ELSEVIER

Applied Numerical Mathematics 30 (1999) 403–423

APPLIED NUMERICAL MATHEMATICS

www.elsevier.nl/locate/apnum

Parallel implementation of a multiblock method with approximate subdomain solution

J. Frank [*], C. Vuik

Delft University of Technology, Faculty of Information Technology and Systems, Department of Technical Mathematics, P.O. Box 5031, 2600 GA Delft, 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 approximate solutions via (1) an inner preconditioned GMRES iteration to fixed tolerance, and (2) incomplete factorization (RILU, restricted to the diagonal) are considered. Numerical experiments for a fundamental test problem are included which show speedups obtained on a cluster of workstations as well as on a distributed memory parallel computer. Additionally, the parallel implementation of GCR is addressed, with particular focus on communication costs associated with orthogonalization processes. This consideration brings up questions concerning the use of Householder reflections with GCR. © 1999 Elsevier Science B.V. and IMACS. All rights reserved.

Keywords: Domain decomposition; Approximate subdomain solution; Parallel Krylov subspace methods; Orthogonalization methods

1. Introduction

Domain decomposition arises naturally in computational fluid dynamics applications on structured grids: complicated geometries are broken down into (topologically) rectangular regions and discretized in general coordinates, see, e.g., [30,39], applying domain decomposition to iteratively arrive at the solution on the global domain. This approach provides easy exploitation of parallel computing resources, and additionally offers a solution to memory limitation problems.

This paper addresses the parallel implementation of a domain decomposition method for the DeFT Navier–Stokes solver described in [30], and is the continuation of work summarized in [7]. Results from a parallel implementation of a Krylov-accelerated Schur complement domain decomposition method are presented in [5]. A serial implementation of nonoverlapping, one-level additive Schwarz method with approximate subdomain solution [6] gave more promising results. For the present research, our goal was

[*] Corresponding author.

to obtain an impression of the behavior of this method in parallel without incurring the programming workload of a full implementation in the DeFT software; which would require fundamental changes. To this end, we report preliminary results for a Poisson problem on a square domain, and refer the reader to a forthcoming article with more realistic experiments. The Poisson problem is representative of the system which must be solved for the pressure correction method used in DeFT.

Theoretical results on approximate solution of subdomain problems for Schur complement domain decomposition methods are given by Börgers [4], Haase et al. [15–17,26], and Cheng [9]. Brakkee [7] gives theoretical and experimental results for non-overlapping Schwarz iterations with variable approximate inner solvers.

In this paper we demonstrate that a reasonable amount of parallel 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 theoretical motivation for approximate subdomain solution.

Much effort has focused on efficient parallelization of Krylov subspace methods. Aside 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 may be very efficient. Inner products, on the other hand, require global communications; therefore, the focus has been on reducing the number of inner products [12,31], overlapping inner product communications with computation [10], or increasing the number of inner products that can be computed with a single communication [2,24].

Some practical points are brought out in Section 3 concerning parallel implementation of orthogonalization procedures for the GCR method. A performance model is developed for comparison of these methods, and the validity of the model is checked against experimental results in Section 4.

Additional results reported in Section 4 include speedup ratios, obtained by comparison of the parallel multiblock computation times to both the single block serial time and the multiblock serial times, and scalability tests for which the number of unknowns per processor is held constant as the number of participating processors is increased. 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. One-level, 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 $x, b \in \mathbb{R}^N$. 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 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 which share a common subdomain to produce a block system:

$$\begin{bmatrix} A_{11} & \dots & A_{1M} \\ \vdots & \ddots & \vdots \\ A_{M1} & \dots & A_{MM} \end{bmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_M \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_M \end{pmatrix}. \tag{2}$$

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, together with the residual, defines a system whose solution provides an approximation of the error. Note that this system may be efficiently solved on parallel computers. It is this form of domain decomposition which we will consider in the rest of the paper.

For a thorough discussion of domain decomposition methods see the book [32] and the review article [8]. Each of these publications contains an extensive bibliography. Convergence theory for domain decomposition methods is discussed in [32]. 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.

2.2. *Krylov subspace acceleration*

In practice (2) 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 [29] for nonsymmetric problems. For our purposes a practical method is GCR [11], shown in Fig. 1. In the algorithm and elsewhere in this paper the Euclidean inner product $\langle x, y\rangle = x^{\mathrm{T}}y$ and associated norm $\|x\| = (x^{\mathrm{T}}x)^{1/2}$ are used.

The function orthonorm() takes input vectors $\widetilde{q}$ and $\widetilde{v}$, orthonormalizes $\widetilde{q}$ with respect to the q_i, $i < k$, updating $\widetilde{v}$ as necessary to preserve the relation $\widetilde{q} = A\widetilde{v}$, and returns the modified vectors q_k and v_k. In serial computations, the modified Gram–Schmidt method, Fig. 2, is often employed for the orthonorm() function. We discuss alternative orthogonalization methods in later sections of this paper.

