Parallel implementation of a multiblock method with approximate subdomain solution

Report 98-11

J. Frank
C. Vuik

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

Parallel implementation of a multiblock method with approximate subdomain solution

J. Frank and C. Vuik

Faculty of Information Technology and Systems
Department of Technical Mathematics
Delft University of Technology, The Netherlands

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 a number of approximate solvers are considered, and numerical experiments are included showing speedups obtained on a cluster of workstations as well as on a distributed memory parallel computer. Additionally, some remarks are made pertaining to the practical application of Householder reflections as an orthogonalization procedure within Krylov subspace methods.

Key words: Domain decomposition; approximate subdomain solution; parallel Krylov subspace methods; orthogonalization methods.

1 Introduction

Domain decomposition arises naturally in computational fluid dynamics: (1) as a means of dealing with geometric complexity and (2) as a source of parallelism. Concerning the first of these, complicated geometries may be broken down into (topologically) rectangular blocks and discretized in general coordinates, see e.g. [3,33,46], applying domain decomposition to iteratively arrive at the solution on the global domain. In this case, since the domain decomposition iteration is used even in the serial computation, the speedups obtained by parallelization of the method may be very significant. 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 [39]. Additionally, parallel domain decomposition may be employed to deal with problems so large as to exceed workstation memory resources.

The current research is motivated by the desire to parallelize an existing domain decomposition implementation for the DeFT software, the continuation of work summarized in [6]. Results from a parallel implementation of a Krylov-accelerated Schur complement domain decomposition method are presented in [7]. A serial implementation of non-overlapping additive Schwarz iterations with approximate subdomain solution [8] gave more promising results. It is the parallelization of this method that we address here.

Theoretical results on approximate solution of subdomain problems for Schur complement domain decomposition methods are given by Börgers [5], Haase, Langer and Meyer [28,19,17,18,16], and Cheng [11]. Brakkee [6] gives theoretical and experimental results for non-overlapping Schwarz iterations with variable approximate inner solvers. Tan [38] shows that for a similar scheme, the subdomains should be solved to a fixed tolerance.

Much effort has focused on efficient parallelization of Krylov subspace methods. Apart from the preconditioning, the main parallel operations required in these methods are distributed matrix-vector multiplications and inner products. For many problems, the matrix-vector multiplications require only nearest neighbor communications, and are thus rather efficient. Inner products, on the other hand, require global communications; therefore, the focus has been on reducing the number of inner products [13,34], overlapping inner product communications with computation [37], or increasing the number of inner products that can be computed with a single communication [2,26].

The additive Schwarz preconditioner is block diagonal and thus may be solved without communication. Other block preconditioning techniques are the multiplicative Schwarz preconditioner—corresponding to a block Gauss-Seidel iteration matrix—and incomplete block factorizations, which are considered by [36,30,45], among others.

In this paper we demonstrate that a reasonable amount of speedup can be observed for a nonoverlapping, one-level additive Schwarz method if the subdomain problems are solved using only a rough approximation. In Section 2 we briefly review the relevant mathematics and give some motivation for approximate subdomain solution.

Some practical points concerning orthogonalization methods are brought out in Section 3. Given a set of k orthogonal vectors $q_1, \ldots, q_k$ and a candidate vector a which is linearly independent of the q_i, an *orthogonalization method* is defined to be a method which produces a new vector q_{k+1}, such that $q_{k+1}^T q_i = 0$, $i = 1, \ldots, k$, and such that $a \in \text{span}\{q_1, \ldots, q_{k+1}\}$. The classical and modified Gram-Schmidt procedures are examples of such orthogonalization methods, as is the Householder implementation of Walker [44]. Also in Section 3, a performance model is developed for comparison of some orthogonalization

methods in parallel. Experimental results in Section 4 confirm the qualitative validity of the model.

Included in Section 4 are speedup results, obtained by comparison of the parallel multiblock computation times to both the single block serial time and the multiblock serial times, as well as results for a scaled problem size (number of unknowns per processor held constant.) The timings were made on a cluster of workstations and a Cray T3E. In particular, our results suggest that the most efficient subdomain approximation in terms of computation time is a simple incomplete factorization.

2 Mathematical Background

2.1 Nonoverlapping domain decomposition

We consider an elliptic partial differential equation discretized using a finite volume or finite difference method on a computational domain Ω. By a computational domain we mean the set of unknown values to be approximated, together with their associated locations in space. Let the domain be the union of M nonoverlapping subdomains Ω_m, $m = 1, \ldots, M$.

Discretization of the PDE results in a sparse linear system

$$Ax = b, \tag{1}$$

with x, $b \in \mathrm{R}^N$. The structure of the matrix A is determined by the stencil of the discretization. Even if there is no overlap between the subdomains, there is still an inter-subdomain coupling due to the stencil. That is, the equation for an unknown adjacent to a subdomain interface is dependent on an unknown across the subdomain boundary.

One technique for solving this problem is to permute the system (1), grouping together into blocks those unknowns whose stencils are in the interior of a common subdomain, and grouping all the remaining unknowns—the interface unknowns, whose stencils transcend a subdomain boundary—into a separate block:

$$\begin{bmatrix} A_{11} & & & A_{1\mathcal{I}} \\ & \ddots & & \vdots \\ & & A_{MM} & A_{M\mathcal{I}} \\ A_{\mathcal{I}1} & \cdots & A_{\mathcal{I}M} & A_{\mathcal{I}\mathcal{I}} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_M \\ x_{\mathcal{I}} \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_M \\ b_{\mathcal{I}} \end{pmatrix}. \tag{2}$$

This system is solved by performing a block decomposition: first solving the Schur complement for the interface unknowns, and then solving the independent subdomain problems. Methods of this type are referred to as *substructuring methods.*

Another approach is to group together the unknowns which share a common subdomain, giving a block system:

$$\begin{bmatrix} A_{11} & \dots & A_{1M} \\ \vdots & \ddots & \vdots \\ A_{M1} & \dots & A_{MM} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_M \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_M \end{pmatrix}. \tag{3}$$

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 *additive Schwarz iteration* introduces the block Jacobi preconditioner:

$$K = \begin{bmatrix} A_{11} & & \\ & \ddots & \\ & & A_{MM} \end{bmatrix},$$

which is easily solved in parallel to approximate the error. We will be primarily concerned with additive Schwarz domain decomposition in this paper.

For a thorough discussion of domain decomposition methods see the book [35] and the review article [9]. Each of these publications contains an extensive bibliography. Convergence theory for domain decomposition methods is discussed in [35]. Roughly speaking, the convergence rate suffers proportionally to the number of subdomains in each direction. If a constant overlap (in physical units) is maintained, the convergence rate is independent of grid size; however, for zero overlap the convergence is relatively poor. The convergence rate may additionally be made independent of the number of subdomains if a coarse subspace correction is applied: for example, the residual is projected onto a single coarse grid domain, where a correction is computed which is then interpolated back to the subdomains.

In practice (3) is solved iteratively, using K as a preconditioner for a Krylov subspace method, such as the conjugate gradient method for symmetric problems or the GMRES method for nonsymmetric problems. A Krylov subspace method seeks, in the kth iteration, an optimal (in some sense) approximation of the solution of $Ax = b$ in the *Krylov subspace*

$$\mathcal{V}_k = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}.$$

Generally, a basis for the subspace is computed recursively, its dimension increasing by one in each iteration. For example, the GMRES method [32] computes in the kth iteration

$$AV_k = V_{k+1}H_k, \tag{4}$$

where the columns of V_k (denoted $v_1, \ldots, v_k$) form an orthonormal basis for $\mathcal{V}_k$, and where the kth column of H_k contains the coefficients of the modified Gram-Schmidt process used to orthonormalize Av_k with respect to V_k to give v_{k+1}. Using a preconditioner in cooperation with GMRES, the problem (1) is replaced by

$$K^{-1}Ax = K^{-1}b.$$

For our purposes the GCR method [12], shown in Figure 1, is useful. In the algorithm and elsewhere in this paper the Euclidean inner product $\langle x, y\rangle = x^T y$ and associated norm $\|x\| = (x^T x)^{1/2}$ are used.

