Parallel domain decomposition with incomplete subdomain solution

Report 96-83

J. Frank
A. Segal
K. 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 domain decomposition with incomplete subdomain solution

J. Frank and A. Segal and K. Vuik
Faculty of Technical Mathematics and Informatics
Delft University of Technology
Mekelweg 4, 2622 EH Delft, The Netherlands

August 15, 1996

Abstract

In this paper we outline a parallel implementation of Krylov-accelerated Schwarz domain decomposition in which subdomain problems are solved to low precision. By so doing, computational time is focused on the convergence of the global iteration rather than wasted on ineffective subdomain iterations. We consider the GCR method using classical Gram-Schmidt and Householder orthogonalization methods. Our goal is to apply this approach to the incompressible Navier-Stokes equations. For the parallel implementation, we assume a distributed memory system with message passing.

1 Schwarz Domain Decomposition

Domain decomposition is a natural approach to solving partial differential equations on complicated domains [6, 15]. It also almost *begs* to be parallelized. In this paper we develop a parallel implementation of domain decomposition for the incompressible Navier-Stokes equations within the framework of the ISNaS program. The ISNaS program solves the incompressible continuity and momentum equations in general coordinates on structured, boundary-fitted grids. We first discuss the domain decomposition method employed.

Domain decomposition, as considered in this paper, is an approach to solving a linear system $Au = f$, often obtained through discretization of a partial differential equation defined on a domain Ω with discrete coordinate elements $x_j \in \Omega$. The approach is to partition Ω into smaller subdomains $\Omega_1, \Omega_2, \cdots, \Omega_M$. To each subdomain Ω_m is associated an index set $J_m = \{j \mid x_j \in \Omega_m\}$, and we define $n_m = \dim J_m$ to be the number of elements in J_m and $n = \dim \Omega$. For a moment we will assume that the J_m are disjoint. We reorder the unknown vector u, collecting elements defined on common subdomains

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix}, \; u_m = \{u^{(j)}, j \in J_m\},$$

where parentheses are placed around an index to indicate an element of a vector, while omission of parentheses indicates a subvector. To aid in notation, we define the trivial restriction operators $R_m : \mathbb{R}^n \to \mathbb{R}^{n_m}$, $m = 1 : M$, with at most one 1 in each row and each column, which achieve the above reordering [6]:

$$u_m = R_m u, \; m = 1 : M.$$

Reordering the coefficient matrix A and the right hand side vector f to agree with the new order of u, we obtain the block system,

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1M} \\ A_{21} & A_{22} & \cdots & A_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ A_{M1} & A_{M2} & \cdots & A_{MM} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_M \end{bmatrix} . \tag{1}$$

Note that the diagonal blocks are given by $A_{mm} = R_m A R_m^t$. For the applications we have in mind, the diagonal blocks are sparse and the off-diagonal blocks are sparse or zero.

In case the J_m are not disjoint, we double the repeated unknowns and augment the matrix A to include the unknowns in each subdomain to which they belong. An example will illustrate this; we consider the two-dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f \tag{2}$$

in Cartesian coordinates with appropriate boundary conditions, defined on $\Omega = (-1, 1) \times (0, 1)$. We construct a grid of size $(2N - 1) \times N$ on Ω with grid spacing $\Delta x \times \Delta y$ as illustrated in Figure 1. Discretizing equation (2) on Ω using vertex-centered central differences, we obtain the five-point stencil

$$\frac{1}{\Delta x^2} u_{i-1,j} + \frac{1}{\Delta y^2} u_{i,j-1} - \left(\frac{2}{\Delta x^2} + \frac{2}{\Delta y^2} \right) u_{ij} + \frac{1}{\Delta y^2} u_{i,j+1} + \frac{1}{\Delta x^2} u_{i+1,j} = f_{ij} . \tag{3}$$

We reorder the unknowns u_{ij} and the right-hand side f_{ij} to produce vectors u and f, obtaining a linear system $Au = f$. where A represents the coefficients of (3).

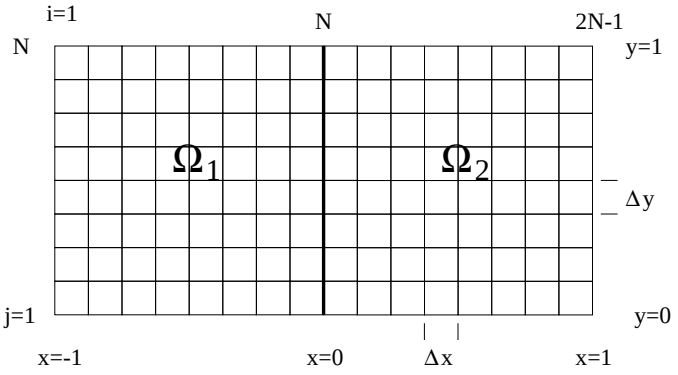

Figure 1: Two subdomain grid for Poisson equation example

We divide Ω into two subdomains along the line $x = 0$, denoting the left subdomain by Ω_1 and the right subdomain by Ω_2 as shown in Figure 1. Letting J_m be the index set of unknowns in Ω_m, we note that J_1 and J_2 have a non-empty intersection corresponding to the interface $i = N$. If, instead, a cell-centered finite volume discretization had been used on the grid in Figure 1, we would have found $J_1 \cap J_2 = \emptyset$, and it would have been trivial to rearrange the unknowns into the block structure (1). However, since there is overlap, we partition the unknowns into three groups with $\tilde{u}_1$ corresponding to unknowns with indices $J_1 \backslash J_2$, $\tilde{u}_2$ corresponding to unknowns with indices $J_1 \cap J_2$, and $\tilde{u}_3$ corresponding to unknowns with indices $J_2 \backslash J_1$. The resulting matrix A has the block structure

$$A = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix} .$$

Now we augment the system, doubling the interface unknowns to get

$$\begin{bmatrix} S_{11} & S_{12} & 0 & S_{13} \\ S_{21} & S_{22} & 0 & S_{23} \\ S_{21} & 0 & S_{22} & S_{23} \\ S_{31} & 0 & S_{32} & S_{33} \end{bmatrix} \begin{bmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \tilde{u}_2' \\ \tilde{u}_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_2 \\ f_3 \end{bmatrix}, \tag{4}$$

for which the invertibility of S_{22} implies $\tilde{u}_2 = \tilde{u}_2'$ [21]. We note here that invertibility of A does not guarantee invertibility of S_{22}; and in particular, if $S_{22} = 0$, then the augmentation (4) results in a loss of independence in the rows of A. For our model problem, S_{22} is the coefficient matrix of the interface equations, disregarding coupling to non-interface unknowns. From (3) we find, taking $h = \Delta x = \Delta y$,

$$S_{22} = \frac{1}{h^2} \begin{bmatrix} -4 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -4 \end{bmatrix},$$

which is strictly diagonally dominant and thus invertible.

In terms of the augmented matrix (4), A can be written in the block form (1), taking $A_{11} = \left(\begin{smallmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{smallmatrix}\right)$, $A_{22} = \left(\begin{smallmatrix} S_{22} & S_{23} \\ S_{32} & S_{33} \end{smallmatrix}\right)$, etc.

