delft University of Technology. Since finishing his PhD in 2000, he has been working as a postdoctoral researcher at the CWI.

Guus Segal obtained his Master's degree in Applied Mathematics from the Delft University of Technology in 1971. Currently he is a staff member of the Numerical Analysis Group at the Delft University of Technology. His research concerns discretization techniques for partial differential equations, mainly based on finite element and finite volume methods.