Domain Decomposition and Multi-Level Type Techniques for General Sparse Linear Systems*

Yousef Saad† Maria Sosonkina ‡ Jun Zhang§

February 12, 1998

Abstract

Domain-decomposition and multi-level techniques are often formulated for linear systems that arise from the solution of elliptic-type Partial Differential Equations. In this paper, generalizations of these techniques for irregularly structured sparse linear systems are considered. An interesting common approach used to derive successful preconditioners is to resort to Schur complements. In particular, we discuss a multi-level domain decomposition-type algorithm for iterative solution of large sparse linear systems based on independent subsets of nodes. We also discuss a Schur complement technique that utilizes incomplete LU factorizations of local matrices.

Key words: Schur complement techniques; Incomplete LU factorization; Schwarz iterations; Multi-elimination; Multi-level ILU preconditioners; Krylov subspace methods.

1 Introduction

A recent trend in parallel preconditioning techniques for general sparse linear systems is to exploit ideas from domain decomposition concepts and develop methods which combine the benefits of superior robustness of ILU-type preconditioning techniques with those of scalability of multi-level preconditioners. Two techniques in this class are the Schur complement technique (Schur-ILU) developed in [19] and the point and block multi-elimination ILU preconditioners (ILUM, BILUM) discussed in [16, 21].

The framework of the Schur-ILU preconditioner is that of a distributed sparse linear system, in which equations are assigned to different processors according to a mapping determined by a graph partitioner. The matrix of the related Schur complement system is also regarded as a distributed object and never formed explicitly. The main difference between our approach and the methods described in [2, 7], is that we do not seek to compute an approximation to the Schur complement. Simply put, our Schur-ILU preconditioner is an approximate solve for the global system which is derived by solving iteratively the Schur complement equations corresponding

*Work supported 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; URL: http://www.cs.umn.edu/~saad.

‡Department of Computer Science, University of Minnesota, Duluth, 320 Heller Hall, 10 University Drive, Duluth, Minnesota 55812-2496. masha@d.umn.edu.

§Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455; e-mail: jzhang@cs.umn.edu; URL: http://www.cs.umn.edu/~jzhang.

to the interface variables. For many problems, it has been observed that the number of steps required for convergence remains about the same as the number of processors and the problem size increase. The overall solution time increases slightly, at a much lower rate than standard Schwarz methods.

ILUM and BILUM exploit successive independent set orderings. One way to think of the idea underlying ILUM is that by a proper reordering of the original variables, the matrix is put in the form

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \tag{1}$$

where B is diagonal so that the Schur complement system associated with the C block remains sparse. Then the idea is applied recursively, computing a sequence of Schur complement (or reduced) systems. The last of these reduced systems is solved by an iterative solver. This recursively constructed preconditioner has a multi-level structure and a good degree of parallelism. Similar preconditioners have been designed and tested in [4] to show near grid-independent convergence for certain type of problems. In a recent report, some of these multi-level preconditioners have been tested and compared favorably with other preconditioned iterative methods and direct methods at least for the Laplace equation [3]. Other multi-level preconditioning and domain decomposition techniques have also been developed in finite element analysis or for unstructured meshes [1, 5].

The ILUM preconditioner has been extended to a block version (BILUM) in which the B block in (1) is block-diagonal. This method utilizes independent sets of small clusters (or blocks), instead of single nodes [4, 21]. In some difficult cases, the performance of this block version is substantially superior to that of the scalar version. The major difference between our approach and the approaches of Botta and Wubs [4] and Reusken [13] is in the choice of variables for the reduced system. In [4] and [13], the nodes in the reduced system, i.e., the unknowns associated with the submatrix C in (1), are those nodes of the independent set itself and this leads to a technique which is akin to an (algebraic) multigrid approach [14]. An approximate inverse technique is used to invert the top-left submatrix B in (1) which is no longer diagonal or block-diagonal. In their implementations, these authors employ a simple approximate inverse technique which usually requires diagonal dominance in the B matrix. In contrast, our approach chooses the nodes of the reduced system to be those unknowns associated with the complement to the independent set. Therefore the difference is that B is associated with the independent set instead of C. This approach is more akin to a domain decomposition technique and it is more generally applicable since it does not require diagonal dominance in either the B or C submatrix.

One aim of the current paper is to further extend BILUM techniques of [21] to include blocks of large size and to treat related issues of keeping sparsity of the BILUM factors. Measurable parameters are introduced to characterize the efficiency of a preconditioner. Numerical results with some hard-to-solve problems are presented to demonstrate the merits of the new implementations. For the Schur-LU preconditioner, we compare the performance of various options for solving the local problems and make some observations and recommendations.

2 Schur Complements and Recursive Schur Complements

Consider a linear system of the form

$$Ax = b, \tag{2}$$

where A is a large sparse nonsymmetric real matrix of size n. To solve such a system on a distributed memory computer, a graph partitioner is usually first invoked to partition the

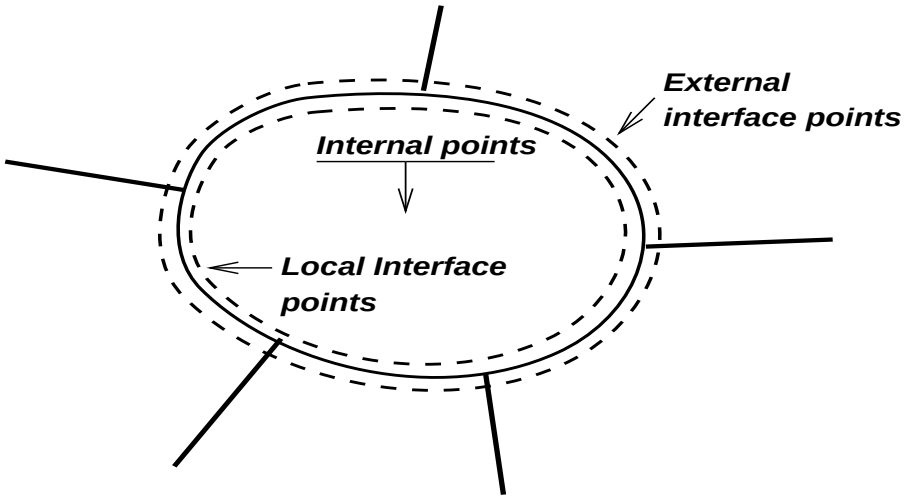

Figure 1: A local view of a distributed sparse matrix.

adjacency graph of A. The data is then distributed to processors such that pairs of equations-unknowns are assigned to the same processor. When this is done, three types of unknowns can be distinguished. (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. This setting which is illustrated in Figure 1, is common to most packages for parallel iterative solution methods [15, 18, 10, 23, 9, 7, 22, 11].

