SIAM J. MATRIX ANAL. APPL.
Vol. 21, No. 1, pp. 279–299

BILUTM: A DOMAIN-BASED MULTILEVEL BLOCK ILUT PRECONDITIONER FOR GENERAL SPARSE MATRICES*

YOUSEF SAAD† AND JUN ZHANG‡

Abstract. This paper describes a domain-based multilevel block ILU preconditioner (BILUTM) for solving general sparse linear systems. This preconditioner combines a high accuracy incomplete LU factorization with an algebraic multilevel recursive reduction. Thus, in the first level the matrix is permuted into a block form using (block) independent set ordering and an ILUT factorization for the reordered matrix is performed. The reduced system is the approximate Schur complement associated with the partitioning, and it is obtained implicitly as a by-product of the partial ILUT factorization with respect to the complement of the independent set. The incomplete factorization process is repeated with the reduced systems recursively. The last reduced system is factored approximately using ILUT again. The successive reduced systems are not stored. This implementation is efficient in controlling the fill-in elements during the multilevel block ILU factorization, especially when large size blocks are used in domain decomposition-type implementations. Numerical experiments are used to show the robustness and efficiency of the proposed technique for solving some difficult problems.

Key words. incomplete LU factorization, ILUT, multilevel ILU preconditioner, Krylov subspace methods, multielimination ILU factorization

AMS subject classifications. 65F10, 65N06

PII. S0895479898341268

1. Introduction.

The preconditioning technique proposed in this paper is based on a multilevel block incomplete LU factorization. It is intended for solving general sparse linear systems of the form

$$Ax = b, \tag{1.1}$$

where A is an unstructured matrix of order n. Such linear systems are often solved by Krylov subspace methods coupled with a suitable preconditioner [52]. The research and design of preconditioners with inherent parallelism have received much attention recently, spurred by the popularity of distributed memory architectures. The main considerations when comparing preconditioners are their intrinsic efficiency, generality, parallelism, and robustness. An experimental study on robustness of a few general purpose preconditioners has been conducted in [22] and a number of ILU-type preconditioners have been tested for solving some difficult problems from computational fluid dynamics in [19, 20].

The incomplete LU factorization without fill-in (ILU(0)) is probably the best known general purpose preconditioner [40]. However, this preconditioner is not robust and is inefficient and fails for many real-life problems. Many extensions of ILU(0), which increase its accuracy and robustness, have been designed, and we refer to [52]

*Received by the editors June 29, 1998; accepted for publication (in revised form) by A. Greenbaum November 23, 1998; published electronically September 22, 1999. This work was supported in part by NSF grant CCR-9618827 and in part by the Minnesota Supercomputer Institute.
http://www.siam.org/journals/simax/21-1/34126.html

†Department of Computer Science and Engineering, University of Minnesota, 4-192 EE/CS Building, 200 Union Street SE, Minneapolis, MN 55455 (saad@cs.umn.edu, http://www.cs.umn.edu/~saad).

‡Department of Computer Science and Engineering, University of Minnesota, 4-192 EE/CS Building, 200 Union Street SE, Minneapolis, MN 55455. Current address: Department of Computer Science, University of Kentucky, 773 Anderson Hall, Lexington, KY 40506-0046 (jzhang@cs.uky.edu, http://www.cs.uky.edu/~jzhang).

for a partial account of the literature and to [25, 26, 33, 44, 60, 64] for just a few ideas described.

The moderate parallelism which can be extracted from the triangular solves in standard ILU factorization [1, 52] is limited, and becomes inadequate for the more accurate ILU factorizations. Alternatives have been considered in the past to develop preconditioners with inherently more parallelism than standard ILU; see, e.g., [48, 52, 58] for references. A standard technique used for this purpose is to exploit "multicolor orderings" or "independent sets" [49]. A well-known drawback of using a multicolor ordering prior to building an ILU factorization is that the quality of the preconditioning on the reordered system is generally worse than that on the original system [29, 30, 32]. However, numerical results in [50] show that some high accuracy ILU-type preconditioners with red-black ordering may eventually outperform their counterparts with natural ordering, if enough fill-in is allowed. Conversely, the higher amount of fill-in usually reduces the parallelism that is available in ILU(0). Therefore, a desirable goal is to achieve high accuracy while retaining parallelism achieved by using multicoloring or independent sets. Other alternatives for developing parallel preconditioners have been proposed that are based on sparse approximate inverse techniques; see, e.g., [9, 16, 21, 23, 35, 37, 62]. These preconditioners afford potential maximum parallelism both in their construction stage (except [9, 62]) and in the application stage which requires only matrix-vector operations. However, these methods tend to become inefficient for handling very large matrices because of their local nature.

The multielimination ILU preconditioner (ILUM), introduced in [51], is based on exploiting the idea of successive independent set orderings. It has a multilevel structure and offers a good degree of parallelism without sacrificing overall effectiveness. Similar preconditioners developed in [12, 55] show near grid-independent convergence for certain types of problems. In a recent report, some of these multilevel preconditioners were tested and compared favorably with other preconditioned iterative methods and direct methods, at least for the Laplace equation [10].

The idea of combining multilevel techniques with ILU is not new. Alternative multilevel approaches that require grid information have been developed. Examples of such approaches include the nested recursive two-level factorization, repeated red-black orderings, and generalized cyclic reduction [2, 4, 11, 31, 43] (see also the survey paper by Axelsson and Vassilevski [3]). Some recently developed methods require only the adjacency graph of the coefficient matrices [12, 46, 51, 55]. For the repeated red-black ordering approach, a near-optimal bound for the condition number of the preconditioned matrix was reported [42]. Other methods which bear some similarity with ILUM-type techniques are the algebraic multigrid methods [13, 18, 46, 47, 59] and certain types of multigrid methods which consider matrix entries [28, 27, 45]. Equally interesting are the multilevel preconditioning techniques based on hierarchical basis, multigraph, or ILU decomposition associated with the finite difference or finite element analysis [6, 8, 7, 15, 36].

A block version of ILUM (BILUM) was recently defined by using small dense matrices as pivots instead of scalars [55, 57]. For some hard-to-solve problems, BILUM may perform much better than ILUM. Tests with large blocks indicate that the larger the block, the more robust the resulting preconditioner. The solution with the independent blocks in BILUM uses the exact inverse or a regularized inverse based on the singular value decomposition (SVD) [55, 57]. These strategies are efficient for blocks of small size, but the cost of such inversion strategies grows cubically as the size of the blocks increases.

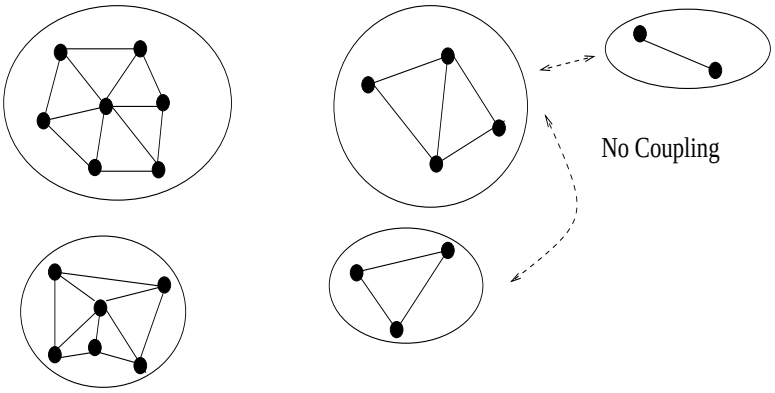

FIG. 2.1. *Independent groups or blocks.*

In this paper we focus on *domain decomposition-based* block ILUM. In this case the blocks can be very large and are associated with a subdomain, as in domain decomposition methods. For these large size blocks, the computational and memory costs of computing exact or regularized inverses used for BILUM become prohibitive. The dropping strategies that have been proposed [57] are not able to deal with both problems (computation and memory costs) simultaneously. It is therefore vital to exploit sparsity for domain decomposition-based block ILUM. This paper introduces an efficient approach to constructing BILUM based on this principle. The construction of such a preconditioner is based on a restricted ILU factorization with a dual dropping strategy (ILUT); see [50]. This multilevel block ILUT preconditioner (BILUTM) retains the efficiency and flexibility of ILUT and offers inherent parallelism that can be exploited on parallel or distributed architectures.