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 Lanczos recursion, but instead requires the storage of an extra set of orthogonal *residual search vectors*. GCR has a number of benefits;

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

Fig. 1. The GCR algorithm.

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

Fig. 2. The modified Gram–Schmidt algorithm.

among them: (1) the preconditioner K need not remain constant (nor even be a linear operator; the GMRESR algorithm in [34] uses GMRES(m) as a preconditioner); (2) one is free to employ truncation strategies such as in [35]; and (3) if the LSQR switch is employed [34], the method will not break down. The importance of allowing a variable preconditioner will be discussed in the next section. In Fig. 1 the GCR method is defined for an unlimited number of iterations, and may incur memory limitations. In practice, therefore, 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 [35] to be very effective in reducing the number of iterations. In the numerical experiments of this paper we do not use any truncation strategies, but simply restart; however, truncation is used in the DeFT software.

2.3. Approximate subdomain solution

Solution for $\widetilde{v}$ from the preconditioning equation $K\widetilde{v} = r_{k-1}$ in the GCR algorithm requires solution of M subdomain systems $A_{mm}\widetilde{v}_m = r_m$, $m = 1, \ldots, M$. Since these problems have a nonzero structure similar to that of the original matrix A, and since they may still be quite large, it is advantageous to solve them using an iterative method. A question which arises naturally, and for purely practical reasons, addresses the tolerance to which these *inner iterations* should converge. Perhaps a very rough approximation would be sufficient. A number of authors have considered approximate solution of 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:

- A second (inner) iterative method (possibly preconditioned) either to a fixed tolerance, to a variable tolerance, or for a predetermined number of iterations.
- Approximate factorization or approximate inversion of the subdomain problems.
- 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 [33] shows that if the inner problems are solved to some tolerance in each outer iteration, then 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.

Brakkee [7] has proven the following theorem. Let $\widetilde{A}_{ii}^{-1}$ 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 $\widetilde{K}^{-1}$ to be the approximate preconditioner consisting of the diagonal blocks $\widetilde{A}_{ii}^{-1}$. If for each subdomain $i = 1, \dots, M$ it holds that $\|I - A_{ii}\widetilde{A}_{ii}^{-1}\| < \varepsilon$, then the condition number of the approximately preconditioned matrix satisfies

$$\kappa\big(A\widetilde{K}^{-1}\big) \leqslant \frac{1+\varepsilon}{1-\varepsilon}\kappa\big(AK^{-1}\big). \tag{3}$$

where $\kappa(A) = \|A\|\|A^{-1}\|$ is the condition number of A. Unfortunately, the condition $\|I - A_{ii}\widetilde{A}_{ii}^{-1}\| < \varepsilon$ is nontrivial to check.

Essential to the proof of the above theorem is the fact that

$$\kappa\big(A\widetilde{K}^{-1}\big) = \kappa\big(AK^{-1}K\widetilde{K}^{-1}\big) \leqslant \kappa\big(AK^{-1}\big)\kappa\big(K\widetilde{K}^{-1}\big).$$

This bound may be clarified by noting that the matrix $B = K\widetilde{K}^{-1}$ is a block diagonal matrix with blocks $B_i = A_{ii}\widetilde{A}_{ii}^{-1}$, $i = 1, \dots, M$. The spectrum of the block diagonal matrix $B^{\mathrm{T}}B$ is a subset of the union of the spectra of the blocks; thus, if there exist α, β bounding the singular values of all blocks: $0 < \alpha \leqslant \min_i \sigma_{\min}(B_i) \leqslant \max_i \sigma_{\max}(B_i) \leqslant \beta$, then

$$\kappa\big(A\widetilde{K}^{-1}\big) \leqslant \frac{\beta}{\alpha}\kappa\big(AK^{-1}\big).$$

Competing against convergence rate for an efficient solution method is the expense of computing the subdomain approximate solutions. The RILUD preconditioner, though a less effective approximate solver than GMRES iterations in terms of convergence rate, is far cheaper, at least for the problems considered here. Thus one makes a tradeoff between effectiveness of an approximate preconditioner in terms of convergence rate and speed in terms of computational expense.

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 GMRES: namely, the

Lanczos recurrence relation no longer holds. To allow the use of a variable preconditioner, Saad [28] has developed the Flexible GMRES (FGMRES) method, which requires storage of an auxiliary set of vectors such as with GCR. However, it is not possible to use truncation strategies with FGMRES. Because we use truncation in our Navier–Stokes code, we consider GCR in the following.