The *additive Schwarz iteration* for solving (1), using an initial guess u^0, is given in [6] by:

$$\begin{aligned} u_1^{k+1} &= u_1^k + A_{11}^{-1} r_1^k \\ &\vdots \\ u_M^{k+1} &= u_M^k + A_{MM}^{-1} r_M^k \end{aligned}$$

where $r^k = f - Au^k$ is the residual. Using the restriction operators R_m, we have

$$\begin{aligned} u_1^{k+1} &= u_1^k + A_{11}^{-1} R_1 (f - Au^k) \\ &\vdots \\ u_M^{k+1} &= u_M^k + A_{MM}^{-1} R_M (f - Au^k) \end{aligned} \tag{5}$$

or,

$$u^{k+1} = u^k + \left(R_1^t A_{11}^{-1} R_1 + \cdots R_M^t A_{MM}^{-1} R_M\right)(f - Au^k).$$

Defining the block diagonal matrix $N = \mathrm{diag}\,(A_{11}, \cdots, A_{MM})$, we obtain the iteration

$$u^{k+1} = u^k + N^{-1}(f - Au^k). \tag{6}$$

This is equivalent to a block Jacobi preconditioner. There is also an analogous *multiplicative Schwarz iteration* [6], in which the residual $(f - Au^k)$ in the right hand side of (5) is updated after each substep of the iteration step. The resulting preconditioner is equivalent to block Gauss-Seidel, but since we are interested in parallelization and since Gauss-Seidel is more difficult to parallelize, it will not be considered here.

In practice, the inverse of N is never computed in iteration (6). Rather, the iteration follows as in Algorithm 1. Note that the solution of $N\delta = r^k$ in the algorithm corresponds to solving a linear system on each subdomain. The solution of each subdomain problem will be accomplished in a secondary iteration, which we will refer to as the *inner iteration loop*. Similarly, the additive-Schwarz iteration is termed the *outer iteration loop*.

Algorithm 1 Schwarz Domain Decomposition

$r^0 = f - Au^0$
for $k = 0, 1, \cdots$, convergence
  solve $N\delta = r^k$
  $u^{k+1} = u^k + \delta$
  $r^{k+1} = r^k - A\delta$
end for

The solution obtained by iteration (6) lies in the subspace $u^0 + K^k$, where K^k is the Krylov subspace:

$$K^k = \text{span}\,\{N^{-1}r^0, (N^{-1}A)N^{-1}r^0, \cdots, (N^{-1}A)^{k-1}N^{-1}r^0\} \tag{7}$$

This can be seen by induction. Given u^0, we have

$$u^1 = u^0 + N^{-1}r^0 \in u_0 + K^1.$$

Suppose $u^k \in u^0 + K^k$. Then

$$r^k = f - Au^k \in r^0 + AK^k,$$

where we have used the notation $A\,\text{span}\,\{x^1, x^2, \cdots\} = \text{span}\,\{Ax^1, Ax^2, \cdots\}$. From this,

$$\begin{aligned} u^{k+1} &= u^k + N^{-1}r^k \\ &\in (u^0 + K^k) + \left[N^{-1}r^0 + (N^{-1}A)K^k\right] \\ &= u^0 + K^k + K^{k+1} = u^0 + K^{k+1}, \end{aligned}$$

, since $K^k \subset K^{k+1}$. This is the same subspace which is searched by a Krylov subspace method for the preconditioned problem

$$N^{-1}Au = N^{-1}f. \tag{8}$$

Krylov subspace methods "optimize" the residual with respect to the Krylov space in each iteration [13]. In this way, using a Krylov subspace method to solve (8) accelerates the convergence of (6), and is thus referred to as *Krylov-accelerated Schwarz domain decomposition.*

2 Incomplete Subdomain Solution

The block structure of the domain-decomposed problem hints at the possibility of parallelization. In the application of domain decomposition to discretized partial differential equations such as (2), the "weight" of the matrix A in (1) is usually concentrated in the blocks on the diagonal. Especially in a problem with a large number of subdomains, most of the off-diagonal blocks are zero. This is easy to see if one notes that the off-diagonal blocks represent coupling between subdomains in the stencil. With a five point stencil such as in (3), each subdomain is coupled only to the nearest row or column of its neighbors. In such a case, the only nonzero off-diagonal blocks are those corresponding to neighboring subdomains. Furthermore, those nonzero blocks are themselves only nonzero where necessary to grab a nearest row or column element from the neighboring subdomain.

As discussed in the previous section, Schwarz domain decomposition amounts to a nested iteration in which the inner loop solves the decoupled subdomain problems and the outer loop, in a sense, resolves the coupling between subdomains. A question which has been considered in [16, 10] with regard to general linear iterative methods is: to what extent may we relax the convergence tolerance in the inner iteration without greatly hurting the convergence of the

outer iteration? For, if the inner loop tolerance may be relaxed, the computational expense presumably would be reduced.

The application of this idea to incomplete solution of subdomain problems in domain decomposition is considered in [2, 12, 7, 20, 3]. A method is presented in [3, 5, 4] which uses incomplete solution of the subdomain problems to accelerate convergence of the global solution. The method uses the GCR algorithm of [8] for the outer iteration loop and GMRES [19] with incomplete subdomain solution for the inner loop. In [3] the method was only implemented sequentially; we will review the method in this section and extend it to parallel in the following sections.

As noted in reference to equation (6), the inverse of the matrix N is never computed. For the Jacobi preconditioner, the matrix N has the block diagonal form

$$N = \begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{MM} \end{bmatrix}. \tag{9}$$

Solution of $N\delta = r_k$ in Algorithm 1 thus amounts to solving the subdomain problems $A_{11}\delta_1 = r_1^k, \cdots, A_{MM}\delta_M = r_M^k$. With incomplete subdomain solution, the result δ_1 is not actually $A_{11}^{-1}r_1^k$, but is dependent on the subdomain solution method used. Consider solving a subdomain problem $Au = g$. Assuming a Krylov subspace method is used, then after k iterations, the solution u^k is an element of $u^0 + \text{span}\{r^0, Ar^0, \cdots, A^{k-1}r^0\}$, where $r^0 = g - Au^0$. If we use an initial guess $u^0 = 0$, we can express u^k as

$$\begin{aligned} u^k &= y^{(0)}g + y^{(1)}Ag + \cdots + y^{(k-1)}A^{k-1}g \\ &= \left[y^{(0)}I + y^{(1)}A + \cdots + y^{(k-1)}A^{k-1}\right]g \end{aligned}$$

for some y. Without implying invertibility we shall refer to the terms in brackets above by $\tilde{A}^{-1}$. Thus $\tilde{A}^{-1}$ is the approximation to the inverse of A achieved by incomplete solution of the subdomain. For Krylov methods, the $y^{(i)}$ are dependent on the right hand side g, and $\tilde{A}^{-1}$ is further dependent on the number of iterations k required to reach the subdomain stopping criterion. If we define $\tilde{N}^{-1}$ to be the block diagonal matrix consisting of these $\tilde{A}_{mm}^{-1}$, then with incomplete subdomain solution we have $\delta = \tilde{N}^{-1}r^k$. It is important to note that because of the dependencies mentioned above, the matrix $\tilde{N}^{-1}$ varies in each outer iteration loop. This will motivate our choice of the GCR method for the outer loop solution.