This paper is organized as following. Section 2 gives an overview and background on multigrid and multilevel preconditioning techniques. Section 3 provides some details on the construction of the reduced system by partial Gaussian elimination. Section 4 discusses the proposed BILUTM. Section 5 describes some numerical experiments, and section 6 gives a few concluding remarks.

2. Multilevel preconditioning techniques. Multilevel preconditioners take advantage of the fact that different parts of the error spectrum can be treated independently on different levels. In construction, multilevel preconditioners also exploit, explicitly or implicitly, the property that a set of unknowns that are not coupled to each other can be eliminated simultaneously in a Gaussian elimination–type process. Such a set is usually called an *independent set* [34]. This concept of independent set can easily be generalized to blocks. Thus a block independent set is a set of groups (blocks) of unknowns such that there is no coupling between unknowns of any two different groups (blocks) [55]. Unknowns within the same group (block) may be coupled. This is illustrated in Figure 2.1.

Thus, point (scalar) independent sets are a particular case which use blocks of uniform size of 1. Various heuristic strategies may be used to find a block independent set with different properties [51, 55]. A simple and usually efficient strategy is a greedy algorithm, which groups the nearest nodes together. Since the focus of this paper is not on finding block independent sets, we assume that this greedy algorithm is used throughout to find block independent sets.

A maximal independent set is an independent set that cannot be augmented by other nodes and still remain independent. Independent sets are often constructed with some other conditions such as guaranteeing certain diagonal dominance for the nodes of the independent set or the vertex cover, which is defined as the complement of the independent set. Thus, in practice, the maximality of an independent set is rarely guaranteed, especially when some dropping strategies are applied [57].

Algebraic and "black box" multigrid methods try to mimic geometric multigrid methods by defining a prolongation operator $\mathbf{I}_{\alpha+1}^{\alpha}$ based on some heuristic arguments; here $0 \leq \alpha < \mathcal{L}$ is an integer used to label the level. For convenience and for satisfying certain conservation laws, the restriction operator, $\mathbf{I}_{\alpha}^{\alpha+1}$, is traditionally defined as the adjoint of the prolongation operator (possibly scaled by a constant), i.e., $\mathbf{I}_{\alpha}^{\alpha+1} = {\mathbf{I}_{\alpha+1}^{\alpha}}^T$ [47]. With $A_0 = A$, the recursive coarse grid operators are then generated by using the Galerkin technique as

$$A_{\alpha+1} = \mathbf{I}_{\alpha}^{\alpha+1} A_{\alpha} \mathbf{I}_{\alpha+1}^{\alpha}. \tag{2.1}$$

Note that, in order for the grid transfer operators to be defined efficiently, a logically rectangular grid is explicitly or implicitly assumed for the black box or matrix-dependent approaches [28]. Most such multigrid methods are designed for two-dimensional problems, and their extensions to higher dimensions are not straightforward [5]. For algebraic multigrid methods, improvements have recently been reported by defining more accurate grid transfer operators [17, 18].

In independent set orderings, the unknowns may be permuted such that those associated with the independent set are listed first, followed by the other unknowns. The permutation matrix P_{α}, associated with such an ordering, transforms the original matrix into a matrix which has the following block structure:

$$A_{\alpha} \sim P_{\alpha} A_{\alpha} P_{\alpha}^T = \begin{pmatrix} D_{\alpha} & F_{\alpha} \\ E_{\alpha} & C_{\alpha} \end{pmatrix}, \tag{2.2}$$

where D_{α} is a block diagonal matrix of dimension m_{α}, and C_{α} is a square matrix of dimension $n_{\alpha} - m_{\alpha}$. In what follows, the notation is slightly abused by not distinguishing the original system from the permuted system, so both permuted and unpermuted matrices will be denoted by A_{α}.

To improve load balancing on parallel computers, it is desirable to have uniformly sized independent blocks. However, this is not a necessary requirement for the techniques described in this paper.

In algebraic multilevel preconditioning techniques, the reduced systems are recursively constructed as the Schur complement with respect to either D_{α} or C_{α}. In the case of BILUM [51, 55], such a construction amounts to performing a block LU factorization of the form

$$\begin{pmatrix} D_{\alpha} & F_{\alpha} \\ E_{\alpha} & C_{\alpha} \end{pmatrix} = \begin{pmatrix} I_{\alpha} & 0 \\ E_{\alpha} D_{\alpha}^{-1} & I_{\alpha} \end{pmatrix} \times \begin{pmatrix} D_{\alpha} & F_{\alpha} \\ 0 & A_{\alpha+1} \end{pmatrix}, \tag{2.3}$$

where $A_{\alpha+1}$ is the Schur complement with respect to C_{α} and I_{α} is the generic identity matrix on level α. Note that $n_{\alpha+1} = m_{\alpha}$. The solution process with the above factorization consists of level-by-level forward elimination, followed by an exact solution on the last reduced system $A_{\mathcal{L}}$. The solution of the original system is obtained by level-by-level backward substitution (with suitable permutation).

The procedure described above is a direct solution method and the reduced systems become denser and denser as the level number increases, a consequence of the

fill-in caused by the elimination process. In BILUM, some dropping strategies are used to control the amount of fill-in by discarding certain elements of small magnitude or by limiting the number of elements allowed in each row of the L and U factors [51, 54, 55]. The resulting incomplete multilevel block LU factorization is then used as a preconditioner in a Krylov subspace method based iterative solver.

In the implementation of BILUM in [55], the block diagonals D_α consist of small size blocks. These small blocks are usually dense and an exact inverse technique is used to compute D_α^{-1} by inverting each small block independently (in parallel). In [57], some regularized-inverse technique based on SVD is used to invert the (potentially near-singular) blocks approximately. As we noted in the introduction, such direct inversion strategies usually produce dense inverse matrices even if the original blocks are highly sparse with large sizes. Thus some heuristic approaches have been proposed to drop small elements from the exactly or approximately inverted blocks to recover sparsity. Obviously this approach cannot reduce the cost of inverting these blocks.

The link between the algebraic multigrid methods and BILUM was discussed briefly in [57]. If we define the grid transfer operators naturally based on the matrix, the reduced system based on the Schur complement technique as in (2.3) also satisfies the Galerkin condition (2.1). In other words, BILUM can be viewed as a naturally defined algebraic multigrid or multilevel technique.

3. Gaussian elimination and ILUT. ILUT is a high-order (high accuracy) preconditioner based on incomplete LU factorization. It uses a dual dropping strategy to control the storage cost (the amount of fill-in) [50]. Its implementation is based on the IKJ variant of Gaussian elimination, which we recall next.

ALGORITHM 3.1. GAUSSIAN ELIMINATION–IKJ VARIANT.

1. *For* $i = 2, n$, *Do*
2. *For* $k = 1, i-1$, *Do*
3. $a_{i,k} := a_{i,k}/a_{k,k}$
4. *For* $j = k+1, n$, *Do*
5. $a_{i,j} := a_{i,j} - a_{i,k} * a_{k,j}$
6. *End Do*
7. *End Do*
8. *End Do*

The ILUT(τ, p) preconditioner attempts to control fill-in elements by applying a dual dropping strategy in Algorithm 3.1. The accuracy of ILUT(τ, p) is controlled by two dropping parameters, τ and p. In Algorithm 3.2, w is a work array, $a_{i,\beta}$ and $u_{k,\beta}$ denote the ith and kth rows of A and U, respectively.

ALGORITHM 3.2. STANDARD ILUT(τ, p) FACTORIZATION [50, 52].

