Parallel Solution of General Sparse Linear Systems[*]

Sergey Kuznetsov[†] Gen-Ching Lo[‡] Yousef Saad[§]

Abstract. This paper discusses a few algorithms and their implementations for solving distributed general sparse linear systems. The preconditioners used are all variations of techniques originating from domain decomposition ideas. In particular we compare a number of variants of Schwarz procedures with Schur complement techniques. Numerical experiments on a few parallel platforms are reported.

1 Introduction

We consider a linear system of the form

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

where A is a large sparse nonsymmetric real matrix of size n. Such a system can arise, for example, from a finite element discretization of a partial differential equation on a certain domain. To solve this system on a distributed memory computer, it is common to partition the finite element mesh by a graph partitioner and assign a cluster of elements which represent a physical subdomain to each processor. Each processor then assembles only the local equations associated with the elements assigned to it. In case the system is already in assembled form, it is natural to assign pairs of equations-unknowns to the same processor, using again some graph partitioner. In either case, each processor will wind up holding a set of equations (rows of the linear system) and a vector of the variables associated with these rows. This natural way of distributing a sparse linear system is fairly general (see [18], [14]) and is closely related to the physical viewpoint.

This paper addresses mainly the issue of defining preconditioners for distributed sparse linear systems. Such systems are regarded a distributed objects and methods are developed for solving the global system by using the distributed data structure. 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 variables and external variables. A preprocessing phase is thus required to determine this and any other information needed during the iteration phase.

Figure 1 shows a 'physical domain' viewpoint of a sparse linear system. This representation borrows from the domain decomposition literature – so the term 'subdomain'

[*]Work supported in part by ARPA under grant NIST 60NANB2D1272 and in part by NSF under grant CCR-9618827.

[†]Institute of Mathematics, Novosibirsk, Russia. This work was carried out while the author was on leave at the University of Minnesota, Department of Computer Science, Minneapolis, MN.

[‡]Morgan-Stanley, New-York

[§]University of Minnesota, Department of Computer Science, Minneapolis, MN.

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. As is often done, we will distinguish between three types of unknowns: (1) Interior variables are those that are coupled only with local variables by the equations; (2) Local interface variables are those coupled with non-local (external) variables as well as local variables; and (3) External interface variables are those variables in other processors which are coupled with local variables.

Along with this figure, we can represent the local equations as shown in Figure 2. The local equations do not correspond to contiguous equations in the original system. The matrix represented in the figure can be viewed as a reordered version of the equations associated with a local numbering of the equations/unknowns pairs.

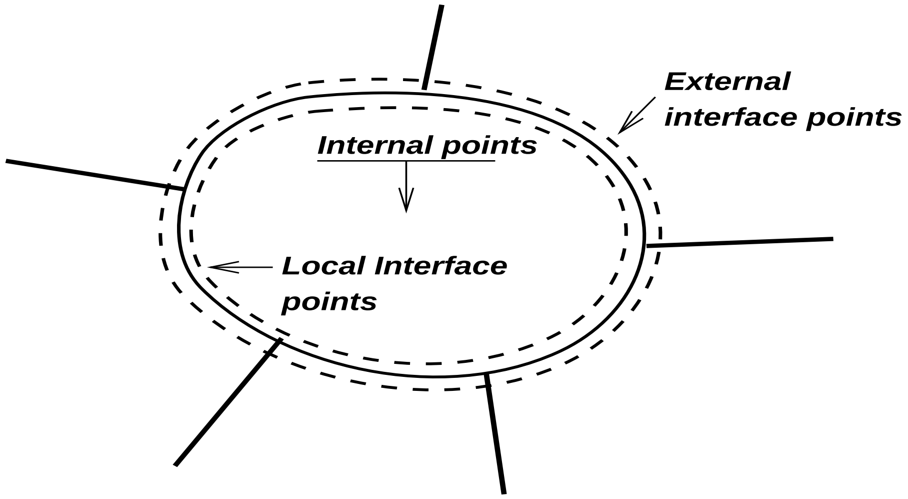

FIG. 1. *A local view of a distributed sparse matrix.*

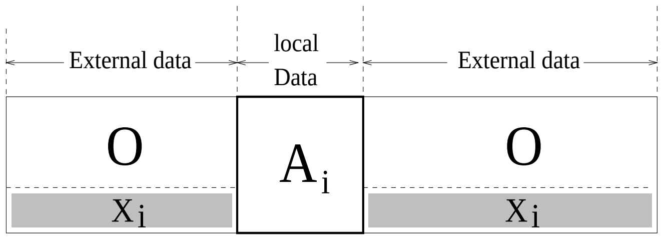

FIG. 2. *A partitioned sparse matrix.*

As can be seen in Figure 2, the rows of the matrix assigned to a certain processor have been split into two parts: a *local* matrix A_i which acts on the local variables and an *interface* matrix X_i which acts on remote variables. These remote variables must be first received from other processor(s) before the matrix-vector product can be completed in these processors. A key feature of the data structure is the separation of the boundary points from the interior points. The interface nodes are always listed last after the interior nodes. This ‘local ordering’ of the data presents several advantages, including more efficient interprocessor communication, and reduced local indirect addressing during matrix-vector products. It should be noted that the using of block sparse row format can also give a sufficient reduction in indirect addressing. The zero blocks shown are due to the fact that local internal nodes are not coupled with external nodes.

Thus, each local vector of unknowns x_i is split in two parts: the subvector u_i of internal nodes followed by the subvector y_i of local interface variables. The right-hand side b_i is conformally split in the subvectors f_i and g_i,

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

The local matrix A_i residing in processor i as defined above is block-partitioned according to this splitting, leading to

$$A_i = \left(\begin{array}{c|c} B_i & E_i \\ \hline F_i & C_i \end{array} \right) . \qquad (2)$$

With this, the local equations can be written as follows.

$$\begin{pmatrix} B_i & E_i \\ F_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix} \qquad (3)$$

The term $E_{ij}y_j$ is the contribution to the local equation from the neighboring subdomain number j and N_i is the set of subdomains that are neighbors to subdomain i. The sum of these contributions, seen on the left side of of (3) is the result of multiplying a certain matrix by the external interface variables. It is clear that the result of this product will affect only the local interface variables as is indicated by the zero in the upper part of the second term in the left-hand side of (3). For practical implementations, the subvectors of external interface variables are grouped into one vector called $y_{i,ext}$ and the notation

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

will be used to denote the contributions from external variables to the local system (3). In effect this represents a local ordering of external variables to write these contributions in a compact matrix form. With this notation, the left-hand side of the (3) becomes

$$w_i = A_i x_i + X_{i,ext} y_{i,ext} \qquad (4)$$

Note that w_i is also the local part the matrix-by vector product Ax in which x is a vector which has the local vector components x_i, $i = 1, \ldots, s$.