Our choice of approximate solution methods is motivated by the results obtained in [6]. In that paper, GMRES was used as to approximately solve subdomain problems to within fixed tolerances of 10^{-4}, 10^{-3}, 10^{-2} and 10^{-1}. Additionally, a blockwise application of the RILUD preconditioner was used. RILUD, a diagonal-restricted variant of the preconditioner introduced in [1], is a weighted average of an ILUD preconditioner [25] and an MILUD preconditioner [14]. The weighting parameter ω, was assigned a value of 0.95 in our experiments. See also [37] 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 [22] and Goossens et al. [13] among others. The results of [6] indicated that coarser tolerances were more effective. However, all numerical results presented therein were obtained from serial runs. In Section 4 we will present numerical results using the above approximate subdomain solution methods in parallel.

2.4. *Orthogonalization methods*

The primary challenges to parallelization of GCR 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. Much of the literature on parallel Krylov subspace methods and parallel orthogonalization methods is focused on orthogonalizing a number of vectors simultaneously. See, e.g., [2,10,20,24,27]. 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 Fig. 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, Fig. 3. In this algorithm all necessary inner products can be computed with a single global communication.

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

Fig. 3. The classical Gram–Schmidt algorithm.

Unfortunately, as shown by Björck [3], the classical Gram–Schmidt method is unstable with respect to rounding errors, so this method is rarely used.

On the other hand, Hoffmann [19] gives experimental evidence indicating that a two-fold application of Fig. 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 [38]. 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 [38] 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.

3.1. Description of the method

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 $A \in \mathbb{R}^{n\times m}$, $m \leqslant n$, 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 $a_k = Qz_k$. It follows that $a_k \in \operatorname{span}\{q_1, \dots, q_k\}$. In other words, 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 have the following properties:

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

In property (ii) e_j is the jth canonical unit vector in $\mathbb{R}^n$. A Householder reflection is given by $P_i = I - 2w_i w_i^{\mathrm{T}}/(w_i^{\mathrm{T}} w_i)$, 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 $q_1, \dots, q_k$ and stored them along with the transformation vectors $w_1, \dots, w_k$ corresponding to $P_1, \dots, P_k$. Given a candidate vector a_{k+1}, one must first apply the previous reflections as described in [38]:

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

where here and elsewhere we denote by W_k the matrix whose columns are $w_1, \dots, w_k$, and where

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

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.

Now having computed $\widetilde{a}$ one wishes to find w_{k+1} such that P_{k+1} satisfies (iii):

$$P_{k+1}\widetilde{a} = z_{k+1} = z_{k+1}^{(1)}e_1 + \cdots + z_{k+1}^{(k+1)}e_{k+1} = \widetilde{a}^{(1)}e_1 + \cdots + \widetilde{a}^{(k)}e_k + \alpha e_{k+1}, \tag{4}$$

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

Because of the relation

$$P_{k+1}\widetilde{a} = \left(I - 2\frac{w_{k+1}w_{k+1}^{\mathrm{T}}}{w_{k+1}^{\mathrm{T}}w_{k+1}}\right)\widetilde{a} = \widetilde{a} - 2\frac{w_{k+1}^{\mathrm{T}}\widetilde{a}}{w_{k+1}^{\mathrm{T}}w_{k+1}}w_{k+1}, \tag{5}$$

one must have $w_{k+1} \in \operatorname{span}\{\widetilde{a}, e_1, \ldots, e_{k+1}\}$. However, Eq. (4) provides the relation which must hold among $\widetilde{a}, e_1, \ldots, e_k$. Let $\widetilde{w}$ be the vector obtained by setting the first k elements of $\widetilde{a}$ to zero. Formally, one has $\widetilde{w} = J_{k+1}\widetilde{a}$, where

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

Thus, $w_{k+1} \in \operatorname{span}\{\widetilde{w}, e_{k+1}\}$. The length of w_{k+1} is a free parameter, so take $w_{k+1} = \widetilde{w} + \beta e_{k+1}$. Substituting into (5) gives

$$P_{k+1}\widetilde{a} = \widetilde{a} - 2\frac{w_{k+1}^{\mathrm{T}}\widetilde{a}}{w_{k+1}^{\mathrm{T}}w_{k+1}}\left(J_{k+1}\widetilde{a} + \beta e_{k+1}\right) = \left(I - 2\frac{w_{k+1}^{\mathrm{T}}\widetilde{a}}{w_{k+1}^{\mathrm{T}}w_{k+1}}J_{k+1}\right)\widetilde{a} - 2\beta\frac{w_{k+1}^{\mathrm{T}}\widetilde{a}}{w_{k+1}^{\mathrm{T}}w_{k+1}}e_{k+1}.$$

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

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

and

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

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

$$w_{k+1} = \widetilde{w} + \operatorname{sign}\left(\widetilde{w}^{(k+1)}\right)\|\widetilde{w}\|e_{k+1}.$$