2.1 Distributed sparse linear systems

The matrix assigned to a certain processor is split into two parts: the *local* matrix A_i, which acts on the local variables and an *interface matrix* X_i, which acts on the external variables. Accordingly, the local equations can be written as follows:

$$A_i x_i + X_i y_{i,ext} = b_i, \tag{3}$$

where x_i represents the vector of local unknowns, $y_{i,ext}$ are the external interface variables, and b_i is the local part of the right-hand side vector. It is common to reorder the local equations in such a way that the interface points are listed last after the interior points. This ordering leads to an improved interprocessor communication and to reduced local indirect addressing during matrix-vector multiplication. Thus, the local variables form a local vector of unknowns x_i which 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. When the block is partitioned according to this splitting, the local equations (3) can be written as follows:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{4}$$

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. These contributions are the result of multiplying X_i by the external interface unknowns:

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

The result of this multiplication affects only the local interface unknowns, which is indicated by a zero in the top part of the second term of the left-hand side of (4).

2.2 Schur complement systems

This section gives a brief background on Schur complement systems; see e.g., [22, 17], for additional details and references. The Schur complement system is obtained 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,

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g'_i, \tag{5}$$

where S_i is the "local" Schur complement

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{6}$$

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:

$$\begin{pmatrix} S_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2 & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \ldots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g'_1 \\ g'_2 \\ \vdots \\ g'_p \end{pmatrix}. \tag{7}$$

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), which we rewrite in the form

$$Sy = g',$$

is the Schur complement system and S is the "global" Schur complement matrix.

2.3 Induced global preconditioners

It is possible to develop preconditioners for the *global system* (2) by exploiting methods that *approximately solve the reduced system* (7). These techniques, termed "induced preconditioners" (see, e.g., [17]), are based on a reordered version of the global system (2) 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,

$$\left(\begin{array}{cccc|c} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & B_p & F_p \\ \hline E_1 & E_2 & \cdots & E_p & C \end{array}\right) \left(\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_p \\ \hline y \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_2 \\ \vdots \\ f_p \\ \hline g \end{array}\right), \tag{8}$$

which also can be rewritten as

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}. \tag{9}$$

Note that the B block acts on the interior unknowns. Consider the block LU factorization

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & F \\ 0 & S \end{pmatrix}, \tag{10}$$

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., [17]). The global system (9) can be preconditioned by an approximate LU factorization constructed such that

$$L = \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} B & F \\ 0 & M_S \end{pmatrix} \tag{11}$$

with M_S being some approximation to S.

Note that the effect of the forward solve is to compute the modified right-hand side g' for the Schur complement system (7). Once this is done, the backward solve with the matrix U consists of two additional steps: solving with M_S to compute approximations to y, and then back-substituting to compute the approximations to the u variables.

Therefore, a global preconditioning operation induced by a Schur complement solve would consist of the following three steps.

1. Compute the reduced right-hand side $g' = g - EB^{-1}f$;
2. Approximately solve $M_S y = g'$;
3. Back-substitute for the u variables, i.e., solve $Bu = f - Fy$.

Each of the above three steps can be accomplished in different ways and this leads to a rich variety of options. For example, in (1) and (2) the linear system with B can be done either iteratively or directly, or approximately using just an ILU factorization for B. The choices for (2) are also numerous. One option considered in [19] starts by replacing (5) by an approximate system of the form,

$$y_i + \tilde{S}_i^{-1} \sum_{j \in N_i} E_{ij} y_j = \tilde{S}_i^{-1} \left[g_i - E_i B_i^{-1} f_i \right], \tag{12}$$

in which $\tilde{S}_i$ is some (local) approximation to the local Schur complement matrix S_i. This can be viewed as a block-Jacobi preconditioned version of the Schur complement system (7) in which the diagonal blocks S_i are approximated by $\tilde{S}_i$. The above system is then solved by an iterative accelerator such as GMRES requiring a solve with $\tilde{S}_i$ at each step. In one method considered in [19] the approximation $\tilde{S}_i$ was extracted from an Incomplete LU factorization of A_i. The idea is based on the following observation (see [17]). Let A_i be the matrix on the left-hand side of (4) and assume it is factored as $A_i = L_i U_i$, where

$$L_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \quad \text{and} \quad U_i = \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix}. \tag{13}$$

Then, $L_{S_i} U_{S_i}$ is equal to the Schur complement S_i associated with the partitioning (4), see [17, 19]. Thus, an approximate LU factorization to S_i can be obtained canonically from an approximate factorization to A_i, by extracting the related parts from the L_i and U_i matrices.

2.4 Recursive Schur complements

The diagonal blocks S_i in the Schur complement system (7) are usually dense blocks. However, the off-diagonal terms E_{ij} are sparse. As a result, it is interesting to consider a particular situation when the blocks B_i are all of small size, e.g., of size one or two. In this situation the coefficient matrix in (7) remains sparse and it is natural to think about a recursive application of the induced preconditioning technique described above. The second level of reduction can be applied to Schur complement system (7) resulting in the second-level Schur complement system ("the Schur complement for the Schur complement"). This process can be continued for a few more levels and the last level system can be solved with a standard iterative method. This idea was exploited in [16] for blocks B_i of the smallest possible size, namely one. In [21], the idea was generalized to blocks larger than one and a number of heuristics were suggested for selecting these blocks.

We recall that an independent set is a set of unknowns which are not coupled by an equation. A maximal independent set is an independent set that cannot be augmented by other elements to form another independent set. These notions can be generalized by considering subsets of unknowns as a group. Thus, a block independent set is a set of such groups of unknowns such that there is no coupling between unknowns of any two different groups. Unknowns within the same group may be coupled.

ILUM and BILUM can be viewed as a recursive applications of domain decomposition in which the subdomains are all of small size. In ILUM the subdomains are all of size one and taken together they constitute an independent set. In BILUM [21], this idea was slightly generalized by using block-independent sets, with groups (blocks) of size two or more instead of just one. As the blocks (subdomains) become larger, Schur complements become denser. However, the resulting Schur complement systems are also smaller and they tend to be better conditioned as well.