To facilitate matrix operations and communication, an important task is to gather the data structure representing the local part of the linear matrix as was just described. In this preprocessing phase it is also important to form any additional data structures required to prepare for the intensive communication that will take place during the solution 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 pieces of data that must be sent and received to/from the "neighboring processors".

The complete description of the data structure associated with this boundary information is given in [33] along with additional implementation details.

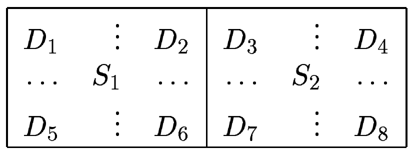

FIG. 3. *An example of splitting eight domains on the plane into two groups*

2 Distributed Krylov Subspace Solvers

The primary accelerator which we use is a flexible variant of GMRES called FGMRES [29]. This is a right-preconditioned variant of GMRES which allows variations in the preconditioner at each step. Details on the implementations of parallel Krylov accelerators are given in [33, 35, 28]. In particular, a reverse communication protocol is used to avoid passing data structures related to the coefficient matrix or the preconditioner. With this implementation, the FGMRES code itself contains no communication calls. All such calls are relegated to either matrix-vector operations which are performed outside the main body of the acceleration code, or dot-product operations which usually rely on special library calls.

In this paper we discuss mostly preconditioning techniques based on domain decomposition ideas. These include the block-Jacobi method (additive Schwarz), multi-color block SOR (multiplicative Schwarz), and Schur complement techniques. Each of the preconditioners can use overlapping of the subdomains.

2.1 Additive Schwarz

The simplest preconditioner is the so-called additive Schwarz procedure. This form of block Jacobi iteration, in which the blocks refer to systems associated with entire domains, is sketched next.

ALGORITHM 2.1. 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 $x_i = x_i + \delta_i$
```

To solve the systems which arise in line 3 of the above algorithm, a standard (sequential) ILUT preconditioner [31] combined with GMRES or one step of an ILU preconditioner is used. Of particular interest in this context are the overlapping additive Schwarz methods. In the domain decomposition literature [1, 3, 37, 19] it is known that overlapping is a good strategy to reduce the number of steps. There are however several different ways of implementing overlapping block Jacobi iterations, for example, we can replace the data in the overlapping subregions by its external version or use some average of the data.

It is sometimes possible to reduce the number of outer iterations for block Jacobi iteration by a 2-level clustering of the subdomains. The subdomains can be been split into groups $S_1, S_2, \ldots, S_p$. The assignment of the subdomains to groups can be determined from knowledge of neighboring subdomains. Figure 3 gives an example of splitting eight domains into two groups. Consider a certain group of subdomains, say S_k, and the set of equations-unknowns corresponding to all the subdomains belonging to S_k. The local system for subdomain $i \in S_k$ can be rewritten in the form:

$$A_i x_i + Z_{i,ext} z_{i,ext} + (X_{i,ext} y_{i,ext} - Z_{i,ext} z_{i,ext}) = w_i$$

where $z_{i,ext}$ is a part of the vector $y_{i,ext}$ corresponding to remote variables in the same group as subdomain i, $Z_{i,ext}$ is a part of the interface matrix $X_{i,ext}$ which acts on the remote variables $z_{i,ext}$. Thus solving the system

$$A_i x_i + Z_{i,ext} z_{i,ext} = w_i$$

in the same manner as an initial system can be considered as a preconditioner for block Jacobi iteration. This form of block Jacobi iteration, referred to as two-level Jacobi iteration, is sketched next.

ALGORITHM 2.2. Hierarchical Block Jacobi Iteration

```
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. Do $i = 1, \ldots, i_1$
4.         Obtain external data $z_{i,ext}$
5.         Compute (update) local residual $s_i = r_i - Z_i z_{i,ext}$
6.         Solve $A_i \delta_i = s_i$
7. End
8. Update solution $x_i = x_i + \delta_i$
```

The number of inner iteration i_1 in the algorithm given above depends on the problem. Often, $i_1 = 2$ is optimal because this choice achieves a good compromise between accuracy for the local block solver and overall performance.

The hierarchical block Jacobi iteration presents several advantages over the traditional block Jacobi iteration. One of these advantages is that it allows us to reduce the number of outer iterations due to increased accuracy in the preconditioning step. On the other hand, this approach increases communication but the cost of communication is small if the processors form small groups.

2.2 Multiplicative Schwarz

The multiplicative Schwarz preconditioner (block SOR) uses the same extended domains as the additive Schwarz method, but it has a sequential component in the solve: every processor uses interface variables defined by preceding local solves. The simplest form of the multiplicative Schwarz is the block Gauss-Seidel algorithm used in domain decomposition techniques [2, 19, 37, 5].

The global ordering can be based on an arbitrary labeling of the processors provided two neighboring domains have a different label. The most common global ordering is a multi-coloring of the domains, which maximizes parallelism [4, 30, 36, 37].

Thus, if the domains are colored, the multiplicative Schwarz as executed in each processor would be as follows.

ALGORITHM 2.3. Multicolor Multiplicative Schwarz procedure

```
1. Do $col = 1, \ldots, numcols$
2.     If $(col.eq.mycol)$ then
```

```
3.          Obtain external data $y_{i,ext}$
4.          Update local residual $r_i = (b - Ax)_i$
5.          Solve $A_i \delta_i = r_i$
6.          Update solution $x_i = x_i + \delta_i$
7.      EndIf
```

Many variations of the above algorithm are possible, including the overlapping of the domains, inaccurate solves in step 5, inclusion of a relaxation parameter ω, etc.

Algorithm 2.3 is executed on each processor and a convergence test on the global residual or some measure of the error must be included. It is known that multicoloring can reduce communication times at the expense of a more complicated code. Another problem with multicoloring is that as the domain associated with the given color is active, all other colors will be inactive. As a result it is typical to obtain only $1/numcol$ efficiency where numcol is the number of colors. To this end, one can further block the local variables into two blocks: interior and interface variables. Then the global SOR iteration is performed with this additional blocking.

In effect, each local matrix A_i is split as

$$A_i = \begin{pmatrix} B_i & E_i \\ F_i & C_i \end{pmatrix} = \begin{pmatrix} B_i & 0 \\ 0 & C_i \end{pmatrix} + \begin{pmatrix} 0 & E_i \\ F_i & 0 \end{pmatrix} \tag{5}$$

where the B_i part corresponds to internal nodes.

The segregated multiplicative Schwarz looks as follows,

ALGORITHM 2.4. Segregated multiplicative Schwarz