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

Fig. 1. The GCR Algorithm

The function orthonorm() takes input vectors $\tilde{q}$ and $\tilde{v}$, orthonormalizes $\tilde{q}$ with respect to the q_i, $i < k$, while preserving the relation $\tilde{q} = A\tilde{v}$, and returns

the modified vectors q_k and v_k. In serial computations, the modified Gram-Schmidt method, Figure 2, is often employed for the orthonorm() function. We discuss alternative orthogonalization methods in later sections of this paper.

Algorithm: Modified Gram-Schmidt
$[q_k, v_k]$ = orthonorm $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$:
  for $i = 1, \ldots, k-1$
    $\alpha = \langle \tilde{q}, q_i \rangle$
    $\tilde{q} = \tilde{q} - \alpha q_i$
    $\tilde{v} = \tilde{v} - \alpha v_i$
  end
  $\beta = \|\tilde{q}\|$
  $q_k = \tilde{q}/\beta$; $v_k = \tilde{v}/\beta$
return

Fig. 2. The modified Gram-Schmidt algorithm

In exact arithmetic, and assuming it does not break down, GCR produces the same iterates as GMRES. However GCR does not take advantage of the recursion (4). Rather, GCR requires the storage of an extra set of orthogonal *residual search vectors*. Three advantages of this method are: (1) the preconditioner K need not remain constant (nor even be a linear operator; the implementation in [40] uses GMRES(m) as a preconditioner); (2) one is free to employ truncation strategies such as in [41]; and (3) the method will not break down if the LSQR switch is employed [40]. As described in Figure 1, the method is unrestricted in the number of iterations, and therefore in the number of vectors v_k and q_k which must be stored. Since most modern computers are equipped with only finite memory, it is necessary either to restart the iteration periodically, discarding all stored vectors, or to maintain only a fixed number of vectors, applying some criterion to determine which vectors will be kept. This second option, referred to as *truncation*, is shown in [41] to be very effective in reducing the number of iterations. The importance of allowing a variable preconditioner will be discussed shortly. Another method allowing variable preconditioners is the FGMRES method [31], but truncation is not possible with that method, and it may break down [40].

2.3 Approximate subdomain solution

Solution for $\tilde{v}$ from the preconditioning equation $K\tilde{v} = r_{k-1}$ in the GCR algorithm requires solution of the M subdomain systems $A_{mm}\tilde{v}_m = r_m$, $m = 1, \ldots, M$. Since these problems have a similar nonzero structure to the original matrix A, and since they may still be quite large, it is reasonable to solve them using a second Krylov subspace iteration. A question which arises naturally, and for purely practical reasons, addresses the tolerance to which these *inner*

iterations should converge. It seems senseless, for example, to solve the subdomain problems with a much smaller tolerance than is desired for the global solution, especially in light of the fact that the global iteration has a finite convergence rate.

A number of authors have considered approximate solution of the subdomain problems. In particular they have considered the consequences of using very fast, rough approximations to reduce the total computing time necessary to solve the global problem.

Some possible strategies for approximating the subdomain solutions are:

- Krylov-subspace method (possibly preconditioned) to a fixed tolerance, to a variable tolerance, or for a predetermined number of iterations.
- Approximation by a single preconditioner solve.
- A few iterations of a basic iterative method such as Jacobi or Gauss-Seidel.
- Do nothing at all. In this case one uses the domain decomposition purely as a form of data distribution and applies the unpreconditioned Krylov method.

Tan [38] has shown that if the inner problems are solved to some tolerance in each outer iteration, than the optimal strategy for choosing the tolerance is a fixed one. That is, it is not necessary to make the subdomain solution tolerance smaller as the global solution converges.

Note that if the subdomains are solved using a Krylov subspace method such as GMRES, then the approximate solution is a function of the right hand side, which is the residual of the outer iteration. Furthermore, if the subdomains are solved to a tolerance, the number of inner iterations may vary from one subdomain to another, and in each outer iteration. The effective preconditioner is therefore nonlinear and varies in each outer iteration.

A variable preconditioner presents a problem for the GMRES method: namely, the recurrence relation (4) no longer holds. To allow the use of a variable preconditioner, Saad [31] has developed the Flexible GMRES (FGMRES) method, which requires storage of an auxiliary set of vectors such as with GCR. On the other hand, the GCR method does not suffer from the use of a variable preconditioner, and has the additional advantage that truncation strategies may be employed, as mentioned previously. For this reason, we shall consider the GCR method in this paper.

Our choice of approximate solution methods is motivated by the results obtained in [8]. In that paper, GMRES was used as an approximate solver, with tolerances varying from 10^{-4} to 10^{-1}; also a blockwise application of the RILUD preconditioner was used. The RILUD preconditioner is a diagonal version of the method introduced in [1]. This factorization is an average of an ILUD preconditioner [27] and an MILUD preconditioner [15], with a weight-

ing parameter ω, assigned a value of 0.95 in our experiments. See also [43] for useful results with RILU factorizations applied to Navier-Stokes equations. The use of incomplete factorizations to obtain subdomain approximations has been advocated by Keyes [24] and Goossens et al. [14] among others.

The results of [8] indicated that the coarser tolerances were more effective. However, all numerical results presented there were obtained from serial runs. In Section 4 we will present numerical experiments with a number of subdomain approximation methods. Simpler subdomain approximations, such as diagonal preconditioning or no preconditioning, frequently did not converge in our experiments, and the results will not be presented here.

Most of the theoretical work on the subject of approximate subdomain solution has focused on the Schur complement system (2). It is also possible to solve (2) iteratively. The construction of the Schur complement for the interface unknowns in (2) is an expensive operation, while the action of the Schur complement on a vector can be computed by solving the subdomain problems. Once the Schur complement solution converges, an additional solution of the subdomain blocks gives the global solution. See [5,28,17,18,11] for discussions on approximate subdomain solution with regard to this approach. These papers in general present results for static preconditioners, such as classical incomplete factorizations.

Brakkee [6] has proven the following result. Let $\tilde{A}_{ii}^{-1}$ be the matrix which represents the approximate inversion of the ith block. In the case of a Krylov subspace method as inner solver, this would be the actual value of the minimizing polynomial applied to A. Similarly define $\tilde{K}^{-1}$ to be the approximate preconditioner consisting of the diagonal blocks $\tilde{A}_{ii}^{-1}$. If for each subdomain $i = 1, \ldots, M$ it holds that $\|I - A_{ii}\tilde{A}_{ii}^{-1}\| < \epsilon$, then the condition number of the approximately preconditioned matrix satisfies

$$\kappa(A\tilde{K}^{-1}) \leq \frac{1+\epsilon}{1-\epsilon}\kappa(AK^{-1}). \tag{5}$$

where $\kappa(A) = \|A\|\|A^{-1}\|$ is the spectral condition number of A.