3 Block Independent Sets with Large Blocks

Heuristics based on local optimization arguments were introduced in [21] to find Block Independent Sets (BIS) having various properties. It has been shown numerically that selecting new subsets according to the lowest possible number of outgoing edges in the subgraph, usually yields better performance and frequently the smallest reduced system. These algorithms were devised for small independent sets. Extending these heuristic algorithms for extracting Block Independent Sets with large block sizes is straightforward. However, these extensions may have undesirable consequences. First, the cost is not linear with respect to the block size and it can become prohibitive as the block size increases. The second undesirable consequence is the rapid increase in the amount of fill-ins in the LU factors and in the inverse of the block diagonal submatrix. As a result, the construction and application of a BILUM preconditioner associated with a BIS having large subsets tend to be expensive [21].

Suppose a block independent set (BIS) with a uniform block size k has been found and the matrix A is permuted into a two-by-two block matrix of the form (with the unknowns of the independent set listed first)

$$A \sim PAP^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix}, \tag{14}$$

where P is a permutation matrix associated with the BIS ordering and $B = \mathrm{diag}[B_1, B_2, \ldots, B_s]$ is a block diagonal matrix of dimension $m = ks$, where s is the number of uniform blocks of size

k. The matrix C is square and of dimension $l = n - m$. In [21], a block ILU factorization of the form (11) is performed, i.e.,

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \approx \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \times \begin{pmatrix} B & F \\ 0 & A_1 \end{pmatrix} = \mathrm{L} \times \mathrm{U}. \tag{15}$$

Here $A_1 = C - EB^{-1}F$ is the Schur complement and I is the identity matrix. In order to maintain sparsity a dropping strategy is adopted when computing the submatrices EB^{-1} and A_1, based on a threshold tolerance. The BILUM preconditioner is obtained by recursively applying the above procedures to these successive Schur complements up to a certain number of levels, say $nlev$. The last reduced system obtained is solved by a direct method or a preconditioned iterative method.

Once the BILUM preconditioner is constructed, the solution process (application of BILUM) consists of the (block) forward and backward steps [16]. At each step (level), we partition the vector x_j as

$$x_j = \begin{pmatrix} y_j \\ x_{j+1} \end{pmatrix} \tag{16}$$

corresponding to the two-by-two block matrix (14) and perform the following steps:

Copy the right-hand side vector b to x_0.
For $j = 0, 1, \ldots, nlev - 1$, do forward sweep:
    Apply permutation P_j to x_j to partition it in the form (16).
    $x_{j+1} := x_{j+1} - E_j B_j^{-1} y_j$.
End do.
Solve with a relative tolerance ε:
    $A_{nlev} x_{nlev} := x_{nlev}$.
For $j = nlev - 1, \ldots, 1, 0$, do backward sweep:
    $y_j := B_j^{-1}(y_j - F_j x_{j+1})$.
    Apply inverse permutation P_j^{-1} to the solution y_j.
End do.

BILUM is, in effect, a recursive application of a domain decomposition technique. In the successive Schur complement matrices obtained, each block contains the internal nodes of a subdomain. The inverse and application of all blocks on the same level can be done in parallel. One distinction with traditional domain decomposition methods [12] is that all subdomains are constructed algebraically and exploit no physical information. In addition, the reduced system (coarse grid acceleration) is solved by a multi-level recursive process akin to a multigrid technique.

We define several measures to characterize the efficiency of BILUM (and other preconditioning techniques). The first one is called the *efficiency ratio* (e-ratio) which is defined as the ratio of the preprocessing time over iteration time, i.e., the ratio of the CPU time spent computing the BILUM preconditioner to that required by GMRES/BILUM to converge. The efficiency ratio determines how expensive it is to compute a preconditioner, relative to the time spent in the iteration phase. It should be used with the second measure which is the *total CPU* time which is the CPU time in seconds that a computer spends to compute the preconditioner and to solve the linear system. Given a total CPU time, a good preconditioner should not be too expensive to compute.

The third measure is the *sparsity ratio* (s-ratio) which is the ratio of the number of nonzeros of the BILUM factors to that of the matrix A. Note that the number of nonzeros of BILUM

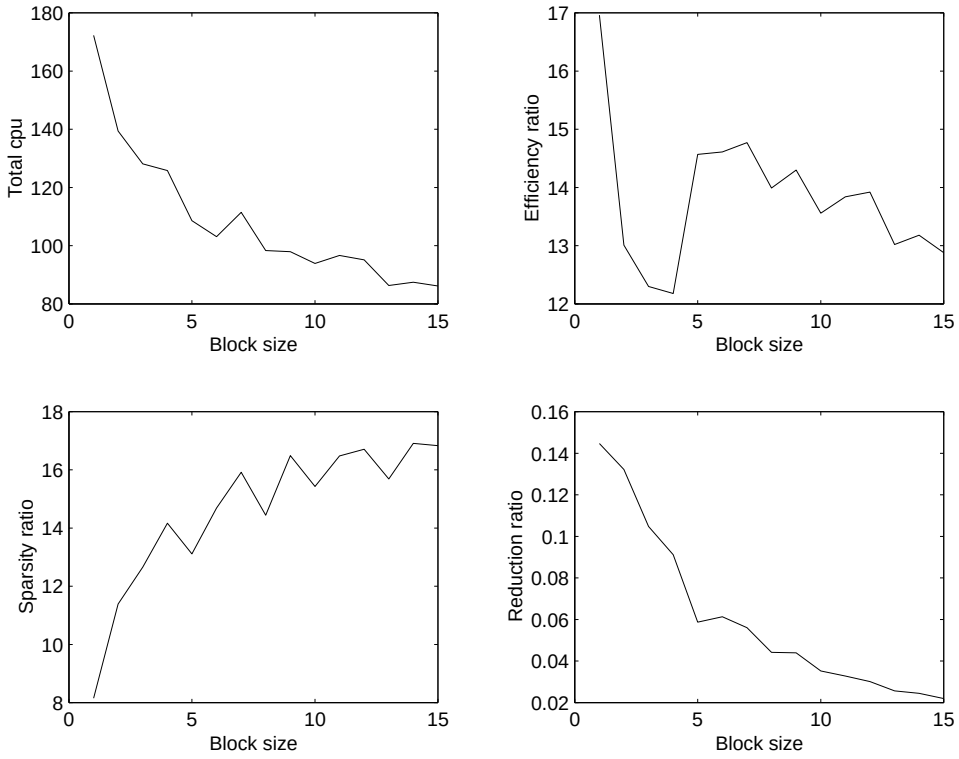

Figure 2: Characteristic measures for solving the 5-POINT matrix for different sizes of uniform blocks.

includes all the nonzeros of the LU factors at all levels plus the last reduced system and its preconditioner. The sparsity ratio determines how much memory is needed to store the given preconditioner, compared with that needed for the simplest preconditioner ILU(0). If the sparsity ratio is too large, a preconditioned iterative solver may lose one of its major advantages over a direct solver. The fourth measure is the *reduction ratio* (r-ratio) which is the ratio of the dimension of the last reduced system to that of the original system A. The reduction ratio determines how good an algorithm finds the independent set. The total CPU time, efficiency ratio and sparsity ratio may be suitable to characterize other preconditioning techniques, but the reduction ratio is mainly for the BILUM-type preconditioners. These four characteristic measures are more informative than the measure provided by the iteration count alone.