```
1. Solve $B_i \delta_{i,x} = r_{i,x}$
2. $x_i := x_i + \delta_{i,x}$
3. Do $col = 1, \ldots, numcols$
4.      If $(col.eq.mycol)$ then
5.          Obtain external data $y_{i,ext}$
6.          Update y-part of residual $r_{i,y}$
7.          Solve $C_i \delta_{i,y} = r_{i,y}$
8.          Update interface unknowns $y_i = y_i + \delta_{i,y}$
9.      EndIf
```

The advantages of this procedure is that the bulk of the computational work in each domain is done in parallel. Loss of parallelism comes from the color loop which involves only solves with interfaces, which are of lower complexity.

2.3 Schur complement techniques

Schur complement techniques refer to methods which iterate on the interface unknowns only, implicitly using internal unknowns as intermediate variables. A general strategy for deriving Schur complement techniques will now be described associated with arbitrary global fixed point iterations.

Consider the simplest case of a block-Jacobi iteration described earlier. The Schur complement system is derived by eliminating the variable u_i from the system (3) extracting from the first equation $u_i = B_i^{-1}(f_i - E_i y_i)$ which yields, upon substitution in the second equation,

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - F_i B_i^{-1} f_i \tag{6}$$

in which S_i is the 'local' Schur complement:

$$S_i = C_i - F_i B_i^{-1} E_i \tag{7}$$

The equations (6) for all subdomains i altogether constitute a system of equations which involves only the interface points y_j, $j = 1, 2, \ldots, s$ and which has a natural block structure associated with these vector variables. The diagonal blocks in this system, namely the matrices S_i, are dense in general but the off-diagonal blocks E_{ij} are sparse. As is known, with a consistent choice of the initial guess, a block-Jacobi (or SOR) iteration with the reduced system is equivalent with a block Jacobi iteration on the global system, see, e.g., [22], [32]. A block Jacobi iteration on the global system takes the following local form:

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

Here a $*$ denotes a nonzero block whose actual expression is unimportant. The important observation is that the y iterates satisfy an independent relation of the form,

$$y_i^{(k+1)} = S_i^{-1} \left[g_i - F_i B_i^{-1} f_i - \sum_{j \in N_i} E_{ij} y_j^{(k)} \right] \tag{8}$$

If we call g_i' the right-hand side of the reduced system (6) then (8) can be rewritten as

$$y_i^{(k+1)} = y_i^{(k)} + S_i^{-1} \left[g_i' - S_i y_i^{(k)} - \sum_{j \in N_i} E_{ij} y_j^{(k)} \right] \tag{9}$$

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

In summary, the sequence of the y-part of the Jacobi vectors on the global system can be viewed as a sequence of Jacobi iterates for the Schur complement system. A similar result holds for the multiplicative Schwarz as well. From a global viewpoint, we have a *primary* iteration for the global variable of the form,

$$x^{(k+1)} = M x^{(k)} + c \tag{10}$$

and the vectors of interface variables y associated with these iterates satisfy an iteration of the form,

$$y^{(k+1)} = G y^{(k)} + h \ . \tag{11}$$

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.

Now the idea is to accelerate the sequence $y^{(k)}$ with a Krylov subspace algorithm such as GMRES. One way to look at this acceleration procedure is that we are attempting to solve the system

$$(I - G) y = h \tag{12}$$

To solve the above system with a Krylov-type method an initial guess and corresponding residual are needed. Also, the Krylov iteration requires a number of matrix-vector product operations. The right-hand side h can be obtained from one step of the iteration (11) 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 original iteration. To compute $w = (I - G)y$ proceed are as follows,

1. Perform one step of the primary iteration $\binom{u'}{y'} = M \binom{0}{y} + c$;
2. set $w := y'$;
3. Compute $w := y - w + h$

This strategy allows to derive a Schur complement technique for any primary fixed-point iteration on the global unknown. Among the possible choices are the Jacobi and SOR iterations as well as iterations derived (somewhat artificially) from ILU preconditioning techniques. The main advantages of this viewpoint are the generality and flexibility of the formulation.

It should be mentioned that the diagonal blocks B_i in representation (3) for local matrices, are sparse and often banded. In these cases, the system

$$(C_i - F_i B_i^{-1} E_i) y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - F_i B_i^{-1} f_i \tag{13}$$

can be solved by a Krylov subspace method using the LU decomposition for banded block B_i or the Cholesky decomposition if B_i is symmetric positive definite. Thus we can easily compute the matrix-vector product $B_i^{-1}x$ using one step of the LU solve. Note that the solve for the system $B_i \delta = \gamma$ must be accurate. We can compute the dense matrix S_i explicitly or solve the above system by using a computation of the matrix-vector product $S_i x$ which can be carried out with three sparse matrix vector multiplies and one linear system solve. As is known (see [37]), because of the large computational expense of these accurate solves, the resulting decrease in iteration counts does not, in itself, make the Schur complement attractive. Therefore, we need to use efficient preconditioners for the Schur complement. From the preconditioning side, using the explicit calculation of Schur complements is more efficient but it requires a large amount of memory and is quite expensive. However, we can use a block Jacobi iteration in the form