Theorem 6.2 in [3] relates the condition number of the preconditioned system (8) to that of the preconditioned system resulting from incomplete subdomain solution. The theorem states that if the subdomain solutions satisfy

$$\|I - A_{mm}\tilde{A}_{mm}^{-1}\|_2 \leq \epsilon < 1, \ m = 1 : M, \tag{10}$$

then the condition numbers of the Jacobi-preconditioned systems resulting from complete and incomplete solution of the subdomain problems satisfy

$$\kappa(\tilde{N}^{-1}A) \leq \frac{1+\epsilon}{1-\epsilon}\kappa(N^{-1}A).$$

That is to say, the condition number should be not much affected for small enough ϵ. Unfortunately, the condition (10) is not easy to check.

The GMRES method searches for a solution of the form $x^k = x^0 + V_k y$ to the problem $Ax = b$. The columns v^j of the matrix V_k form an orthonormal basis for the Krylov space $K_k = \text{span}\{r^0, Ar^0, \cdots, A^{k-1}r^0\}$. The weight vector y is chosen to minimize the functional

$$J(y) = \left\| \|r_0\|_2 e_1 - \bar{H}_k y \right\|_2,$$

where the matrix $\bar{H}_k$ is an augmentation of the upper Hessenberg matrix $H_k = V_k^t A V_k$ and e_1 is the first canonical unit vector in $\mathbb{R}^k$. It is necessary for the matrix A to be constant during the GMRES iteration. Since the coefficient matrix $\tilde{N}$ "varies" in each iteration, it is thus impractical to use GMRES for this problem. There is a Flexible GMRES algorithm in which the matrix A can vary [18]; however, we choose to use the GCR method, which can be optimized [23].

The GCR method produces a series of A^tA-orthogonal (conjugate) vectors, along each of which the residual is minimized. By first computing the new search direction v^{k+1} to be

$$v^{k+1} = \tilde{N}^{-1} r^k = \tilde{N}^{-1}(f - Au^k) \tag{11}$$

and then applying an orthogonalization procedure with respect to all previous search vectors, the GCR algorithm searches in the same Krylov subspace as (6).

The above computation of v^{k+1} corresponds to a single iteration of Algorithm 1 with incomplete solution and with an initial value $u^k = 0$. That is, the update to the solution vector becomes the new search vector. This new search vector is orthogonalized with respect to all of the previous search directions using the modified Gram-Schmidt method. Finally, the residual is minimized along the new search direction. GCR requires computation of the scalar products $\langle Av^{k+1}, Av^j \rangle$, $j = 1, \cdots, k+1$ and $\langle f, Av^k \rangle$, plus the matrix-vector product Av^k itself. An additional requirement is the storage of the vectors v^i and Av^i. The sequential version of GCR with incomplete subdomain solution is given in Algorithm 2.

Algorithm 2 GCR

Given: the solution u^0
  $r^0 = f - Au^0$
  for $k = 0, 1, \cdots,$ convergence
    solve $Nv^{k+1} = r^k$, incomplete subdomain solution using GMRES
    compute Av^{k+1}
    /* Orthogonalize: */
    for $i = 1, \cdots, k$
      $\alpha = \langle Av^{k+1}, Av^i \rangle$
      $v^{k+1} = v^{k+1} - \alpha v^i$
      $(Av^{k+1} = Av^{k+1} - \alpha Av^i)$
    end for
    $\beta = ||Av^{k+1}||$
    $v^{k+1} = v^{k+1}/\beta$
    $(Av^{k+1} = Av^{k+1}/\beta)$
    /* Minimize: */
    $\gamma = \langle r^k, Av^{k+1} \rangle$
    $u^{k+1} = u^k + \gamma v^{k+1}$
    $r^{k+1} = r^k - \gamma Av^{k+1}$
  end for

The individual subdomain problems $A_{mm}u_m = g_m$ are solved using GMRES with ILUD preconditioning. Defining the matrices L, U and D by

$$\begin{aligned} l_{ij} &= a_{ij}, \ j < i \\ u_{ij} &= a_{ij}, \ j > i \\ u_{ii} &= l_{ii} = d_i \end{aligned}$$

where the diagonal matrix D may be chosen in various ways, according to the well-known techniques of ILU factorization. The solution of the preconditioned subdomain problem

$$U_m^{-1} D_m L_m^{-1} A_{mm} u_m = U_m^{-1} D_m L_m^{-1} g_m$$

by GMRES requires the repeated computation of the matrix-vector product $w = U_m^{-1} D_m L_m^{-1} v$. This product is achieved as follows:

$$\begin{aligned} &\textbf{solve } L_m \bar{v} = v \\ &\textbf{compute } \bar{w} = D_m \bar{v} \\ &\textbf{solve } U_m w = \bar{w} \end{aligned}$$

With the subdomain solution algorithm thus implemented, the GCR iteration is given in Algorithm 2.

In [5], Algorithm 2 with minimal subdomain solution ($\epsilon = 10^{-1}$) was found to reduce computation time by as much as 85% for 4 subdomains, decaying to 50% for 64 subdomains, over that of the algorithm with complete subdomain solution ($\epsilon = 10^{-8}$). At the same time, the solution accuracy and the required number of outer iterations remained relatively unaffected. The reduction in computational time became less marked as the number of subdomains was increased. We use the success of Algorithm 2 as motivation to attempt a parallelized version in the following section.

3 Parallel Implementation

In this section, we would like to distinguish between local problems, corresponding to the inner iteration loop, which are solved on individual processors with limited dependence on neighboring subdomains; and the global problem, associated with the outer iteration loop, which mainly involves the convergence of the GCR algorithm. The local problems are comprised of tasks to be performed in parallel, and the global problem is the essentially sequential host task. There are three operations in Algorithm 2 which contribute to the global problem: the initial construction of the coefficient matrix A and the right-hand side vector f, the matrix-vector multiplications Au and Av^{k+1}, and the inner products $\langle Av^{k+1}, Av^i \rangle$, $\|Av^{k+1}\|$, and $\langle Av^{k+1}, r^k \rangle$.

Taking advantage of the block structure of (1), we will develop a subdomain-parallel implementation of Algorithm 2. In other words, to each subdomain we will associate a processor node which will handle storage of the corresponding partitions of the vectors u^k, r^k and v^k in addition to the corresponding diagonal block of A and the "coupling rules" for that block, which encompass the off-diagonal blocks of the same block row of A. These coupling rules allow local computation of the local subvector of Av^{k+1} given the vector v^{k+1}. This partitioning of data is illustrated in Figure 2. Here we tacitly assume a one-to-one correspondence between available nodes and subdomains, as well as a distributed memory parallel architecture. There is no reason why more than one subdomain cannot be handled by the same processor, however. To summarize, given the system (1), we have M processor nodes. Node m holds the subvectors u_m^k, f_m, r_m^k, and v_m^i and $(Av^i)_m$ $(i = 1, \cdots, k)$, plus the blocks $A_{m1}, \cdots, A_{mm}, \cdots, A_{mM}$ of the matrix A.