Essential to the proof of this relation is the fact that

$$\kappa(A\tilde{K}^{-1}) = \kappa(AK^{-1}K\tilde{K}^{-1}) \leq \kappa(AK^{-1})\kappa(K\tilde{K}^{-1}).$$

We obtain a slightly more general result in the case of a symmetric matrix A with symmetric K and $\tilde{K}$. Note that the matrix $B = K\tilde{K}^{-1}$ is a block diagonal matrix with blocks $B_i = A_{ii}\tilde{A}_{ii}^{-1}$, $i = 1, \ldots, M$. It is not difficult to see that the eigenvalues of such a matrix are the union of the eigenvalues of the blocks. It follows easily that if there exist α, β such that $0 < \alpha \leq \min_i |\lambda_{\min}(B_i)| \leq$

$\max_i |\lambda_{\max}(B_i)| \leq \beta$, then

$$\kappa(A\tilde{K}^{-1}) \leq \frac{\beta}{\alpha}\kappa(AK^{-1}).$$

One can gain a qualitative understanding of this result by considering the special case in which the blocks A_{ii} have identical spectra, and a static, classical preconditioner, such as an MILU preconditioner, is applied to each block. In this case, the fraction β/α is simply the spectral condition number of any block. It is known that for the block Jacobi preconditioner, $\kappa(AK^{-1}) = O(h^{-1}H^{-1})$, where h is the grid spacing and H is the subdomain diameter (see, e.g. [10]). Furthermore, for the MILU preconditioner, it has been shown that $\kappa(A_{ii}\tilde{A}_{ii}^{-1}) = O(h^{-1})$ (see [15].) It follows that

$$\kappa(A\tilde{K}^{-1}) = O(h^{-2}H^{-1}).$$

Applying the same analysis as above, the use of a grid-independent preconditioner such as multigrid would give

$$\kappa(A\tilde{K}^{-1}) = O(1)O(h^{-1}H^{-1}) = O(h^{-1}H^{-1}),$$

indicating that the convergence rate of the approximately preconditioned system would be about the same as that of the exactly preconditioned system.

Convergence rate is not everything, however. Another important factor is the expense of computing the subdomain approximate solutions. The RILUD preconditioner, though least effective in terms of convergence rate, is far cheaper than iterative solution of subdomain problems by a Krylov method, at least for the problem size considered here. One makes a tradeoff thus between effectiveness of an approximate preconditioner in terms of convergence rate and cheapness in terms of computational expense.

2.4 Orthogonalization methods

The primary challenges to parallelization of Figure 1 are parallelization of the preconditioning—a difficulty which disappears when a block preconditioner K is used—and parallel computation of the inner products. Inner products require global communication and therefore do not scale. Most of the literature on parallel Krylov subspace methods and parallel orthogonalization methods is focused on orthogonalizing a number of vectors simultaneously. See, e.g. [29,22,2,37,34,26]. 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.

The modified Gram-Schmidt method of Figure 2 suffers from the fact that the inner products must be computed using successive communications, and the

number of these inner products increases by one with the iteration number. This is not the case if one uses the classical Gram-Schmidt method, Figure 3. In this algorithm all necessary inner products can be computed with a single

Algorithm: Classical Gram-Schmidt
$[q_k, v_k]$ = orthonorm $(\tilde{q}, \tilde{v}, q_i, v_i, i < k)$:
  $\beta = \langle \tilde{q}, \tilde{q} \rangle$
  for $i = 1, \ldots, k-1$
    $\alpha_i = \langle \tilde{q}, q_i \rangle$
  end
  $\beta = \sqrt{\beta - \sum_{i=1}^{k-1} \alpha_i^2}$
  $q_k = \beta^{-1} \left(\tilde{q} - \sum_{i=1}^{k-1} \alpha_i q_i \right)$
  $v_k = \beta^{-1} \left(\tilde{v} - \sum_{i=1}^{k-1} \alpha_i v_i \right)$
return

Fig. 3. The classical Gram-Schmidt algorithm

global communication. Unfortunately, Björck [4] has shown that the classical Gram-Schmidt method is unstable with respect to rounding errors, so this method is rarely used.

On the other hand, Hoffmann [21] gives experimental evidence indicating that a twofold application of Figure 3 *is* stable. Furthermore, it appears that if orthogonality is important, such a re-orthogonalization is also required even for the more stable modified Gram-Schmidt algorithm.

A third method which has been suggested is the parallel implementation of Householder transformations, introduced by Walker [44]. We shall reformulate that method for GCR in the following section. Additionally, we will present a simple parallel performance analysis for comparison of these three orthogonalization procedures.

3 Householder orthogonalization

Walker [44] has proposed a GMRES variant using a vectorized version of Householder transformations as an alternative to the modified Gram-Schmidt procedure. The Householder method has the advantage that it requires only a fixed number of communications per GMRES iteration. In this section we describe the GCR implementation and discuss some practical details concerning its use.

In the following discussion 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 QR, where Q is orthogonal and R is upper triangular. Then the kth column of A is given by $a_k = Qr_k$. It follows that $a_k \in \text{span}\{q_1, \ldots, q_k\}$. In other words, the columns of Q form an orthonormal basis for the span of the columns of A.

We take Q as the product of a series of Householder reflections, $Q = P_1 \cdots P_m$, used to transform A into R. The matrices P_i have the following properties:

i) $P_i^2 = I = P_i^T P_i$.
ii) $P_i e_j = e_j$, if $j < i$.
iii) $P_i(P_{i-1} \cdots P_1)a_i = r_i$.

In property ii) e_j is the jth canonical unit vector in $\mathbb{R}^n$. A Householder reflection is given by $P_i = I - 2\frac{w_i w_i^T}{w_i^T w_i}$, for some $w_i \in \mathbb{R}^n$. Note that such a matrix has property i). Property ii) is ensured by requiring the first $i-1$ components of w_i be zero: $w_i^{(j)} = 0$ for $j < i$.

Suppose one has already produced k orthogonal basis vectors $q_1, \ldots, q_k$ and stored them along with the transformation vectors $w_1, \ldots, w_k$ corresponding to $P_1, \ldots, P_k$. Given a candidate vector a_{k+1}, one must first apply the previous reflections as described in [44]:

$$\tilde{a} = P_k \cdots P_1 a_{k+1} = (I - 2W_k L_k^{-1} W_k^T)a_{k+1}$$

where here and elsewhere we denote by W_k 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 $2m-1$ inner products, but they may all be computed using only a single global communication.

Now having computed $\tilde{a}$ one wishes to find w_{k+1} such that P_{k+1} satisfies

$$\begin{aligned} P_{k+1}\tilde{a} = r_{k+1} &= r_{k+1}^{(1)}e_1 + \cdots + r_{k+1}^{(k+1)}e_{k+1} \\ &= \tilde{a}^{(1)}e_1 + \cdots + \tilde{a}^{(k)}e_k + \alpha e_{k+1}, \end{aligned} \tag{6}$$

where property ii) has been used for the last equality.

Because of the relation

$$P_{k+1}\tilde{a} = (I - 2\frac{w_{k+1}w_{k+1}^T}{w_{k+1}^T w_{k+1}})\tilde{a} = \tilde{a} - 2\frac{w_{k+1}^T\tilde{a}}{w_{k+1}^T w_{k+1}}w_{k+1}, \tag{7}$$

one must have $w_{k+1} \in \text{span}\{\tilde{a}, e_1, \ldots, e_{k+1}\}$. However equation (6) provides the relation which must hold among $\tilde{a}, e_1, \ldots, e_k$. Let $\tilde{w}$ be the vector obtained by setting the first k elements of $\tilde{a}$ to zero. Formally, one has $\tilde{w} = J_{k+1}\tilde{a}$, where

$$J_{k+1} = \begin{bmatrix} 0_k & \\ & I_{n-k} \end{bmatrix}.$$

Thus, $w_{k+1} \in \text{span}\{\tilde{w}, e_{k+1}\}$. The length of w_{k+1} is a free parameter, so take $w_{k+1} = \tilde{w} + \beta e_{k+1}$. Substituting into (7) gives

$$\begin{aligned} P_{k+1}\tilde{a} &= \tilde{a} - 2\frac{w_{k+1}^T\tilde{a}}{w_{k+1}^T w_{k+1}}(J_{k+1}\tilde{a} + \beta e_{k+1}) \\ &= (I - 2\frac{w_{k+1}^T\tilde{a}}{w_{k+1}^T w_{k+1}}J_{k+1})\tilde{a} - 2\beta\frac{w_{k+1}^T\tilde{a}}{w_{k+1}^T w_{k+1}}e_{k+1}. \end{aligned}$$