$$y_i^{(k+1)} = C_i^{-1} \left[g_i' + F_i B_i^{-1} E_i y_i^{(k)} - \sum_{j \in N_i} E_{ij} y_j^{(k)} \right] \tag{14}$$

as a preconditioner.

3 Numerical Experiments

In this section, we report on some results obtained for solving distributed sparse linear systems on an IBM SP2 with 14 nodes, an IBM cluster of 8 workstations, and an SGI

Challenge cluster and a 64 processor CRAY-T3E. The SGI challenge workstation cluster consists of three 4-processor Challenge L servers and one 8-processor Challenge XL server. The processors on the Challenge L and XL are the same (R10,000) but the memory sizes are different. Communication between different SGI cluster workstations (respectively, IBM RS/6000 workstations) can be performed via a HiPPI (High Performance Parallel Interface) switch or a Fibre-Channel switch. On the IBM cluster, an ATM switch is used. The MPI communication library is used for all communication calls. The ATM and HiPPI are high speed interfaces and can transfer data at 155 Mbps and 800 Mbps, respectively. The IBM RS/6000 Model 590 workstations are based on the Power2 architecture. The SP2 nodes communicate with an internal switch which achieves a bandwidth of 320 Mbps bandwidth and has a latency of about 40 microseconds. The CRAY-T3E has 64 nodes and 128 Megabytes of memory per node. The processors are connected in a 3-D torus by a network capable of 480 Mbps in each direction for each link.

The main Krylov accelerator used in the examples is the flexible variant of GMRES known as FGMRES. This is a right-preconditioned variant which allows the preconditioning to vary at each step. For further details on the algorithm, see [29]. Reverse communication (see e.g. [8],[27], [11]) allows to have exactly the same codes work on parallel and sequential platforms.

3.1 Experiments with block Jacobi preconditioning

The execution time depends on the time spent in the local solver. If GMRES is used as a inner solver, factors which can affect convergence are the tolerance, the level of fill for the ILUT preconditioner and the number of inner iterations. There exists a unique set of the parameters mentioned above that minimize the execution time for the fixed number of processors. Figure 4 shows the execution time in seconds as a function of the level of fill for a ILUT preconditioner, using a relative tolerance of $\varepsilon = 10^{-6}$, a Krylov subspace dimension of $m = 30$, and a number of inner iterations of 10. The numerical experiments are done for the RAEFSKY3 matrix from the SIMON collection on the SGI cluster for three processors (using the HiPPI switch). This matrix of dimension 21,200 has 1,448,768 non-zero entries. There were no convergence in 250 FGMRES steps through level of fill from 0 to 15. It can be seen from this figure that an increase in the level of fill from 26 does not minimize the execution time. We observed the same behavior for the execution time as a function of the number of inner iterations and for other types of the block Jacobi iteration. The optimal choice of a level of fill and a number of inner iterations depends on the number of processors. As in the sequential case, using the best level of fill can give significant reduction in the total time.

We now compare the results obtained with the distributed block Jacobi iteration using the three different overlapping options as described in Section 4. These results are summarized in Figures 5 and 6. In the figures, *jacno* stands for block Jacobi with no overlapping, *jaco_av* for block Jacobi with overlapping and averaging of the overlapping data, *jac* for block Jacobi with overlapping and exchange of overlapped data.

A standard ILUT preconditioner combined with GMRES was used as a local solver and the number of inner iterations was equal to $its = 11$. We select $its = 11$ because it minimizes the execution time for a small number of processors if preconditioned GMRES is used as a local solver. Note that the optimal set of parameters depends on the number of processors and the problem considered. It is sometimes reasonable to use a smaller number for the level of fill or a smaller number of inner iterations if the number of processors is

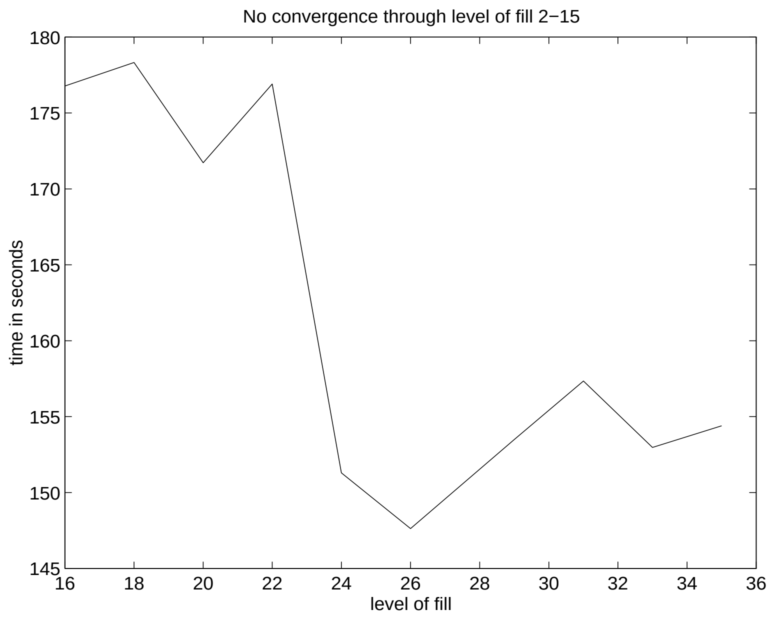

FIG. 4. *Total time in seconds as a function of level of fill for the local ILUT preconditioner*

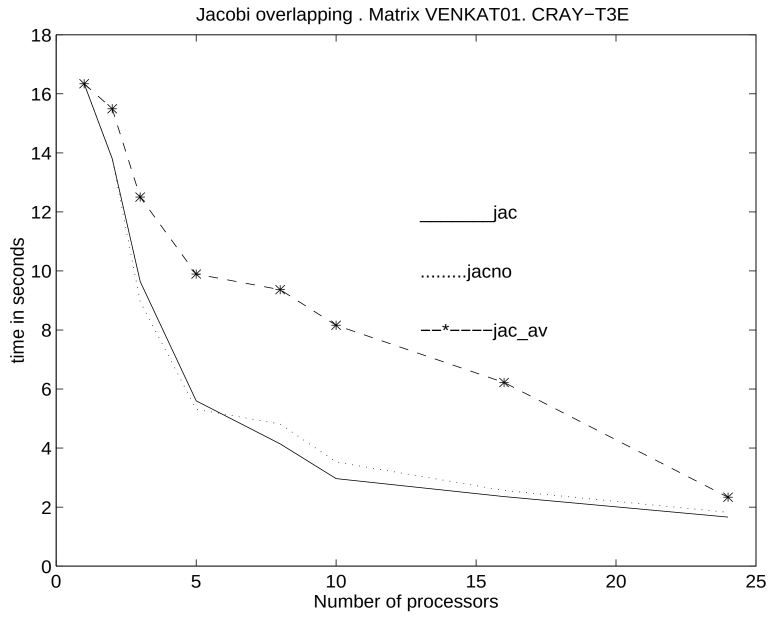

FIG. 5. *Comparison of FGMRES with distributed block Jacobi preconditioner for three different overlapping strategies on the CRAY-T3E*

large. The results are for the matrix VENKAT01 of dimension 62,424 with 1,717,792 non-zero entries on the CRAY-T3E, using a relative tolerance of $\varepsilon = 10^{-6}$, a Krylov subspace dimension of $m = 50$ and a level of fill 25 for the ILUT preconditioner. The comparison shows that overlapping can reduce the number of outer iterations and *jac* usually requires the smaller number of iterations.

Our numerical tests show that using one step of an ILU solve with an accurate ILUT

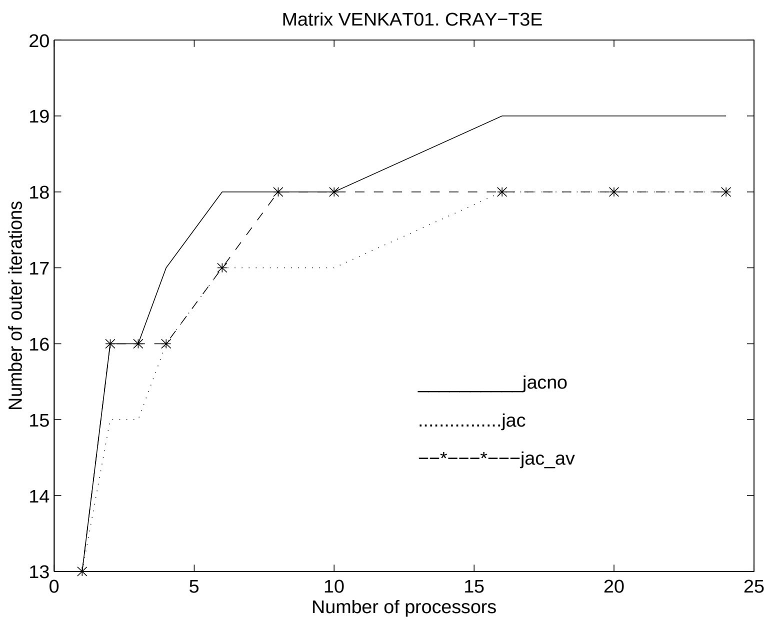