In practice, the w_k are normalized to length one. Since α is the $(k+1)$th component of $P_{k+1}\widetilde{a}$, substitution of the above relation into (5), and noting that

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

gives

$$\alpha = \widetilde{a}^{(k+1)} - \widetilde{w}^{(k+1)} - \operatorname{sign}\left(\widetilde{w}^{(k+1)}\right)\|\widetilde{w}\| = -\operatorname{sign}\left(\widetilde{w}^{(k+1)}\right)\|\widetilde{w}\| = -\beta,$$

and the length of w_{k+1} can be expressed as

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

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 (4) by $P_1 \cdots P_{k+1}$ gives

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

from which it follows that

$$q_{k+1} = \frac{1}{\alpha} \left[a_{k+1} - \sum_{i=1}^{k} \widetilde{a}^{(i)} q_i \right]. \tag{6}$$

Within the GCR algorithm, the same linear combination must be applied to the v_i to obtain v_{k+1}. As pointed out by an anonymous referee, use of Eq. (6) may result in subtractive cancellation, causing the reduced stability of this Householder implementation observed in the next section. However, a relation of this form is needed to be able to enforce the identity $q_{k+1} = Av_{k+1}$, necessary for GCR.

Our implementation requires three communications in the $(k+1)$th iteration, namely:

(1) The computation of $\widetilde{a}$ 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 $\widetilde{a}$.
(3) A communication is required to compute $\|\widetilde{w}\|$ for determining α.

The algorithm is shown in Fig. 4, with $\widetilde{q}$ 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 used for orthogonalization.

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 one floating point operation. Similar computation/communication models are used, for example, in [10,18,23,27].

Let p denote the number of processes, and define a function $f(p)$ which gives the maximum number of non-simultaneous sends necessary for any given process participating in a broadcast operation to $p-1$ processes. The function $f(p)$ is machine-dependent and also dependent on the distribution of processes

```
Algorithm: Householder orthogonalization
[$q_k, v_k$] = 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
      $L_{k-1} = \begin{bmatrix} L_{k-2} & 0 \\ 2w_{k-1}^{\mathrm{T}} W_{k-2} & 1 \end{bmatrix}$
    end
    $y = W_{k-1}^{\mathrm{T}} \tilde{q}$
    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 $w_k^{(i)} = 0, i = 1, \ldots, k-1$
  $\alpha = -\mathrm{sign}(\tilde{a}^{(k)}) \| w_k \|$
  $w_k^{(k)} = w_k^{(k)} - \alpha$
  $q_k = q_k / \alpha$
  $v_k = v_k / \alpha$
  $w_k = w_k / \sqrt{2\alpha(\alpha - \tilde{a}^{(k)})}$
return
```

Fig. 4. The Householder orthogonalization algorithm.

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, see Table 1.

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 given in Table 2. 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.

Table 1

Operation	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

Table 2
Number of operations in the kth iteration of GCR

Modified Gram–Schmidt	k SIP(1)
	$2k - 1$ axpy
Re-orthogonalized CGS	2 SIP(k)
	$3k - 1$ axpy
Householder	SIP($2k - 3$)
	SIP(1)
	FS($k - 1$)
	$3k - 3/2$ axpy
	1 $B(p,k)$

Based on the communication model outlined in Tables 1 and 2 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

$$t_{\mathrm{MGS}} = \frac{s(s+1)}{2}\left[6n^2\phi + 2f(p)(t_0 + \beta)\right] - s\left(2n^2\phi\right), \tag{7}$$

$$t_{\mathrm{CGS2}} = \frac{s(s+1)}{2}\left[10n^2\phi + 4f(p)\beta\right] + s\left[4f(p)t_0 - 2n^2\phi\right], \tag{8}$$

$$\begin{aligned} t_{\mathrm{HH}} = \phi\frac{s(s+1/2)(s+1)}{3} + \frac{s(s+1)}{2}\left[(10n^2 - 2)\phi + 5f(p)\beta\right] \\ + s\left[f(p)(6t_0 - 4\beta) - (7n^2 - 1)\phi\right]. \end{aligned} \tag{9}$$

