SIAM J. SCI. COMPUT.
Vol. 21, No. 4, pp. 1337–1356

DISTRIBUTED SCHUR COMPLEMENT TECHNIQUES FOR GENERAL SPARSE LINEAR SYSTEMS*

YOUSEF SAAD† AND MARIA SOSONKINA‡

Abstract. This paper presents a few preconditioning techniques for solving general sparse linear systems on distributed memory environments. These techniques utilize the Schur complement system for deriving the preconditioning matrix in a number of ways. Two of these preconditioners consist of an approximate solution process for the global system, which exploits approximate LU factorizations for diagonal blocks of the Schur complement. Another preconditioner uses a sparse approximate-inverse technique to obtain certain local approximations of the Schur complement. Comparisons are reported for systems of varying difficulty.

Key words. parallel preconditioning, distributed sparse linear systems, Schur complement techniques, domain decomposition

AMS subject classifications. 65F10, 65F50, 65N55, 65Y05

PII. S1064827597328996

1. Introduction.

The successful solution of many "grand challenge" problems in scientific computing depends largely on the availability of adequate large sparse linear system solvers. In this context, *iterative solution* techniques are becoming a mandatory replacement for direct solvers due to their more moderate computational and storage demands. A typical grand challenge application requires the use of powerful parallel computing platforms along with parallel solution algorithms to run on these platforms. In distributed memory environments, iterative methods are relatively easy to implement compared with direct solvers, and so they are often preferred in spite of their unpredictable performance for certain types of problems.

However, users of iterative methods do face a number of issues that do not arise in direct solution methods. In particular, it is not easy to predict how fast a linear system can be solved to a certain accuracy or whether it can be solved at all by certain types of iterative solvers. This depends on the algebraic properties of the matrix, such as the condition number and the clustering of the spectrum.

With a good preconditioner, the total number of steps required for convergence can be reduced dramatically, at the cost of a slight increase in the number of operations per step, resulting in much more efficient algorithms in general. In distributed environments, an additional benefit of preconditioning is that it reduces the parallel overhead, and therefore it decreases the total parallel execution time. The parallel overhead is the time spent by a parallel algorithm in performing communication tasks or in idling due to synchronization requirements. The algorithm will be efficient if the construction and the application of the preconditioning operation both have a small parallel overhead. A parallel preconditioner may be developed in two distinct ways: by extracting parallelism from efficient sequential techniques or by designing a preconditioner from the start, specifically for parallel platforms. Each of these two

*Received by the editors October 20, 1997; accepted for publication (in revised form) March 24, 1999; published electronically December 21, 1999. This work was supported in part by NSF under grants CCR-9618827 and DMR-9525885 and in part by the Minnesota Supercomputer Institute.
http://www.siam.org/journals/sisc/21-4/32899.html
†Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455 (saad@cs.umn.edu).
‡Department of Computer Science, University of Minnesota–Duluth, 320 Heller Hall, 10 University Drive, Duluth, MN 55812-2496 (masha@d.umn.edu).

approaches has its advantages and disadvantages. In the first approach, the preconditioners yield the same good convergence properties as those of a sequential method but often have a low degree of parallelism, leading to inefficient parallel implementations. In contrast, the second approach usually yields preconditioners that enjoy a higher degree of parallelism but that may have inferior convergence properties.

This paper mainly addresses the issue of developing preconditioners for distributed sparse linear systems by regarding these systems as distributed objects. This viewpoint is common in the framework of parallel iterative solution techniques [22, 19, 26, 14, 31, 1, 10] and borrows ideas from domain decomposition methods that are prevalent in the PDE literature. The key issue is to develop preconditioners for the global linear system by exploiting its distributed data structure. Recently, a number of methods have been developed which exploit the Schur complement system related to interface variables; see for example [17, 1, 10]. In particular, several distributed preconditioners included in the ParPre package [10] employ variants of Schur complement techniques. One difference between our work and [1] is that our approach does not construct a matrix to approximate the global Schur complement. Instead, the preconditioners constructed are entirely local. However, they also have a global nature in that they do attempt to solve the global Schur complement system approximately by an iterative technique.

The paper is organized as follows. Section 2 gives a background regarding distributed representations of sparse linear systems. Section 3 starts with a general description of the class of domain decomposition methods known as Schur complement techniques. This section also presents several distributed preconditioners that are defined via various approximations to the Schur complement. The numerical experiment section (section 4) contains a comparison of these preconditioners for solving various distributed linear systems. Finally, a few concluding remarks are made in section 5.

2. Distributed sparse linear systems. Consider a linear system of the form

$$Ax = b, \tag{2.1}$$

where A is a large sparse nonsymmetric real matrix of size n. Often, to solve such a system on a distributed memory computer, a graph partitioner is first invoked to partition the adjacency graph of A. A number of graph partitioners are available and several packages can be readily obtained [21, 12, 15]. Based on the resulting partitioning, the data is distributed to processors such that pairs of equations-unknowns are assigned to the same processor. Thus each processor holds a set of equations (rows of the linear system) and vector components associated with these rows.

A good distributed data structure is crucial for the development of effective sparse iterative solvers. It is important, for example, to have a convenient representation of the local equations as well as the dependencies between the local and external vector components. A preprocessing phase is thus required to determine these dependencies and any other information needed during the iteration phase. The approach described here follows the one used in the PSPARSLIB package; see [26, 29, 19] for additional details.

2.1. Background and notation. Figure 2.1 shows a "physical domain" viewpoint of a sparse linear system. This representation borrows from the domain decomposition literature. Thus the term "subdomain" is often used here instead of the more proper term "subgraph." Note that the concepts of a subdomain and a "point" are defined algebraically and do not necessarily have direct geometrical representations.