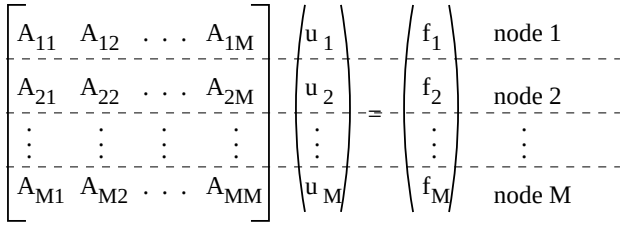

Figure 2: Distribution of the partitioned linear system $Au = f$ across parallel nodes.

There must be some information exchange between nodes to build the subdomain coefficient matrices A_{mm}, coupling matrices A_{mi}, and right-hand side vectors f_m. With a five-point stencil

such as in (3), the only information required by a given subdomain is the nearest row or column of its neighbor subdomains. Formally, we define the virtual-cell restriction operators:

DEFINITION 1
The virtual-cell restriction matrix R_{mn} with at most one 1 in each row and each column and all other elements 0, selects from the unknowns of subdomain m, those which are coupled to subdomain n.

For the two subdomain problem of Figure 1, $R_{1,2}$ is an $N \times N^2$ matrix. If u_1 represents the unknowns in subdomain Ω_1, then $R_{1,2}u_1$ corresponds to the unknowns on $i = N-1$. We define the neighbor set of subdomain m to be the set of indices $B_m = \{n \mid \Omega_n \text{ is a neighbor of } \Omega_m\}$. With this we can implement two message passing functions:

- vc_send(vector $R_{mn}u_m$, node n)
- vc_receive(vector $R_{nm}u_n$, node n)

These functions allow implementation of Algorithm 1 in parallel with complete subdomain solution.

To compute the local component of Av^{k+1}, we need those elements of v^{k+1} from the surrounding subdomains to which the local subdomain is coupled. But those are exactly the elements which are passed by the functions *vc_send()* and *vc_receive()*. As noted previously, the off-diagonal blocks of the matrix A represent coupling to neighboring subdomains. Therefore, to implement the matrix-vector multiply, we use the virtual cell concept, copying the coupled unknowns from the surrounding subdomains into virtual rows/columns around the local grid. Then we apply the stencil (3) to each of the local subdomain elements, using the virtual cells where needed. See Figure 3. We denote the function which accomplishes this product locally as $(Av^{k+1})_m = Amult(A_m, v^{k+1})$.

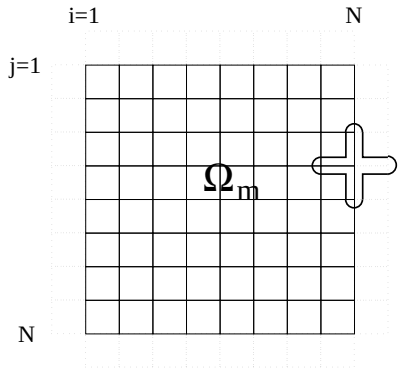

Figure 3: Virtual cells and stencil used in the local computation of Av^{k+1} in subdomain Ω_m

To compute the inner product of two vectors a and b, stored in blocks a_m and b_m, $m = 1, \cdots, M$ on M processors, we compute the local inner products $\langle a_m, b_m \rangle$, $m = 1, \cdots, M$ and take the global inner product as the sum of these:

$$\langle a, b \rangle = \sum_{m=1}^{M} \langle a_m, b_m \rangle. \tag{12}$$

For this we need to define two more message passing functions, this time between each subdomain processor and a common host processor:

- ip_send(vector local_ips, node host)

- ip_receive(vector global_ips, node host)

where the variables *local_ips* and *global_ips* are declared as vectors to allow a set of inner products to be computed simultaneously as discussed below. It should be noted that the host machine does not have to be dedicated to the task of computing the sums (12). The host may be any of the subdomain processors. These four message passing functions now handle all of the required communication for our parallel implementation.

Algorithm 2 utilizes the modified Gram-Schmidt orthogonalization process. The communication requirements of this method are high: $k+1$ communications per outer iteration. It is possible to reduce the number of communications by calculating all of the inner products of Algorithm 2 with a single call to the *ip_send()* and *ip_receive()* functions. Adopting the notation of [3], we let $s^i = Av^i,\ i = 1:k$. Further, let v be the solution of $Nv = r^k$, prior to orthogonalization, and $s = Av$. Inner products are required for:

- coefficients for orthogonalization of s,
- the length of s for normalization, and
- calculation of the minimization coefficient γ.

We will consider these three cases separately in the following three paragraphs.

In the classical Gram-Schmidt orthogonalization procedure, all of the orthogonalization coefficients

$$\alpha^{(i)} = \langle s, s^i \rangle,\ i = 1:k$$

are computed prior to orthogonalizing the vector s, which can subsequently be carried out locally on each node:

$$s^{k+1} = s - \sum_{i=1}^{k} \alpha^{(i)} s^i \tag{13}$$

Classical Gram-Schmidt may be subject to serious round-off error [17]. Our choice is motivated by the reduction in communication which can be achieved with the classical method, and we give some justification here. It is shown in [17] that the orthogonalization errors $\epsilon_{ij} = \langle w^i, w^j \rangle$ for previously orthogonalized w^i and w^j (i.e. the ϵ_{ij} should be zero) are amplified by a factor $1/\|\hat{w}^{k+1}\|$, where $\hat{w}^{k+1}$ is that component of the vector to be orthogonalized which is normal to all preceding vectors. It is therefore desirable to make this component as large as possible. We note that for Algorithm 2, the new search direction is given by $v = N^{-1}r^k$, so that $s = AN^{-1}r^k$. If the preconditioner N^{-1} is a good approximation for A^{-1}, then "one can hope" that s will be approximately in the same direction as r^k, and since $r^k - s^i,\ i \leq k$, the component of s normal to the previous s^i would be large. One possible solution for the problem of stability of classical Gram-Schmidt is to re-orthogonalize when the vectors are not "orthogonal enough." Another possibility is to use an iterative Gram-Schmidt method [14]. An alternative approach, to be discussed in the next section, is the use of Householder orthogonalization.

By comparison with (13), we can also compute the orthogonalized vector length by first computing the inner product prior to orthogonalization,

$$\alpha^{(k+1)} = \langle s, s \rangle = \|s\|^2,$$

and then updating this using the Pythagorean theorem to get $\beta = \|s^{k+1}\|$. Keeping in mind that the $s^i, i \leq k$ are already of length one,

$$\beta^2 = \|s^{(k+1)}\|^2 = \alpha^{(k+1)} - \sum_{i=1}^{k} (\alpha^{(i)})^2.$$

Finally, the inner product $\gamma = \langle r^k, s^{k+1} \rangle$ may be computed before orthogonalization, using the property $r^k - s^i,\ i \leq k$

$$\begin{aligned} \langle r^k, s^{k+1} \rangle &= \langle r^k, s - \sum_{i=1}^{k} \alpha^{(i)} s^i \rangle \\ &= \langle r^k, s \rangle - \sum_{i=1}^{k} \alpha^{(i)} \langle r^k, s^i \rangle \\ &= \langle r^k, s \rangle. \end{aligned}$$

Thus we can compute all vector products, before orthogonalization, with a single communication call. It remains to be determined if this method will be stable with respect to rounding errors.

