Distributed Schur Complement Techniques
for General Sparse Linear Systems *

Yousef Saad†  Masha Sosonkina‡

October 16, 1997

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 exploit 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.

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 to direct solvers due to their more moderate computational and storage demands. A typical “Grand-Challenge” application requires the use of powerful parallel computing platforms as well as 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 and 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

*This work was supported in part by ARPA under grant number NIST 60NANB2D1272, in part by NSF under grant CCR-9618827, and in part by the Minnesota Supercomputer Institute.
†Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455, e-mail:saad@cs.umn.edu.
‡Department of Computer Science, 320 Heller Hall, 10 University Drive, Duluth, Minnesota 55812-2496. masha@d.umn.edu.
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: extracting parallelism from efficient sequential techniques or designing a preconditioner from the start specifically for parallel platforms. Each of these two 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 addresses mainly 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 [15, 14, 18, 20, 10, 1, 2, 8] 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, [12, 2, 8]. In particular, several distributed preconditioners included in the ParPre package [8] employ variants of Schur complement techniques. One difference between our work and [2] 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

$$Ax = b,$$  \hspace{1cm} (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 $A$. 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 that used in the PSPARSLIB package, see [20, 22, 14] for additional details.

Figure 1 shows a “physical domain” viewpoint of a sparse linear system. This representation borrows from the domain decomposition literature – so the term “subdomain” is often used instead of the more proper term “subgraph”. 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 non-local (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 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.

![Diagram of local view of a distributed sparse matrix](image)

Figure 1: A local view of a distributed sparse matrix.

In Figure 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:

$$A_i x_i + X_i y_{i,\text{ext}} = b_i.$$  \hspace{1cm} (2)

where $x_i$ represents the vector of local unknowns, $y_{i,\text{ext}}$ 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 $y_{i,\text{ext}}$ from other processors, and finally multiply the received data by $X_i$ and add the result to that already obtained with $A_i$. 

3
Figure 2: A partitioned sparse matrix.

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.,

\[
x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix} ; \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix}.
\]

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

\[
A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix},
\]

so the local equations (2) can be written as follows:

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

(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 (4).

The preprocessing phase should construct the 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 [20, 22, 14].

3 Derivation of Schur complement techniques

Schur complement techniques refer to 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 equation (2) and its block form (4). Schur complement systems are derived by eliminating the variable $u_i$ from the system (4). Extracting from the first equation $u_i = B_i^{-1}(f_i - F_i y_i)$ yields, upon substitution in the second equation,

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

where $S_i$ is the “local” Schur complement

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

The equations (5) 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:

$$
\begin{pmatrix}
S_1 & E_{12} & \cdots & E_{1p} \\
E_{21} & S_2 & \cdots & E_{2p} \\
\vdots & \ddots & \ddots & \vdots \\
E_{p1} & E_{p-1,2} & \cdots & S_p
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2 \\
\vdots \\
y_p
\end{pmatrix}
= 
\begin{pmatrix}
g'_1 \\
g'_2 \\
\vdots \\
g'_p
\end{pmatrix}
$$

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 global system (4), are sparse.

The system (7) can be written as

$$S y = g', $$

5
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

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

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 (7) 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 (resp. SOR) on the global system (see, e.g., [11], [19]). The $k$-th step of a block-Jacobi iteration on the global system takes the following local form:

$$x_i^{(k+1)} = x_i^{(k)} + A_i^{-1} r_i^{(k)}$$

$$= x_i^{(k)} + A_i^{-1} \left( b_i - A_i x_i^{(k)} - \left( \sum_{j \in N_i} E_{ij} y_j^{(k)} \right) \right)$$

$$= A_i^{-1} \left( g_i - \sum_{j \in N_i} E_{ij} y_j^{(k)} \right)$$

$$= \left( -S_i^{-1} E_i B_i^{-1} \right) \left( g_i - \sum_{j \in N_i} E_{ij} y_j^{(k)} \right).$$

(8)

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

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

(9)

or equivalently

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

(10)

which is nothing but a Jacobi iteration on the Schur complement system (7).