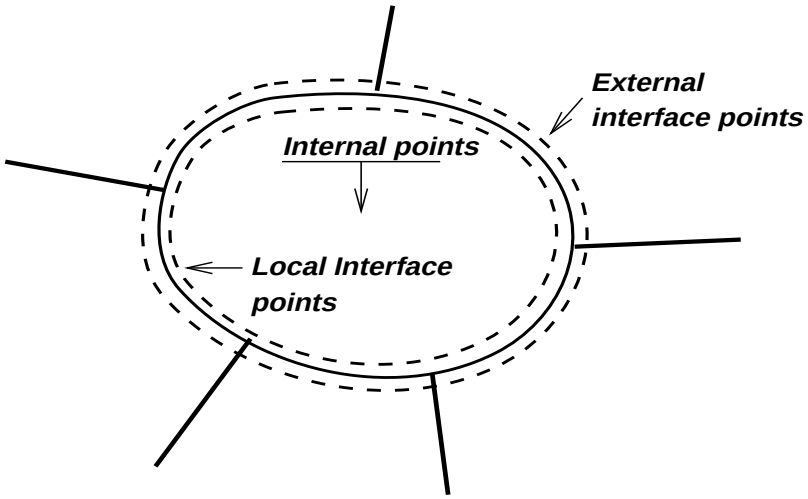

FIG. 2.1. *A local view of a distributed sparse matrix.*

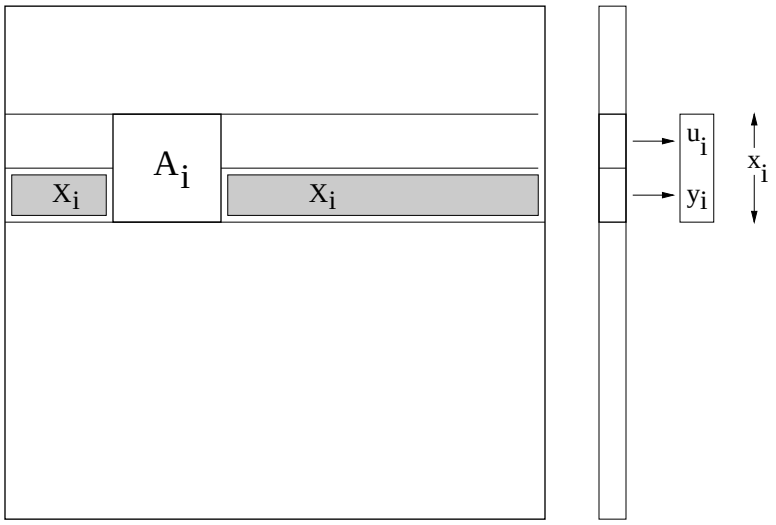

FIG. 2.2. *A partitioned sparse matrix and vector.*

Each point (node) belonging to a subdomain is actually a pair representing an equation and an associated unknown. It is common to distinguish between three types of unknowns: (1) interior unknowns that are coupled only with local equations; (2) local interface unknowns that are coupled with both nonlocal (external) and local equations; and (3) external interface unknowns that belong to other subdomains and are coupled with local equations. The matrix in Figure 2.2 can be viewed as a reordered version of the equations associated with a local numbering of the equation-unknown pairs. Note that local equations do not necessarily correspond to contiguous equations in the original system.

In Figure 2.2, the rows of the matrix assigned to a certain processor have been split into two parts: the *local* matrix A_i, which acts on the local vector components, and

the rectangular *interface matrix* X_i, which acts on the external vector components. Accordingly, the local equations can be written as follows:

$$A_i x_i + X_i y_{i,ext} = b_i, \tag{2.2}$$

where x_i is the vector of local unknowns, $y_{i,ext}$ are the external interface variables, and b_i is the local part of the right-hand side vector. Similarly, a (global) matrix-vector product Ax can be performed in three steps. First, multiply the local vector components x_i by A_i, then receive the external interface vector components $y_{i,ext}$ from other processors, and finally multiply the received data by X_i and add the result to that already obtained with A_i.

The preprocessing phase should construct a data structure for representing the matrices A_i and X_i. It should also form any additional data structures required to prepare for the intensive communication that takes place during the iteration phase. In particular, each processor needs to know (1) the processors with which it must communicate, (2) the list of interface points, and (3) a break-up of this list into sublists that must be communicated among neighboring processors. For further details see [26, 29, 19]. An important feature of the data structure used is the separation of the interface points from the interior points. In each processor, local points are ordered such that the interface points are listed last after the interior points. Such ordering of the local data presents several advantages, including more efficient interprocessor communication, and reduced local indirect addressing during matrix-vector multiplication.

With this local ordering, each local vector of unknowns x_i is split into two parts: the subvector u_i of internal vector components followed by the subvector y_i of local interface vector components. The right-hand side b_i is conformally split into the subvectors f_i and g_i, i.e.,

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix}; \qquad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix}.$$

When block partitioned according to this splitting, the local matrix A_i residing in processor i has the form

$$A_i = \left(\begin{array}{c|c} B_i & F_i \\ \hline E_i & C_i \end{array}\right), \tag{2.3}$$

so the local equations (2.2) can be written as follows:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{2.4}$$

Here, N_i is the set of indices for subdomains that are neighbors to the subdomain i. The term $E_{ij} y_j$ is a part of the product $X_i y_{i,ext}$ which reflects the contribution to the local equation from the neighboring subdomain j. The sum of these contributions is the result of multiplying X_i by the external interface unknowns

$$\sum_{j \in N_i} E_{ij} y_j \equiv X_i y_{i,ext}.$$

It is clear that the result of this multiplication affects only the local interface unknowns, which is indicated by zero in the top part of the second term of the left-hand side of (2.4).

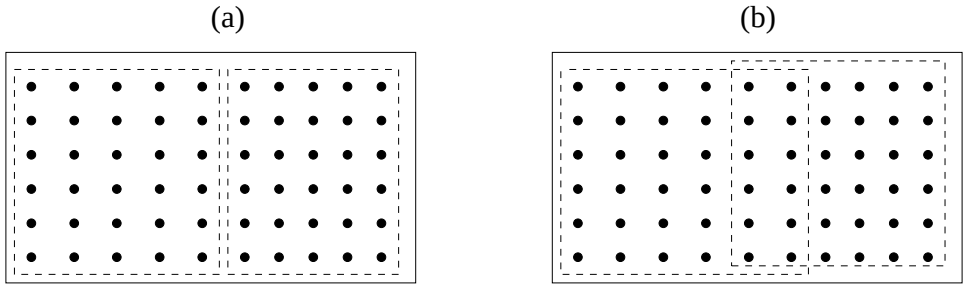

FIG. 2.3. (a) *Partition of a domain into two nonoverlapping subdomains.* (b) *Resulting overlapping partition after one level expansion.*

2.2. Distributed additive Schwarz preconditioning. Figure 2.1 can be used again to illustrate a framework for preconditioning construction based on domain decomposition techniques, the simplest of which is the additive Schwarz procedure. This form of a block Jacobi iteration, in which blocks refer to the systems associated with entire domains, is sketched next.

ALGORITHM 2.1. BLOCK JACOBI ITERATION (ADDITIVE SCHWARZ).

1. Obtain external data $y_{i,ext}$.
2. Compute (update) local residual $r_i = (b - Ax)_i = b_i - A_i x_i - X_i y_{i,ext}$.
3. Solve $A_i \delta_i = r_i$.
4. Update solution $x_i = x_i + \delta_i$.

Observe that the required communication, as well as the overall structure of the routine, is identical to that of a matrix-vector multiplication.

Of particular interest are the overlapping Jacobi methods. In the domain decomposition literature [3, 4, 5, 11] overlapping is used as a strategy for improving the convergence rate. In this paper, we consider mainly one-level overlapping: after an initial (nonoverlapping) partitioning, each subgraph is expanded by one level and the additional level-set is added to this initial subdomain. Figure 2.3 provides an illustration for this one-level overlapping. The data in the overlapping subregion will have two versions, each residing in one of the processors involved. When exchanging data during the iteration phase, we can either (1) replace the local version of the data by its external version or (2) replace it by some average. Note that for illustration purposes we used two subdomains, but in the overlapping case, it is common that a given data is assigned to more than two processors, so several different versions of the data have to be averaged in some way. Our experience is that the distinction between these different implementations of overlapping is rather minimal [18]. It is easy to generalize the one-level overlapping described above to overlappings that involve expansions by more than several levels.

3. Derivation of Schur complement techniques. Schur complement techniques refer to the methods which iterate on the interface unknowns only, implicitly using internal unknowns as intermediate variables. A few strategies for deriving Schur complement techniques will now be described. First, the Schur complement system is derived.