In the parallel GCR Algorithm 3, we have defined an additional function *build(matrix A, vector f)* which constructs the local subdomain problems. Note that Algorithm 3 is written from the point of view of the local node, and not from that of the global job. We use here a "node-only" model, as it is described in the PVM Users' Guide [9] (p. 33). Multiple instances of algorithm are executed on different processors, exchanging information with each other as needed, and communicating with the host task to compute global inner products and to output the solution periodically.

Algorithm 3 Parallel GCR with incomplete subdomain solution

/* Algorithm for subdomain (processor) m, with neighbors $n \in B_m$ */
Given: the solution u^0
call vc_send($R_{mn}u^0$, neighbor n) ($n \in B_m$)
call vc_receive($R_{nm}u^0$, neighbor n) ($n \in B_m$)
build matrix A, vector f
call Au^0 =Amult(A,u^0)
$r^0 = f - Au^0$
for $k = 0, 1, \cdots$, convergence
solve $Nv^{k+1} = r^k$, incomplete subdomain solution using GMRES
call vc_send ($R_{mn}v^{k+1}$, neighbor n) ($n \in B_m$)
call vc_receive ($R_{nm}v^{k+1}$, neighbor n) ($n \in B_m$)
call Av^{k+1} = Amult (A,v^{k+1})
for $i = 1 : k+1$
$\alpha\text{local}^{(i)} = \langle Av^{k+1}, Av^i \rangle$
end for
$\gamma\text{local} = \langle Av^{k+1}, r^k \rangle$
call ip_send(γlocal, αlocal$^{(i)}$ ($i = 1 : k+1$), host)
call ip_receive(γ, $\alpha^{(i)}$, $i = 1 : k+1$, host)
$Av^{k+1} = Av^{k+1} - \sum_{i=1}^{k} \alpha^{(i)} Av^i$
$v^{k+1} = v^{k+1} - \sum_{i=1}^{k} \alpha^{(i)} v^i$
$\beta = \|Av^{k+1}\| = \left[\alpha^{(k+1)} - \sum_{i=1}^{k} (\alpha^{(i)})^2\right]^{1/2}$
$Av^{k+1} = Av^{k+1}/\beta$
$v^{k+1} = v^{k+1}/\beta$
$\gamma = \gamma/\beta$
$u^{k+1} = u^k + \gamma v^{k+1}$
$r^{k+1} = r^k - \gamma Av^{k+1}$
end for

4 Householder Orthogonalization

To avoid the threat of round-off error from the classical Gram-Schmidt orthogonalization method, we consider using Householder transformations, which are known to be more stable than Gram-Schmidt.

Suppose a matrix A with columns $a^1, \cdots, a^k$ could be factored as $A = QR$, where Q is orthogonal and R is upper triangular. Then it follows that the columns $q_1, \cdots, q_k$ of Q form an orthonormal basis for the span of the a_i, since $a_i = QR_i = R_i^{(1)} q_1 + \cdots + R_i^{(i)} q_i$, where R_i denotes the ith column of R. One way of obtaining Q is to apply a series of k orthogonal reflection matrices $P_1, \cdots, P_k$ to A. Such matrices have the properties $P_i^2 = I = P_i^t P_i$. We choose the P_i such that

$$P_k \cdots P_1 A = R.$$

In particular, we want the P_i to have the properties:

1. $P_i(P_{i-1} \cdots P_1 a_i) = R_i = R_i^{(1)} e_1 + \cdots + R_i^{(i)} e_i$
2. $P_i e_j = e_j, \ j < i$

where e_i is the ith canonical unit vector in the space of the same dimension as the columns of A. Property 1 says that, having applied the previous $i-1$ transformations to A, we want P_i to transform the ith column of the modified A so that the elements below the ith are all zero. Property 2 assures that P_i has no effect on the previously transformed $i-1$ columns.

Having factored A completely, we can let $Q = (P_k \cdots P_1)^t = P_1 \cdots P_k$, and Q is orthogonal since the P_i are. Note that the ith column of Q is given by

$$\begin{aligned} q_i &= Q e_i \\ &= P_1 \cdots P_i P_{i+1} \cdots P_k e_i \\ &= P_1 \cdots P_i e_i \end{aligned}$$

by property 2 above, and we see that it is not necessary to have the complete factorization of A to get a particular orthonormal basis vector q_i. In fact, it is not even necessary to have the complete set of a_i prior to transformation, Given a_1, we can find P_1 and thus q_1; given a_1 and a_2, we can find P_1, P_2, q_1, and q_2; etc. The P_i which achieve our goal are called Householder transformations and are defined by:

$$P_i = I - 2\frac{w_i w_i^t}{w_i^t w_i},$$

where w_i, yet to be determined, is of the same dimension as a_i, and $w_i^{(j)} = 0, \ j < i$. This second condition on w_i is imposed to ensure property 2.

Now, following the derivation of [11], suppose we have a_1. We would like to find w_1 which generates P_1 such that

$$P_1 a_1 = \left(I - 2\frac{w_1 w_1^t}{w_1^t w_1}\right) a_1 = a_1 - 2\frac{w_1^t a_1}{w_1^t w_1} w_1 \ \in \text{span}\{e_1\}.$$

We see that $w_1 \in \text{span}\{a_1, e_1\}$, so let $w_1 = a_1 + \beta e_1$. Then

$$w_1^t a_1 = a_1^t a_1 + \beta a_1^{(1)},$$

and

$$w_1^t w_1 = a_1^t a_1 + 2\beta a_1^{(1)} + \beta^2$$

and therefore

$$P_1 a_1 = \left(1 - 2\frac{a_1^t a_1 + \beta a_1^{(1)}}{a_1^t a_1 + 2\beta a_1^{(1)} + \beta^2}\right) a_1 - 2\beta \frac{w_1^t a_1}{w_1^t w_1} e_1 .$$

To have the coefficient of a_1 equal zero, we set $\beta = \pm\|a_1\|$. Thus, $w_1 = a_1 \pm \|a_1\| e_1$. We can choose the sign of β to reduce cancelation errors. Once w_1 is determined, the transformation P_1 may be applied to the columns of A, and a w_2 can be found in the same manner which zeros the subdiagonal elements of $P_1 a_2$. This process can be repeated for each of the k columns of A.

We now develop a parallel implementation of this orthogonalization method for application to the GCR algorithm with incomplete subdomain solution. We must therefore allow for the modification of the v^{k+1} as we orthogonalize the $s^{k+1} = Av^{k+1}$. For this reason, and because our vectors are distributed across several processors, the following approach is attractive.

According to [24], the transformation $P_k \cdots P_1 s$ can be computed as follows:

$$\tilde{s} = P_k \cdots P_1 s = s - 2d^{(1)} w_1 - \cdots - 2d^{(k)} w_k , \tag{14}$$

where the coefficients $d^{(i)}$ are given by

$$\begin{aligned} d^{(1)} &= w_1^t s \\ d^{(k)} &= w_k^t s - 2d^{(1)} w_k^t w_1 - \cdots - 2d^{(k-1)} w_k^t w_{k-1}, \ k \geq 2. \end{aligned} \tag{15}$$

To calculate the $d^{(i)}$ it is thus necessary to have the inner products $\langle w_i, s\rangle$, $i = 1 : k$ and $\langle w_k, w_j\rangle$, $j = 1 : k-1$. Formula (15) then costs the same work as a forward substitution of dimension k. Note that it will be necessary to store all of the previous inner products $\langle w_i, w_j\rangle$, $0 < j < i < k$ on the host processor.