To ensure that all elements below the $(k+1)$th are zero, one requires $1 - 2\frac{w_{k+1}^T\tilde{a}}{w_{k+1}^T w_{k+1}} = 0$. But,

$$w_{k+1}^T\tilde{a} = (\tilde{w} + \beta e_{k+1})^T\tilde{a} = \|\tilde{w}\|^2 + \beta\tilde{a}^{(k+1)},$$

and

$$w_{k+1}^T w_{k+1} = \|(\tilde{w} + \beta e_{k+1})\|^2 = \|\tilde{w}\|^2 + 2\beta\tilde{a}^{(k+1)} + \beta^2.$$

Substituting these numbers into the above relation, one finds $\beta = \pm\|\tilde{w}\|$, and the sign of β is chosen to be the same as that of $\tilde{w}^{(k+1)}$ to eliminate the risk of subtractive cancelation:

$$w_{k+1} = \tilde{w} + \text{sign}\,(\tilde{w}^{(k+1)})\|\tilde{w}\|e_{k+1}.$$

In practice, the w_k are normalized to length one. Since α is the $(k+1)$th component of $P_{k+1}\tilde{a}$, substitution of the above relation into (7), and noting

that

$$2\frac{w_{k+1}^T \tilde{a}}{w_{k+1}^T w_{k+1}} = 1,$$

gives $\alpha = \tilde{a}^{(k+1)} - \tilde{w}^{(k+1)} - \text{sign}\,(\tilde{w}^{(k+1)})\|\tilde{w}\| = -\text{sign}\,(\tilde{w}^{(k+1)})\|\tilde{w}\| = -\beta$, and the length of w_{k+1} can be expressed as

$$\|w_{k+1}\| = \sqrt{2\alpha^2 - 2\alpha\tilde{w}^{(k+1)}}.$$

The $(k+1)$th column of Q is the new orthonormal basis vector,

$$Qe_{k+1} = P_1 \cdots P_{k+1} e_{k+1},$$

and because of property ii), the yet to be computed reflections will not affect this column. Multiplying both sides of (6) by $P_1 \cdots P_k$ gives:

$$a = \tilde{a}^{(1)} q_1 + \cdots + \tilde{a}^{(k)} q_k + \alpha q_{k+1},$$

from which it follows that

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

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

Our implementation requires three communications in each iteration, namely:

(1) The computation of $\tilde{a}$ using Walker's approach, requires $2k-1$ inner products, all of which can be performed with a single communication.
(2) A second communication is necessary to broadcast the first $k+1$ elements of $\tilde{a}$.
(3) A communication is required to compute $\|\tilde{w}\|$ for α.

The implementation is as in Figure 4, with $\tilde{q}$ playing the role of a in the above discussion.

Comparing the Householder implementation with modified Gram-Schmidt,

- In the kth iteration the Householder method requires three communications, whereas Gram-Schmidt requires $k+1$.
- Householder requires approximately twice as many inner products as Gram-Schmidt, plus $1\frac{1}{2}$ times the number of 'axpy' operations.
- The Householder method requires the storage of an extra set of k vectors.

A drawback of the Householder method is that there appears to be no simple way to incorporate truncation schemes in the GCR method if Householder is

Algorithm: Householder Orthogonalization

$[q_k, v_k] = \text{orthonorm } (\tilde{q}, \tilde{v}, q_i, v_i, i < k)$:

if $k == 1$

$w_1 = \tilde{q}$

else

if $k == 2$

$L_1 = 1$

else

$$L_{k-1} = \begin{bmatrix} L_{k-2} & 0 \\ 2w_{k-1}^T W_{k-2} & 1 \end{bmatrix}$$

end

$y = W_{k-1}^T \tilde{q}$

Solve $L_{k-1} d = y$

$w_k = \tilde{q} - 2W_{k-1} d$

end

$\tilde{a}^{(i)} = w_k^{(i)}$, $i = 1, \ldots, k$

Broadcast $(\tilde{a})$

$q_k = \tilde{q} - \sum_{i<k} \tilde{a}^{(i)} q_i$

$v_k = \tilde{v} - \sum_{i<k} \tilde{a}^{(i)} v_i$

Set $w_k^{(i)} = 0$, $i = 1, \ldots, k-1$

$\alpha = -\text{sign}(\tilde{a}^{(k)}) \|w_k\|$

$w_k^{(k)} = w_k^{(k)} - \alpha$

$q_k = q_k / \alpha$

$v_k = v_k / \alpha$

$w_k = w_k / \sqrt{2\alpha(\alpha - \tilde{a}^{(k)})}$

return

Fig. 4. The Householder orthogonalization algorithm

used for orthogonalization. For truncation one would like to be able to discard an arbitrary vector q_j and the corresponding transformation vector w_j, adding a new pair q_{k+1}, w_{k+1} to the respective spaces. Below we give a mathematical motivation for the difficulty of doing this if Householder orthogonalization is used:

Suppose one has $Q = P_1 \cdots P_k$, with orthonormal basis $q_i = Qe_i$, $i = 1, \ldots, k$. Now consider the matrix obtained by deleting P_j and adding a new transformation P_{k+1}, i.e. $\tilde{Q} = P_1 \cdots P_{j-1} P_{j+1} \cdots P_{k+1}$, and let $\tilde{q}_{k+1} = Qe_k$ be the kth column of this matrix. Then from properties i) and ii) one finds in general:

$$\begin{aligned} \langle \tilde{q}_{k+1}, q_i \rangle &= \langle P_1 \cdots P_{j-1} P_{j+1} \cdots P_{k+1} e_k, P_1 \cdots P_i e_i \rangle \\ &= \langle e_k, e_i \rangle = 0, \text{ for } i < j, \end{aligned}$$

however,

$$\langle \tilde{q}_{k+1}, q_i \rangle = \langle P_{j+1} \cdots P_{k+1} e_k, P_j \cdots P_i e_i \rangle \neq 0, \text{ for } i \geq j.$$

There is therefore no obvious way to do this.

In the next section we develop a performance model for comparison of the Householder and Gram-Schmidt methods.

3.2 Performance model

To give insight into the choice of an orthogonalization procedure, consider a simple performance model. Let the time required for communication of a message of n floating point numbers be given by

$$t_{\text{comm}} = t_0 + \beta n,$$

where t_0 is the fixed time required for a message of length zero, and β is the time per floating point number (bandwidth). Let the time for n floating point operations be given by

$$t_{\text{comp}} = \phi n,$$

where ϕ is the time for 1 floating point operation. Similar computation/communication models are used, for example, in [37,29,20,25].

Let p denote the number of processes, and define a function $f(p)$ which gives the maximum number of a set of communications which must be performed sequentially. The function $f(p)$ is machine-dependent and also dependent on the distribution of processes on the machine. For example, assuming perfect connectivity and that processes not participating in a given communication are free to participate in a concurrent communication, the broadcast of a message among p processes requires $f(p) = \lceil \log_2 p \rceil$ consecutive send operations from the broadcasting process. An Ethernet broadcast requires $f(p) = p - 1$ consecutive send operations.