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 (8) 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 (7)–(9), and assuming $f(p) = p - 1$ for the workstation cluster and $f(p) = \lceil \log_2 p \rceil$ for the Cray T3E, the quantities

$$\mathcal{F}_{\mathrm{HH}} = \frac{\text{orthog. time MGS}}{\text{orthog. time HH}}, \tag{10}$$

$$\mathcal{F}_{\mathrm{CGS2}} = \frac{\text{orthog. time MGS}}{\text{orthog. time CGS2}} \tag{11}$$

are plotted as functions of n for $s = 60$ and $p = 4, 9$ ($p = 4, 9, 25$ for the Cray T3E) in Fig. 5. The Householder (respectively CGS2) method is faster at points in the figure where $\mathcal{F}_{\mathrm{HH}} > 1$ (respectively $\mathcal{F}_{\mathrm{CGS2}} > 1$). 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 lower than the ones predicted here. Another issue of relevance to the choice of an orthogonalization method is the stability of the method with respect to rounding errors. Fig. 6 shows a comparison of the classical, modified, and re-orthogonalized classical Gram–Schmidt methods and the Householder implementation for the test matrix of [3]:

$$A = \begin{bmatrix} 1 & 1 & \cdots \\ \varepsilon & & \\ & \varepsilon & \\ & & \ddots \end{bmatrix}.$$

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

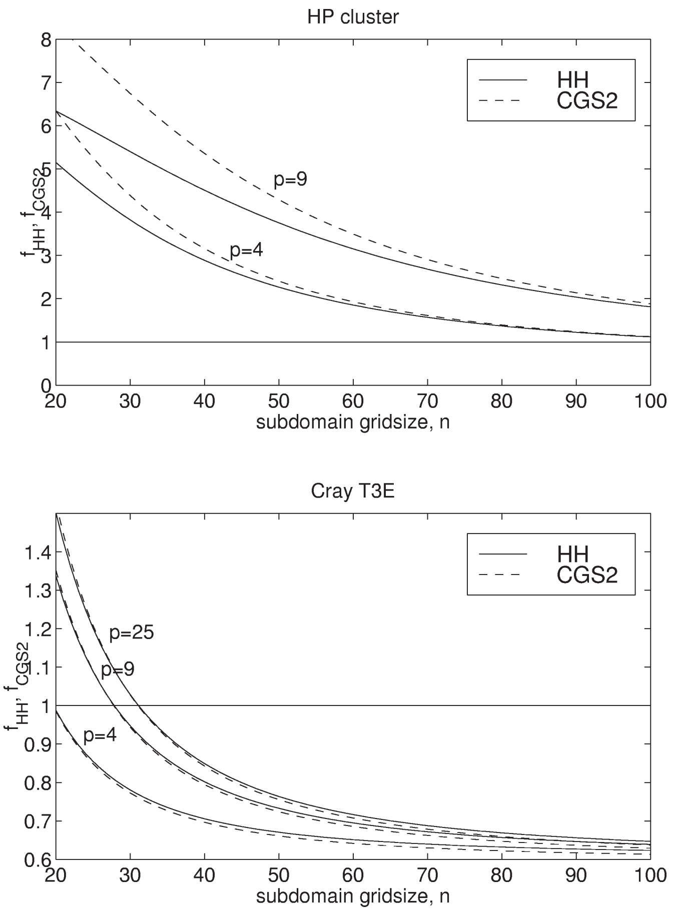


Fig. 5. Predicted speedup with Householder orthogonalization.

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.

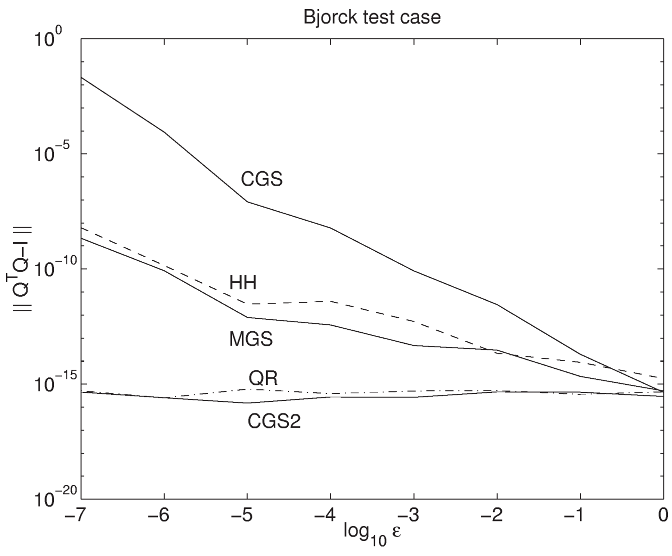


Fig. 6. Comparison of orthogonalization error for classical (CGS), modified (MGS), re-orthogonalized (CGS2) Gram–Schmidt methods, Householder (HH) method, and Matlab QR function on Björck test problem.