1. *For* $i = 2, n$, *Do*
2. $w := a_{i,\beta}$
3. *For* $k = 1, i-1$ *and when* $w_k \neq 0$, *Do*
4. $w_k := w_k/a_{k,k}$
5. *If* $|w_k| < \tau * \text{nzavg}(a_{i,\beta})$, *set* $w_k := 0$
6. *If* $w_k \neq 0$, *then*
7. $w := w - w_k * u_{k,\beta}$
8. *End If*
9. *End Do*
10. *Apply a dropping strategy to row* w
11. *Set* $l_{i,j} := w_j$ *for* $j = 1, \ldots, i-1$ *whenever* $w_j \neq 0$
12. *Set* $u_{i,j} := w_j$ *for* $j = i, \ldots, n$ *whenever* $w_j \neq 0$

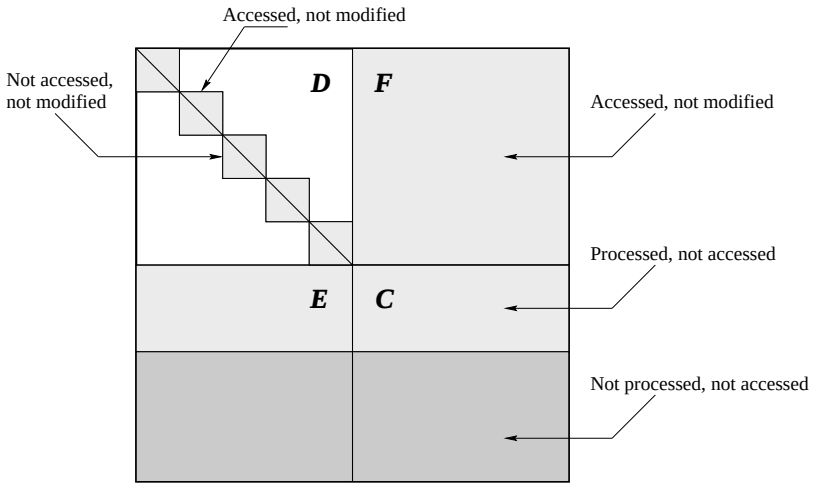

FIG. 3.1. *Illustration of restricted IKJ version of Gaussian elimination.*

13. *Set* $w := 0$
14. *End Do*

In Line 5, the function nzavg($a_{i,\beta}$) returns the average magnitude of the nonzero elements of a given sparse row. Elements with relatively small magnitude are dropped. In Line 10, a different dropping strategy is applied. First, small elements are dropped according to the relative magnitude similar to the criterion used in Line 5. Then a sorting operation is performed and only the largest p elements in absolute value of the L and U factors are kept. After the dual dropping strategy, there are at most p elements kept in each row of the L and U factors.

We now consider a slightly different elimination procedure. Assume that the first m equations are associated with the independent set as in the left-hand side of (3.2). If we perform the LU factorization (Gaussian elimination) to the upper part (the first m rows) of the matrix, i.e., to the submatrix $(D\ F)$, we have

$$(D\ \ F) = (LU\ \ L^{-1}F). \tag{3.1}$$

We then continue the Gaussian elimination to the lower part, but the elimination is only performed with respect to the submatrix E; i.e., we only eliminate those elements $a_{i,k}$ for which $m < i \le n, 1 \le k \le m$. Appropriate linear combinations are also performed with respect to the C submatrix, in connection with the eliminations in the E submatrix, as in the usual Gaussian elimination. Note that when doing these operations on the lower part, the upper part of the matrix is only accessed but not modified; see Figure 3.1. The processed rows of the lower part are never accessed again. This gives the following "restricted" version of Algorithm 3.1.

ALGORITHM 3.3. RESTRICTED IKJ VERSION OF GAUSSIAN ELIMINATION.

1. *For* $i = 2, n$, *Do*
2. *For* $k = 1, \min(i-1, m)$, *Do*
3. $a_{i,k} := a_{i,k}/a_{k,k}$
4. *For* $j = k+1, n$, *Do*
5. $a_{i,j} := a_{i,j} - a_{i,k} * a_{k,j}$
6. *End Do*
7. *End Do*
8. *End Do*

Here m is a parameter which defines the size of the matrix D. Algorithm 3.3 performs a block factorization of the form

$$\begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ EU^{-1} & I \end{pmatrix} \times \begin{pmatrix} U & L^{-1}F \\ 0 & A_1 \end{pmatrix} = \mathbf{LU}. \tag{3.2}$$

In other words, the $a_{i,k}$'s (of the lower part) for $k \leq m$ are the elements in EU^{-1} and the other elements are those in A_1.

PROPOSITION 3.4. *The matrix A_1 computed by Algorithm* 3.3 *is the Schur complement of A with respect to C.*

Proof The part of the matrix after the upper part Gaussian elimination (with respect to the independent set, see (3.1)) that is accessed is $(U\ L^{-1}F)$; the L part is never accessed again. So we may write the active part of the (partially processed) matrix A as

$$\begin{pmatrix} U & \tilde{F} \\ E & C \end{pmatrix} = \begin{pmatrix} U & L^{-1}F \\ E & C \end{pmatrix}.$$

In order to eliminate an element in E, say, $e_{i,j}(= a_{i,j})$ with $m < i \leq n, 1 \leq j \leq m$, we perform a linear combination of the ith row of A and jth row of the U-part $(U\ L^{-1}F)$. Hence, the elements in C are modified according to the operations

$$\tilde{c}_{i,k} = c_{i,k} - \frac{e_{i,j}}{u_{j,j}} \tilde{f}_{j,k}.$$

After eliminating all $e_{i,j}$'s in E, the elements of the C matrix are changed to

$$\tilde{c}_{i,k} = c_{i,k} - \sum_{j=1}^{m} \frac{e_{i,j}}{u_{j,j}} \tilde{f}_{j,k}.$$

It follows that

$$A_1 = \tilde{C} = C - EU^{-1}\tilde{F} = C - EU^{-1}L^{-1}F = C - ED^{-1}F. \qquad \square$$

Note that $D = LU$ is factored. However, even in exact factorization, LU is usually sparser than D^{-1}. The submatrices DU^{-1} and $L^{-1}F$ are formed automatically, and the Schur complement is formed implicitly, during the partial Gaussian elimination with respect to the lower part of A.

Dropping strategies similar to those used in Algorithm 3.2 can be applied to Algorithm 3.3, resulting in an incomplete LU factorization with an approximate Schur complement A_1. We formally describe the restricted ILUT factorization as in Algorithm 3.5.

ALGORITHM 3.5. RESTRICTED ILUT(τ, p) FACTORIZATION.

1. *For* $i = 2, n$, *Do*
2. $w := a_{i,\beta}$
3. *For* $k = 1, \min(i-1, m)$ *and when* $w_k \neq 0$, *Do*
4. $w_k := w_k / a_{k,k}$
5. *Set* $w_k := 0$ *if* $|w_k| < \tau * \text{nzavg}(a_{i,\beta})$
6. *If* $w_k \neq 0$, *then*
7. $w := w - w_k * u_{k,\beta}$
8. *End If*
9. *End Do*

10. *Apply a dropping strategy to row w*
11. *Set* $l_{i,j} := w_j$ *for* $j = 1, \ldots, \min(i-1, m)$ *whenever* $w_j \neq 0$
12. *Set* $u_{i,j} := w_j$ *for* $j = \min(i, m), \ldots, n$ *whenever* $w_j \neq 0$
13. *Set* $w := 0$
14. *End Do*

Algorithm 3.5 yields an ILU factorization of the form

$$A = \mathbf{LU} + R, \tag{3.3}$$

where R is the residual matrix representing the difference between A and $\mathbf{LU}$. The ILUT implementation gives an easy representation of the residual matrix.

PROPOSITION 3.6. *The elements of the residual matrix* R *as in* (3.3) *are those elements dropped in Algorithm* 3.5.

Proof. The proof can be formulated from the arguments in [52, p. 274] and [56]. □

Clearly, Algorithm 3.5 will fail when any individual ILUT fails on at least one of the blocks due to zero pivots. There are at least three strategies that can be used to deal with this situation. First, one can use pivoting as in ILUTP [52], a variant of ILUT which incorporates column pivoting. Second, we may use a diagonal threshold strategy as was done in ILUM [56]. In this technique, nodes with small absolute diagonal values are put in the vertex cover. This strategy may reduce the size of the independent set. Third, we may replace a small (absolute) diagonal value by a larger one and proceed with the normal ILUT. The third strategy is suitable and is almost free of cost since ILUT is not an exact factorization anyway. In our implementation we chose the third strategy and replaced a zero diagonal value by a value that is determined by the dropping tolerance and the absolute average nonzero elements of the current row.