FIG. 6. *Iteration count for block Jacobi on the CRAY-T3E*

factorization is usually more economical. Table 1 gives timing results, speed-ups and iteration count for the block Jacobi preconditioner with overlapping for two different local solvers. In the table *its* is the number of FGMRES iterations. These results are given for the VENKAT01 matrix on the CRAY-T3E, using a relative tolerance of $\varepsilon = 10^{-6}$, a Krylov subspace dimension of $m = 50$ and a level of fill of 25 for the ILUT preconditioner. A comparison of the results shows that using an ILU solves as a local solver is more efficient than preconditioned GMRES with the ILUT preconditioner.

Number of processors		1	3	5	10	16	24
preconditioned GMRES	time	16.34	9.64	5.59	2.96	2.35	1.66
	speed-up		1.69	2.92	5.52	6.95	9.84
	its	13	16	17	18	19	19
ILU solver	time	8.85	3.78	2.49	1.39	1.04	0.77
	speed-up		2.34	3.55	6.37	8.51	11.49
	its	15	18	20	21	21	21

TABLE 1

Execution times in seconds, speed-up and number of outer iterations for the block Jacobi preconditioner with two different local solvers

The block Jacobi iteration considered in this paper is a synchronous algorithm. Synchronization among processors take place in the orthogonalization step in FGMRES and during the computation of matrix-vector product. As a rule, we compute matrix-vector product after the preconditioning step and processors must wait until the communication network delivers all messages issued by the processors to their destinations.

There are several approaches that can be used to improve load balanced among processors for the block Jacobi preconditioned GMRES. An approach is to change the number of inner iterations for each processor starting from a certain iteration of FGMRES, say k, based on the time in the preconditioning step for a few first iterations. This 'forced load balancing' approach allows to increase the accuracy in the preconditioning step and, as a result, reduce the number of outer iterations. For example, the number n^i of inner iteration for processor i can be defined by

$$n^i = [n^i_{\text{old}} * \frac{t_{\max}}{t^i_{\max}}] \tag{15}$$

where n^i_{old} is the initial number of inner iterations, $t_{\max}$ is the maximum time in the preconditioner step for all processors, $t^i_{\max}$ is the maximum time in the preconditioner step for processor i.

In our numerical tests on the sp2 and on the ibm rs6000 cluster we found that the approach mentioned above can give a 10% to 25 % reduction in the execution time when the number of processors is small. Table 2 gives timing results and iteration counts for the block Jacobi preconditioner with overlapping when two local solver is based on preconditioned GMRES. In the table *its* is the number of FGMRES iteration, the last two lines in the table correspond to the block Jacobi iteration with preconditioned GMRES as a local solver where the number of inner iterations was fixed. The first two lines correspond to the Jacobi preconditioner iteration with GMRES where the number of inner iteration were defined by (15) starting from the third iteration. These results are given the VENKAT01 matrix on the IBM RS6000 cluster with an ATM switch, using a relative tolerance of $\varepsilon = 10^{-6}$, a Krylov subspace dimension of $m = 20$ and a level of fill of 15 for the ILUT preconditioner. The initial number of inner steps is 10.

Number of processors		2	3	4	6	7
MODIFIED	time	16.07	11.74	9.19	8.50	7.17
	its	9	9	9	10	11
STANDARD	time	26.96	15.29	13.21	10.63	9.64
	its	11	10	11	11	13

TABLE 2

Execution times in seconds, number of outer iterations for the block Jacobi preconditioner with two different local solvers on on the IBM RS6000 cluster with an ATM switch.

The next matrix, referred to as BARTH1S, was supplied by T. Barth of NASA Ames [7]. It is for a 2D high Reynolds number airfoli problem, with a turbulence model and this matrix has a 5x5 block structure. Two rows in each block are for the momentum equations, and one row each is for mass balance, energy balance, and turbulence balance. The matrix is of order 189,370 with 6,260,236 nonzero entries. The condition number of this matrix is very high (see [7]). The system with BARTH1S is solved with the deflated GMRES [6]. The local systems are solved using one step an ILU solve, where the LU factors are obtained from a block ILU(k) preconditioner (see [7]) where k denotes a level of fill. Table 2 gives the timing results, and the number of matrix-vector multiplications. The number in the first column indicates the number of processors involved.

The following parameters have been used: the level of fill was 4, the number of deflated eigenvectors was 8, the Krylov subspace dimension was 50, and the tolerance threshold was 1.E-5.

It should be noted that this problem could not be solved for 8 or fewer processors due to limitation in memory space. It was also impossible to find the solution for a level of fill less than 3 and using fewer than 12 deflated eigenvectors. On the other hand, an increase

PEs	Seconds	Matvecs
8	308.2158	399
10	254.3837	493
16	170.8574	589
20	165.6654	727

TABLE 3

Execution times (seconds) and number of matrix-vector multiplications for the BARTH1S matrix

in the number of processors leads to worse convergence. We partitioned this matrix block by block so that each processor has its own part of the physical domain.