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

$$x^{(k+1)} = M x^{(k)} + c.$$ (11)

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

$$y^{(k+1)} = G y^{(k)} + h.$$ (12)
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 [13, 12]. The sequence $y^{(k)}$ can be accelerated with a Krylov subspace algorithm, such as GMRES [21]. One way to look at this acceleration procedure is to consider the solution of the system

$$(I - G)y = h. \quad (13)$$

The right-hand side $h$ can be obtained from one step of the iteration (12) 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

$$s^{(0)} = h - (y^{(0)} - Gy^{(0)}) = y^{(1)} - y^{(0)}.$$ 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 $B\delta = \gamma$ (needed to operate with the matrix $S_i$) should be accurate. We can compute the dense matrix $S_i$ explicitly or solve system (5) 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 [23]), 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 (1) by exploiting methods that approximately solve the reduced system (7). These techniques, termed “induced preconditioners” (see, e.g., [19]), can be best explained by considering a reordered version of the global system (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

$$\begin{pmatrix} B_1 & F_1 \\ B_2 & F_2 \\ \vdots & \vdots \\ B_p & F_p \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_p \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_p \end{pmatrix}, \quad (14)$$
which also can be rewritten as
\[
\begin{pmatrix}
B & F \\
E & C
\end{pmatrix}
\begin{pmatrix}
u \\
g
\end{pmatrix} =
\begin{pmatrix}
f \\
g
\end{pmatrix}.
\]  
(15)

Note that the \(B\) block acts on the interior unknowns. Eliminating these unknowns from the system leads to the Schur complement system (7).

Induced preconditioners for the global system are obtained by exploiting a block LU factorization for \(A\). Consider the factorization
\[
\begin{pmatrix}
B & F \\
E & C
\end{pmatrix} =
\begin{pmatrix}
B & 0 \\
E & S
\end{pmatrix}
\begin{pmatrix}
I & B^{-1}F \\
0 & I
\end{pmatrix},
\]  
(16)

where \(S\) is the global Schur complement
\[
S = C - EB^{-1}F.
\]

This Schur complement matrix is identical to the coefficient matrix of system (7) (see, e.g., [19]).

The global system (15) can be preconditioned by an approximate LU factorization constructed such that
\[
L = 
\begin{pmatrix}
B & 0 \\
E & M_S
\end{pmatrix} \quad \text{and} \quad U = 
\begin{pmatrix}
I & B^{-1}F \\
0 & I
\end{pmatrix},
\]  
(17)

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. The preconditioners presented next are based on the global system of equations for the Schur unknowns (system (5) or the equivalent form (7)) and on the global block LU factorization (16), or rather its approximated version (17).

### 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 (4) and (5). As is readily seen from (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
\[
B_iu_i = f_i - F_iy_i
\]
with the matrix \(B_i\) in each processor. In practice, it is often simpler to solve a slightly larger system obtained from (2) or
\[
A_ix_i = b_i - X_iy_{i,\text{ext}}
\]  
(18)

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, (5) is rewritten as a preconditioned system with the diagonal blocks:
\[
y_i + S_i^{-1} \sum_{j \in \mathcal{N}_i} E_{ij}y_j = S_i^{-1} \left[ y_i - E_i B_i^{-1} f_i \right].
\]  
(19)
Note that this is simply a block-Jacobi preconditioned Schur complement system. System (19) 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 [19]). Let $A_i$ have the form (3) and be factored as $A_i = L_iU_i$, where

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

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 (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:

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

from which the result $S_i = L_{S_i} U_{S_i}$ follows by comparison with (16).

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 (5), or equivalently (19), the following approximate (local) Schur complement system derived from (19) can be considered on each processor $i$:

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

(20)

The global system related to equations (20) 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. equation (17)). The global preconditioner (17) can then be defined from $M_S$.

Given a local ILU factorization

$$A_i = L_i U_i + R_i,$$
with which the factorization
\[ S_i = L_S U_S + R_S, \]
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 (20). Then, the interior vector components are calculated using equation (18) (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 vector \(g_i\) associated with the interface variables.