We mention that in Algorithm 3.5 the diagonals of the approximate Schur complement (A_1) are not dropped regardless of their values. From Figure 3.1 the accuracy of the EU^{-1} part is related to that of the LU part; the accuracy of the A_1 part is related to that of the $L^{-1}F$ part. It may be profitable to use different dropping parameters (τ, p) for the upper and lower parts of the ILU factorizations in Algorithm 3.5. We did some numerical experiments and did not find overwhelming evidence for supporting the use of different dropping parameter set for most test problems. However, even if other problems may be tested to show certain advantages, the increased difficulty of determining more parameters for a general purpose preconditioner may offset the gain in convergence. Thus, the numerical results reported in this paper all use uniform dropping parameters during the construction phase. We even kept the parameters the same between different levels.

We point out that the inherent parallelism in the construction phase is excellent. The construction of the upper part factorization is parallelizable with respect to individual blocks. The construction of the lower part factorization is fully parallelizable relative to individual rows, as processing each row needs only information from (access to) the upper part. In addition, parallel algorithms for finding independent sets are available [38, 39].

4. Multilevel block ILUT. The BILUTM is based on the restricted ILUT Algorithm 3.5. On each level α, an incomplete LU factorization is performed and an approximate reduced system $A_{\alpha+1}$ is formed as in Algorithm 3.5. Formally, we have

$$\begin{pmatrix} D_\alpha & F_\alpha \\ E_\alpha & C_\alpha \end{pmatrix} = \begin{pmatrix} L_\alpha & 0 \\ E_\alpha U_\alpha^{-1} & I_\alpha \end{pmatrix} \times \begin{pmatrix} U_\alpha & L_\alpha^{-1} F_\alpha \\ 0 & A_{\alpha+1} \end{pmatrix} = \mathbf{L}_\alpha \mathbf{U}_\alpha. \tag{4.1}$$

The whole process of finding block independent sets, permuting the matrix, and performing the restricted ILUT factorization is recursively repeated on the matrix $A_{\alpha+1}$. The recursion is stopped when the last reduced system $A_{\mathcal{L}}$ is small enough. Then a standard ILUT factorization $\mathbf{L}_{\mathcal{L}}\mathbf{U}_{\mathcal{L}}$ is performed on $A_{\mathcal{L}}$ (Algorithm 3.2). However, we do not store any reduced systems on any level, including the last one. Instead, we store two sparse matrices on each level

$$\mathbf{L}_\alpha = \begin{pmatrix} L_\alpha & 0 \\ E_\alpha U_\alpha^{-1} & I_\alpha \end{pmatrix} \qquad \text{and} \qquad \mathbf{U}_\alpha = \begin{pmatrix} U_\alpha & L_\alpha^{-1}F_\alpha \\ 0 & 0 \end{pmatrix}, \quad \text{for} \quad 0 \le \alpha < \mathcal{L}-1,$$

along with the factors $\mathbf{L}_{\mathcal{L}}$ and $\mathbf{U}_{\mathcal{L}}$.

The approximate solution on the last level is obtained by applying one sweep of ILUT of the last reduced system using the factors $\mathbf{L}_{\mathcal{L}}\mathbf{U}_{\mathcal{L}}$. This is different from the implementation of BILUM [55], where the last reduced system is solved to certain accuracy by a Krylov subspace method preconditioned by ILUT. The advantage of BILUTM includes the added flexibility in controlling the amount of fill-in (and the computation costs during the construction), especially when large-sized blocks are used.

Suppose the right-hand side b and the solution vector x are partitioned according to the independent set ordering as in (2.2); then we would have, on each level,

$$x_\alpha = \begin{pmatrix} x_{\alpha,1} \\ x_{\alpha,2} \end{pmatrix} \qquad \text{and} \qquad b_\alpha = \begin{pmatrix} b_{\alpha,1} \\ b_{\alpha,2} \end{pmatrix}.$$

The forward elimination is performed by solving for a temporary vector y_α, i.e., for $\alpha = 0, 1, \ldots, \mathcal{L}-1$, by solving

$$\begin{pmatrix} L_\alpha & 0 \\ E_\alpha U_\alpha^{-1} & I_\alpha \end{pmatrix} \begin{pmatrix} y_{\alpha,1} \\ y_{\alpha,2} \end{pmatrix} = \begin{pmatrix} b_{\alpha,1} \\ b_{\alpha,2} \end{pmatrix}, \text{ with } \begin{array}{lrcl} \text{(F1)}: & y_{\alpha,1} & = & L_\alpha^{-1} b_{\alpha,1}, \\ \text{(F2)}: & y_{\alpha,2} & = & b_{\alpha,2} - E_\alpha U_\alpha^{-1} y_{\alpha,1}. \end{array}$$

We then solve the last reduced system as

$$\mathbf{L}_{\mathcal{L}}\mathbf{U}_{\mathcal{L}} x_{\mathcal{L}} = b_{\mathcal{L}}.$$

A backward substitution is performed to obtain the solution by solving, for $\alpha = \mathcal{L}-1, \ldots, 1, 0$,

$$\begin{pmatrix} U_\alpha & L_\alpha^{-1}F_\alpha \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_{\alpha,1} \\ x_{\alpha,2} \end{pmatrix} = \begin{pmatrix} y_{\alpha,1} \\ y_{\alpha,2} \end{pmatrix}, \text{ with } \begin{array}{lrcl} \text{(B1)}: & x_{\alpha,1} & = & y_{\alpha,1} - L_\alpha^{-1}F_\alpha x_{\alpha,2}, \\ \text{(B2)}: & x_{\alpha,1} & = & U_\alpha^{-1} x_{\alpha,1}. \end{array}$$

The backward substitution will work since $x_{\alpha,2} = x_{\alpha+1,1}$ and $x_{\mathcal{L}-1,2} = x_{\mathcal{L}}$. The preconditioned iteration process is reminiscent of a multigrid V-cycle algorithm [14], see Figure 4.1. A Krylov subspace iteration is performed on the finest level acting as a smoother; the residual is then transferred level-by-level to the coarsest level, where one sweep of ILUT is used to yield an approximate solution. In the current situation, the coarsest-level ILUT is actually a direct solver with limited accuracy comparable to the accuracy of the whole preconditioning process.

Let us rewrite (4.1) as

$$\text{(4.2)} \qquad \begin{pmatrix} I_\alpha & 0 \\ E_\alpha U_\alpha^{-1} L_\alpha^{-1} & I_\alpha \end{pmatrix} \times \begin{pmatrix} L_\alpha U_\alpha & 0 \\ 0 & A_{\alpha+1} \end{pmatrix} \times \begin{pmatrix} I_\alpha & U_\alpha^{-1} L_\alpha^{-1} F_\alpha \\ 0 & I_\alpha \end{pmatrix}$$

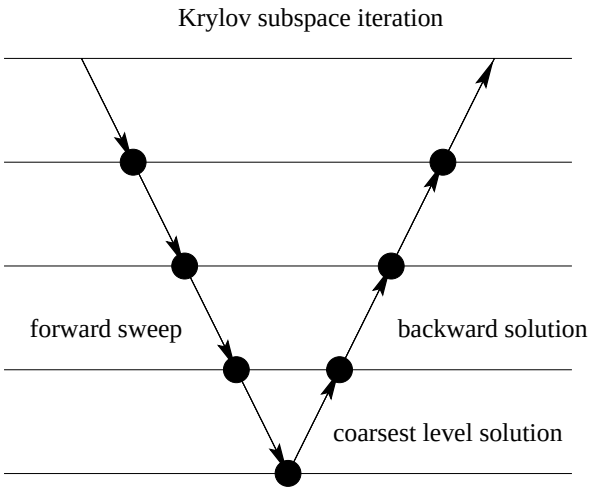

FIG. 4.1. *The multilevel structure of the BILUTM preconditioned Krylov subspace solver.*

and examine a few interesting properties. It is clear that the central part of (4.2) is an operator acting on the full vector, say, x_α ($L_\alpha U_\alpha$ on $x_{\alpha,1}$ and $A_{\alpha+1}$ on $x_{\alpha,2}$). In a two-level analysis, we may define

$$\mathbf{I}_\alpha^{\alpha+1} = \begin{pmatrix} -E_\alpha U_\alpha^{-1} L_\alpha^{-1} & I_\alpha \end{pmatrix} \qquad \text{and} \qquad \mathbf{I}_{\alpha+1}^{\alpha} = \begin{pmatrix} -U_\alpha^{-1} L_\alpha^{-1} F_\alpha \\ I_\alpha \end{pmatrix}$$

as the restriction and interpolation operators, respectively. Then the following results linking BILUTM with the algebraic multigrid methods can be verified directly; see [57].