Our iterative algorithm consists of GMRES with a small restart value as an accelerator and BILUM as a preconditioner. The last reduced system is solved iteratively by another GMRES preconditioned by an ILUT preconditioner [17]. The dropping strategy that we used in [21] is the simplest one. Elements in EB^{-1} in the L factor and in the reduced system A_1 are dropped whenever their absolute values are less than a threshold tolerance *droptol* times the average value of the current row. For BILUM with large size BIS formed by the greedy algorithm, this simple single dropping strategy is not sufficient to keep BILUM sparse enough. Figure 2 shows the behavior of the four characteristic measures as the block size changes when an algorithm using this single dropping strategy is used to solve a system with the 5-POINT matrix described in [21] (some information about this matrix is given in Section 4). Here, a 20-level BILUM with single dropping strategy was used. The coarsest level solve was preconditioned by ILUT$(10^{-4}, 10)$. The iteration counts are 5 for block sizes ≤ 4, and 4 otherwise. It can be seen that although

Table 1: Description of test matrices.

matrix	size	nonzeros	description
5-POINT	40 000	199 200	5-point upwind scheme of convection-diffusion
RAEFSKY3	21 200	1 488 768	Fluid structure interaction turbulence problem
VENKAT50	62 424	1 717 792	Unstructured 2D Euler solver, time step = 50
WIGTO966	3 864	238 252	Euler equation model

BILUM with large block sizes reduced all three other measures (not monotonically), it increased the storage cost substantially. The sparsity ratio was doubled when the block size increased from 1 to 15. Such an uncontrolled large storage requirement may cause serious problems in large scale applications.

Inspired by the dual threshold dropping strategy of ILUT [17], we propose a similar dual threshold dropping for BILUM. We first apply the single dropping strategy as above to the EB^{-1} and A_1 matrices and keep only the largest $lfil$ elements (absolute value) in each row.

Another cause of loss of sparsity comes from the matrix B^{-1}. In general, each block of B is sparse, but the inverse of the block is dense. For BIS with large blocks this results in a matrix B^{-1} that is much denser than B. However, if a block is diagonally dominant, the elements of the block inverse are expected to decay away from the diagonal rapidly. Hence, small elements of B^{-1} can be dropped without sacrificing the quality of the preconditioner too much. In practice, we may use a double dropping strategy similar to the one just suggested, for the EB^{-1} and A_1 matrices, possibly with different parameters.

4 Numerical Experiments

The Schur-ILU factorization has been implemented in the framework of the PSPARSLIB package [18, 20, 15] and was tested on a variety of machines. The experiments reported here have been performed on a CRAY T3E-900. ILUM and Block ILUM have been implemented and tested on sequential machines. Additional experiments with BILUM, specifically with small blocks, have been reported elsewhere, see [21]. We begin with experiments illustrating the Schur-ILU preconditioners.

Some information on the test matrices is given in Table 1. The Raefsky matrix was supplied to us by H. Simon from Lawrence Berkeley National Laboratory. The Venkat matrix was supplied by V. Venkatakrishnan from NASA and the Wigton matrix by L. Wigton from Boeing Commercial Airplane Group. Despite being small, the Wigton matrix is fairly difficult to solve by iterative methods. In all the tests, the right-hand sides were generated artificially by assuming that the solution is a vector of all ones.

4.1 Tests with approximate Schur-ILU preconditioning

Two test problems RAEFSKY3 and VENKAT50, described in Table 1, as well as a 5-point PDE problem are considered for the numerical experiments in this subsection. In these test problems, the matrix rows followed by the columns were scaled by 2-norm. The initial guess was set to zero. A flexible variant of restarted GMRES (FGMRES) [17] with a subspace dimension of 20 has been used to solve these problems to reduce the residual norm by 10^6. In the preconditioning phase of the solution, the related Schur complement systems have been solved by ILUT-preconditioned

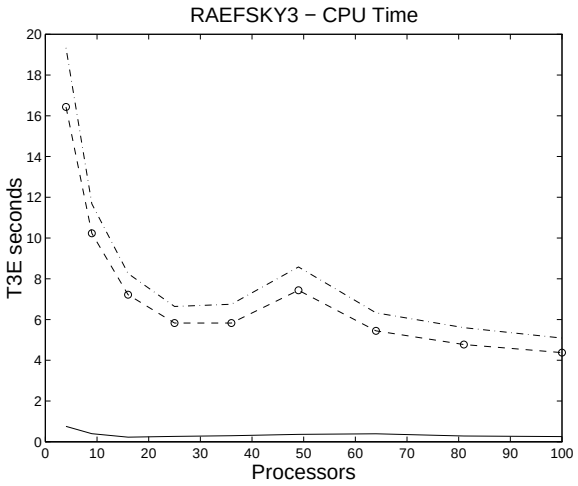

Figure 3: Time results on different processor numbers: total solution time (dash-dotted line), preconditioning operation time (dash-circled line), and FGMRES time (solid line).

Table 2: Schur-ILU preconditioning parameter choices for RAEFSKY3 and VENKAT50.

Problem	*Label*	$lfil$	*tol*	*itmax*
RAEFSKY3	Rprec1	40	10^{-3}	5
	Rprec2	90	10^{-4}	30
VENKAT50	Vprec1	20	10^{-3}	5
	Vprec2	50	10^{-4}	30

GMRES, and thus preconditioning operations differed from one FGMRES iteration to another. Furthermore, varying the preconditioning parameters, such as the number of fill-in elements for ILUT, the maximum number of iterations, and tolerance for GMRES, affects the overall cost of the solution.