**Algorithm 3.1** Approximate Schur-LU solution step with GMRES

1. Given: local right-hand side \(\text{rhs} = (f_i)\)
2. Define an \((m + 1) \times m\) matrix \(\tilde{H}_m\) and set \(\tilde{H}_m := 0\).
3. Arnoldi process:
   4. \(y_i := 0\)
   5. \(r := (L_i U_i)^{-1} \text{rhs}\)
   6. \(v_1 := P r / \| P r \|_2\)
   7. For \(j = 1, \ldots , m\) do
      8. Exchange interface vector components \(y_i\)
      9. \(t := (L_S U_S)^{-1} X_i y_{i, \text{ext}}\)
      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 \(V_m := [v_1, \ldots , v_m]\).
18. Form the approximate solution for interface variables:
   19. Compute \(y_i := y_i + V_m z_m\), where
   20. \(z_m = \arg \min_z \| \beta e_1 - \tilde{H}_m z \|_2\) and \(e_1 = [1, 0, \ldots , 0]^T\).
21. Find other local unknowns:
   22. Exchange interface vector components \(y_i\)
   23. \(t := X_i y_{i, \text{ext}}\)
   24. \(\text{rhs} := \text{rhs} - \begin{pmatrix} 0 \\ t \end{pmatrix}\)
   25. \(\begin{pmatrix} u_i \\ y_i \end{pmatrix} := (L_i U_i)^{-1} \text{rhs}\).

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 (8), we have
\[
A_i^{-1} \begin{pmatrix} f_i \\ g_i \end{pmatrix} = \left( S_i^{-1} (g_i - E_i B_i^{-1} f_i) \right),
\]
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 that on the right-hand side of (20), 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 (20).

### 3.5 Schur complements via approximate inverses

Equation (17) describes in general terms an approximate block LU factorization for the global system (15). 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 [3]:

\[
\min_{Q \in S} \| I - AQ \|_F^2,
\]

in which \( S \) is a certain set of \( n \times n \) sparse matrices. This minimization problem can be decoupled into \( n \) minimization problems of the form

\[
\min_{m_j} \| e_j - Am_j \|_2^2, \quad j = 1, 2, ..., n,
\]

where \( e_j \) and \( m_j \) are the \( j \)th columns of the identity matrix and a matrix \( Q \in S \), 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 [4] and [9]. In [9] the initial sparsity pattern is taken to be diagonal with further fill-in allowed depending on the improvement in the minimization. The work [4] 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 \( j = 1, 2, ..., n \). In this paper, the approximate-inverse technique proposed in [4] and [5] 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 (16). 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 [5]). The resulting matrix \( Y_i \) is sparse and therefore

\[
M_{S_i} = C_i - E_i Y_i
\]

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 (7), i.e., by solving its approximated
\[
\tilde{S}_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i, \tag{22}
\]

the right-hand side of which can be also computed approximately. System (22) 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 \( \tilde{M} \) to the matrix \( A \) can be defined from considering the global system (14), also written as (15). The Schur variables correspond to the bottom part of the linear system. The global preconditioner \( M \) is given by the block factorization (17) in which \( M_S \) is the approximation to \( S \) obtained by \textit{iteratively solving system} (22).

Thus, the block forward-backward solves with the factors (17) will amount to the following three-step procedure.

1. Solve \( Bu = f \);
2. Solve (iteratively) the system (22) to obtain \( y_i \);
3. Compute \( u := u - B^{-1} F_i y_i \).

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 \( \text{rhs} = (f_i, g_i) \)
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 \( M_{S_i} y_i + X_i y_{i, \text{ext}} = \tilde{g}_i \)
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 (22) is calculated. In Line 5, there are several choices for approximating \( t := B_i^{-1} F_i y_i \). 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 \( M_{S_i} \) (equation (21)).
4 Numerical experiments

In the experiments, we compared the performance of the described preconditioners and the distributed Additive Schwarz preconditioning (see, e.g., [19]) on 2-D elliptic PDE problems and on several problems with the matrices from the Harwell-Boeing and Davis collections [7]. A flexible variant of restarted GMRES (FGMRES) [16] 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 1fil fill-in elements. Recall that ILUT [17] 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: $M_S$ and $B_i^{-1}F_i$ are approximated using the matrix $Y_i$, constructed using the approximate-inverse technique described in [4];

**SAPINV$S$** Distributed approximate block LU factorization: $M_S$ is approximated using the approximate-inverse technique in [4], 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 $S_i \approx L_S U_S$;

**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

$$Lu = \frac{\partial}{\partial x} \left( a \frac{\partial}{\partial x} u \right) + \frac{\partial}{\partial y} \left( b \frac{\partial}{\partial y} u \right) + \frac{\partial}{\partial x} (du) + \frac{\partial}{\partial y} (eu) = u$$

(23)

on rectangular regions with Dirichlet boundary conditions.