PROPOSITION 4.1. *Suppose the factorization* (4.2) *exists and is exact. Then*

1. *the reduced system $A_{\alpha+1}$ satisfies the Galerkin condition* (2.1);
2. *if, in addition, A_α is symmetric, then $\mathbf{I}_\alpha^{\alpha+1} = \mathbf{I}_{\alpha+1}^{\alpha}{}^T$.*

The above discussion of the solution procedure omitted the permutation and inverse permutation that must be performed before and after each operation on each level. This is also the approach that we used in our current implementation (and that of BILUM [55]). On the other hand, we may permute the matrices on each level in the construction phase. In this case, only the global permutation is needed before and after the application of the preconditioner [51].

All of the steps of the procedure, including the construction of the preconditioner, are fully parallelizable, except potentially the solution of the last reduced system.[1] For example, the step (F1) of the forward solve with L_α^{-1} can be performed in parallel because only the unknowns within each block are coupled. The same is true for the backward solve with U_α^{-1} in step (B2). All other parts are just matrix-vector operations and vector updates.

Finally, the sparsity of BILUTM depends primarily on the parameter p used to control the amount of fill-in allowed and the size of the block independent sets.

PROPOSITION 4.2. *Let m_α be the size of the block independent set on level α. The number of nonzeros of BILUTM with $\mathcal{L}$ levels of reductions is bounded by $p(2n + \sum_{\alpha=1}^{\mathcal{L}} \alpha m_\alpha)$.*

[1] For this, we may employ a sparse approximate inverse technique [21] or a multicoloring strategy [52] to solve the last reduced system; see [61].

Proof. On each level $0 \le \alpha < \mathcal{L}-1$, the L and U factors of the upper part have at most p elements in each row. The left-hand side of the lower part also has at most p elements in each row; the right-hand side (the reduced system) is not stored. Suppose the sizes of the block independent set and the vertex cover are m_α and r_α, respectively. On level α, the total number of nonzeros is bounded by $2pm_\alpha + pr_\alpha$. Since the last reduced system is factored by ILUT(τ, p), the amount of nonzeros is bounded by $2pn_{\mathcal{L}} = 2pm_{\mathcal{L}}$ and $r_{\mathcal{L}} = 0$. Summing all levels yields the bound

$$\sum_{\alpha=0}^{\mathcal{L}-1}(2pm_\alpha + pr_\alpha) + 2pm_{\mathcal{L}} = p\left(2\sum_{\alpha=0}^{\mathcal{L}} m_\alpha + \sum_{\alpha=0}^{\mathcal{L}-1} r_\alpha\right). \tag{4.3}$$

Since the nodes of all independent sets and the last reduced system constitute the order of the matrix, we have

$$\sum_{\alpha=0}^{\mathcal{L}} m_\alpha = n. \tag{4.4}$$

Note that $r_\alpha = m_{\alpha+1} + r_{\alpha+1}$ for $0 \le \alpha < \mathcal{L}-1$, and $r_{\mathcal{L}} = 0$. It is easy to verify

$$\sum_{\alpha=0}^{\mathcal{L}-1} r_\alpha = \sum_{\alpha=1}^{\mathcal{L}} \alpha m_\alpha. \tag{4.5}$$

Substituting (4.4) and (4.5) into (4.3) gives the bound for the number of nonzeros of BILUTM as

$$2pn + p\sum_{\alpha=1}^{\mathcal{L}} \alpha m_\alpha = p\left(2n + \sum_{\alpha=1}^{\mathcal{L}} \alpha m_\alpha\right). \tag{4.6}$$

□

We remark that in (4.6), the term $2pn$ is the bound for the number of nonzeros of the standard ILUT. Due to the block structure of BILUTM, the first few rows of each block of the upper L factors have less than p elements and the overall nonzero number of BILUTM is actually smaller. The term $p\sum_{\alpha=1}^{\mathcal{L}} \alpha m_\alpha$ represents the extra nonzeros for the multilevel implementation. Note that m_0 is not in the second term and the factor α grows as the level increases. It is therefore advantageous to have large block independent sets in the first few levels.

5. Numerical experiments. Implementations of ILUM and BILUM have been described in detail in [51, 55]. One significant difference between BILUTM and BILUM and ILUM is that we do not use an inner iteration to solve the last reduced system. Instead, backward and forward solution steps are performed with the incomplete LU factors $\mathbf{L}_{\mathcal{L}}\mathbf{U}_{\mathcal{L}}$ of the last reduced system. Unless otherwise explicitly indicated, we used the following default parameters for our preconditioned iterative solver: GMRES with a restart value of 50 was used as the accelerator; the maximum number of reductions (levels) allowed was 10, i.e., $\mathcal{L} = 10$; the threshold dropping tolerance was set to be $\tau = 10^{-4}$, and the block sizes were chosen to be equal to the parameter p used to control the number of fill-in elements.

All matrices were considered general sparse matrices and any available structures were not exploited. The right-hand side was generated by assuming that the solution is a vector of all ones and the initial guess was a vector of some random numbers. The computations were terminated when the 2-norm of the residual was reduced by a factor

TABLE 5.1
Comparison of BILUTM and ILUT for solving the convection-diffusion problem with different Re.

	BILUTM					ILUT				
Re	p	iter	total	solu	spar	p	iter	total	solu	spar
1	10	56	30.8	16.1	3.53	8	58	15.0	14.2	3.21
10	10	63	32.2	17.6	3.53	8	42	13.1	11.9	3.21
100	10	39	25.4	10.6	3.55	9	21	6.83	5.30	3.60
1000	10	13	17.5	2.80	3.39	9	5	2.09	1.08	3.32
10000	20	22	24.1	6.40	5.76	17	22	10.4	7.18	5.82
100000	100	43	43.8	22.5	15.2	180	25	192.5	44.7	71.5

of 10^7. We also set an upper bound of 100 for the GMRES iteration. 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.

In all tables with numerical results, "bsize" is the size of the uniform blocks (used only when bsize $\neq p$), "iter" shows the number of GMRES iterations, "total" shows the CPU time in seconds for the preprocessing and solution phases, "solu" shows the CPU time for the solution phase only, and "spar" shows the sparsity ratio which is the ratio between the number of nonzeros of the preconditioner to that of the original matrix. The symbol "–" indicates lack of convergence. We mainly compare BILUTM with (single-level) ILUT and sometimes BILUTM with different parameters.

Convection-diffusion problem. We first consider a convection-diffusion problem

$$u_{xx} + u_{yy} + \mathrm{Re}(\exp(xy-1)u_x - \exp(-xy)u_y) = 0, \tag{5.1}$$

defined on the unit square. Here Re is the so-called Reynolds number. A Dirichlet boundary condition was assumed, but the linear systems used the artificially generated right-hand side as stated above. We used the standard 5-point central difference discretization scheme with a uniform mesh $h = 1/201$. The resulting matrices with different values of Re have 40,000 unknowns and 199,200 nonzeros. The percentage of the rows with diagonal dominance becomes smaller as Re increases [63].

Table 5.1 gives some performance data of BILUTM and ILUT for solving (5.1) with different Re. Here p was varied so that BILUTM and ILUT used approximately the same storage space. There was an exception for $\mathrm{Re} = 10^5$ when ILUT did not converge for $p < 180$ while BILUTM converged for $p = 100$. It can be seen that for simple problems (small Re), ILUT was more efficient than BILUTM. They performed similarly for $\mathrm{Re} = 10^4$. For $\mathrm{Re} \geq 10^5$, ILUT failed to converge unless it used very large storage space. Conversely, BILUTM did very well for this difficult problem.

Note that BILUTM took more steps to converge for small Re problems. However, note also that the inherent parallelism in BILUTM is far superior to that in ILUT. Table 5.2 gives another set of tests with larger values for p. We see that BILUTM performed better than ILUT (solution time) with high accuracy. This improvement comes without sacrificing potential for parallelism but the cost of preprocessing increased somewhat.

The next test is to show how the performance of BILUTM is affected by the block sizes. Here we chose $p = 10$ and $\mathrm{Re} = 10^3$. The size of the uniform blocks varied from 1 to 400. Note that for the large block sizes, we actually had only three levels of reductions. The results are given in Table 5.3.