In general, the more accurate solves with the Schur complement system, the faster (in terms of iteration numbers) the convergence of the original system. However, even for rather large numbers of processors, preconditioning operations account for the largest amount of time spent in the iterative solution. Consider, for example, the comparison of the total solution time for RAEFSKY3 versus the preconditioning time (Figure 3). For the two test problems, Figure 4 displays the time and iteration number results with various choices (see Table 2) for the preconditioning parameters. In Table 2, each set of choices is assigned a name stated in column *Label* and the GMRES parameters tolerance and maximum number of iterations are shown in columns *tol* and *itmax*, respectively. For the Schur-ILU preconditioning, the parameter $lfil$ specifies the amount of fill-ins in the whole local matrix A_i in the processor i (see equation (3)). Thus, with an increase in $lfil$, as it can be inferred from equation (13), the accuracy of the approximations to the parts of L_i and U_i increases. However, an increase in the accuracy of the preconditioning operation does not necessarily lead to a smaller CPU time cost (cf., for example, the numerical results for VENKAT50 in Figure 4).

Notice the upward jump of the execution time which occurs for the RAEFSKY3 example for exactly 36 processors. This behavior is common for parallel iterative solvers for irregularly structured problems. For a reason that would be difficult to determine, the local linear systems

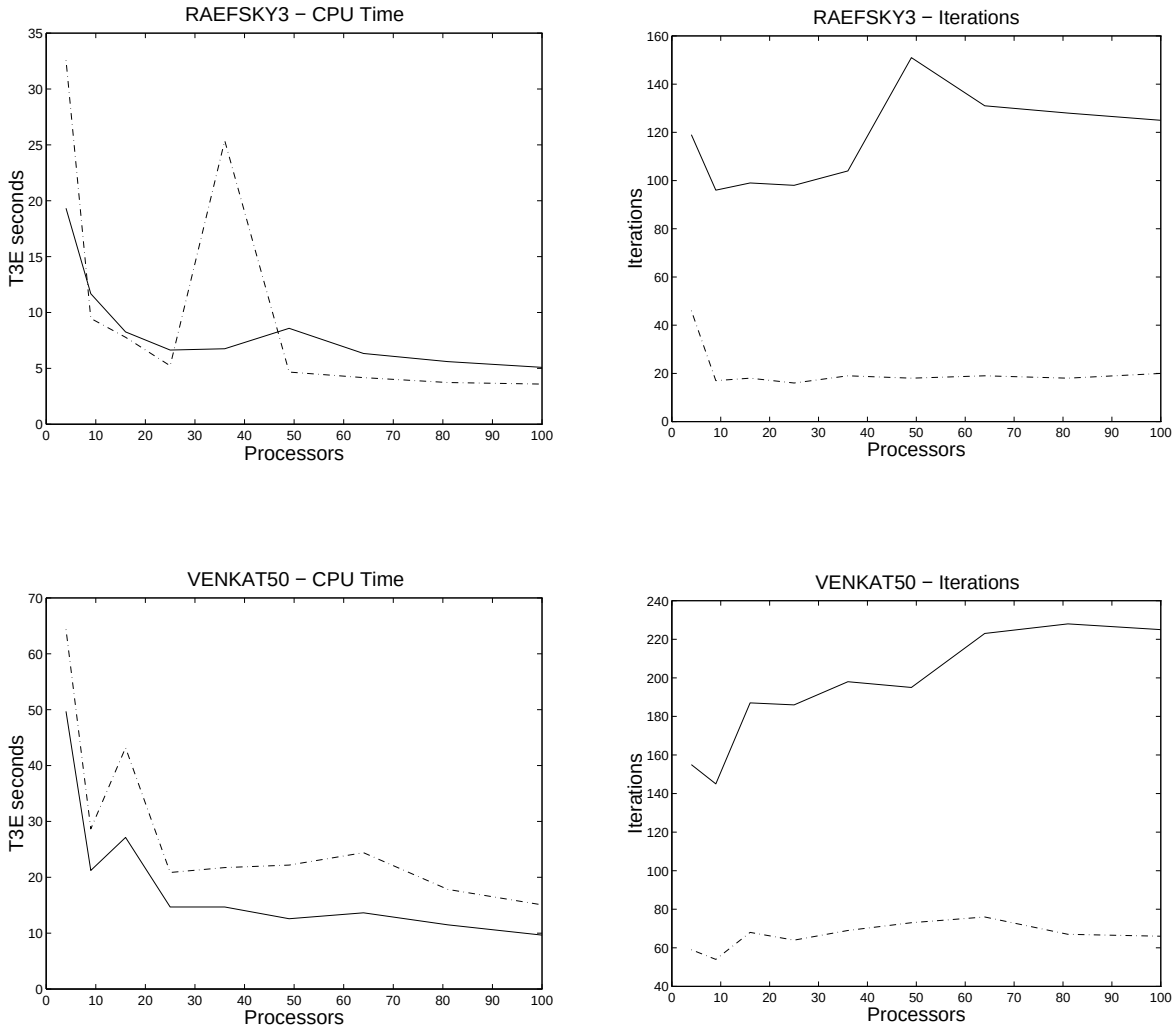

Figure 4: Solution time and iterations when different accuracy is used in Schur-ILU preconditioning: For RAEFSKY3, Rprec1 (solid line) and Rprec2 (dash-dotted line); for VENKAT50, Vprec1 (solid line) and Vprec2 (dash-dotted line).

suddenly become hard to solve for this particular number of processors, causing these solves to take the maximum number of steps (30) in each or most of the calls. The difficulty disappears as soon as we restrict the maximum number of steps in each preconditioning operation to a smaller number (e.g., 5). Unfortunately, these difficulties are hard to predict in advance, at the time when the partitioning is performed. The problem does not occur for 37 or 35 processors.

Finally, we should point out that the previous two test matrices are relatively small for a machine of the size of the T3E, and so the fact that the execution times do not decrease substantially beyond 20 or 30 processors should not be too surprising.

Next we consider a linear system which arises from a 2-dimensional regular mesh problem. Specifically, consider the elliptic equation:

$$-\Delta u + 100\exp(x*y)\frac{\partial u}{\partial x} + 100\exp(-x*y)\frac{\partial u}{\partial y} - 100u = f$$

on the square $(0,1)^2$ with Dirichlet boundary conditions. When discretized using centered difference so that there are 720 interior mesh points in each direction, the resulting linear system is of size $n = 720^2 = 518,400$. The shift term $-100u$ makes the problem indefinite.