As a test example, we consider a Poisson problem, discretized with the finite volume method on a square domain. The pressure correction matrix, which we solve in each time step of an incompressible Navier–Stokes simulation to enforce the divergence-free constraint [21], is similar to a Poisson problem, but with asymmetry arising from the use of curvilinear coordinates. Solution of this system requires about 75% of the computing effort. So that we can obtain a useful indication of the performance of our method on the pressure correction matrix, we do not exploit the symmetry of the Poisson 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

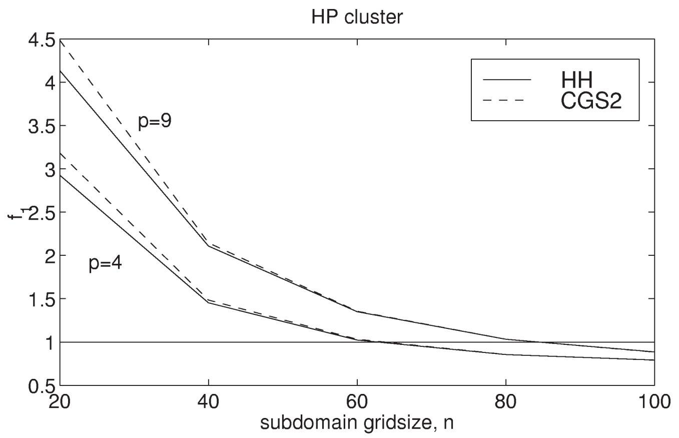
$$4u_{ij} - u_{i+1j} - u_{i-1j} - u_{ij-1} - u_{ij+1} = h^2 f_{ij}.$$

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.

4.1. Evaluation of performance model, Householder orthogonalization

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.

Fig. 7 is the experimental analog of Fig. 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 ($M = 2$ and 3, respectively) on the HP cluster and 4, 9 and 25 processors ($M = 2, 3$ and 5, respectively) on the Cray T3E.



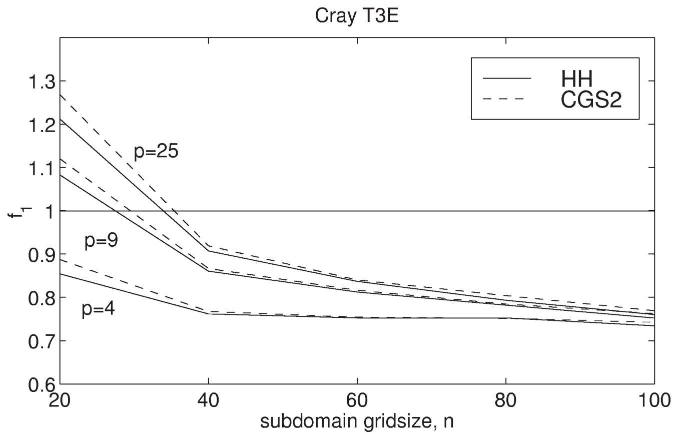


Fig. 7. Measured speedup with Householder (HH) orthogonalization and re-orthogonalized classical Gram–Schmidt (CGS2), restart value $s = 60$.

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.

4.2. *Evaluation of approximate subdomain solvers*

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 as follows:

- GMR6 = restarted GMRES with a tolerance of 10^{-6} (preconditioned with RILUD),

Table 3

	$n = 60$	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

Table 4

	$n = 60$	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

- 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.

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 for the HP cluster and for the Cray T3E are given in Tables 3 and 4, respectively. Note that GCR preconditioned with GMRES iterations gives a variation of the GMRESR method of [34]. All three lead to approximately the same solution time. This is in agreement with the findings of [34–36] 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 for the HP cluster and for the Cray T3E are given in Tables 5 and 6, respectively.

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 methods 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

Table 5

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

Table 6

	$p=4$	$p=9$	$p=16$	$p=25$
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

Table 7

	$p=4$	$p=9$	$p=16$	$p=25$
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)

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), see Table 7. 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.