3.1. Schur complement system. Consider (2.2) and its block form (2.4). Schur complement systems are derived by eliminating the variable u_i from the system (2.4). Extracting from the first equation $u_i = B_i^{-1}(f_i - F_i y_i)$ yields, upon substitution

in the second equation,

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i', \tag{3.1}$$

where S_i is the "local" Schur complement

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{3.2}$$

The equations (3.1) for all subdomains i ($i = 1, \ldots, p$) constitute a system of equations involving only the interface unknown vectors y_i. This reduced system has a natural block structure related to the interface points in each subdomain:

$$\begin{pmatrix} S_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2 & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \ldots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}. \tag{3.3}$$

The diagonal blocks in this system, the matrices S_i, are dense in general. The off-diagonal blocks E_{ij}, which are identical with those involved in the system (2.4), are sparse.

The system (3.3) can be written as

$$Sy = g',$$

where $y = (y_1, \ldots, y_p)^T$ is the vector of all the interface variables and $g' = (g_1', \ldots, g_p')^T$ is the right-hand side vector. Throughout the paper, we will abuse the notation slightly for the transpose operation, by defining

$$(y_1, \ldots, y_p)^T \equiv \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}$$

rather than the actual transpose of the matrix with column vectors $y_j, j = 1, \ldots, p$. The matrix S is the "global" Schur complement matrix, which will be exploited in section 3.3.

3.2. Schur complement iterations. One of the simplest ideas that comes to mind for solving the Schur complement system (3.3) is to use a block-relaxation method associated with the blocking of the system. Once the Schur complement system is solved the interface variables are available and the other variables are obtained by solving local systems. As is known, with a consistent choice of the initial guess, a block-Jacobi (or SOR) iteration with the reduced system is equivalent to a block-Jacobi iteration (SOR, respectively) on the global system (see, e.g., [16, 25]). The kth step of a block-Jacobi iteration on the global system takes the following local form:

$$\begin{aligned} x_i^{(k+1)} &= x_i^{(k)} + A_i^{-1} r_i^{(k)} \\ &= x_i^{(k)} + A_i^{-1} \left(b_i - A_i x_i^{(k)} - \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j^{(k)} \end{pmatrix} \right) \\ &= A_i^{-1} \begin{pmatrix} f_i \\ g_i - \sum_{j \in N_i} E_{ij} y_j^{(k)} \end{pmatrix} \\ &= \begin{pmatrix} * & * \\ -S_i^{-1} E_i B_i^{-1} & S_i^{-1} \end{pmatrix} \begin{pmatrix} f_i \\ g_i - \sum_{j \in N_i} E_{ij} y_j^{(k)} \end{pmatrix}. \end{aligned} \tag{3.4}$$

Here, an asterisk denotes a nonzero block whose actual expression is unimportant. A worthwhile observation is that the iterates with interface unknowns y satisfy an independent relation

$$y_i^{(k+1)} = S_i^{-1} \left[g_i - E_i B_i^{-1} f_i - \sum_{j \in N_i} E_{ij} y_j^{(k)} \right] \tag{3.5}$$

or equivalently

$$y_i^{(k+1)} = y_i^{(k)} + S_i^{-1} \left[g_i' - S_i y_i^{(k)} - \sum_{j \in N_i} E_{ij} y_j^{(k)} \right], \tag{3.6}$$

which is nothing but a Jacobi iteration on the Schur complement system (3.3).

From a global viewpoint, a *primary* iteration for the global unknowns is

$$x^{(k+1)} = M x^{(k)} + c. \tag{3.7}$$

As was explained above, the vectors of interface unknowns y associated with the primary iteration satisfy an iteration (called a Schur complement iteration)

$$y^{(k+1)} = G y^{(k)} + h. \tag{3.8}$$

The matrix G is not known explicitly, but it is easy to advance the above iteration by one step from an arbitrary (starting) vector v, meaning that it is easy to compute $Gv + h$ for any v. This viewpoint was taken in [18, 17].

The sequence $y^{(k)}$ can be accelerated with a Krylov subspace algorithm, such as GMRES [27]. One way to look at this acceleration procedure is to consider the solution of the system

$$(I - G)y = h. \tag{3.9}$$

The right-hand side h can be obtained from one step of the iteration (3.8), computed for the initial vector 0, i.e.,

$$h = (G \times 0 + h).$$

Given the initial guess $y^{(0)}$, the initial residual $s^{(0)} = h - (I - G)y^{(0)}$ can be obtained from

$$s^{(0)} = h - (y^{(0)} - G y^{(0)}) = y^{(1)} - y^{(0)}.$$

Matrix-vector products with $I - G$ can be obtained from one step of the primary iteration. To compute $w = (I - G)y$, proceed as follows:

(1) Perform one step of the primary iteration $\begin{pmatrix} u' \\ y' \end{pmatrix} = M \begin{pmatrix} 0 \\ y \end{pmatrix} + c$;
(2) Set $w := y'$;
(3) Compute $w := y - w + h$.

The presented global viewpoint shows that a Schur complement technique can be derived for any primary fixed-point iteration on the global unknowns. Among the possible choices of the primary iteration there are Jacobi and SOR iterations as well as iterations derived (somewhat artificially) from ILU preconditioning techniques.

The main disadvantage of solving the Schur complement system is that the solve for the system $B_i\delta = \gamma$ (needed to operate with the matrix S_i) should be accurate. We can compute the dense matrix S_i explicitly or solve system (3.1) by using a computation of the matrix-vector product $S_i y$, which can be carried out with three sparse matrix-vector multiplies and one accurate linear system solve. As is known (see [30]), because of the large computational expense of these accurate solves, the resulting decrease in iteration counts is not sufficient to make the Schur complement iteration competitive. Numerical experiments will confirm this.

3.3. Induced preconditioners. A key idea in domain decomposition methods is to develop preconditioners for the *global system* (2.1) by exploiting methods that *approximately solve the reduced system* (3.3). These techniques, termed "induced preconditioners" (see, e.g., [25]), can best be explained by considering a reordered version of the global system (2.1) in which all the internal vector components $u = (u_1, \ldots, u_p)^T$ are labeled first, followed by all the interface vector components y. Such reordering leads to a block system

$$\left(\begin{array}{cccc|c} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & B_p & F_p \\ \hline E_1 & E_2 & \cdots & E_p & C \end{array}\right) \left(\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_p \\ \hline y \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_2 \\ \vdots \\ f_p \\ \hline g \end{array}\right), \tag{3.10}$$

which also can be rewritten as

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{3.11}$$

Note that the B block acts on the interior unknowns. Eliminating these unknowns from the system leads to the Schur complement system (3.3).

Induced preconditioners for the global system can be obtained by exploiting a block LU factorization for A. Consider the factorization

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} B & 0 \\ E & S \end{pmatrix} \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix}, \tag{3.12}$$

where S is the global Schur complement

$$S = C - EB^{-1}F.$$

This Schur complement matrix is identical to the coefficient matrix of system (3.3) (see, e.g., [25]).

The global system (3.11) can be preconditioned by an approximate LU factorization constructed such that

$$L = \begin{pmatrix} B & 0 \\ E & M_S \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix} \tag{3.13}$$

with M_S being some approximation to S.

Two techniques of this type are discussed in the rest of this section. The first one exploits the relation between an LU factorization and the Schur complement matrix, and the second uses approximate-inverse techniques to obtain approximations to the local Schur complements.