Assume each processor is responsible for an $n \times n$ subdomain with n^2 unknowns. Define the times for some basic operations:

Op.	Communication	Computation	Definition
send (k)	$t_0 + \beta k$		send a message of length k
flop (n)		$n\phi$	n floating point operations
$B(p,k)$	$f(p)(t_0 + \beta k)$		broadcast k elements
$G(p,k)$	$2f(p)(t_0 + \beta k)$		global sum k elements
SIP(k)	$G(p,k)$	$2kn^2\phi$	k inner prod. simult. comms.
FS(k)		$k^2\phi$	forward substitution, order k
axpy		$2n^2\phi$	$z = ax + y$, scalar a

Note that we distinguish between inner products that can be computed simultaneously (i.e. with a single communication) and inner products that cannot. For example, k simultaneous inner products are denoted SIP(k), whereas k non-simultaneous inner products are denoted k SIP(1). The modified and re-orthogonalized classical Gram-Schmidt and Householder routines can be broken down into components as follows. In the kth iteration of GCR:

Mod. Gram-Schmidt	k SIP(1)
	$2k-1$ axpy
Re-orthog. CGS	2 SIP (k)
	$3k-1$ axpy
Householder	SIP $(2k-3)$
	SIP(1)
	FS $(k-1)$
	$3k-3/2$ axpy
	1 Broadcast (k)

We have implemented the re-orthogonalized classical Gram-Schmidt method so that in the kth iteration, the candidate residual search vector $\tilde{q}$ is twice orthogonalized against the basis $q_1, \ldots, q_{k-1}$ to obtain q_k, and only then is the search vector v_k computed. This eliminates a series of vector updates and explains why there are only $3k-1$ 'axpy' operations.

Based on the communication model outlined in the previous two tables, the orthogonalization time required for s iterations of GCR (without restart) using the modified Gram-Schmidt (MGS), re-orthogonalized classical Gram-Schmidt

(CGS2) and Householder (HH) methods, respectively, is given by:

$$t_{\mathrm{MGS}} = \frac{s(s+1)}{2}\left[6n^2\phi + 2f(p)(t_0+\beta)\right] - s(2n^2\phi), \tag{8}$$

$$t_{\mathrm{CGS2}} = \frac{s(s+1)}{2}\left[10n^2\phi + 4f(p)\beta\right] + s\left[4f(p)t_0 - 2n^2\phi\right], \tag{9}$$

$$\begin{aligned} t_{\mathrm{HH}} = {} & \phi\frac{s(s+\frac{1}{2})(s+1)}{3} + \frac{s(s+1)}{2}\left[(10n^2-2)\phi + 5f(p)\beta\right] \\ & + s\left[f(p)(6t_0-4\beta) - (7n^2-1)\phi\right]. \end{aligned} \tag{10}$$

If the forward substitution in the Householder algorithm is negligible, the model becomes

$$t_{\mathrm{HH}} = \frac{s(s+1)}{2}\left[10n^2\phi + 5f(p)\beta\right] + s\left[f(p)(6t_0-4\beta) - 7n^2\phi\right].$$

Comparing this expression with (9) for the re-orthogonalized classical Gram-Schmidt algorithm, we see that the two methods are very similar in cost, while we shall see later that re-orthogonalized Gram-Schmidt is much more stable than Householder for the standard test problem. The similarity in cost is confirmed by our experiments.

Tests were performed on a cluster of HP workstations to obtain representative values for the parameters t_0, β and ϕ:

$$t_0 \approx 4.7 \times 10^{-4}, \qquad \beta \approx 7.5 \times 10^{-6}, \qquad \phi \approx 4.9 \times 10^{-8}.$$

Similar tests were performed on a Cray T3E using MPI communications with the results:

$$t_0 \approx 2.4 \times 10^{-5} \qquad \beta \approx 5.4 \times 10^{-8} \qquad \phi \approx 5.8 \times 10^{-8}.$$

Assuming the models (8), (9)and (10), and assuming $f(p) = p - 1$ for the workstation cluster and $f(p) = \lceil \log_2 p \rceil$ for the Cray T3E, the quantities

$$\mathcal{F}_{\mathrm{HH}} = \frac{\text{orthog. time MGS}}{\text{orthog. time HH}}, \tag{11}$$

$$\mathcal{F}_{\mathrm{CGS2}} = \frac{\text{orthog. time MGS}}{\text{orthog. time CGS2}} \tag{12}$$

are plotted as a function of n for $s = 60$ and $p = 4, 9$ ($p = 4, 9, 25$ for the Cray T3E) in Figure 5. The Householder (resp. CGS2) method is faster at points in the figure where $\mathcal{F}_{\mathrm{HH}} > 1$ (resp. $\mathcal{F}_{\mathrm{CGS2}} > 1$). The model predicts that the alternative methods (HH) and (CGS2) are only advantageous for small enough subdomain size. On the workstation cluster this size may be about 10000 unknowns on 4 processors and somewhat more 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. Note also that the model indicates that the computational efforts of Householder and re-orthogonalized classical Gram-Schmidt are very similar, with the Gram-Schmidt variant to be preferred in the useful range.

In the following section we shall see that, while the model is qualitatively correct, the observed performance curves are a bit lower than the ones predicted here.

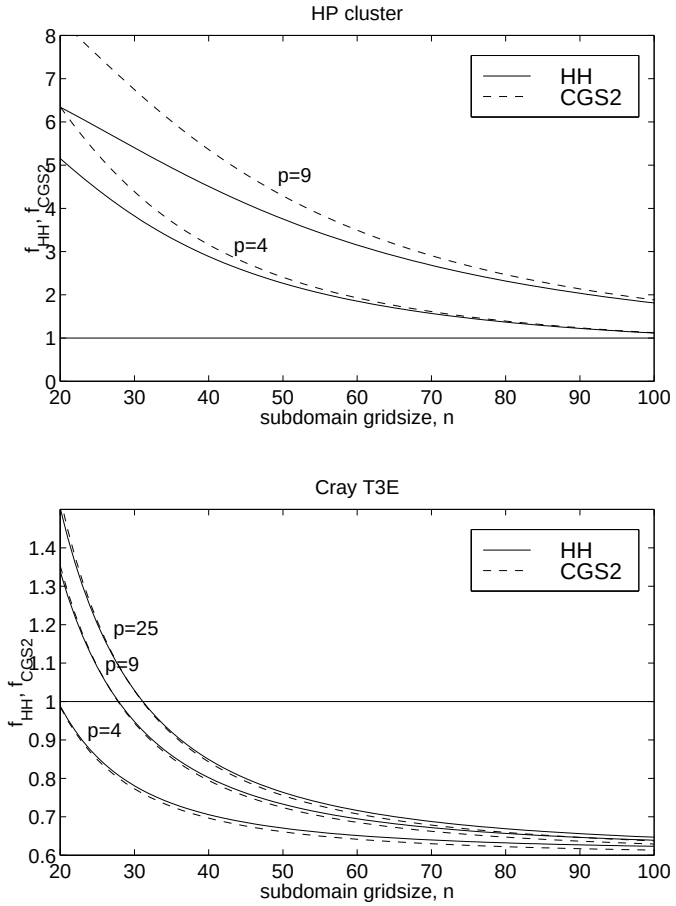

Fig. 5. Predicted speedup with Householder orthogonalization

To put the importance of the orthogonalization time into better perspective, the computation time required for one iteration of GCR can be broken down as follows:

$$t_k = t_{\text{mat}} + t_{\text{precond}} + t_{\text{orth}}$$

where t_{precond} is the time required to solve the preconditioner, t_{orth} is the orthogonalization time and t_{mat} represents the remaining operations (i.e. matrix-vector multiplication and vector sums). If the system to be solved is very large

and the preconditioner is expensive, then the time required for orthogonalization can become unimportant.

Another issue of relevance to the choice of an orthogonalization method is the stability of the method with respect to rounding errors. Figure 6 shows a comparison of the classical, modified, and re-orthogonalized classical Gram-Schmidt methods and the Householder implementation for the test matrix of [4]:

$$A = \begin{bmatrix} 1 & 1 & \dots \\ \epsilon & & \\ & \epsilon & \\ & & \ddots \\ & & \end{bmatrix}.$$

The comparison method is the QR decomposition function of Matlab. We see that the re-orthogonalized classical Gram-Schmidt method gives the smallest orthogonalization error of all methods for this test case.