Fig. 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

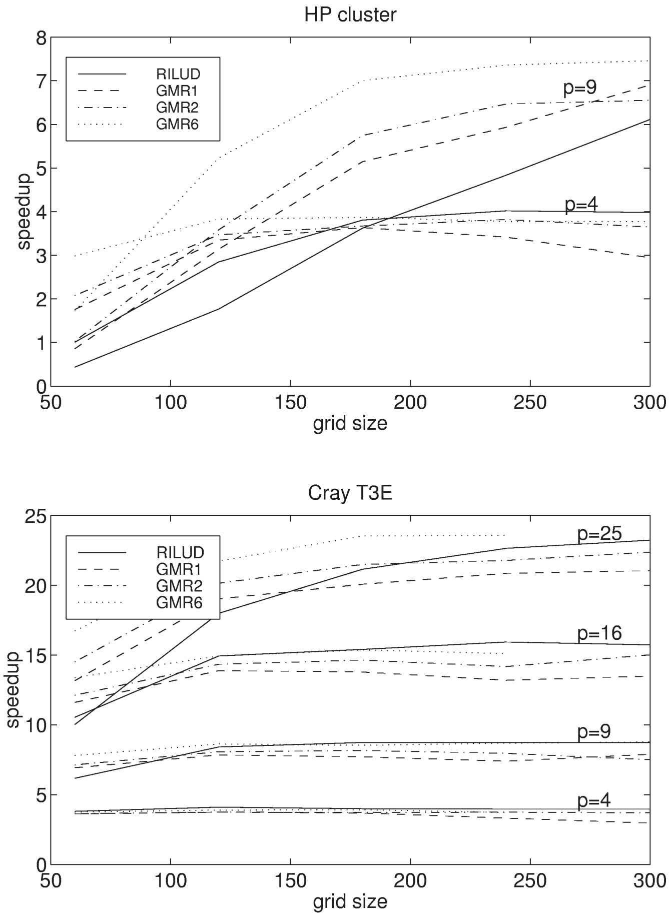


Fig. 8. Speedup versus multiblock solution on the cluster of workstations and the Cray T3E.

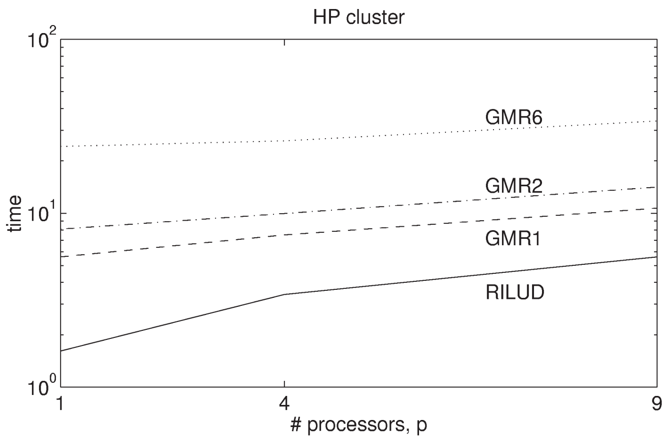
for geometric reasons or due to memory limitations, a speedup can be achieved by parallelization and subdomain approximation with an RILUD preconditioner.

4.2.3. Multiblock case, scaled problem size

Fig. 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



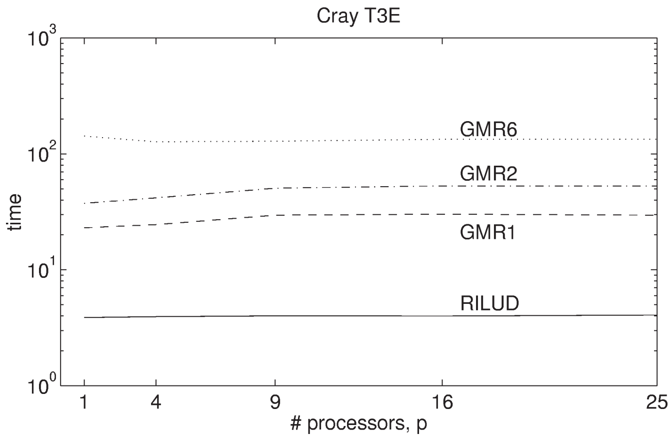


Fig. 9. Computation time for fixed subdomain size of 120×120.

approximately. A reasonable speedup with respect to the single block serial solution method is also attainable, particularly 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.

Acknowledgements

The authors are indebted to Pieter Wesseling and an anonymous referee for critical remarks on the manuscript.

References

[1] O. Axelsson and G. Linskog, On the eigenvalue distribution of a class of preconditioning methods, *Numer. Math.* 48 (1986) 479–498.
[2] Z. Bai, D. Hu and L. Reichel, A Newton-basis GMRES implementation, *IMA J. Numer. Anal.* 14 (1994) 563–581.
[3] A. Björck, Solving linear least squares problems by Gram–Schmidt orthogonalization, *BIT* 7 (1967) 1–21.
[4] C. Börgers, The Neumann–Dirichlet domain decomposition method with inexact solvers on the subdomain, *Numer. Math.* 55 (1989) 123–136.
[5] 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 (1995) 185–205.
[6] E. Brakkee, C. Vuik and P. Wesseling, Domain decomposition for the incompressible Navier–Stokes equations: Solving subdomain problems accurately and inaccurately, *Internat. J. Numer. Methods Fluids* 26 (1998) 1217–1237.
[7] E. Brakkee, Domain decomposition for the incompressible Navier–Stokes equations, Ph.D. Thesis, Delft University of Technology, Delft, The Netherlands (April 1996).
[8] T.F. Chan and T.P. Mathew, Domain decomposition algorithms, in: A. Iserles, ed., *Acta Numerica* (University Press, Cambridge, 1994) 61–143.
[9] H. Cheng, On the effect of using inexact solvers for certain domain decomposition algorithms, *East–West J. Numer. Math.* 2 (4) (1994) 257–284.
[10] E. de Sturler and H.A. van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, *Appl. Numer. Math.* 18 (1995) 441–459.
[11] S.C. Eisenstat, H.C. Elman and M.H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, *SIAM J. Numer. Anal.* 20 (2) (1983) 345–357.
[12] J. Erhel, A parallel GMRES version for general sparse matrices, *Electronic Trans. Numer. Anal.* 3 (1995) 160–176 (`http://etna.mcs.kent.edu`).
[13] 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, eds., *Proceedings High Performance Computing and Networking '96*, Lecture Notes in Computer Science, Vol. 1067 (Springer, New York, 1996) 619–626.
[14] I. Gustafsson, A class of first order factorization methods, *BIT* 18 (1978) 142–156.
[15] G. Haase, U. Langer and A. Meyer, The approximate Dirichlet domain decomposition method, Part I: An algebraic approach, *Computing* 47 (1991) 137–151.
[16] G. Haase, U. Langer and A. Meyer, The approximate Dirichlet domain decomposition method, Part I: Applications to 2nd-order elliptic BVPs, *Computing* 47 (1991) 153–167.
[17] G. Haase, U. Langer and A. Meyer, Domain decomposition preconditioners with inexact subdomain solvers, *J. Numer. Linear Algebra Appl.* 1 (1) (1991) 27–41.