3.4. Approximate Schur LU preconditioner. The idea outlined in the previous subsection is that, if an approximation $\tilde{S}$ to the Schur complement S is available, then an approximate solve with the whole matrix A for all the global unknowns can be obtained, which will require (approximate or exact) solves with $\tilde{S}$ and B. It is also possible to think locally in order to act globally. Consider (2.4) and (3.1). As is readily seen from (2.4), once approximations to all the components of the interface unknowns y_i are available, corresponding approximations to the internal components u_i can be immediately obtained from solving

$$B_i u_i = f_i - F_i y_i$$

with the matrix B_i in each processor. In practice, it is often simpler to solve a slightly larger system obtained from (2.2) or

$$A_i x_i = b_i - X_i y_{i,ext} \tag{3.14}$$

because of the availability of the specific local data structure.

Now return to the problem of finding approximate solutions to the Schur unknowns. For convenience, (3.1) is rewritten as a preconditioned system with the diagonal blocks:

$$y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} \left[g_i - E_i B_i^{-1} f_i \right]. \tag{3.15}$$

Note that this is simply a block-Jacobi preconditioned Schur complement system. System (3.15) may be solved by a GMRES-like accelerator, requiring a solve with S_i at each step. There are at least three options for carrying out this solve with S_i:

(1) Compute each S_i exactly in the form of an LU factorization. As will be seen shortly, this representation can be obtained directly from an LU factorization of A_i.
(2) Use an approximate LU factorization for S_i, which is obtained from an approximate LU factorization for A_i.
(3) Obtain an approximation to S_i using approximate-inverse techniques (see the next subsection) and then factor it using an ILU technique.

The methods in options (1) and (2) are based on the following observation (see [25]). Let A_i have the form (2.3) and be factored as $A_i = L_i U_i$, where

$$L_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \quad \text{and} \quad U_i = \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix}.$$

Then, a rather useful result is that $L_{S_i} U_{S_i}$ is equal to the Schur complement S_i associated with the partitioning (2.3). This result can be easily established by "transferring" the matrices U_{B_i} and U_{S_i} from the U-matrix to the L-matrix in the factorization:

$$\begin{aligned} A_i &= \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix} \\ &= \begin{pmatrix} L_{B_i} U_{B_i} & 0 \\ E_i & L_{S_i} U_{S_i} \end{pmatrix} \begin{pmatrix} I & U_{B_i}^{-1} L_{B_i}^{-1} F_i \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} B_i & 0 \\ E_i & L_{S_i} U_{S_i} \end{pmatrix} \begin{pmatrix} I & B_i^{-1} F_i \\ 0 & I \end{pmatrix}, \end{aligned}$$

from which the result $S_i = L_{S_i} U_{S_i}$ follows by comparison with (3.12).

When an approximate factorization to A_i is available, an approximate LU factorization to S_i can be obtained canonically by extracting the related parts from the L_i and U_i matrices. In other words, *an ILU factorization for the Schur complement is the trace of the global ILU factorization on the unknowns associated with the Schur complement.* For a local Schur complement, the ILU factorization obtained in this manner leads to an approximation $\tilde{S}_i$ of the local Schur complement S_i. Instead of the exact Schur complement system (3.1), or equivalently (3.15), the following approximate (local) Schur complement system derived from (3.15) can be considered on each processor i:

$$y_i + \tilde{S}_i^{-1} \sum_{j \in N_i} E_{ij} y_j = \tilde{S}_i^{-1} \left[g_i - E_i B_i^{-1} f_i \right]. \tag{3.16}$$

The global system related to (3.16) can be solved by a Krylov subspace method, e.g., GMRES. The matrix-vector operation associated with this solve involves a certain matrix M_S (cf. (3.13)). The global preconditioner (3.13) can then be defined from M_S.

Given a local ILU factorization

$$A_i = L_i U_i + R_i,$$

with which the factorization

$$S_i = L_{S_i} U_{S_i} + R_{S_i}$$

is associated, the following algorithm applies, in each processor, the global approximate Schur LU preconditioner to a block vector $(f_i, g_i)^T$ to obtain the solution $(u_i, y_i)^T$. The algorithm uses m iterations of GMRES without restarting to solve the local part of the Schur complement system (3.16). Then, the interior vector components are calculated using (3.14) (lines 21–25). In the description of Algorithm 3.1, P represents the projector that maps the whole block vector $(f_i, g_i)^T$ into the subvector g_i associated with the interface variables.

ALGORITHM 3.1. APPROXIMATE SCHUR-LU SOLUTION STEP.

1. *Given: local right-hand side* $rhs = \begin{pmatrix} f_i \\ g_i \end{pmatrix}$
2. *Define an* $(m+1) \times m$ *matrix* $\bar{H}_m$ *and set* $\bar{H}_m := 0$.
3. **Arnoldi process:**
4. $y_i := 0$
5. $r := (L_i U_i)^{-1} rhs$
6. $v_1 := Pr / \|Pr\|_2$
7. *For* $j = 1, ..., m$ *do*
8. *Exchange interface vector components* y_i
9. $t := (L_{S_i} U_{S_i})^{-1} X_i y_{i,ext}$
10. $w := v_j + t$
11. *For* $l = 1, \ldots, j$ *Do:*
12. $h_{l,j} := (w, v_l)$
13. $w := w - h_{l,j} v_l$
14. *EndDo*
15. $h_{j+1,j} := \|w\|_2$ *and* $v_{j+1} := w / h_{j+1,j}$
16. *EndDo*
17. *Define* $V_m := [v_1, ..., v_m]$.
18. **Form the approximate solution for interface variables:**

19. *Compute* $y_i := y_i + V_m z_m$, *where*
20. $z_m = \text{argmin}_z \|\beta e_1 - \bar{H}_m z\|_2$ *and* $e_1 = [1, 0, \ldots, 0]^T$.
21. **Find other local unknowns:**
22. *Exchange interface vector components* y_i
23. $t := X_i y_{i,ext}$
24. $rhs := rhs - \begin{pmatrix} 0 \\ t \end{pmatrix}$
25. $\begin{pmatrix} u_i \\ y_i \end{pmatrix} := (L_i U_i)^{-1} rhs.$

A few explanations are in order. Lines 4–6 compute the initial residual for the GMRES iteration with initial guess of zero and normalize this residual to obtain the initial vector of the Arnoldi basis. According to the expression for the inverse of A_i in (3.4), we have

$$A_i^{-1} \begin{pmatrix} f_i \\ g_i \end{pmatrix} = \begin{pmatrix} * \\ S_i^{-1}(g_i - E_i B_i^{-1} f_i) \end{pmatrix},$$

which is identical to the expression in line 5 with A_i replaced by its approximation $L_i U_i$. Comparing the bottom part of the right-hand side of the above expression with the right-hand side of (3.16), it is seen that the vector Pr obtained in line 6 of the algorithm is indeed an approximation to the local right-hand side of the Schur complement system. Lines 8–10 correspond to the matrix-vector product with the preconditioned Schur complement matrix, i.e., with the computation of the left-hand side of (3.16).

3.5. Schur complements via approximate inverses. Equation (3.13) describes in general terms an approximate block LU factorization for the global system (3.11). A particular factorization stems from approximating the Schur complement matrix S using one of several approximate-inverse techniques described next.