3.2 Experiments with hierarchical block Jacobi

We now present a comparison between the Jacobi overlapping with the forward-backward LU solver and the hierarchical Jacobi preconditioner for elliptic problems on the plane. It was assumed that all processors have been split into groups by means of the following function $f_b(j)$ defined by:

$$f_b(j) = \lfloor (j-1)/b \rfloor + 1$$

where b is a blocking factor for processors. Table 4 gives timing results and iteration count for the hierarchical block Jacobi preconditioner using four different numbers of inner iterations. The results are given for the Laplace operator on the unit square. A standard forward-backward local ILU solver has been applied in these numerical tests. The blocking factor for processors was 2, the number of processors of 8, a tolerance of 10^{-6}, the level of fill of 5, the Krylov subspace dimension of 30.

Inner its	Preconditioner	Seconds	Matvecs
1	hierarchical	15.2319	238
2	hierarchical	9.8849	108
3	hierarchical	15.0807	127
5	hierarchical	18.0245	106
	ILU solve	15.1467	238

TABLE 4

Execution times in seconds, number of matrix-vector multiplications for the hierarchical Jacobi preconditioner as a function of number of inner iterations. The number of processors is 8, the problem size is 90,000.

Table 4 shows that an increase in the number of inner iterations leads to a significant reduction in the number of outer iterations. On the other hand, the execution time increases as the number of inner iterations increases from 2 to 5 due to an increase in computational costs. The number of inner iterations of 2 seems to give the best compromise in this case. Table 5 gives some other timing results and iteration counts. Here the number of inner iterations is fixed to 2, the blocking factor is two subdomains, the tolerance is 10^{-6}, the level of fill is 5, and the Krylov subspace dimension is 30.

3.3 Experiments with Multiplicative Schwarz

Figure 7 shows a comparison of multiplicative Schwarz for different types of local solver on the CRAY-T3E for the VENKAT01 matrix.

Problem size	Preconditioner	PEs	Seconds	Matvecs
25,600	hierarchical	4	2.5507	39
25,600	ILU solve	4	2.9977	70
51,200	hierarchical	8	4.3356	61
51,200	ILU solve	8	5.2002	114
102,400	hierarchical	16	5.2211	73
102,400	ILU solve	16	5.8928	126
158,000	hierarchical	20	5.0396	81
158,000	ILU solve	20	5.8001	135

TABLE 5

Comparison of the hierarchical Jacobi preconditioner and standard Jacobi preconditioner with ILU solve.

In the figure the "multicolor segregated SOR, ILU solver" refers to the segregated multicolor multiplicative Schwarz in which the interface data is solved for after the interior data. The "multicolor SOR with ILU solver" refers to the segregated multicolor multiplicative Schwarz in which the interface data is only solved for. On each subdomain the systems for both options mentioned above are solved using one step an ILU solve, with LU factors obtained from an ILUTP factorization (incomplete LU factorization with dual truncation mechanism) for the first option and an ILUT factorizations for the second one. The "multicolor SOR" refers to the multicolor multiplicative Schwarz in which a standard GMRES with the ILUT preconditioner is used for finding the interior unknowns as well as the interface unknowns. The "multicolor segregated SOR, ILU solver and GMRES" refers to the segregated multicolor multiplicative Schwarz procedure as described by Algorithm 2.4 in which a standard GMRES with the ILUT preconditioner is used for finding the interior unknowns but the systems for the interface unknowns are solved using a single forward-backward ILU solve, with LU factors obtained from the ILUT factorization. The set of parameters (level of fill of 15, number of inner iterations of 3, and Krylov subspace dimension of 50) was fixed for all the options listed above. As was already explained, the Multicolor SOR scheme used here involves substantial idle time for each processor since only one color is active at any given time. The figure shows once again that requiring more accuracy for ILU local solves leads to a reduction in the total time. Also the use of a single ILU solve leads to good savings in computational time.

3.4 Experiments with Schur complement techniques

Before discussing the results obtained with the Schur complement technique, it is worth pointing out that these techniques are often implemented in conjunction with direct solvers. These solvers are invoked either to compute the actual Schur complement matrix, in forming the Schur complement system, or for solving the successive linear systems which arise during the iterative process. If an iterative process is to be used instead of a direct solver, it is important to note that the solves involved with the Schur complement iteration process must be accurate. This represents the main weakness of Schur complement techniques.

Table 6 gives the timing results, the number of matrix-vector multiplications for solving the systems for the interface data using a relative tolerance of $\varepsilon = 10^{-5}$, a Krylov subspace dimension of $m = 50$, a level of fill of 25, and a number of inner iteration of 10. All of the computation were done according to the description of Section 2.3. A preconditioned GMRES using the ILUT preconditioner has been used as the local solves.