[18] R.W. Hockney and C.R. Jesshope, *Parallel Computers 2: Architecture, Programming and Algorithms* (Adam Hilger, Bristol, 1988).
[19] W. Hoffman, Iterative algorithms for Gram–Schmidt orthogonalization, *Computing* 41 (1989) 335–348.
[20] W. Jalby and B. Philippe, Stability analysis and improvement of the block Gram–Schmidt algorithm, *SIAM J. Sci. Statist. Comput.* 12 (5) (1991) 1058–1073.
[21] J. van Kan, A second-order accurate pressure-correction scheme for viscous incompressible flow, *SIAM J. Sci. Statist. Comput.* 7 (3) (1986) 870–891.
[22] D.E. Keyes, Aerodynamic applications of Newton–Krylov–Schwarz solvers, in: S.M. Despande, S.S. Desai and R. Narasimha, eds., *Proceedings of 14th International Conference on Numerical Methods in Fluid Dynamics* (Springer, Berlin, 1995) 1–20.
[23] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing; Design and Analysis of Algorithms* (Benjamin/Cummings, Redwood City, 1994).
[24] G. Li, A block variant of the GMRES method on massively parallel processors, *Parallel Computing* 23 (1997) 1005–1019.
[25] J.A. Meijerink and H.A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, *Math. Comp.* 31 (1977) 148–162.
[26] A. Meyer, A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain, *Computing* 45 (1990) 217–234.
[27] D.P. O'Leary and P. Whitman, Parallel QR factorization by Householder and modified Gram–Schmidt algorithms, *Parallel Computing* 16 (1990) 99–112.
[28] Y. Saad, A flexible inner–outer preconditioned GMRES algorithm, *SIAM J. Sci. Statist. Comput.* 14 (1993) 461–469.
[29] Y. Saad and M.H. Schultz, GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 7 (3) (1986) 856–869.
[30] A. Segal, P. Wesseling, J. van Kan, C.W. Oosterlee and K. Kassels, Invariant discretization of the incompressible Navier–Stokes equations in boundary-fitted co-ordinates, *Internat. J. Numer. Methods Fluids* 15 (1992) 411–426.
[31] R.B. Sidje, Alternatives for parallel Krylov subspace basis computation, *Numer. Linear Algebra Appl.* 4 (4) (1997) 305–331.
[32] B.F. Smith, P.E. Bjørstad and W.D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations* (Cambridge University Press, Cambridge, 1996).
[33] K.H. Tan, Local coupling in domain decomposition, Ph.D. Thesis, Utrecht University, Utrecht, The Netherlands (April 1996).
[34] H.A. van der Vorst and C. Vuik, GMRESR: A family of nested GMRES methods, *Numer. Linear Algebra Appl.* 1 (4) (1994) 369–386.
[35] C. Vuik, Further experiences with GMRESR, *Supercomputer* 55 (1993) 13–27.
[36] C. Vuik, New insights in GMRES-like methods with variable preconditioners, *J. Comput. Appl. Math.* 61 (1995) 189–204.
[37] C. Vuik, Fast iterative solvers for the discretized incompressible Navier–Stokes equations, *Internat. J. Numer. Methods Fluids* 22 (1996) 195–210.
[38] H.F. Walker, Implementation of the GMRES method using Householder transformations, *SIAM J. Sci. Statist. Comput.* 9 (1) (1988) 152–163.
[39] P. Wesseling, A. Segal, C.G.M. Kassels and H. Bijl, Computing flows on general two-dimensional nonsmooth staggered grids, *J. Engrg. Math.* 34 (1998) 21–44.