Given an arbitrary matrix A, approximate-inverse preconditioners consist of finding an approximation Q to its inverse, by solving approximately the optimization problem [2]

$$\min_{Q \in \mathcal{S}} \|I - AQ\|_F^2,$$

in which $\mathcal{S}$ is a certain set of $n \times n$ sparse matrices and $\|\cdot\|_F$ is the Frobenius norm. This minimization problem can be decoupled into n minimization problems of the form

$$\min_{m_j} \|e_j - Am_j\|_2^2, \qquad j = 1, 2, \ldots, n,$$

where e_j and m_j are the jth columns of the identity matrix and a matrix $Q \in \mathcal{S}$, respectively. Note that each of the n columns can be computed independently. Different strategies for selecting a nonzero structure of the approximate-inverse are proposed in [7] and [13]. In [13] the initial sparsity pattern is taken to be diagonal with further fill-in allowed depending on the improvement in the minimization. The work [7] suggests controlling the sparsity of the approximate inverse by dropping certain nonzero entries in the solution or search directions of a suitable iterative method (e.g., GMRES). This iterative method solves the system $Am_j = e_j$ such that $\min_{m_j} \|e_j - Am_j\|_2^2$, for $j = 1, 2, \ldots, n$. In this paper, the approximate-inverse technique proposed in [7] and [6] is used.

Consider the local matrix A_i blocked as

$$A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}$$

and its block LU factorization similar to the one given by (3.12). The sparse approximate-inverse technique can be applied to approximate $B_i^{-1}F_i$ with a certain matrix Y_i (as it is done in [6]). The resulting matrix Y_i is sparse and therefore

$$M_{S_i} = C_i - E_i Y_i \tag{3.17}$$

is a sparse approximation to S_i. A further approximation can be constructed using an ILU factorization for the matrix M_{S_i}.

As in the previous subsection, an approximation M_S to the global Schur complement S can be obtained by approximately solving the reduced system (3.3), i.e., by solving its approximated version

$$\tilde{S}_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i, \tag{3.18}$$

the right-hand side of which can be also computed approximately. System (3.18) requires an approximation $\tilde{S}_i$ to the local Schur complement $S_i = C_i - E_i B_i^{-1} F_i$. The matrix M_{S_i} defined from the approximate-inverse technique outlined above can be used for $\tilde{S}_i$. Now that an approximation to the Schur complement matrix is available, an induced global preconditioner M to the matrix A can be defined from considering the global system (3.10), also written as (3.11). The Schur variables correspond to the bottom part of the linear system. The global preconditioner M is given by the block factorization (3.13) in which M_S is the approximation to S obtained by *iteratively solving system* (3.18).

Thus, the block forward-backward solves with the factors (3.13) will amount to the following three-step procedure:

(1) Solve $Bu = f$;
(2) Solve (iteratively) the system (3.18) to obtain y;
(3) Compute $u := u - B^{-1}Fy$.

This three-step procedure translates into the following algorithm executed by Processor i.

Algorithm 3.2. Approximate-inverse Schur complement solution step with GMRES.

1. Given: local right-hand side $rhs = \binom{f_i}{g_i}$.
2. Solve $B_i u_i = f_i$ approximately.
3. Calculate the local right-hand side $\tilde{g}_i := g_i - E_i u_i$.
4. Use GMRES to solve the distributed system $M_{S_i} y_i + X_i y_{i,ext} = \tilde{g}_i$.
5. Compute an approximation to $t := B_i^{-1} F_i y_i$.
6. Compute $u_i := u_i - t$.

Note that the steps in lines 2, 5, and 6 do not require any communication among processors, since matrix-vector operations in these steps are performed with the local vector components only. In contrast, the solution of the global Schur system invoked in line 4, involves global matrix-vector multiplications with the "interface exchange matrix" consisting of all the interface matrices X_i. The approximate solution of $B_i u_i = f_i$ (line 2) can be carried out by several steps of GMRES or by the forward-backward solves with incomplete L and U factors of B_i (assuming that a factorization

$B_i \approx L_{B_i} U_{B_i}$ is available). Then an approximation $\tilde{g}_i$ (line 3) to the local right-hand side of system (3.18) is calculated. In line 5, there are several choices for approximating $t := B^{-1} F_i y_i$. It is possible to solve the linear system $B_i t = F_i y_i$ using GMRES as in line 2. An alternative is to exploit the matrix Y_i that approximates $B_i^{-1} F_i$ in construction of M_{S_i} (3.17).

4. Numerical experiments. In the experiments, we compared the performance of the described preconditioners and the distributed Additive Schwarz preconditioning on two-dimensional elliptic PDE problems and on several problems with the matrices from the Harwell–Boeing [9] and Davis [8] collections.

Two parallel computing platforms have been used: an Intel Paragon (at Virginia Tech) and a Cray T3E-900 (at the Minnesota Supercomputer Institute). These computers have distributed-memory architectures. Thus, data sharing among processors is performed by message passing (MPI) [20]. For the Intel Paragon and Cray T3E, the interconnection networks are a two-dimensional mesh and a three-dimensional torus, respectively. The physical interconnection of processors emulates a fully connected network. The Intel Paragon was equipped with 100 compute nodes each having two Intel i860XP processors (50 MHz)—one dedicated to applications and the other to message passing—and 32 MB of memory. The Cray T3E-900 had 256 application nodes, each equipped with a DEC Alpha processor (450 MHz) and 512 MB of memory.

A flexible variant of restarted GMRES (FGMRES) [23] has been used to solve the original system since this accelerator permits a change in the preconditioning operation at each step. This is useful when, for example, an iterative process is used to precondition the input system. Thus, it is possible to use ILUT-preconditioned GMRES with `lfil` fill-in elements. Recall that ILUT [24] is a form of incomplete LU factorization with a dual threshold strategy for dropping fill-in elements.

For convenience, the following abbreviations will denote preconditioners and solution techniques used in the numerical experiments:

SAPINV	Distributed approximate block LU factorization: M_{S_i} and $B_i^{-1} F_i$ are approximated using the matrix Y_i, constructed using the approximate-inverse technique described in [7];
SAPINVS	Distributed approximate block LU factorization: M_{S_i} is approximated using the approximate-inverse technique in [7], but $B_i^{-1} F_i$ is applied using one matrix-vector multiplication followed by a solve with B_i;
SLU	Distributed global system preconditioning defined via an approximate solve with M_S, in which $S_i \approx L_{S_i} U_{S_i}$;
BJ	Approximate additive Schwarz, where ILUT-preconditioned GMRES(k) is used to precondition each submatrix assigned to a processor;
SI	"Pure" Schur complement iteration as described in section 3.2.

4.1. Elliptic problems. Consider the elliptic partial differential equation

$$\Delta u = f \tag{4.1}$$

on rectangular regions with Dirichlet boundary conditions, discretized with a five-point centered finite-difference scheme.