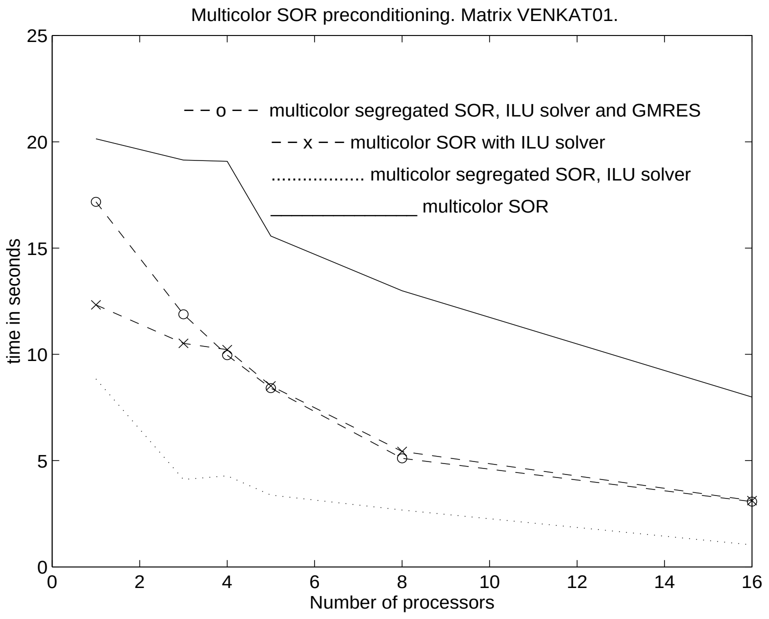

FIG. 7. *Comparison of the multiplicative Schwarz with four different options.*

PEs	Seconds	Matvecs
2	30.80	9
3	20.51	9
6	14.35	11
8	10.58	10
16	5.75	11

TABLE 6

Execution times in seconds, number of matrix-vector multiplications, number of processors for the Schur complement technique

The table 7 gives the comparison between a block Jacobi iteration with overlapping and a Schur complement for the SHERMAN3 matrix on the CRAY-T3E, using a relative tolerance of $\varepsilon = 10^{-6}$, a Krylov subspace dimension of $m = 50$ and a level of fill of 25, a number of iteration for the preconditioning step of 10. This matrix is of order 5,005 with 20,033 non-zero entries. For the block Jacobi iteration we used exchange of overlapped data and a preconditioned GMRES with the ILUT preconditioner as a local solver. The Schur complement solved by a preconditioned Krylov method where the preconditioner defined by equation (14) has been used. We used one step of an ILU solve associated with an ILUT factorization of C_i instead of the exact C_i^{-1}. The matrices B_i^{-1} are factored exactly by an LU factorization. The numerical tests show that the Schur complement technique can be competitive with standard preconditioners if the subdomains assigned to each processor are relatively small, i.e., they become competitive for larger number of processors. It should be noted that in fact, most of the time is spent in computing the LU decomposition. It is observed that there are significant savings in the orthogonalization time (FGMRES) due to the shorter vectors involved in the FGMRES iteration. The explicit calculations of Schur complements are usually more expensive for this matrix than the approach described above.

Number of processors		2	3	6	8	12
Jacobi preconditioning	time	1.37	0.94	0.89	0.71	0.50
	its	15	15	28	33	30
Schur complement	time	6.11	3.19	1.35	0.56	0.39
	its	1	10	33	33	29

TABLE 7

Execution times in seconds, number of outer iterations for the block Jacobi preconditioner and the Schur complement technique on on the CRAY-T3E for the SHERMAN3 matrix.

3.5 Where is the time spent?

Figure 8 presents a breakdown of the times spent in a typical solution on the CRAY-T3E. Times are given in seconds. The results are for the matrix VENKAT01, using a relative tolerance of $\varepsilon = 10^{-6}$, a Krylov subspace dimension of $m = 50$ and a level of fill of 25 for the ILUT preconditioner. The solution was obtained by a block Jacobi preconditioner using one step of an ILU solve. The figure shows that all contributions decrease as the number of processors increases. A contribution of each part can vary depending on network characteristics and the type of preconditioner used. For example, we observed that the FGMRES time on the SGI cluster with a Fibre-Channel switch can increase slightly as the number of processors increases due to the high latency.

4 Conclusion

Our main conclusion is that an iterative method can be effectively used to solve very large sparse linear systems on computers with distributed memory architectures. Small systems can be handled more efficiently on a small number of processors or a single processor. The break-even point depends on many factors and is a function of the dimension of the matrix, the architecture parameters and iterative solution techniques used. For example, any improvements in latency and bandwidth will allow to solve smaller problems more efficiently. For very large problems, communication is small relative to computation, and the issues of avoiding idle time and achieving more effective utilization of the memory

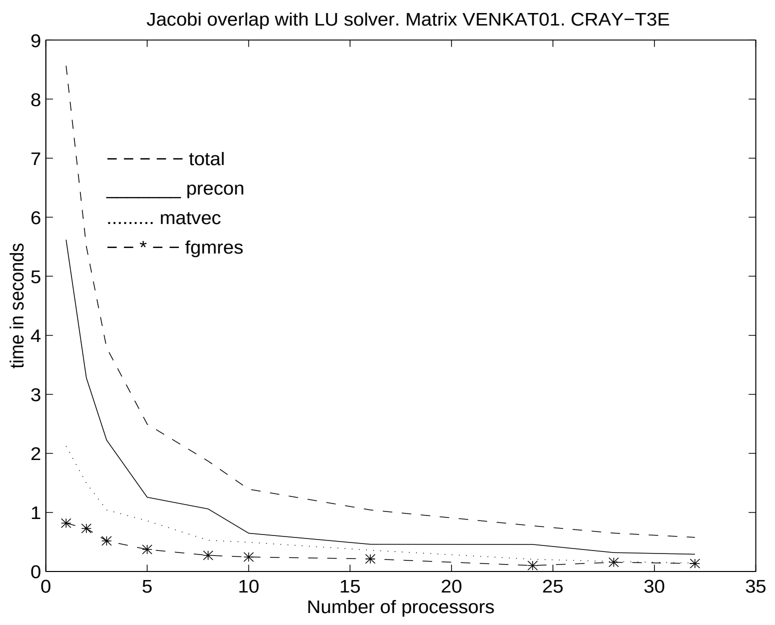

FIG. 8. *Contributions to total execution time for a distributed block Jacobi preconditioner with overlapping*

become more important.

Acknowledgement. The authors would like to acknowledge the support of the Minnesota Supercomputer Institute which provided the computer facilities and an excellent research environment to conduct this research. Additional computational resources were provided by the University of Minnesota IBM Shared Project.

References

[1] P. E. Bjørstad. *Multiplicative and Additive Schwarz Methods: Convergence in the 2 domain case.* In Tony Chan, Roland Glowinski, Jacques Périaux, and O. Widlund, editors, Domain Decomposition Methods, Philadelphia, PA, 1989. SIAM.

[2] P. E. Bjørstad and O. B. Widlund. *Iterative methods for the solution of elliptic problems on regions partitioned into substructures.* SIAM Journal on Numerical Analysis, 23(6):1093–1120, 1986.

[3] P. E. Bjørstad and O. B. Widlund. *To overlap or not to overlap: A note on a domain decomposition method for elliptic problems.* SIAM Journal on Scientific and Statistical Computing, 10(5):1053–1061, 1989.

[4] X. C. Cai and Y. Saad. *Overlapping domain decomposition algorithms for general sparse matrices.* Technical Report 93-027, Army High Performance Computing Research Center, Minneapolis, MN, 1993.

[5] T. F. Chan and T. P. Mathew. *Domain decomposition algorithms.* Acta Numerica, pages 61–143, 1994.

[6] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Tech. Report UMSI 95/181, Minnesota Supercomputer Institute, 1995.

[7] A. Chapman, Y. Saad, and L. Wigton *High-order ILU preconditioners for CFD problems*, Tech. Report UMSI 96/14, Minnesota Supercomputer Institute, 1996.