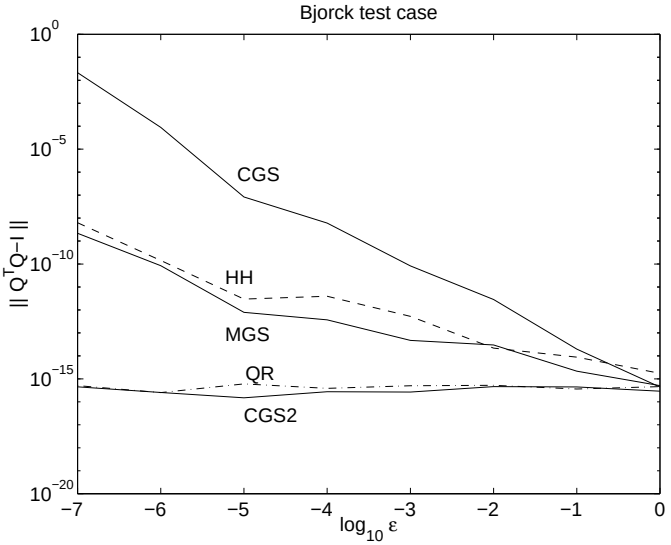

Fig. 6. Comparison of orthogonalization error for classical (CGS), modified (MGS), re-orthogonalized (CGS2) Gram-Schmidt methods, Householder (HH) method, and Matlab QR function on Björck test problem.

In conclusion, we mention that there does not seem to be any reason to prefer the parallel Householder method over the re-orthogonalized classical Gram-Schmidt method, at least in this context. In terms of parallel efficiency the two methods are almost identical. However the Gram-Schmidt variant is simpler to implement, provides significantly less orthogonalization error, and allows

truncation strategies to be employed in a natural way.

4 Numerical experiments

In this section, we give numerical results which provide useful insights into approximate solution techniques. Numerical results were obtained from both a cluster of HP-735 and HP-755 workstations (99-125 MHz) and from a Cray T3E parallel computer. All communications were handled with MPI. Reported times are obtained from the MPI timing functions, and are the minimum time achieved over three runs. For our interests, the workstation results are as important (or more so) than those from the parallel machine, due to the immediate availability and relative cheapness of workstations.

Speedup results will be given both for fixed problem size and for fixed subdomain size (problem size proportional to the number of processors). For the serial case, we are interested both in the single subdomain and multiblock cases, since we mostly use domain decomposition for geometric reasons.

Finally, the question we wish to address is: what is the best form of approximate subdomain solution with respect to total computational expense?

As a test example, we consider a Poisson problem, discretized with the finite volume method on a square domain. This example is relevant to our work, because a similar system must be solved in each time step of an incompressible Navier-Stokes simulation to enforce the divergence-free constraint, see [23]. Solution of this system requires about 75% of the computing effort. The actual system to be solved in incompressible Navier-Stokes flows is nonsymmetric due to boundary conditions, but we consider only a symmetric matrix in these experiments. The domain is composed of an $M \times M$ array of subdomains, each with an $n \times n$ grid. With $h = \Delta x = \Delta y = 1.0/(Mn)$ the discretization is

$$4u_{ij} - u_{i+1j} - u_{i-1j} - u_{ij-1} - u_{ij+1} = h^2 f_{ij}.$$

The right hand side function is $f_{ij} = f(ih, jh)$, where $f(x, y) = -32(x(1 - x) + y(1 - y))$. Homogeneous Dirichlet boundary conditions $u = 0$ are defined on $\partial\Omega$, implemented by adding a row of ghost cells around the domain, and enforcing the condition, for example, $u_{0j} = -u_{1j}$ on boundaries. This ghost cell scheme allows natural implementation of the domain decomposition as well.

The performance model for the orthogonalization methods in the previous section predicts that the modified Gram-Schmidt algorithm is to be preferred for large subdomain problems. We wish to investigate this experimentally, to confirm the model predictions. The results presented here were computed for a fixed number of iterations s, equal to the restart value.

Figure 7 is the experimental analog of Figure 5. The parameters $\mathcal{F}_{\mathrm{HH}}$ and $\mathcal{F}_{\mathrm{CGS2}}$ are plotted for subdomain grid sizes of $n = 20, 40, 60, 80, 100$ and a fixed number of iterations $s = 60$. Measurements were made for 4 and 9 processors ($M = 2, 3$, respectively,) on the HP cluster and 4, 9 and 25 processors ($M = 2, 3, 5$, respectively) on the Cray T3E.

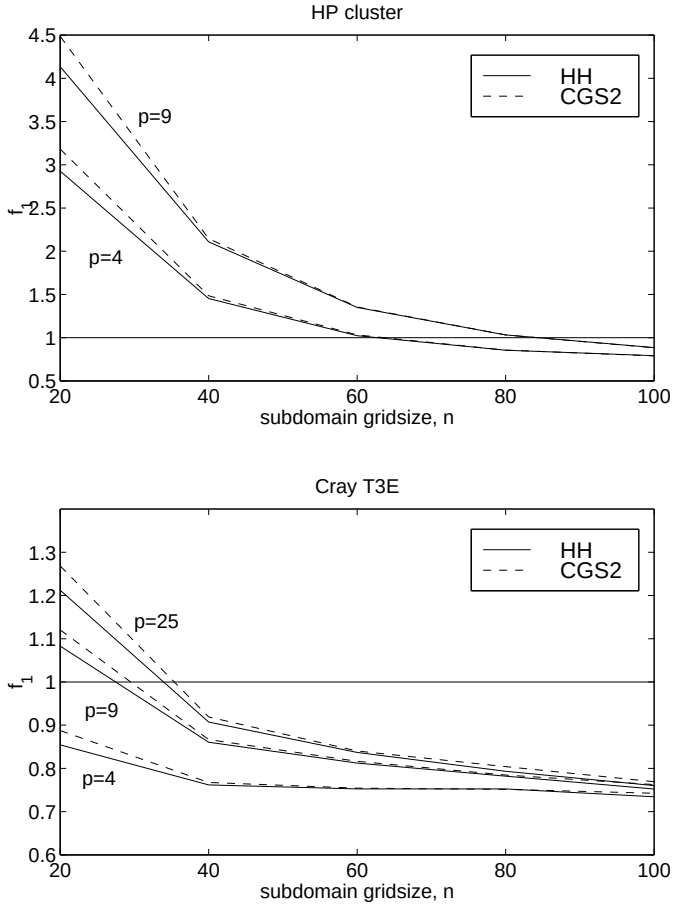

Fig. 7. Measured speedup with Householder (HH) orthogonalization and re-orthogonalized classical Gram-Schmidt (CGS2), restart value $s = 60$.

By comparison one sees that the model developed in the previous section is qualitatively correct, but is rather optimistic with respect to the range of problem sizes for which Householder is more effective than Gram-Schmidt.

In this section we compare speedups obtained with a number of approximate subdomain solvers to get an impression of which solvers might be effectively used with the Navier-Stokes equations. For the tests of this section, a fixed restart value of $s = 30$ was used, and modified Gram-Schmidt was used as the orthogonalization method for all computations. The solution was computed to a fixed tolerance of 10^{-6} unless noted otherwise. The performance measure is computation time, after initialization, taken as the minimum achieved over three runs.

The subdomain approximations will be denoted:

- GMR6 = restarted GMRES with a tolerance of 10^{-6}, (preconditioned with RILUD)
- GMR2 = restarted GMRES with a tolerance of 10^{-2}, (preconditioned with RILUD)
- GMR1 = restarted GMRES with a tolerance of 10^{-1}, (preconditioned with RILUD)
- RILUD = one application of an RILUD preconditioner.

Speedups are compared both to single and multiblock serial computations. The motivation for this is that sometimes one needs domain decomposition for geometric reasons, and sometimes as a means of parallelism.