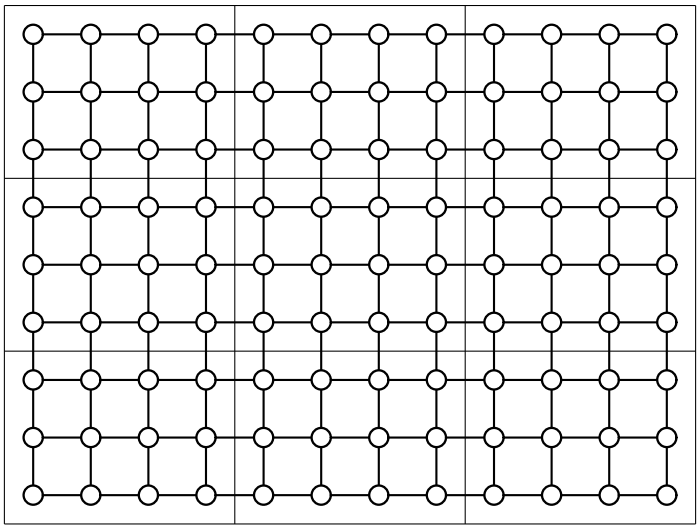

FIG. 4.1. *Domain decomposition and assignment of a* 12×9 *mesh to a* 3×3 *virtual processor grid.*

If the number of points in the x and y directions (respectively) are n_x and n_y, excluding the boundary points, then the mesh is mapped to a virtual $p_x \times p_y$ grid of processors, such that a subrectangle of n_x/p_x points in the x direction and n_y/p_y points in the y direction is mapped to a processor. In fact, each of the subproblems associated with these subrectangles is generated in parallel. Figure 4.1 shows a domain decomposition of a mesh and its mapping onto a virtual processor grid.

A comparison of timing results and iteration numbers for a global 360×360 mesh mapped to (virtual) square processor grids of increasing size is given in Figure 4.2. All the reported timing results are obtained by measuring the wall-clock time of a linear system solution excluding local ILU factorization and approximate inverse constructions. (In Figure 4.2, we omit the solution time for the BJ preconditioning on four processors, which is 95.43 seconds.) The residual norm reduction by 10^{-6} was achieved by flexible GMRES(10). In preconditioning, ILUT with $lfil = 15$ and the dropping tolerance 10^{-4} was used as a choice of an incomplete LU factorization. GMRES was used in the application of BJ and SLU, such that the GMRES convergence was detected at a relative tolerance of 10^{-2} or a maximum of five iterations. For SAPINV, forward-backward solves with $B_i \approx L_{B_i} U_{B_i}$ were performed in Line 2 of Algorithm 3.2.

Since the problem (mesh) size is fixed, with an increase in number of processors the subproblems become smaller and the overall time decreases. Both preconditioners based on Schur complement techniques are less expensive than the Additive Schwarz preconditioning. This is especially noticeable for small numbers of processors.

Keeping *subproblem sizes* fixed while increasing the number of processors increases the overall size of the problem making it harder to solve and thus increasing the solution time. In ideal situations of perfectly scalable algorithms, the execution time should remain constant. Timing results for fixed local subproblem sizes of 15×15, 30×30, 50×50, and 70×70 are presented in Figure 4.3. (Premature termination of the curves for SI indicates nonconvergence in 300 iterations.) The growth in the solution

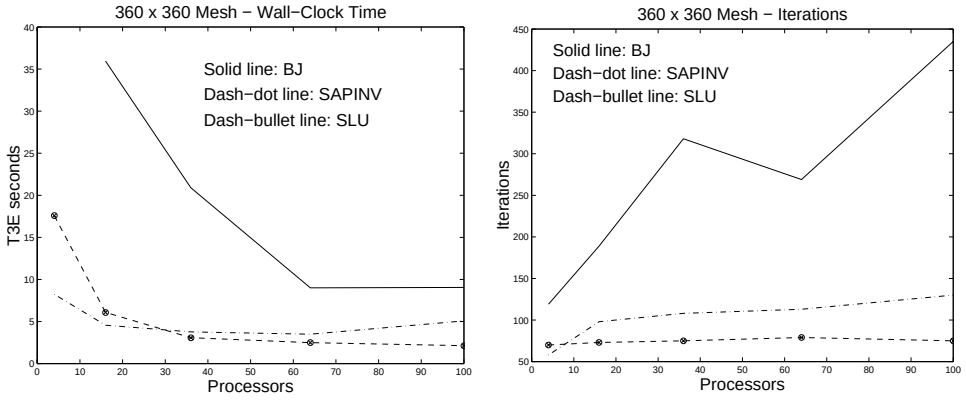

FIG. 4.2. *Solution times (wall-clock) and iteration counts for solving a* 360×360 *discretized Laplacean problem with* 3 *different preconditioners using flexible* GMRES(10).

time as the number of processors increases is rather pronounced for the "pure" Schur complement iteration and additive Schwarz, whereas it is rather moderate for the Schur complement-based preconditioners.

The additive Schwarz preconditioning performs at its best when the partitioning includes some domain overlapping. We compared the proposed Schur complement-based preconditioning with the overlapping additive Schwarz algorithm. We used a version in which the local overlapping data are exchanged, meaning that overlapping data that belong to a given processor are replaced by the external version (see section 2.2). In cases of more than two overlapping domains, data is exchanged with only one of the overlapping neighboring domains. Figure 4.4 shows the wall-clock timing results and iteration numbers for the one- and two-level overlappings used in both BJ and SLU preconditionings. Note that the two-level overlapping refers to the case when two level sets are added to an initial nonoverlapping partitioning (neighbors of the interface points and the neighbors of these neighbors). As indicated in Figure 4.4, BJ benefits much more than SLU from overlapping. In fact, if the execution time is the main criterion used, then the overlapping version of SLU is more expensive than the nonoverlapping one. However, Schur LU retains its superior performance over block-Jacobi. At this point it is interesting to note that the Schur-LU preconditioner can be viewed as a two-level technique. At the second level, a "coarse" (global) problem consisting of all the interface variables is solved approximately. A prolongation to the "fine" problem is then obtained and the process is repeated until convergence, with the outer loop accelerated by a flexible Krylov accelerator. Therefore, it is not too surprising that the number of outer steps does not vary too much as the number of processors increases.

4.2. General problems. Table 4.1 describes three test problems from the Harwell–Boeing and Davis collections. The column Pattern specifies whether a given problem has a structurally symmetric matrix. In all three test problems, the matrix rows followed by the columns were scaled by 2-norm. Also, in the partitioning of a problem the one-level overlapping with data replacing was used (see section 2.2).

Tables 4.2–4.4 show iteration numbers required by FGMRES(20) with SAPINV, SAPINVS, SLU, and BJ until convergence on different numbers of processors. An asterisk indicates nonconvergence. In the preconditioning phase, approximate solves