We note that for most values of the block size, the performance of BILUTM has no significant difference. This property is desirable since it implies that, for this test

TABLE 5.2
Comparison of high accuracy BILUTM and ILUT for solving the convection-diffusion problem with different Re.

	BILUTM					ILUT				
Re	p	iter	total	solu	spar	p	iter	total	solu	spar
1	50	10	14.1	3.29	9.30	24	13	9.63	5.27	9.47
10	50	11	14.3	3.64	9.29	24	14	10.1	5.71	9.46
100	50	7	12.5	2.21	8.83	22	7	6.54	2.62	8.57
1000	50	3	7.91	0.76	5.93	15	3	1.76	0.71	3.78
10000	50	6	14.2	2.04	9.95	43	6	8.47	2.38	9.81

TABLE 5.3
Performance of BILUTM$(10^{-4}, 10)$ *as a function of the block size. Convection-diffusion problem with* $\mathrm{Re} = 10^3$.

bsize	1	5	10	30	50	90	170	200	250	290	350	380	400
iter	17	14	13	12	11	11	11	11	11	12	11	12	12
total	112	28.9	17.6	9.3	7.5	6.6	6.9	7.4	8.8	8.5	10.2	14.3	11.5
solu	3.8	3.1	2.8	2.4	2.2	2.2	2.2	2.2	2.2	2.4	2.2	2.4	2.4
spar	2.5	3.2	3.4	3.2	3.2	3.3	3.4	3.4	3.4	3.4	3.4	3.5	3.5

TABLE 5.4
Description of the TOKAMAK matrices.

Name	Unknowns	Nonzeros	Condition number	Diagonal dominance
UTM300	300	3 155	1.50(+06)	no
UTM1700a	1 700	21 313	6.24(+06)	no
UTM1700b	1 700	21 509	1.16(+07)	no
UTM3060	3 060	42 211	3.94(+07)	no
UTM5940	5 940	83 842	1.91(+09)	no

TABLE 5.5
Comparison of BILUTM and ILUT for solving the first four TOKAMAK matrices.

	BILUTM					ILUT				
Matrices	p	iter	total	solu	spar	p	iter	total	solu	spar
UTM300	20	26	0.11	0.045	4.25	20	17	0.039	0.021	2.38
UTM1700a	20	36	1.09	0.63	3.98	30	30	0.82	0.42	3.64
UTM1700b	20	27	0.86	0.44	3.82	30	29	0.77	0.40	3.56
UTM3060	30	26	2.18	0.99	4.70	38	25	1.90	0.99	4.63

problem and with current test conditions, the convergence rate of BILUTM would not be very sensitive to the number of processors had our test been implemented on a parallel computer.

TOKAMAK matrices. The TOKAMAK matrices are real unsymmetric matrices which arise from nuclear fusion plasma simulations in a tokamak reactor.[2] These are part of the SPARSKIT collections and have been provided by P. Brown of Lawrence Livermore National Laboratory. Table 5.4 shows some data on these matrices.

The solution details for the first four TOKAMAK matrices are listed in Table 5.5, and those for UTM5940 are listed in Table 5.6. We note that for the first three matrices of small sizes, ILUT seemed to outperform BILUTM, given a similar memory consumption. They were almost tied for UTM3060. For the largest TOKAMAK matrix, BILUTM performed much better than ILUT by virtually all measures (Table 5.6). In fact, ILUT could not converge for $p \leq 70$, while BILUTM still converged

[2]The TOKAMAK matrices are available online from the matrix market of the National Institute of Standards Technology at http://math.nist.gov/MatrixMarket.

TABLE 5.6
Solving the UTM5940 matrix by BILUTM and ILUT with different parameters.

BILUTM						ILUT					
p	τ	iter	total	solu	spar	p	τ	iter	total	solu	spar
100	10^{-4}	19	10.4	2.29	9.93	130	10^{-4}	25	14.5	4.04	13.5
90	10^{-4}	21	9.05	2.38	9.13	120	10^{-4}	28	13.8	4.32	12.7
80	10^{-4}	23	8.67	2.56	8.78	110	10^{-4}	31	13.4	4.57	11.8
70	10^{-4}	26	8.82	2.79	8.21	100	10^{-4}	35	12.8	4.90	10.9
60	10^{-4}	26	6.64	2.52	7.13	90	10^{-4}	37	11.8	4.90	9.96
50	10^{-4}	27	6.62	2.49	6.45	80	10^{-4}	46	12.1	5.78	8.94
40	10^{-4}	36	6.58	3.14	5.64	70	10^{-5}	–	–	–	–
30	10^{-4}	75	8.65	6.05	4.72	70	10^{-4}	–	–	–	–
20	10^{-4}	96	8.64	6.82	3.44	70	10^{-3}	–	–	–	–

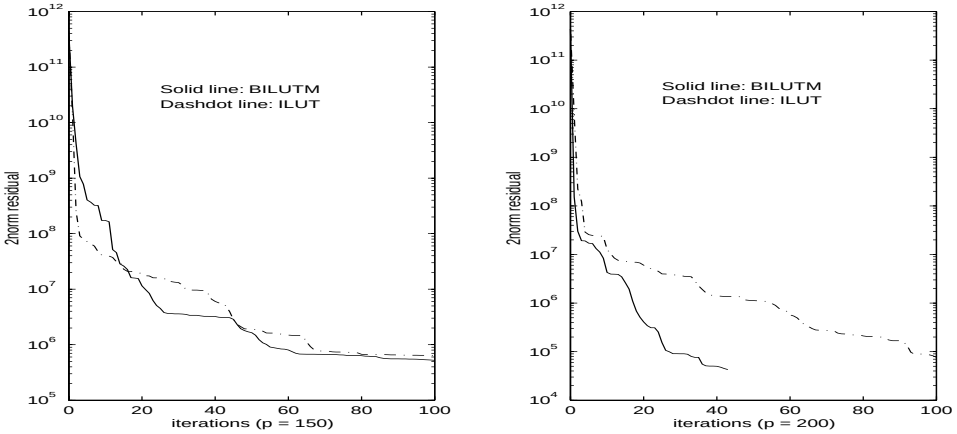

FIG. 5.1. *Convergence history of BILUTM and ILUT for solving the RAEFSKY4 matrix.*

with $p = 20$. It can be seen that BILUTM needed less than half the storage required for ILUT to converge. With more storage space made available for ILUT, BILUTM still outperformed ILUT with a faster convergence rate (and less memory consumption.)

RAEFSKY4 matrix. The RAEFSKY4 matrix[3] has 19,779 unknowns and 1,328,611 nonzeros. It is from the buckling problem for the container model and was supplied by H. Simon from Lawrence Berkeley National Laboratory (originally created by A. Raefsky from Centric Engineering). This is probably the hardest one in the total of six RAEFSKY matrices. (BILUM with diagonal threshold techniques was able to solve all but the RAEFSKY4 matrix [56].) In order for BILUTM to converge fast, we found it necessary to use a larger restart value (100) for GMRES. Figure 5.1 shows the convergence history of BILUTM and ILUT with $p = 150$ and 200, respectively. In both tests, the block size was 200 for BILUTM. We note that with $p = 150$, both preconditioners had similar lack of full convergence. However, with $p = 200$, BILUTM converged in 43 iterations while ILUT was still not fully converged in 100 iterations.

Figure 5.2 depicts performance comparisons when BILUTM was used with differ-

[3]The RAEFSKY matrices are available online from the University of Florida sparse matrix collection [24] at http://www.cise.ufl.edu/~davis/sparse.

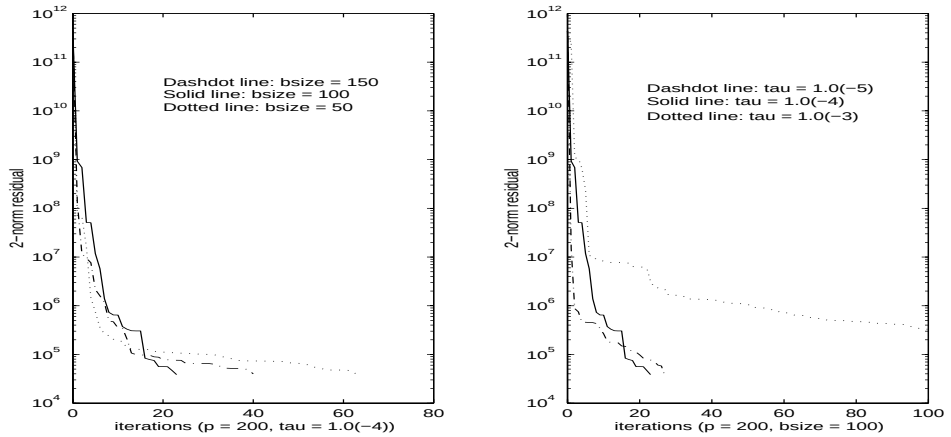

FIG. 5.2. *Convergence history of BILUTM with different parameters for solving the RAEFSKY4 matrix. Left: different block size. Right: different dropping threshold τ.*

TABLE 5.7
Solving the WIGTO966 matrix by BILUTM and ILUT with different parameters.