If there are $n_x$ points in the $x$ direction and $n_y$ points in the $y$ direction, 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 belongs to a processor. In fact, each of the subproblems associated with these subrectangles are generated in parallel. Figure 3 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 $360 \times 360$ mesh mapped to (virtual) square processor grids of increasing size is given in Figure 4. (In Figure 4, we omit the solution time for the BJ preconditioning, 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. Five iterations
of GMRES with the relative tolerance $10^{-2}$ were used in the application of BJ and SLU. For SAPIINV, 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 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 $15 \times 15$, $30 \times 30$, $50 \times 50$, and $70 \times 70$ are presented in Figure 5. (Premature termination of the curves for SI indicates nonconvergence in 300 iterations). The growth in the solution 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.

### 4.2 General problems

Table 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 one level of overlapping with data exchanging was used following [13].

Tables 2–4 show iteration numbers required by FGMRES(20) with SAPIINV, SAPIVNS, SLU,
Figure 4: Times and iteration counts for solving a 360 x 360 discretized Laplacean problem with 3 different preconditioners using flexible GMRES(10).

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>nz</th>
<th>Pattern</th>
<th>Discipline</th>
</tr>
</thead>
<tbody>
<tr>
<td>af23560</td>
<td>23560</td>
<td>484256</td>
<td>Symm</td>
<td>Airfoil, eigenvalue calculation</td>
</tr>
<tr>
<td>raefsky1</td>
<td>3242</td>
<td>22316</td>
<td>Unsymm</td>
<td>Incompressible flow in pressure driven pipe</td>
</tr>
<tr>
<td>sherman3</td>
<td>5005</td>
<td>20033</td>
<td>Symm</td>
<td>Oil reservoir modeling</td>
</tr>
</tbody>
</table>

Table 1: Description of test problems

and BJ till convergence on different numbers of processors. An asterisk indicates nonconvergence. In the preconditioning phase, approximate solves in each processor were carried out by GMRES to reduce the residual norm by $10^{-3}$ but no more than for five steps were allowed. As a choice of ILU factorization, ILUT with $1f1l$ fill-in elements (shown in the column $1f1l$) was used in the experiments here. $1f1l$ 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, $1f1l$ specifies the number of nonzeros in the blocks of preconditioning; for SLU, $1f1l$ is the total number of nonzeros in the preconditioning, therefore, the number of nonzeros in a given approximation $S$ is not 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 2). The main explanation for this is the fact that the approximations to the local and global Schur complement matrices from
Figure 5: Solution times for a Laplacean problem with various local subproblem sizes using FGMRES(10) with 3 different preconditioners (BJ, SAPINV, SLU) and the Schur complement iteration (SI).
<table>
<thead>
<tr>
<th>Name</th>
<th>Precon</th>
<th>lfil</th>
<th>16</th>
<th>24</th>
<th>32</th>
<th>40</th>
<th>56</th>
<th>64</th>
<th>80</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>raefty1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAPINV</td>
<td>10</td>
<td>14</td>
<td>13</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAPINVS</td>
<td>10</td>
<td>16</td>
<td>13</td>
<td>11</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>13</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLU</td>
<td>10</td>
<td>215</td>
<td>197</td>
<td>198</td>
<td>194</td>
<td>166</td>
<td>171</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>48</td>
<td>50</td>
<td>40</td>
<td>42</td>
<td>41</td>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BJ</td>
<td>10</td>
<td>85</td>
<td>171</td>
<td>173</td>
<td>273</td>
<td>252</td>
<td>263</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td>82</td>
<td>170</td>
<td>173</td>
<td>271</td>
<td>259</td>
<td>259</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Number of FGMRES(20) iterations for the RAEFSKY1 problem.

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

Table 3: Number of FGMRES(20) iterations for the AF23560 problem.

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

Table 4: Number of FGMRES(20) iterations for the SHERMAN3 problem.
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 [5] 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 MR method. Smaller numbers of iterations were also tested. Their effect on the overall solution process amounted to an average one extra iteration of FGMRES(20) for the problems considered here.

5 Conclusions

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 [16], 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 Computing resources for this work were provided by the Minnesota Supercomputer Institute and the Virginia Tech Computing Center.
References