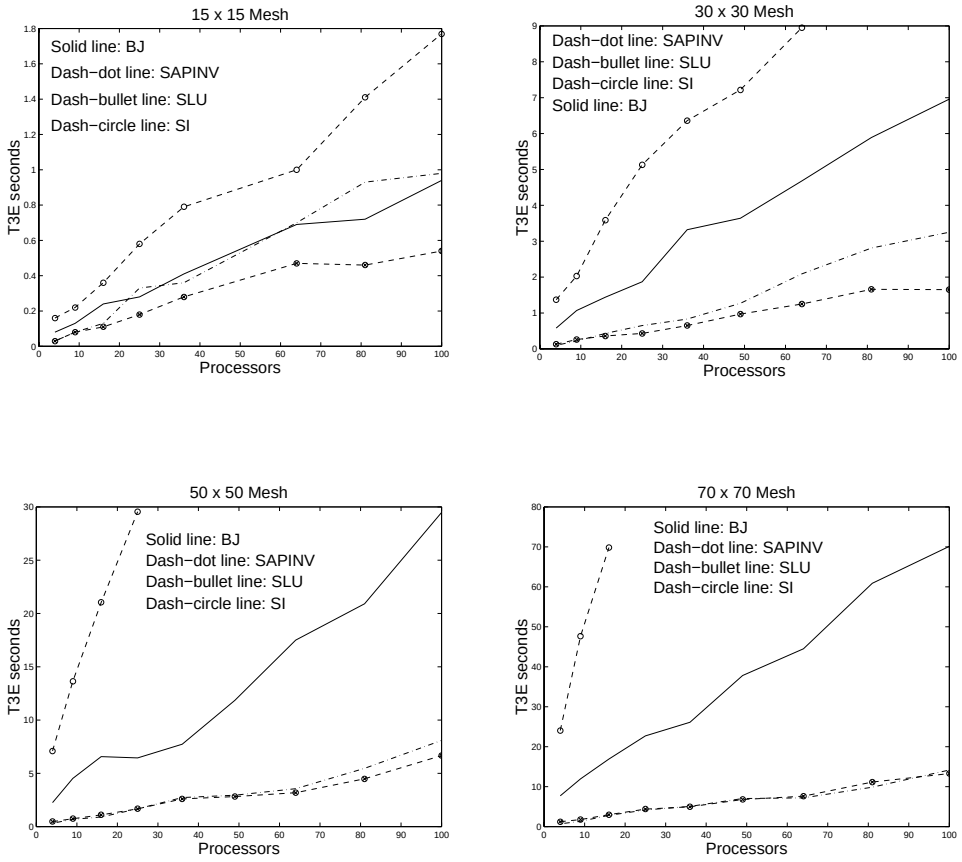

FIG. 4.3. *Solution times (wall-clock) for a Laplacean problem with various local subproblem sizes using flexible* GMRES(10) *with* 3 *different preconditioners (BJ, SAPINV, SLU) and the Schur complement iteration (SI).*

TABLE 4.1
Description of test problems.

Name	n	n_z	Pattern	Discipline
af23560	23560	484256	Symm	Airfoil, eigenvalue calculation
raefsky1	3242	22316	Unsymm	Incompressible flow in pressure driven pipe
sherman3	5005	20033	Symm	Oil reservoir modeling

in each processor were carried out by GMRES to reduce the residual norm by 10^{-3} but no more than five steps were allowed. As a choice of ILU factorization, ILUT with `lfil` fill-in elements (shown in the column `lfil`) was used in the experiments here. `lfil` corresponds also to the number of elements in a matrix column created by the approximate-inverse technique. In general, it is hard to compare the methods since the number of fill-in elements in each of the resulting preconditioners is different. In other words, for SAPINV and SAPINVS, `lfil` specifies the number of nonzeros in the blocks of preconditioning; for SLU, `lfil` is the total number of nonzeros in the preconditioning, therefore, the number of nonzeros in a given approximation S is not

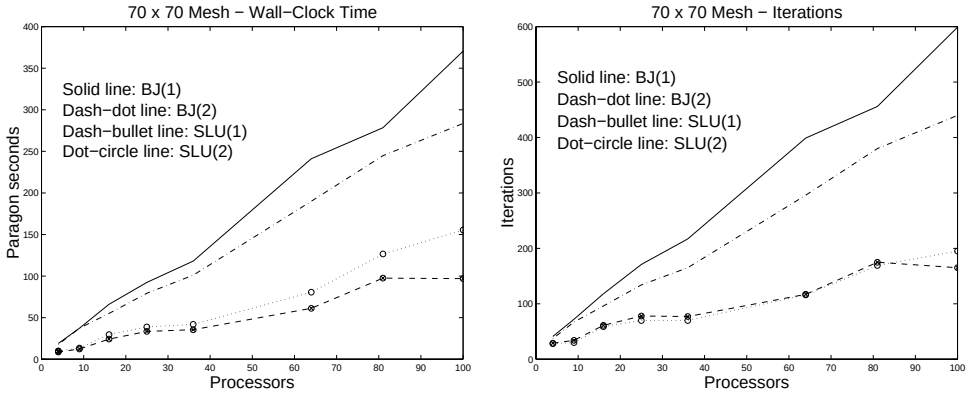

FIG. 4.4. *Times and iteration counts for solving a Laplacean problem with* 70×70 *local subproblem size using BJ and SLU preconditioners with one- and two-level overlappings—*$BJ(1)$*,* $BJ(2)$*,* $SLU(1)$*, and* $SLU(2)$*, respectively.*

TABLE 4.2
Number of FGMRES(20) *iterations for the* RAEFSKY1 *problem.*

Name	Precon	`lfil`	4	8	16	24	36	40
raefsky1	SAPINV	10	14	13	10	11	8	8
		20	12	11	9	9	8	8
	SAPINVS	10	16	13	10	11	8	8
		20	13	11	9	9	8	8
	SLU	10	215	197	198	194	166	171
		20	48	50	40	42	41	41
	BJ	10	85	171	173	273	252	263
		20	82	170	173	271	259	259

known exactly.

For a given problem, iteration counts for the SAPINV and SAPINVS suggest a clear trend of achieving convergence in fewer iterations with increasing number of processors, which means that a high degree of parallelism of these preconditioners does not impede convergence, and may even enhance it significantly (cf. rows 1–4 of Table 4.2). The main explanation for this is the fact that the approximations to the local and global Schur complement matrices, from which the global preconditioner M is derived, actually improve as the processor numbers become larger since these matrices become smaller. Furthermore, SAPINV, SAPINVS, and SLU do not suffer from the information loss as happens with BJ (since BJ disregards the local matrix entries corresponding to the external interface vector components). Note that the effectiveness of BJ degrades with increasing number of processors (cf. Subsection 4.1). Comparison of SAPINV and SAPINVS (for RAEFSKY1 and SHERMAN3) confirms the conclusions of [6] that using Y_i to approximate $B_i^{-1}F_i$ (Line 5 in Algorithm 3.2) is more efficient than applying $B_i^{-1}F_i$ directly, which is also computationally expensive. For general distributed matrices, this is especially true, since iterative solves with B_i may be very inaccurate.

In the experiments, sparse approximations of Y_i appear to be quite accurate (usually reducing the Frobenius norm to 10^{-2}), which could be attributed to the small dimensions of the matrices used in approximations. This reduction in the Frobenius norm was attained in 10 iterations of the Minimal Residual (MR) method (see, e.g., [25]). Smaller numbers of iterations were also tested. Their effect on the overall

TABLE 4.3
Number of FGMRES(20) *iterations for the* AF23560 *problem.*

Name	Precon	lfil	16	24	32	40	56	64	80	96
af23560	SAPINV	20	32	36	27	29	73	35	71	61
		30	32	35	23	29	46	60	33	52
	SAPINVS	20	32	35	24	29	55	35	37	59
		30	32	34	23	28	43	45	23	35
	SLU	20	81	105	94	88	90	76	85	71
		30	38	34	37	39	38	39	38	35
	BJ	20	37	153	53	60	77	80	95	*
		30	36	41	53	57	81	87	97	115

TABLE 4.4
Number of FGMRES(20) *iterations for the* SHERMAN3 *problem.*

Name	Precon	lfil	16	24	32	40	48	56	60
sherman3	SAPINV	10	52	42	21	17	31	21	20
		20	54	40	18	17	29	20	19
	SAPINVS	10	54	48	21	18	33	21	21
		20	55	51	19	16	30	19	19
	SLU	10	34	32	16	20	20	23	17
		20	21	23	12	16	15	18	13
	BJ	10	158	155	72	208	110	122	84
		20	163	155	72	206	111	120	85

solution process amounted to on average one extra iteration of FGMRES(20) for the problems considered here.