BILUTM							ILUT					
bsize	p	τ	iter	total	solu	spar	p	τ	iter	total	solu	spar
200	200	10^{-5}	7	29.6	1.39	6.21	400	10^{-4}	16	72.0	5.05	9.65
100	200	10^{-5}	8	27.8	1.70	6.69	400	10^{-3}	18	52.8	5.14	8.57
100	150	10^{-5}	11	21.3	2.10	5.87	360	10^{-5}	18	76.8	5.21	9.48
100	100	10^{-5}	33	18.5	5.17	4.45	360	10^{-4}	20	68.7	5.64	9.17
100	100	10^{-4}	44	19.7	6.96	4.43	360	10^{-3}	33	61.4	8.97	8.71
70	70	10^{-5}	25	11.0	3.20	3.17	340	10^{-5}	28	76.6	7.91	9.14
30	60	10^{-5}	43	11.4	5.27	2.80	340	10^{-4}	44	76.2	13.2	8.92
30	60	10^{-4}	41	11.0	4.94	2.74	340	10^{-3}	42	59.5	10.9	8.14
30	40	10^{-4}	86	12.3	8.52	2.02	320	10^{-5}	–	–	–	–
20	40	10^{-5}	93	12.7	8.96	1.89	320	10^{-4}	41	71.3	11.9	8.59
20	40	10^{-4}	86	11.8	8.16	1.86	320	10^{-3}	39	54.2	10.5	7.90
20	35	10^{-4}	89	11.3	8.04	1.70	300	10^{-4}	–	–	–	–
20	35	10^{-3}	95	11.4	8.33	1.60	300	10^{-3}	–	–	–	–

ent parameters. The left part of Figure 5.2 shows that BILUTM with block size 100 gave the best results. Larger and smaller block sizes resulted in deterioration of convergence. The right part of Figure 5.2 shows that $\tau = 10^{-4}$ was the best among the three values tested for this parameter. It is interesting to note that higher accuracy ($\tau = 10^{-5}$) did not yield faster convergence.

WIGTO966 matrix. The WIGTO966 matrix[4] has 3,864 unknowns and 238,252 nonzeros. It comes from an Euler equation model and was supplied by L. Wigton from Boeing. It is solvable by ILUT with large values of p [19]. This matrix was also used to compare BILUM with ILUT in [54] and to test point and block preconditioning techniques in [20, 22]. BILUM (with GMRES(10)) was shown to be six times faster than ILUT with only one-third of the memory required by ILUT [54]. In our current tests, we chose several values for τ and p for BILUTM and ILUT, and the size of the blocks in case of BILUTM. We tabulate the results in Table 5.7. Amazingly BILUTM converged for this problem with a sparsity ratio of 1.60. The smallest sparsity ratio that yields convergence for ILUT is 7.90. In addition, BILUTM converged almost five

[4]The WIGTO966 matrix is available from the authors.

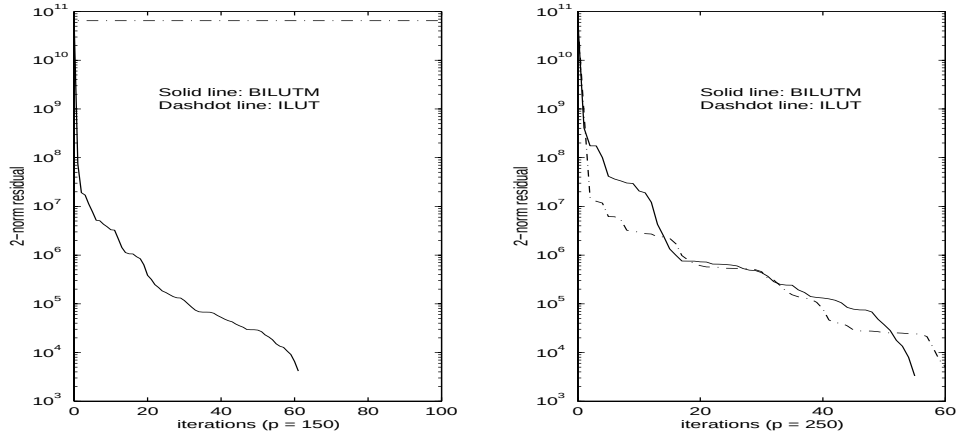

FIG. 5.3. *Convergence history of BILUTM and ILUT with different amount of fill-in (p) for solving the OLAFU matrix.*

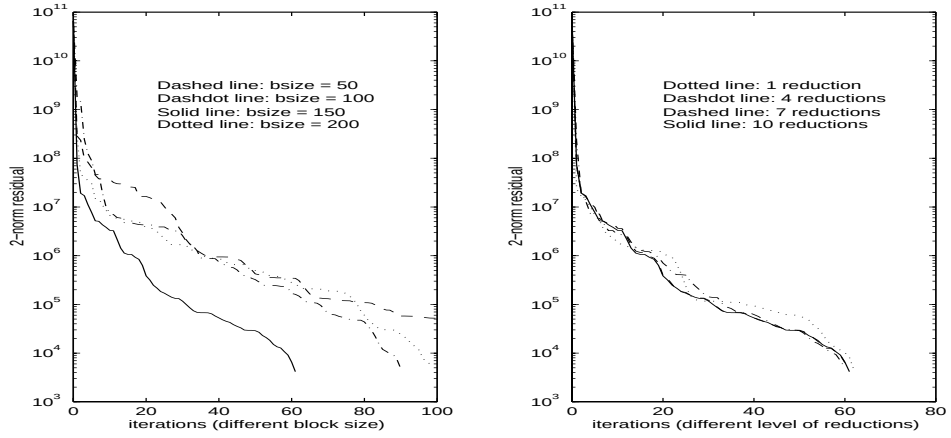

FIG. 5.4. *Convergence history of BILUTM with different parameters and* $p = 150$ *for solving the OLAFU matrix. Left: different block size. Right: different level of reductions.*

times faster (total CPU time) than ILUT and used just about one-fifth of the memory that was required by ILUT.

OLAFU matrix. The OLAFU matrix[5] has 16,146 unknowns and 1,015,156 nonzeros. It is a structural modeling problem from NASA Langley. The tests with OLAFU also used GMRES(100) as the accelerator. Figure 5.3 shows the comparison between BILUM and ILUT with two different values of p. We point out that with $p = 150$, ILUT did not show any signs of convergence (see the left part of Figure 5.3), while BILUTM converged within 61 iterations. The right part of Figure 5.3 shows that ILUT did converge with more fill-in ($p = 200$), but BILUTM was still faster. These results indicate that the OLAFU matrix cannot be solved by ILUT without sufficient accuracy. We remark that for the comparison shown in Figure 5.3, both BILUTM

[5]The OLAFU matrix is available online from the University of Florida sparse matrix collection [24] at http://www.cise.ufl.edu/~davis/sparse.

TABLE 5.8
Description of the BARTH matrices.