It is interesting to observe various measures of parallel performance for this problem. Strictly speaking, each run is different since the preconditioners are different, resulting from different partitionings. In particular, the number of iterations required to converge increases substantially from 4 processors to 100 processors, as shown in Figure 5 (top-left plot). This contrasts with the earlier example and other tests seen in [19]. Recall that the problem is indefinite. The increase in the number of iterations adds to the deterioration of the achievable speed-up for larger numbers of processors. It is informative to have a sense of how much of the loss of efficiency is due to convergence deterioration versus other factors, such as communication time, load imbalance, etc. For example, if we were to factor out the loss due to increased iteration numbers, then for 60 processors over 4 processors, the speed-up would be about 13, compared with the perfect speed-up of 15. In this case, the efficiency would be about 80%. However, the increase in the number of the iterations required for convergence reduces the speed-up to about 9 and the efficiency decreases to about 55%. The Speed-up and Efficiency plots in Figure 5 show the 'adjusted' measures which factor out iteration increases.

4.2 Experiments with BILUM

Standard implementations of ILUM and BILUM have been described in detail in [16, 21]. We used GMRES(10) as an accelerator for both the inner and outer iterations. The outer iteration process was preconditioned by BILUM with dual dropping strategy as was discussed earlier. The inner iteration process to solve the last reduced system approximately was preconditioned by ILUT [17]. Exceptions are stated explicitly. The construction and application of the BILUM preconditioner was similar to those described in [21], except that here the dual dropping strategy was applied from the first level. The initial guess was random numbers.

The numerical experiments were conducted on a Power-Challenge XL Silicon Graphics workstation equipped with 512 MB of main memory, two 190 MHZ R10000 processors, and 1 MB secondary cache. We used FORTRAN 77 in 64-bit precision.

The inner iteration was stopped when the (inner iteration) residual in the 2-norm was reduced by a factor of 10^2 or the number of iterations exceeded 10, whichever occurred first. The outer iteration was stopped when the residual in the 2-norm was reduced by a factor of 10^7. We also set an upper bound of 200 for the outer GMRES(10) iteration. (A symbol "–" in a table indicates that convergence did not reached in 200 outer iterations.) We used $droptol = 10^{-3}$

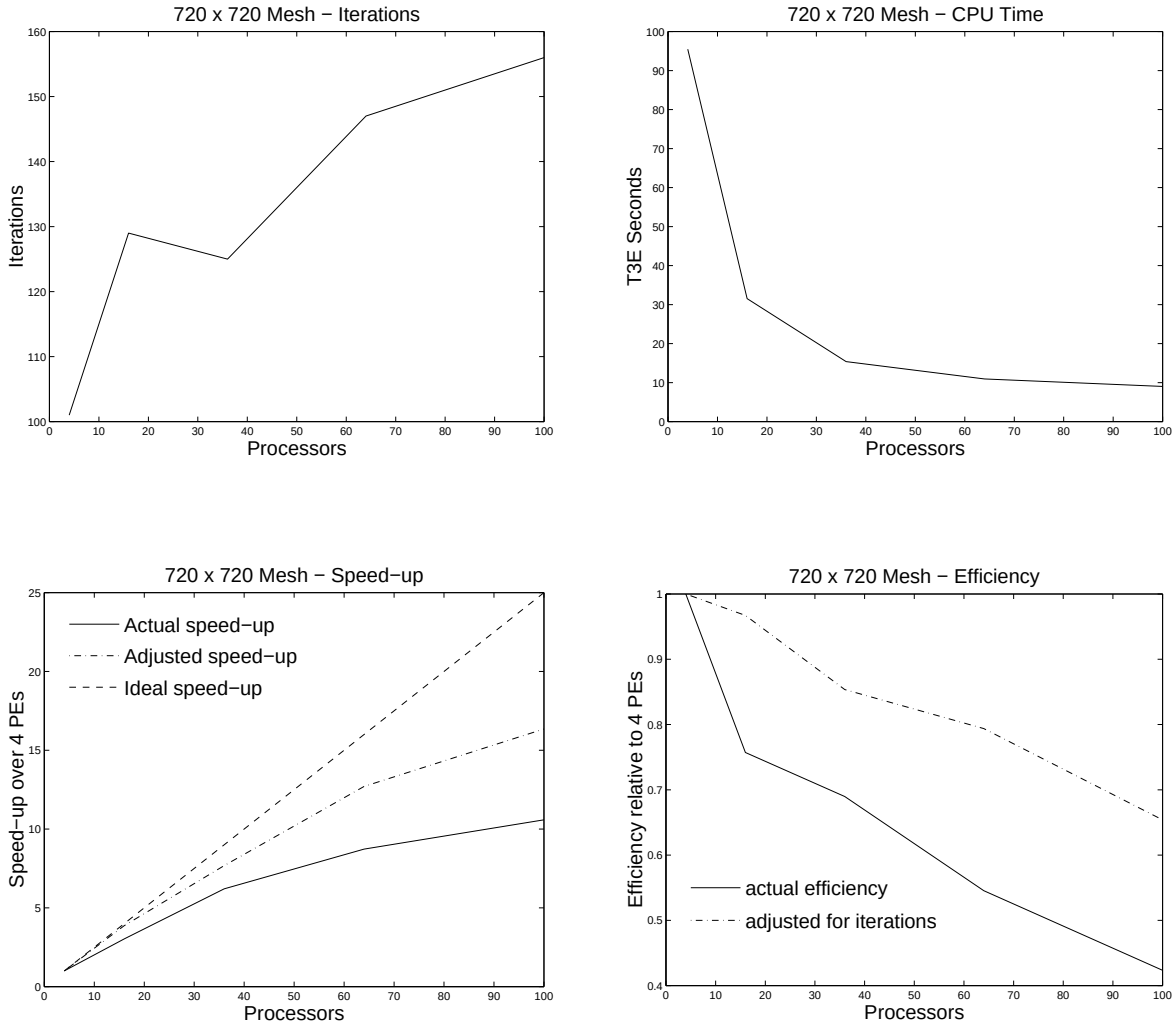

Figure 5: Performance measures for solving a linear system arising from a 5-point matrix on a 720×720 mesh. The adjusted speed-ups and efficiencies are defined as speed-ups and efficiencies divided by the gain (loss) ratios in the iteration count.