Suppose we are in the $(k+1)$th iteration of GCR. Thus we have already computed and stored $s^1, \cdots, s^k$ and $w_1, \cdots, w_k$. After solving $v = \tilde{N}^{-1} r_k$, the new vector to be orthogonalized is $s = Av$. We can compute the effect of the previous Householder transformations on s, obtaining $\tilde{s}$ from (14). Now we want to find w_{k+1} with elements $w_{k+1}^{(i)} = 0$, $i < k+1$ such that

$$P_{k+1}\tilde{s} = P_{k+1} P_k \cdots P_1 s = \tilde{s}^{(1)} e_1 + \cdots + \tilde{s}^{(k)} e_k + \alpha e_{k+1} . \tag{16}$$

To get w_{k+1}, we formally define the matrix J_{k+1} which sets the first k elements of $\tilde{s}$ to zero,

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

and take $\tilde{w} = J_{k+1}\tilde{s}$. Note that $\tilde{w}^t \tilde{s} = \tilde{w}^t \tilde{w} = \|w\|^2$. Now we have, before normalization,

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

from which

$$\begin{aligned} w_{k+1}^t \tilde{s} &= \tilde{w}^t \tilde{s} + \mathrm{sign}(\tilde{s}^{(k+1)}) \|\tilde{w}\| \tilde{s}^{(k+1)} \\ &= \|\tilde{w}\|^2 + |\tilde{s}^{(k+1)}| \|\tilde{w}\| \end{aligned}$$

and similarly

$$\begin{aligned} w_{k+1}^t w_{k+1} &= \tilde{w}^t \tilde{w} + 2\mathrm{sign}(\tilde{s}^{(k+1)}) \|\tilde{w}\| \tilde{s}^{(k+1)} + \|\tilde{w}\|^2 \\ &= 2\|\tilde{w}\|^2 + 2|\tilde{s}^{(k+1)}| \|\tilde{w}\| = 2 w_{k+1}^t \tilde{s} \end{aligned} \tag{17}$$

Now, to determine α in (16), using the non-normalized w_{k+1}, we get

$$P_{k+1}\tilde{s} = (I - 2\frac{w_{k+1} w_{k+1}^t}{w_{k+1}^t w_{k+1}})\tilde{s} = \tilde{s} - \frac{2 w_{k+1}^t \tilde{s}}{w_{k+1}^t w_{k+1}} w_{k+1} = \tilde{s} - w_{k+1}$$

and since α is the $(k+1)$th component of the resulting vector,

$$\alpha = \tilde{s}^{(k+1)} - \tilde{w}^{(k+1)} - \text{sign}(\tilde{s}^{(k+1)})\|\tilde{w}\|$$

or since $\tilde{s}^{(k+1)} = \tilde{w}^{(k+1)}$,

$$\alpha = -\text{sign}(\tilde{s}^{(k+1)})\|\tilde{w}\|. \tag{18}$$

It is assumed in (15) that the w_i are of unit length. Using the definition (18) of α, equation (17) becomes $\|w_{k+1}\|^2 = 2\alpha^2 - 2\alpha\tilde{s}^{(k+1)}$, so that the normalized vector w_{k+1} is given by

$$w_{k+1} = (J_{k+1}\tilde{s} - \alpha e_{k+1})/\sqrt{2\alpha^2 - 2\alpha\tilde{s}^{(k+1)}}, \tag{19}$$

and the denominator is zero only when α is. Note that computation of α from (18) requires an additional inner product, which may not be evaluated at the same time as the inner products of (15), resulting in an additional host communication. Alternatively, we could use the fact that $\|\tilde{s}\| = \|s\|$:

$$\alpha = -\text{sign}(\tilde{s}^{(k+1)})\left\|(\tilde{s}^{(k+1)}, \cdots, \tilde{s}^{(n)})\right\| \tag{20}$$

$$= -\text{sign}(\tilde{s}^{(k+1)})\sqrt{\|s\|^2 - \|\tilde{s}^{(1)} \cdots \tilde{s}^{(k)}\|^2}, \tag{21}$$

which is more efficiently parallelized. Note that for typically large domain decomposition problems, all of the first $k+1$ elements of $\tilde{s}$ will reside on the same "first" subdomain. The term $\|s\|$ can be calculated in the same communication as the inner products $\langle w_i, s\rangle$, and then all that are needed to calculate α on each process individually are the $\tilde{s}^{(i)}, i = 1 : k+1$. These values may be broadcast to all other processes simultaneously, and as we shall see, they are required for computation of s^{k+1} and v^{k+1} anyway.

After doing some numerical experiments, we observe that computing α from (21) can result in large round-off errors. To illustrate this, consider the example of [1]. The vectors to be orthogonalized are

$$S = \begin{bmatrix} 1 & 1 & 1 \\ \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix}.$$

Here, ϵ is chosen small enough that $1 + \epsilon^2$ evaluates to 1 in computer arithmetic. Thus the length of each column of S evaluates to 1. The first vector remains unchanged in the first orthogonalization step, and w_1 is defined to be $(1\ \epsilon/2\ 0\ 0)^T$. In the second step, $s = (1\ 0\ \epsilon\ 0)^T$, $\|s\|$ and $d^{(1)}$ evaluate to 1; $\tilde{s} = s - 2w_1 = (-1\ -\epsilon/2\ \epsilon\ 0)^T$. Equation (21) then gives

$$\alpha = -\text{sign}(-\epsilon/2)\sqrt{(1)^2 - \|1\|^2} = 0,$$

and the algorithm breaks down, cf. equation (19) and equations (23) and (24), yet to be derived. To avoid this, α may calculated with (20). Figure 4 compares the various orthogonalization methods for this test problem. In the figure, the sum of the absolute value of the inner products of the normalized vectors:

$$\textit{orthogonality error} = \sum_i \sum_{j<i} |\langle s^i, s^j \rangle|$$

is plotted for values of ϵ from 10^{-1} to 10^{-7}. Of course this is a concocted example, and it remains to be seen whether the matrices encountered in real examples will require the communication increase posed by (20).