4.2.1 Single block serial case

The single block serial solution times in seconds on grids of dimension $n = 60$, 120, 180, 240 and 300 are, on the HP cluster:

	$n = 60$	120	180	240	300
GMR6	0.788	7.56	28.1	82.5	195
GMR2	0.862	8.00	34.7	75.8	180
GMR1	0.815	6.75	29.3	82.1	166
RILUD	1.10	11.0	41.6	117	292

and on the Cray T3E:

	$n = 60$	120	180	240	300
GMR6	0.483	3.98	11.9	34.7	80.6
GMR2	0.563	4.24	14.8	32.0	74.9
GMR1	0.552	3.62	13.2	35.4	69.3
RILUD	0.666	5.49	17.2	49.9	119

Note that GCR preconditioned with GMRES iterations gives a variation of the GMRESR method of [40]. All three lead to approximately the same solution time. This is in agreement with the findings of [40–42] for the GMRESR method. The fourth case is equivalent to solving the problem with GCR, preconditioned with the RILUD preconditioner. It is also in keeping with the findings of the above papers that this method is slower than GMRESR.

4.2.2 Multiblock solution, fixed problem size

In this section we compare results for a fixed problem size on the 300×300 grid with 4 and 9 processors on the workstation cluster and 4, 9, 16 and 25 processors on the Cray T3E. We use one processor per block. The timing results in seconds are, for the HP cluster:

	$p = 4$	$p = 9$
GMR6	1430	386
GMR2	346	220
GMR1	457	261
RILUD	157	89

and for the Cray T3E:

	$p = 4$	$p = 9$	$p = 16$	$p = 25$
GMR6	685	178	143	79.3
GMR2	167	102	63.3	37.1
GMR1	222	118	65.6	38.9
RILUD	65.3	25.9	21.9	14.9

On both systems one observes that the method using RILUD as the subdomain approximation gives a faster computation time than the fastest serial computation times from the previous subsection. On the Cray T3E, the meth-

ods GMR1 and GMR2 are also somewhat faster than the fastest serial time, for $p = 16$ and $p = 25$ processors.

Furthermore, one sees that among those methods in which GMRES is used to solve the subdomain problems, a tolerance of 10^{-2} gives a faster solution time than a tolerance of 10^{-1}. Thus some subdomain convergence appears to be desirable. On the other hand, the fastest solutions in each case are obtained with the least accurate subdomain approximation—namely, the RILUD preconditioner.

To give insight into these results, it is useful to look at the iteration counts: both the number of outer iterations and the average number of inner iterations (in parentheses).

	$p = 4$	$p = 9$	$p = 16$	$p = 25$
GMR6	78(68.4)	83(38.7)	145(31.4)	168(26.4)
GMR2	86(15.7)	118(15.7)	168(13.7)	192(10.9)
GMR1	139(13.6)	225(9.3)	287(7.1)	303(5.9)
RILUD	341(1)	291(1)	439(1)	437(1)

Note the large increase in the number of outer iterations incurred for GMR1 over GMR2, which helps to explain the faster time for GMR2. Apparently, an inner loop tolerance of 10^{-1} is insufficient for fast global convergence, yet is still a very expensive subdomain approximation. The RILUD approximation, on the other hand, though it gives the worst convergence rate of the outer loop, is very cheap to apply; in fact, cheap enough to make it the fastest method.

Figure 8 illustrates the speedup against the multiblock serial solution, obtained on the workstation cluster and on the Cray using GMR6, GMR2, GMR1 and RILUD subdomain approximations. We would expect nearly perfect speedup, especially for large problems, since the work required for preconditioning is proportional to the total number of unknowns, while the amount of communication is proportional to the length of subdomain interfaces. The observed speedup is quite good on the Cray; however, on the workstation cluster, especially for $p = 9$ the subdomain grid size needs to be quite large to obtain a high speedup. In any case we can conclude that if domain decomposition is going to be used anyway for geometric reasons or due to memory limitations, a speedup can be achieved by parallelization and subdomain approximation with an RILUD preconditioner.

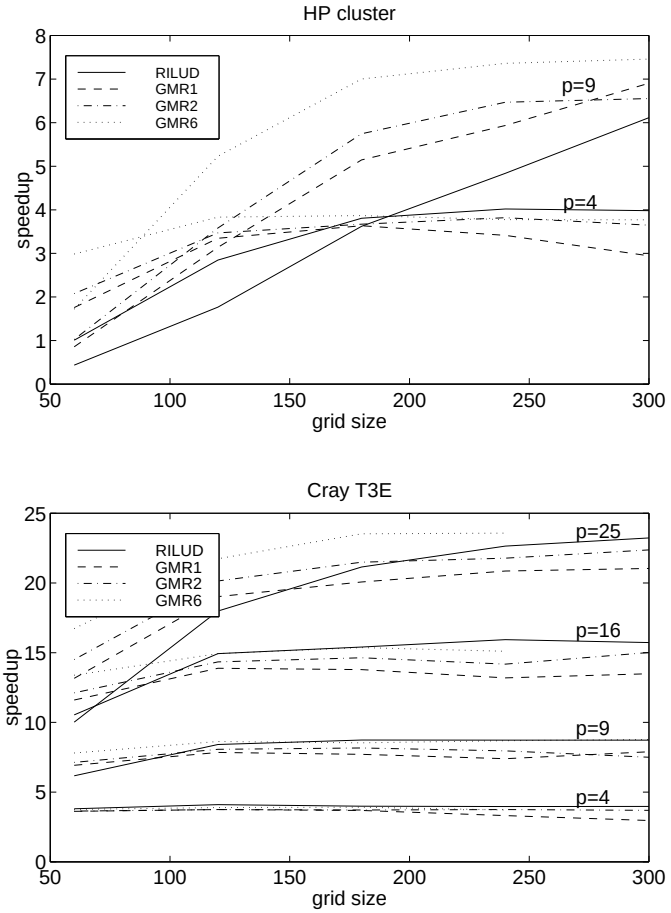

Fig. 8. Speedup vs. multiblock solution on the cluster of workstations and the Cray T3E.

4.2.3 Multiblock case, scaled problem size

Figure 9 shows a comparison of the parallel scalability of the domain decomposition method with approximate subdomain solution. The figure shows computation times on 1, 4 and 9 processors (1, 4, 9, 16 and 25 processors for the Cray T3E) with a fixed subdomain size of 120×120. A fixed number of outer iterations (30) were computed. Note that the method scales almost perfectly on the Cray for this range of processors. On the workstation cluster, the scaling is somewhat poorer, but reasonable.

5 Conclusions

For applications which require domain decomposition for some reason other than parallelism, it is possible to achieve a great reduction of computation time by solving subdomain problems approximately. A reasonable speedup with respect to the single block serial solution method is also attainable, particularly