[8] *Documentation for Harwell library subroutines*, NS02, EA 15, March 1984.

[9] M. Dryja and O. B. Widlund. *Towards a unified theory of domain decomposition algorithms for elliptic problems.* In Tony Chan, Roland Glowinski, Jacques Périaux, and Olof Widlund, editors, Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, held in Houston, Texas, March 20-22, 1989. SIAM, Philadelphia, PA, 1990.

[10] J. Dongarra, J. Bunch, C. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, PA, 1979.

[11] J. Dongarra, V. Eijkhout, and V. Kahlan, *Reverse communication interface for linear algebra templates for iterative methods*, Tech. report CS-95-291, University of Tennessee, 1995.

[12] J. A. George and J. W. Liu, *Computer solution of large sparse positive definite systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[13] T. Goehring and Y. Saad, *Heuristic algorithms for automatic graph partitioning*, Technical Report UMSI 94-29, University of Minnesota Supercomputer Institute, Minneapolis, MN, 1994.

[14] W. Gropp, L. C. McInnes, and B. Smith, *Scalable libraries for solving systems of nonlinear equations and unconstrained minimization problems*, Proceeding of the Scalable Parallel Libraries Conference, IEEE Computer Society Press, pp. 60-67, 1995.

[15] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, 1994.

[16] B. Hendrickson and R. Leland *An improved spectral graph partitioning algorithm for mapping parallel computations*, Tech. Report SAND92-1460, Sandia National Laboratories, Albuquerque, New Mexico, 1992.

[17] B. Hendrickson and R. Leland *The chaco user's guide, version 1. 0*, Tech. Report SAND93-2339, Sandia National Laboratories, Albuquerque, New Mexico, 1993.

[18] S. A. Hutchinson, J. N. Shadid, and R. Tuminaro, *Aztec user's guide*, Tech. Report SAND95-1559, Sandia National Laboratories, Albuquerque, New Mexico, 1995.

[19] M. T. Jones and P. E. Plassmann *BlockSolve v1. 1: Scalable library software for the parallel solution of sparse linear systems*, Tech. Report ANL-92/46, Mathematics and Computer Science Division, Argonne National Laboratory, December, 1992.

[20] G. Karypis, *Graph partitioning and its application to scientific computing*, Ph. D. thesis, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1996.

[21] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Tech. Report TR95-037 , Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.

[22] D. E. Keyes and W. D. Gropp, *A comparison of domain decomposition techniques for elliptic partial differential equation and their parallel implementation*, SIAM J. Sci. Statist. Comput. , 8 (1987), pp. 856-869.

[23] G. -C. Lo and Y. Saad, *Iterative solution of general sparse linear systems on clusters of workstations*, Tech. Report UMSI 96/117, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1995.

[24] A. Pothen, H. D. Simon, and P. K. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix. Anal. Appl. , 11 (1990), pp. 430–452.

[25] A. Quarteroni, Yu. Kuznetsov, J. Periaux, and O. Widlund, editors. *Domain decomposition methods in science and engineering: The Sixth Iternational Conference on Domain Decomposition*, Como, Italy, June 15-19, 1992, Vol. 15. AMS, Providence, RI, 1994.

[26] Y. Saad, *Preconditioned Krylov subspace methods for CFD applications*, Tech. Report UMSI-94-171, Minnesota Supercomputer Institute, Minneapolis, MN 55415, August 1994.

[27] Y. Saad, *Data structures and algorithms for domain decomposition and distributed sparse matrix computations*, Tech. report 95-014, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995.

[28] , Y. Saad, *Krylov Subspace Methods in Distributed Computing Environments*, In *State of the art in CFD*, M. Hafez, editor, pp. 741-755, (1995).

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

[30] Y. Saad. *Highly parallel preconditioners for general sparse matrices.* In G. Golub, M. Luskin, and A. Greenbaum, editors, Recent Advances in Iterative Methods, IMA Volumes in Mathematics and Its Applications, volume 60, pages 165–199, New York, 1994. Springer Verlag.

[31] Y. Saad. *ILUT a dual threshold incomplete ILU factorization.* Numerical Linear Algebra with Applications, 1:387–402, 1994.

[32] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS publishing, New York, 1996.

[33] Y. Saad and A. Malevsky. *PSPARSLIB: A portable library of distributed memory sparse iterative solvers.* In V. E. Malyshkin et al. , editor, Proceedings of Parallel Computing Technologies (PaCT-95), 3-rd international conference, St. Petersburg, Sept. 1995, 1995.

[34] Y. Saad and M. H. Schultz. *GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems.* SIAM Journal on Scientific and Statistical Computing, 7:856–869, 1986.

[35] Y. Saad and K. Wu. *Design of an iterative solution module for a parallel sparse matrix library (P_SPARSLIB)*, In W. Shonauer, editor, Proceedings of the IMACS conference, Georgia, 1994.

[36] J. N. Shadid and R. S. Tuminaro. *A comparison of preconditioned nonsymmetric krylov methods on a large-scale mimd machine.* SIAM J. Sci Comput. , 15(2):440–449, 1994.

[37] B. Smith, P. Bjørstad, W. Gropp,*Domain decomposition: Parallel multilevel methods for elliptic partial differential equations*, Cabridge University Press, New-York, NY, 1996.