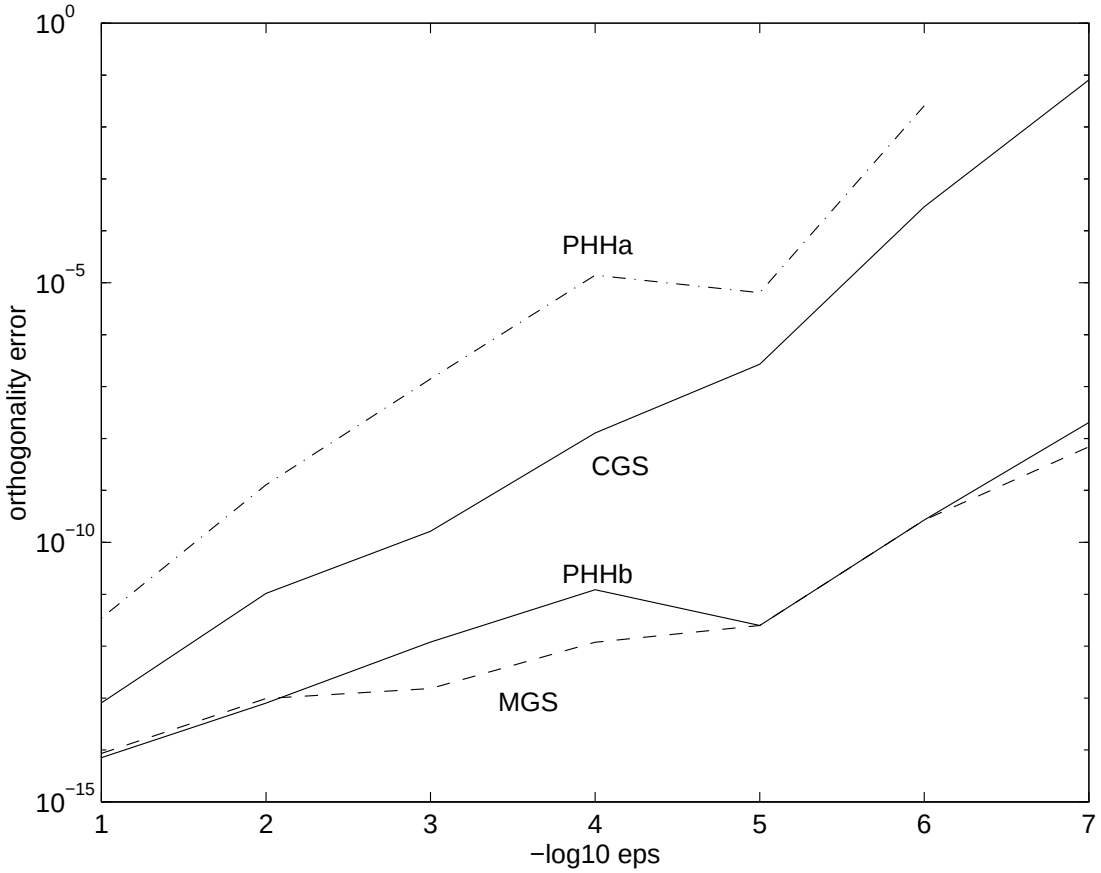

Figure 4: Comparison of orthogonalization methods for a poorly-behaved system. The 2-norm of the orthogonality error is plotted for decreasing magnitude of ϵ. CGS: classical Gram-Schmidt, MGS: modified Gram-Schmidt, PHHa: parallel Householder with (21), PHHb: parallel Householder with (20).

Since the $(k+1)$th column of the orthogonal matrix Q is given by Qe_{k+1}, we obtain the newly orthogonalized vector s_{k+1} from:

$$s^{k+1} = Qe_{k+1} = P_1 \cdots P_n e_{k+1} = P_1 \cdots P_{k+1} e_{k+1}, \tag{22}$$

which is to say that we can orthogonalize as we go along, without having to start with a complete set of vectors. Using this property, we multiply both sides of (16) by $P_1 \cdots P_{k+1}$, getting

$$\begin{aligned} s =& \tilde{s}^{(1)} P_1 e_1 + \tilde{s}^{(2)} P_1 P_2 e_2 + \cdots + \alpha P_1 P_2 \cdots P_{k+1} e_{k+1} \\ =& \tilde{s}^{(1)} s^1 + \tilde{s}^{(2)} s^2 + \cdots + \alpha s^{k+1}. \end{aligned}$$

Solving for s^{k+1},

$$s^{k+1} = \frac{1}{\alpha} \left[s - \sum_{i=1}^{k} \tilde{s}^{(i)} s^i \right]. \tag{23}$$

To maintain the relationship $s^{k+1} = Av^{k+1}$, we let

$$v^{k+1} = \frac{1}{\alpha}\left[v - \sum_{i=1}^{k} \tilde{s}^{(i)} v^i\right]. \tag{24}$$

When we distribute $\tilde{s}^{(1)}, \cdots, \tilde{s}^{(k)}$ to calculate s^{k+1} and v^{k+1} from equations (23) and (24), we can also distribute $\tilde{s}^{(k+1)}$ and the computation of w_{k+1} from (19) can be carried out without further communication.

To minimize the residual in the new search direction v^{k+1}, making it A-orthogonal to all searched directions v^j, $j \leq k+1$, we take

$$r^{k+1} = r^k - \gamma s^{k+1},$$

where $\gamma = \langle s^{k+1}, r^k \rangle$. By comparison with equation (23), we see that $\gamma = \langle s, r_k \rangle / \alpha$. That is, we can compute γ using s before orthogonalization since $r^k - s^i$, $i \leq k$.

For parallel GCR with Householder orthogonalization, we define a host process *host()* as in Algorithm 4. Then the subdomain processes may be defined as in Algorithm 5.

Algorithm 4 GCR/Householder: Host process

Define function host (Input: Subdomain contributions to: $\langle s, s \rangle$, $\langle s, r^k \rangle$, $\langle w_i, s \rangle$, $i = 1:k$ and $\langle w_k, w_j \rangle$, $j = 1:k-1$; Output: γ, $\|s\|$, $d^{(1:k)}$)
/* Initialize running sums */
$\gamma = \sigma = 0$
$d^{(i)} = 0$, $i = 1:k$
$\nu^{(j)} = 0$, $j = 1:k-1$
for $m = 1:M$ /* Compute global inner products */
  $\gamma = \gamma + \langle s, r^k \rangle_m$
  $\sigma = \sigma + \langle s, s \rangle_m$
  for $i = 1:k$
    $d^{(i)} = d^{(i)} + \langle w_k, s \rangle_m$
    if $i < k$ **then**
      $\nu_k^{(i)} = \nu_k^{(i)} + \langle w_k, w_i \rangle_m$
    end if
  end for
end for
Store the $\nu_k^{(i)}$ on the host processor.
/* Finish the $d^{(i)}$: */
for $i = 2:k$
  for $j = 1:i-1$
    $d^{(i)} = d^{(i)} - 2d^{(j)}\nu_i^{(j)}$
  end for
end for
$\|s\| = \sqrt{\sigma}$
return

The algorithm will breakdown when $\alpha = 0$ (cf. Eqn. (23), (24), and (19).) This occurs when $\tilde{s}^{(i)} = 0$, $i = k+1:n$. But in this case,

$$\tilde{s} = P_k \cdots P_1 s = \tilde{s}^{(1)} e_1 + \cdots + \tilde{s}^{(k)} e_k,$$

or multiplying both sides by $P_1 \cdots P_k$,

$$\begin{aligned} s =& \tilde{s}^{(1)} s^1 + \cdots \tilde{s}^{(k)} s^k \\ s \in& \operatorname{span}\{s^1, \cdots, s^k\}. \end{aligned}$$

That is to say, the algorithm breaks down when the new search direction is in the space already searched. This further implies $\gamma = \langle s, r^k \rangle = 0$, so that no update of r^k and u^k occurs, and the algorithm stalls. This should be a rare occurrence with large domain decomposition problems, but we can avoid breakdown by checking if $\alpha = 0$, and if so, letting $v = A^t r^k$, $s = Av$. This is equivalent to the "LSQR-switch" of [22]. Then we find

$$\begin{aligned} \gamma = \langle s, r^k \rangle &= \langle AA^t r^k, r^k \rangle \\ &= \langle A^t r^k, A^t r^k \rangle = \|A^t r^k\| \\ &> 0 \end{aligned}$$

as long as $r^k \neq 0$, which is the case unless the method has converged.