We point out that a few other experiments using methods described in this paper as well as related methods are reported in [28].

5. Conclusion. In this paper, several preconditioning techniques for distributed linear systems are derived from approximate solutions with the related Schur complement system. The preconditioners are built upon the already available distributed data structure for the original matrix, and an approximation to the global Schur complement is *never* formed explicitly. Thus no communication overhead is incurred to construct a preconditioner, making the preprocessing phase simple and highly parallel. The preconditioning operations utilize the communication structure precomputed for the original matrix.

The preconditioning to the global matrix A is defined in terms of a block LU factorization which involves a solve with the global Schur complement system at each preconditioning step. This system is in turn solved approximately with a few steps of GMRES exploiting approximations to the local Schur complement for preconditioning. Two different techniques, incomplete LU factorization and approximate-inverse, are used to approximate these local Schur complements. Distributed preconditioners constructed and applied in this manner allow much flexibility in specifying approximations to the local Schur complements and local system solves and in defining the global induced Block-LU preconditioner to the original matrix.

With an increasing number of processors, a Krylov subspace method, such as FGMRES [23], preconditioned by the proposed techniques exhibits a very moderate growth in the execution time for scaled problem sizes. Experiments show that the proposed distributed preconditioners based on Schur complement techniques are superior to the commonly used Additive Schwarz preconditioning. In addition, this advantage comes at no additional cost in code-complexity or memory usage, since the same data

structures as those for additive Schwarz preconditioners can be used.

Acknowledgments. The authors thank the referees for their valuable comments and suggestions. Computing resources for this work were provided by the Minnesota Supercomputer Institute and the Virginia Tech Computing Center.

REFERENCES

[1] T. Barth, T. F. Chan, and W.-P. Tang, *A Parallel Algebraic Non-overlapping Domain Decomposition Method for Flow Problems*, Tech. Report NAS 98-002, 1998.

[2] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Math., 22 (1982), pp. 127–140.

[3] P. E. Bjørstad, *Multiplicative and Additive Schwarz methods: Convergence in the two-domain case*, in Domain Decomposition Methods, T. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., SIAM, Philadelphia, PA, 1989.

[4] P. E. Bjørstad and O. B. Widlund, *To overlap or not to overlap: A note on a domain decomposition method for elliptic problems*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1053–1061.

[5] X.-C. Cai, W. D. Gropp, and D. E. Keyes, *A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems*, Numer. Linear Algebra Appl., 1 (1994), pp. 477–504.

[6] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM J. Sci. Comput., 18 (1997), pp. 1657–1675.

[7] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.

[8] T. Davis, *University of Florida sparse matrix collection*, NA Digest, 97 (1997); also available online at http://www.cise.ufl.edu/~davis/sparse.

[9] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[10] V. Eijkhout and T. Chan, *ParPre a Parallel Preconditioners Package, Reference Manual for Version* 2.0.17, Tech. Rep. CAM Report 97-24, UCLA, 1997.

[11] W. D. Gropp and B. F. Smith, *Experiences with domain decomposition in three dimensions: Overlapping Schwarz methods*, in Proceedings of the Sixth International Symposium on Domain Decomposition Methods, AMS 1993, pp. 323–333.

[12] B. Hendrickson and R. Leland, *The Chaco User's Guide, Version* 2.0, Tech. Rep. SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1994.

[13] T. Huckle and M. Grote, *A New Approach to Parallel Preconditioning with Sparse Approximate Inverses*, Tech. Rep. SCCM-94-03, Scientific Computing and Computational Mathematics Program, Stanford University, Stanford, CA, 1994.

[14] S. A. Hutchinson, J. N. Shadid, and R. S. Tuminaro, *Aztec User's Guide. Version* 1.0, Tech. Rep. SAND95-1559, Sandia National Laboratories, Albuquerque, NM, 1995.

[15] G. Karypis and V. Kumar, *Metis: Unstructured Graph Partitioning and Sparse Matrix Ordering System*, Tech. Rep., Department of Computer Science, University of Minnesota, 1995; also available online at http://www.cs.umn.edu/~karypis/metis.

[16] D. E. Keyes and W. D. Gropp, *A comparison of domain decomposition techniques for elliptic partial differ ential equation and their parallel implementation*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. s166–s202.

[17] S. Kuznetsov, G. C. Lo, and Y. Saad, *Parallel Solution of General Sparse Linear Systems*, Tech. Rep. UMSI 97/98, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1997.

[18] G.-C. Lo and Y. Saad, *Iterative Solution of General Sparse Linear Systems on Clusters of Workstations*, Tech. Reps. UMSI 96/117, UM-IBM 96/24, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1996.

[19] S. Ma and Y. Saad, *Distributed ILU*(0) *and SOR Preconditioners for Unstructured Sparse Linear Systems*, Tech. Rep. 94–027, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, 1994.

[20] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, *MPI: The Complete Reference*, 2nd ed., Vol. 2, *The MPI-2 Extensions*, MIT Press, Cambridge, MA, 1998.

[21] A. Pothen, H. D. Simon, and K. P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.
[22] Y. Saad, *Krylov subspace methods on parallel computers*, in Solving Large-Scale Problems in Mechanics: Parallel and Distributed Computer Applications, M. Papadrakakis, ed., John Wiley, New York, 1996.
[23] Y. Saad, *A flexible inner-outer preconditioned* GMRES *algorithm*, SIAM J. Sci. Statist. Comput., 14 (1993), pp. 461–469.
[24] Y. Saad, *ILUT: A dual threshold incomplete ILU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.
[25] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, New York, 1996.
[26] Y. Saad and A. Malevsky, PSPARSLIB: *A portable library of distributed memory sparse iterative solvers*, in Proceedings of Parallel Computing Technologies (PaCT-95), 3rd International Conference, St. Petersburg, Russia, 1995, V. E. Malyshev et al., eds., 1995.
[27] Y. Saad and M. H. Schultz, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[28] Y. Saad, M. Sosonkina, and J. Zhang, *Domain Decomposition and Multi-Level Type Techniques for General Sparse Linear Systems*, Contemp. Math. 218, AMS, Providence, RI, 1998.
[29] Y. Saad, *Parallel sparse matrix library (P_SPARSLIB): The iterative solvers module*, in Advances in Numerical Methods for Large Sparse Sets of Linear Equations 10, Matrix Analysis and Parallel Computing, Keio University, Yokohama, Japan, 1994, pp. 263–276.
[30] B. Smith, P. Bjørstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, New York, 1996.
[31] B. Smith, W. D. Gropp, and L. C. McInnes, *PETSc* 2.0 *User's Manual*, Tech. Rep. ANL-95/11, Argonne National Laboratory, Argonne, IL, 1995.