Table 3: Characteristic parameters for solving the RAEFSKY3 matrix for different sizes of uniform blocks. BILUM with 20 levels and double dropping strategy was used.

	$lfil = 40, droptol = 10^{-3}$					$lfil = 50, droptol = 10^{-3}$				
k	iter.	tot-CPU	e-rato	s-rato	r-rato	iter.	tot-CPU	e-rato	s-rato	r-rato
5	–	–	–	1.15	0.247	–	–	–	1.33	0.258
10	–	–	–	1.44	0.213	22	90.35	2.47	1.62	0.216
15	84	140.73	0.45	1.55	0.179	17	72.71	2.70	1.72	0.178
20	150	222.60	0.28	1.83	0.156	19	81.89	2.75	2.02	0.158
25	106	160.58	0.39	1.90	0.140	16	71.19	2.96	2.07	0.129
30	–	–	–	1.87	0.122	51	103.71	0.85	2.07	0.114
35	30	74.66	1.27	1.99	0.127	14	68.48	3.31	2.24	0.120
40	27	64.85	1.25	1.92	0.100	13	60.50	3.17	2.19	0.106
45	–	–	–	1.99	0.100	160	222.78	0.25	2.24	0.106
50	–	–	–	2.07	0.106	18	66.39	2.27	2.33	0.094

and tested two values for $lfil$. The block size k varied from 5 to 50. The first test was with the matrix RAEFSKY3. The test results are presented in Table 3.

The results suggest that in order to increase the size of the independent set, it is preferable to have large enough blocks. However, block sizes should not be too large. In the present tests it seems that a good upper limit is the average number of nonzero elements kept in each row (the $lfil$ parameter) during the BILUM factorization. Figure 6 shows the convergence history of BILUM with different block sizes for solving the RAEFSKY3 matrix. With these block sizes, initial convergence was fast in all cases. However, after a few iterations, only BILUM using large block sizes was able to continue converging at a good rate.

The second test is with the matrix VENKAT50. Each row has 28 nonzeros on average. We again used $droptol = 10^{-3}$ and tested two values for $lfil$. The test results are shown in Table 4. The test results for both RAEFSKY3 and VENKAT50 indicate that $lfil$ should be chosen large enough to allow a sufficient amount of fill-ins and the block size k should also be large enough to insure a good reduction rate. We point out that both tests yielded slow convergence for $k = 5$.

Each row of the matrix WIGTO966 has 62 nonzeros on average. This matrix is very hard to solve by ILUT [6] which only worked with a large value of $lfil$, i.e., a large number of elements per row. Test results using ILUT with different $lfil$ and $droptol$ are given in Table 5. We note the large values for the efficiency ratio and the sparsity ratio. Table 6 shows the test results for the matrix WIGTO966 solved by BILUM with 10 levels of reduction (an exception was indicated explicitly). Different block size k, $lfil$, and $droptol$ were tested. We find that for this example, BILUM was 6 times faster than ILUT and used only one-third of the storage space required by ILUT. Once again, we see that the block size k should not be larger than the number of nonzeros kept in each row.

The option of making the diagonal blocks sparse at each level is not tested in this paper. Its potential success obviously depends on the diagonal dominance of the submatrices. There are other techniques that may be used to enhance stability of the inverse of the blocks so that preconditioning effects may be improved. A typical example is the employment of singular value decomposition. Special blocks such as arrow-head matrices may also be constructed that has no additional fill-in in the inverse [8]. Furthermore, it is not necessary that all blocks should be of the same size. For unstructured matrices, blocks with variable sizes may be able to capture

Table 4: Characteristic parameters for solving the VENKAT50 matrix for different sizes of uniform blocks. BILUM with 10 levels and double dropping strategy was used.

	$lfil = 30, droptol = 10^{-3}$					$lfil = 40, droptol = 10^{-3}$				
k	iter.	tot-CPU	e-rato	s-rato	r-rato	iter.	tot-CPU	e-rato	s-rato	r-rato
5	–	–	–	2.53	0.218	–	–	–	2.30	0.260
10	180	458.11	0.27	3.10	0.146	136	426.62	0.40	3.55	0.170
15	166	412.81	0.26	3.43	0.109	126	377.43	0.41	3.94	0.130
20	157	373.77	0.26	3.42	0.087	124	368.74	0.39	4.27	0.110
25	166	382.25	0.23	3.52	0.074	115	324.93	0.41	4.40	0.095
30	192	397.33	0.19	2.91	0.060	121	334.93	0.37	4.10	0.079
35	180	374.07	0.19	2.97	0.054	109	306.89	0.40	4.16	0.073
40	179	366.01	0.19	2.95	0.048	110	302.77	0.37	4.21	0.066
45	161	333.82	0.20	2.99	0.042	115	308.04	0.35	4.23	0.056
50	174	353.54	0.18	3.00	0.040	105	287.42	0.37	4.31	0.052

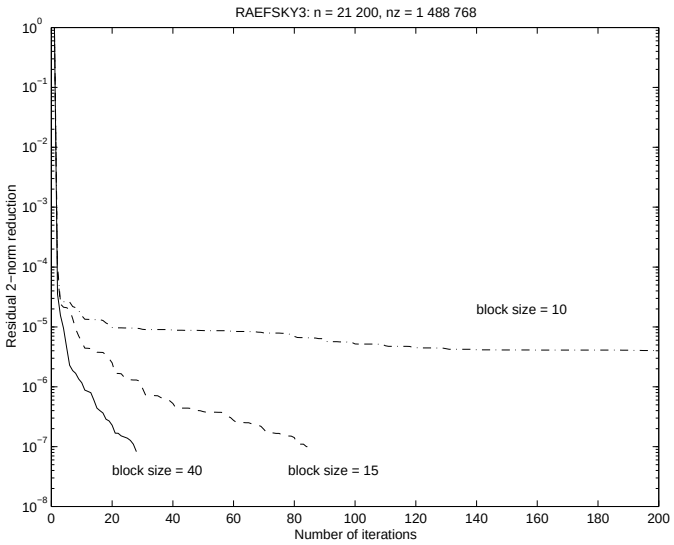

Figure 6: Convergence history of BILUM with different block sizes to solve the RAEFSKY3 matrix. BILUM with 20 levels and double dropping strategy was used and the coarsest level solve was preconditioned by ILUT($10^{-3}, 40$).

Table 5: Test results of WIGTO966 matrix solved with ILUT.