Algorithm 5 GCR/Householder: Node process

/* Algorithm for subdomain (processor) m, with neighbors $n \in B_m$ */
Given: the solution u^0
call vc_send($R_{mn}u^0$, neighbor n) ($n \in B_m$)
call vc_receive($R_{nm}u^0$, neighbor n) ($n \in B_m$)
build matrix A, vector f
call Au^0 =Amult(A,u^0)
$r^0 = f - Au^0$
for $k = 0, 1, \cdots$, convergence
solve $Nv = r^k$, incomplete subdomain solution using GMRES
call vc_send ($R_{mn}v$, neighbor n) ($n \in B_m$)
call vc_receive ($R_{nm}v$, neighbor n) ($n \in B_m$)
call s = Amult (A,v)
/* Compute local contributions to inner products */
$\sigma\text{local} = \langle s, s \rangle$
$\gamma\text{local} = \langle s, r^k \rangle$
for $i = 1 : k$
$\delta\text{local}^{(i)} = \langle w_i, s \rangle$
if $(i < k)$ **then**
$\nu\text{local}^{(i)} = \langle w_k, w_i \rangle$
end if
end for
call γ, $\|s\|, d^{(1:k)}$ = host(σlocal, γlocal, $\delta\text{local}^{(1:k)}$, $\beta\text{local}^{(1:k-1)}$)
$\tilde{s} = s - 2\sum_{i=1}^{k} d^{(i)} w_i$
distribute $\tilde{s}^{(1)}, \cdots, \tilde{s}^{(k+1)}$ to all nodes
$\alpha = -\text{sign}(\tilde{s}^{(k+1)})\sqrt{\|s\|^2 - \|\tilde{s}^{(1)} \cdots \tilde{s}^{(k)}\|^2}$
$s^{k+1} = (1/\alpha)[s - \sum_{i=1}^{k} \tilde{s}^{(i)} s^i]$
$v^{k+1} = (1/\alpha)[v - \sum_{i=1}^{k} \tilde{s}^{(i)} v^i]$
$w_{k+1} = (J_{k+1}\tilde{s} - \alpha e_{k+1})/\sqrt{2\alpha^2 - 2\alpha\tilde{s}^{(k+1)}}$
$\gamma = \gamma/\alpha$
$r^{k+1} = r^k - \gamma s^{k+1}$
$u^{k+1} = u^k + \gamma v^{k+1}$
end for

5 Conclusions

To conclude, we compare the computational efficiencies of the three orthogonalization procedures: modified Gram-Schmidt, classical Gram-Schmidt, and Householder. Assuming a minimum storage requirement of the most recently computed solution and residual vectors u^k and

r^k, as well as the subspace vectors v^i and s^i, $i = 1 : k$, we consider the amount of work required in the orthogonalization procedures in the kth iteration of GCR (the outer iteration). The modified Gram-Schmidt method, if implemented in parallel, would require $k + 2$ inner products, including $k + 1$ host communications for each node, since the vector to be orthogonalized must be updated within the orthogonalization loop. Modified Gram-Schmidt would require no additional storage. Classical Gram-Schmidt also requires $k + 2$ inner products, but only 1 host communication per node, and no additional storage. Householder orthogonalization requires $2k + 1$ inner products ($2k + 2$ if (20) is used) and 2 communications—1 host communication per node and 1 multicast communication (or only 2 host communications per node if (20) is used.) Additional storage is required for the vectors w_i, $i = 1 : k$. An additional k vector sums are required in (14), plus a (likely negligible) back-substitution of dimension k on the host process. These results are summarized in Table 1.

Table 1: Comparison of computational expense (in the kth outer iteration)

Orthog. Procedure	*Inner Products*	*Communications*	*Additional Storage*	*Additional Computations*
Modified Gram-Schmidt	$k + 2$	$k + 1$	none	none
Classical Gram-Schmidt	$k + 2$	1	none	none
Householder	$2k + 1$	2	$w_i,\ i = 1 : k$	k vector additions dim. k back-subst.

In a setting where inter-processor communication is expensive, such as is the case with PVM on a cluster of workstations, the classical Gram-Schmidt and Householder procedures are to be preferred. Classical Gram-Schmidt would be the cheapest solution if the problem to be solved is "well enough behaved." On the other hand, if stability is a concern and if the memory requirements are not, the well-known superior stability of Householder orthogonalization makes it the safer solution. We thus recommend an implementation providing the user with a choice between Householder orthogonalization and classical Gram-Schmidt, possibly with re-orthogonalization.

References

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

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

[3] 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.

[4] E. Brakkee, C. Vuik, and P. Wesseling. Domain decomposition for the incompressible Navier-Stokes equations: Solving subdomain problems accurately and inaccurately. Technical Report 95–37, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, the Netherlands. `http://www.twi.tudelft.nl`, 1995.

[5] E. Brakkee, C. Vuik, and P. Wesseling. An investigation of Schwartz domain decomposition using accurate and inaccurate solution of subdomains. Technical Report 95–18, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, the Netherlands. `http://www.twi.tudelft.nl`, 1995.

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

[7] H. Cheng. On the effect of using inexact solvers for certain domain decomposition algorithms. *East-West J. Numer. Math.*, 2(4):257–284, 1993.

[8] 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.

[9] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing.* The MIT Press, 1994.

[10] G. H. Golub and M. L. Overton. The convergence of inexact Chebyshev and Richardson iterative methods for solving linear systems. *Numerische Mathematik*, 53:571–593, 1988.

[11] G. H. Golub and C. F. van Loan. *Matrix Computations.* Johns Hopkins University Press, 2 edition, 1989.

[12] G. Haase, U. Langer, and A. Meyer. Domain decomposition preconditioners with inexact subdomain solvers. *Numerical Linear Algebra with Applications*, 1:27–42, 1992.

[13] W. Hackbush. *Iterative Solution of Large Sparse Systems of Equations.* Springer-Verlag New York, Inc., 1994.

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

[15] P. le Tallec. Domain decomposition methods in computational mechanics. In J. T. Oden, editor, *Computational Mechanics Advances*, volume 1, pages 121–220. Elsevier Science, B.V., Amsterdam, 1994.

[16] N. K. Nichols. On the convergence of two-stage iterative processes for solving linear equations. *SIAM Journal on Numerical Analysis*, 10(3):460–469, 1973.

[17] J. R. Rice. Experiments on Gram-Schmidt orthogonalization. *Mathematics of Computation*, 20:325–328, 1966.

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

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

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

[21] W. P. Tang. Generalized Schwartz splittings. *SIAM Journal on Scientific and Statistical Computating*, 13:573–595, 1992.

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

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

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