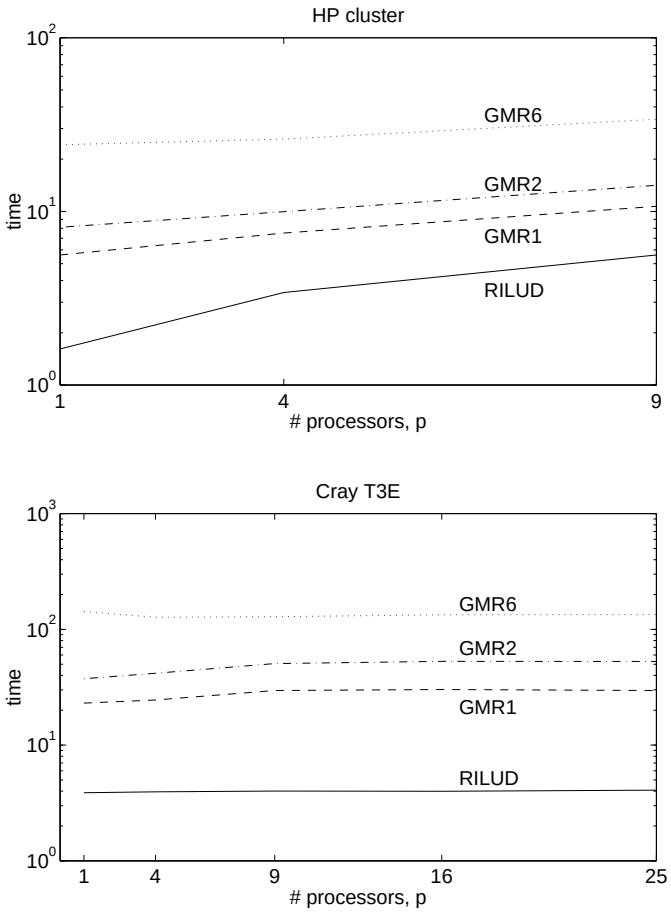

Fig. 9. Computation time for fixed subdomain size of 120×120.

when using many processors of a massively parallel distributed memory machine. This speedup is less impressive when computing on a cluster of workstations, due to the increased communication latency.

In our experience, the best subdomain approximation method in parallel is a simple incomplete factorization restricted to the diagonal: the RILUD factorization. With this preconditioner used as a subdomain approximation, the approximate solves become so cheap (and yet sufficiently accurate) that they offset the increased number of global iterations resulting from inaccurate subdomain solution.

A performance model for the modified Gram-Schmidt, re-orthogonalized classical Gram-Schmidt, and Householder orthogonalization methods indicates that classical Gram-Schmidt and Householder require approximately the same amount of work and communication, making the classical Gram-Schmidt more attractive, since it is easier to implement and more stable. The Householder and re-orthogonalized Gram-Schmidt methods are most effective for relatively small problems: using nine processors, up to about 900 unknowns per processor for a Cray T3E, or 8000 unknowns per processor for a cluster of workstations.

One promising area of application for these procedures is in long-time simulations of systems of this size.

References

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

[2] Z. Bai, D. Hu, and L. Reichel. A Newton-basis GMRES implementation. *IMA Journal of Numerical Analysis*, 14:563–581, 1994.

[3] P. v. Beek, R. R. P. van Nooyen, and P. Wesseling. Accurate discretization on non-uniform curvilinear staggered grids. *Journal of Computational Physics*, 117:364–367, 1995.

[4] A. Björck. Solving linear least squares problems by Gram-Schmidt orthogonalization. *BIT*, 7:1–21, 1967.

[5] C. Börgers. The Neumann-Dirichlet domain decomposition method with inexact solvers on the subdomain. *Numerische Mathematik*, 55:123–136, 1989.

[6] E. Brakkee. *Domain Decomposition for the Incompressible Navier-Stokes Equations*. PhD thesis, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands, Apr. 1996.

[7] 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.

[8] E. Brakkee, C. Vuik, and P. Wesseling. Domain decomposition for the incompressible Navier-Stokes equations: Solving subdomain problems accurately and inaccurately. *International Journal for Numerical Methods in Fluids*, 1998. To appear.

[9] T. F. Chan and T. P. Mathew. Domain decomposition algorithms. In A. Iserles, editor, *Acta Numerica*, pages 61–143. Cambridge University Press, 1994.

[10] M. Y.-M. Chang and M. H. Schultz. Bounds on block diagonal preconditioning. *Parallel Algorithms and Applications*, 1:141–164, 1993.

[11] H. Cheng. On the effect of using inexact solvers for certain domain decomposition algorithms. *East-West J. Numer. Math.*, 2(4):257–284, 1994.

[12] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM Journal on Numerical Analysis*, 20(2):345–357, Apr. 1983.

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

[14] S. Goossens, E. Issman, G. Degrez, and D. Roose. Block $ILP^{-1}U(0)$ preconditioning for a GMRES based Euler/Navier-Stokes solver. In H. Liddell, A. Colbrook, B. Herzberger, and P. Sloot, editors, *High Performance Computing and Networking '96*, Lecture Notes in Computer Science 1067, pages 619–626. Springer-Verlag, 1996.

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

[16] G. Haase and U. Langer. The efficient parallel solution of PDEs. *Computers Math. Applic.*, 31(4/5):151–159, 1996.

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

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

[19] G. Haase, U. Langer, and A. Meyer. Domain decomposition preconditioners with inexact subdomain solvers. *Journal of Numerical Linear Algebra with Applications*, 1(1):27–41, 1991.

[20] R. W. Hockney and C. R. Jesshope. *Parallel Computers 2: Architecture, Programming and Algorithms*. Adam Hilger, Bristol, 1988.

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

[22] W. Jalby and B. Philippe. Stability analysis and improvement of the block Gram-Schmidt algorithm. *SIAM Journal of Scientific and Statistical Computing*, 12(5):1058–1073, 1991.

[23] J. v. Kan. A second-order accurate pressure-correction scheme for viscous incompressible flow. *SIAM Journal on Scientific and Statistical Computing*, 7(3):870–891, 1986.

[24] D. E. Keyes. Aerodynamic applications of Newton-Krylov-Schwarz solvers. In S. M. Despande, S. S. Desai, and R. Narasimha, editors, *Fourteenth International Conference on Numerical Methods in Fluid Dynamics*, pages 1–20. Springer, Berlin, 1995.

[25] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing; Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, 1994.

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

[27] 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. *Mathematics of Computation*, 31:148–162, 1977.

[28] A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, 45:217–234, 1990.

[29] D. P. O'Leary and P. Whitman. Parallel QR factorization by Householder and modified Gram-Schmidt algortihms. *Parallel-Computing*, 16:99–112, 1990.

[30] G. Radicati and Y. Robert. Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric linear systems on a vector multiprocessor. *Parallel Computing*, 11:223–239, 1989.

[31] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific and Statistical Computing*, 14:461–469, 1993.

[32] Y. Saad and M. H. Schultz. GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, July 1986.

[33] A. Segal, P. Wesseling, J. van Kan, C. Oosterlee, and K. Kassels. Invariant discretization of the incompressible Navier-Stokes equations in boundary-fitted co-ordinates. *International Journal for Numerical Methods in Fluids*, 15:411–426, 1992.

[34] R. B. Sidje. Alternatives for parallel Krylov subspace basis computation. *Numerical Linear Algebra with Applications*, 4(4):305–331, 1997.

[35] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996.

[36] E. d. Sturler. Incomplete block LU preconditioners on slightly overlapping subdomains for a massively parallel computer. *Applied Numerical Mathematics*, 19:129–146, 1995.

[37] E. d. Sturler and H. A. v. d. Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics*, 18:441–459, 1995.

[38] K. H. Tan. *Local Coupling in Domain Decomposition.* PhD thesis, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, The Netherlands, Apr. 1996.

[39] E. F. v. d. Velde. Domain decomposition vs. concurrent multigrid. Technical Report Caltech 217-50, Center for Research on Parallel Computation, California Institute of Technology, 1994.

[40] H. A. v. d. Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Numerical Linear Algebra with Applications*, 1(4):369–386, 1994.

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

[42] C. Vuik. New insights in GMRES-like methods with variable preconditioners. *Journal of Computational and Applied Mathematics*, 61:189–204, 1995.

[43] C. Vuik. Fast iterative solvers for the discretized incompressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 22:195–210, 1996.

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

[45] T. Washio and K. Hayami. Parallel block preconditioning based on SSOR and MILU. *Numerical Linear Algebra with Applications*, 1(6):533–553, 1994.

[46] P. Wesseling, A. Segal, C. G. M. Kassels, and H. Bijl. Computing flows on general two-dimensional staggered grids. *Journal of Engineering Mathematics*, 1998. To appear.