$lfil$	$droptol$	iter.	tot-CPU	e-rato	s-ratio
340	10^{-5}	110	81.54	2.48	9.14
350	10^{-5}	41	86.16	6.78	9.33
400	10^{-5}	22	87.07	12.60	10.06
340	10^{-4}	–	–	–	–
330	10^{-5}	–	–	–	–
330	10^{-6}	–	–	–	–

Table 6: Test results of WIGTO966 matrix with 10 level BILUM.

k	$lfil$	$droptol$	iter.	tot-CPU	e-rato	s-rato	r-rato
100	50	10^{-4}	100	18.72	0.47	3.21	0.017
100	60	10^{-4}	–	–	–	–	–
100	100	10^{-4}	22	15.90	3.11	5.34	0.042
80	80	10^{-4}	–	–	–	–	–
62	50	10^{-4}	–	–	–	–	–
62	50	10^{-5}	–	–	–	–	–
62	62	10^{-4}	40	15.10	1.60	3.83	0.101
62	65	10^{-4}	42	16.50	1.59	3.91	0.101
62	70	10^{-4}	–	–	–	–	–
50	50	10^{-4}	44	12.64	1.30	3.12	0.094
50	50	10^{-3}	48	13.02	1.18	3.08	0.094
50	60	10^{-3}	35	13.48	1.85	3.50	0.081
50	60	10^{-4}	32	13.10	2.02	3.52	0.081
40	50	10^{-4}	52	14.80	1.21	3.10	0.120
40	60	10^{-4}	28	14.07	2.58	3.57	0.099
$30^{\ddagger}$	40	10^{-4}	69	14.71	0.91	2.65	0.006

‡: 20 levels of reduction were used.

more physical information than those with uniform size. The major difficulty of applying these and other advanced techniques is the complexity of programming. We will examine these and other ideas experimentally and the results will be reported elsewhere.

5 Concluding Remarks

We discussed two techniques based on Schur complement ideas for deriving preconditioners for general sparse linear systems. The Schur-ILU preconditioning is an efficient yet simple to implement preconditioner aimed at distributed sparse linear systems. The simplicity of this preconditioner comes from the fact that only local data structures are used. The multi-level domain-type algorithm (BILUM) for solving general sparse linear systems is based on a recursive application of Schur complement techniques using small subdomains. We proposed a dual dropping strategy to improve sparsity in the BILUM factors. Several parameters were introduced to characterize the efficiency of a preconditioner. Numerical results showed that the proposed strategy works well for reducing the storage cost of BILUM with large block sizes. This class of methods offers a good alternative to the standard ILU preconditioners with threshold, in view of their robustness and efficiency. However, their implementation on parallel platforms may prove to be more challenging than the single-level Schur complement preconditioners such as the approximate Schur-LU technique.

References

[1] R. E. Bank and C. Wagner. Multilevel ILU decomposition. Technical report, Department of Mathematics, University of California at San Diego, San-Diego, CA, 1997.

[2] T. Barth, T. F. Chan, and W.-P. Tang. A parallel algebraic non-overlapping domain decomposition method for flow problems. Technical report, NASA Ames Research Center, Moffett Field, CA, 1998. In preparation.

[3] E. F. F. Botta, K. Dekker, Y. Notay, A. van der Ploeg, C. Vuik, F. W. Wubs, , and P. M. de Zeeuw. How fast the Laplace equation was solved in 1995. *Applied Numerical Mathematics*, 24:439–455, 1997.

[4] E. F. F. Botta and W. Wubs. MRILU: it's the preconditioning that counts. Technical Report W-9703, Department of Mathematics, University of Groningen, The Netherlands, 1997.

[5] T. F. Chan, S. Go, , and J. Zou. Multilevel domain decomposition and multigrid methods for unstructured meshes: algorithms and theory. Technical Report 95-24, Department of Mathematics, University of California at Los Angeles, Los Angeles, CA, 1996.

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

[7] V. Eijkhout and T. Chan. ParPre a parallel preconditioners package, reference manual for version 2.0.17. Technical Report CAM Report 97-24, UCLA, 1997.

[8] G. Golub and J. M. Ortega. *Scientific Computing: An Introduction with Parallel Computing*. Academic Press, Boston, 1993.

[9] W. D. Gropp and B. Smith. User's manual for KSP: data-structure neutral codes implementing Krylov space methods. Technical Report ANL-93/23, Argonne National Laboratory, Argonne, IL, March 1993.

[10] Scott A. Hutchinson, John N. Shadid, and R. S. Tuminaro. Aztec user's guide. version 1.0. Technical Report SAND95-1559, Sandia National Laboratories, Albuquerque, NM, 1995.

[11] M. T. Jones and P. E. Plassmann. BlockSolve95 users manual: Scalable library software for the solution of sparse linear systems. Technical Report ANL-95/48, Argonne National Lab., Argonne, IL., 1995.

[12] J. Mandel. Balancing domain decomposition. *Communications in Applied Numerical Methods*, 9:233–241, 1993.

[13] A. A. Reusken. Approximate cyclic reduction preconditioning. Technical Report RANA 97-02, Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands, 1997.

[14] J W. Ruge and K Stüben. Efficient solution of finite difference and finite element equations. In In D. J. Paddon and H. Holstein, editors, *Multigrid Methods for Integral and Differential Equations*, pages 169–212, Oxford, 1985. Clarendon Press.

[15] Y. Saad. Parallel sparse matrix library (P_SPARSLIB): The iterative solvers module. In *Advances in Numerical Methods for Large Sparse Sets of Linear Equations, Number 10, Matrix Analysis and Parallel Computing, PCG 94*, pages 263–276, Keio University, Yokohama, Japan, 1994.

[16] Y. Saad. ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing*, 17(4):830–847, 1996.

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

[18] 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, Russia, Sept. 1995*, 1995.

[19] Y. Saad and M. Sosonkina. Distributed Schur complement techniques for general sparse linear systems. Technical Report UMSI 97/159, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1997.

[20] Y. Saad and K. Wu. Design of an iterative solution module for a parallel sparse matrix library (P_SPARSLIB). In W. Schonauer, editor, *Proceedings of IMACS conference, Georgia, 1994*, 1995.

[21] Y. Saad and J. Zhang. BILUM: Block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems. Technical Report UMSI 97/126, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1997.

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

[23] B. Smith, W. D. Gropp, and L. C. McInnes. PETSc 2.0 user's manual. Technical Report ANL-95/11, Argonne National Laboratory, Argonne, IL, July 1995.