Name	Unknowns	Nonzeros	Descriptions
BARTHT1A	14 075	481 125	Small airfoil 2D Navier–Stokes, distance 1
BARTHS1A	15 735	539 225	Small airfoil 2D Navier–Stokes, distance 1
BARTHT2A	14 075	1 311 725	Small airfoil 2D Navier–Stokes, distance 2
BARTHS2A	15 735	1 510 325	Small airfoil 2D Navier–Stokes, distance 2

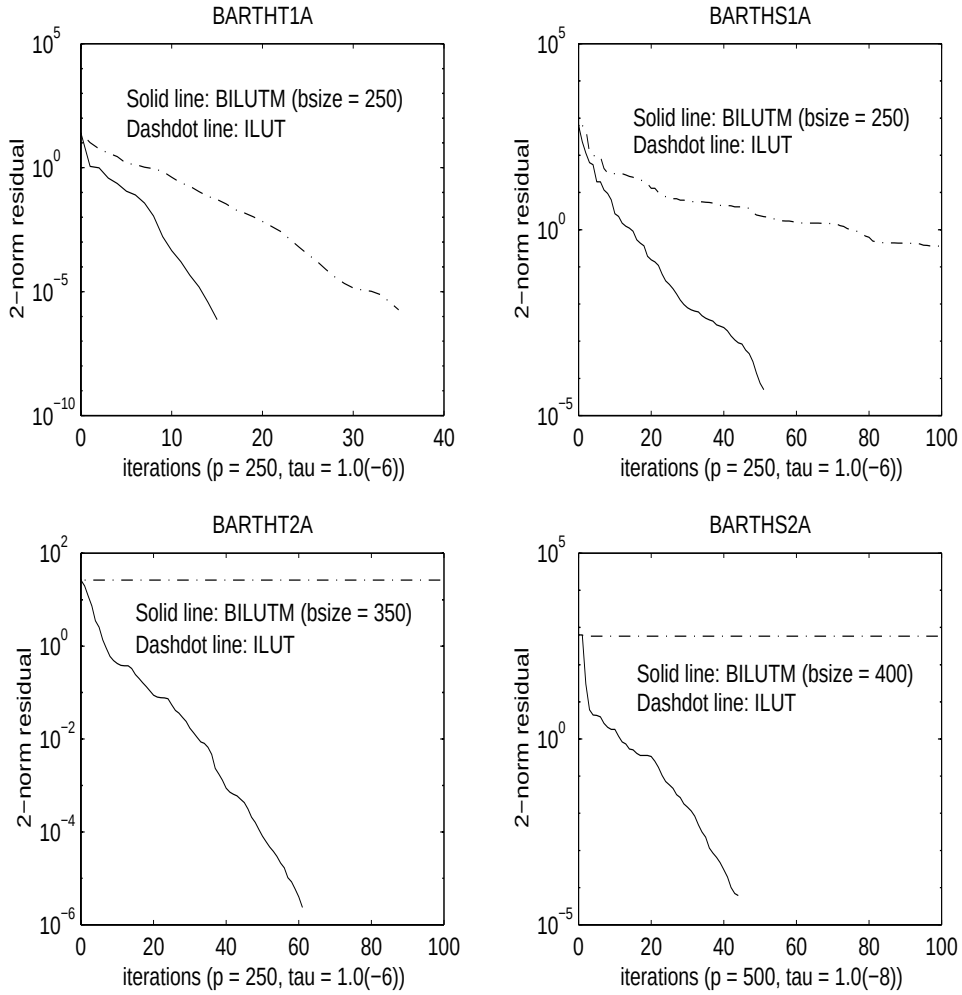

FIG. 5.5. *Convergence history of BILUTM and ILUT for solving the BARTH matrices.*

and ILUT used approximately the same memory space for the same value of p.

In Figure 5.4, we show test results of using BILUTM with different parameters to solve OLAFU. The parameter $p = 150$ was fixed. The left part of Figure 5.4 shows that the size of the blocks did affect the convergence of BILUTM for this large problem. It seems that taking the block size to be equal to the fill-in parameter p yielded the best results. The right part of Figure 5.4 indicates that the number of levels (reductions) did not have significant effect on the convergence of BILUTM. This is probably because the largest independent set had been factored out in the

first level and the use of ILUT on the coarsest level gave comparable accuracy. Recall that there is a big difference between one level of reduction and no reduction at all, since Figure 5.3 shows that BILUTM without reduction (actually equivalent to ILUT) failed to converge.

BARTH matrices. The BARTH matrices[6] were supplied by T. Barth of NASA Ames. They are for a two-dimensional high Reynolds number airfoil problem, with one equation turbulence model. The S and T matrices are results of using different grids. The grid of the T matrices has a concentration of elements unrealistically close to the airfoil. The four BARTH matrices are described in Table 5.8. Note that in order for ILUT and BILUTM to work properly, zero diagonals are added. The BARTH matrices have been used as test matrices for other ILU-type techniques in [19], but none of them has been solved by enhanced BILUM techniques [57], partly because of the prohibitive computation and memory costs associated with the use of very large-sized blocks (on the given computer).

We present in Figure 5.5 only one set of comparisons of BILUTM and ILUT by solving the BARTH matrices using large size blocks and GMRES(100). We remark that for this set of test parameters, ILUT took about three times more CPU time (BARTHT1A) and used about 20% more memory than BILUTM did. We found that BILUTM converged much faster than ILUT for these indefinite matrices with small and zero diagonals. For the two largest BARTH matrices, ILUT almost completely failed to reduce the residual norm within 100 iterations, while BILUTM converged satisfactorily.

6. Concluding remarks. We have presented a mulitlevel block ILU preconditioner with a dual dropping strategy for solving general sparse matrices. The method offers flexibility in controlling the amount of fill-in during the ILU factorization when large size blocks are used for domain decomposition based implementation of multilevel ILU preconditioning method. A particular merit of BILUTM is that both the construction and application phases of the preconditioner have a high level of inherent parallelism.

We gave an upper bound for the number of nonzeros of the preconditioner. We showed that the extra storage costs of multilevel implementation are not substantial if large block independent sets can be found in the first few reductions. It may also be beneficial not to have too many levels, especially when the size of the block independent sets becomes small.

Our numerical experiments with several matrices show that the proposed technique indeed demonstrates the anticipated flexibility and effectiveness. As a parallelizable high accuracy preconditioner, BILUTM is comparable with sequential ILUT for solving easy problems. For some difficult problems, where high accuracy preconditioning is a must, BILUTM is more robust and is more efficient than ILUT and usually requires less memory. In other words, this preconditioner does not sacrifice convergence in order to improve parallelism. This is in sharp contrast with lower order preconditioner such as ILU(0) or the high-order single-level preconditioner such as ILUT.

Although we did not directly compare BILUTM with other multilevel preconditioning techniques, we did remark that BILUTM solved several difficult matrices that might not be solved by BILUM efficiently on the given computer because of the computation and memory costs associated with the use of very large size blocks.

[6]The BARTH matrices are available from the authors.

Implementations of grid-based multilevel methods on parallel and vector computers can be found in [41] (for structured matrices) and those of domain-based (two-level) methods can be found in [53, 54]. Those and other implementations on shared-memory machines [11] demonstrate the advantage of the inherent parallelism of the multilevel preconditioning methods. Conversely, implementing multilevel preconditioning methods on distributed-memory machines requires the consideration of cost trade-off between communications and computations. It is obviously not advantageous to have too small blocks and too many levels. The parallel solution of the last reduced system may also be desirable in certain applications.

REFERENCES

[1] E. C. Anderson and Y. Saad, *Solving sparse triangular systems on parallel computers*, Internat. J. High Speed Comput., 1 (1989), pp. 73–96.

[2] O. Axelsson and M. Neytcheva, *Algebraic multilevel iteration method for Stieltjes matrices*, Numer. Linear Algebra Appl., 1 (1994), pp. 216–236.

[3] O. Axelsson and P. S. Vassilevski, *A survey of multilevel preconditioned iterative methods*, BIT, 29 (1989), pp. 769–793.

[4] O. Axelsson and P. S. Vassilevski, *Algebraic multilevel preconditioning methods.* II, SIAM J. Numer. Anal., 27 (1990), pp. 1569–1590.

[5] V. A. Bandy, *Black Box Multigrid for Convection-Diffusion Equations on Advanced Computers*, Ph.D. thesis, University of Colorado at Denver, Denver, CO, 1996.

[6] R. E. Bank and R. K. Smith, *The incomplete factorization multigraph algorithm*, SIAM J. Sci. Comput., 20 (1999), pp. 1349–1364.

[7] R. E. Bank and C. Wagner, *Multilevel ILU decomposition*, Numer. Math., to appear.

[8] R. E. Bank and J. Xu, *The hierarchical basis multigrid method and incomplete LU decomposition*, in the Seventh International Symposium on Domain Decomposition Methods for Partial Differential Equations, D. Keyes and J. Xu, eds., AMS, Providence, RI, 1994, pp. 163–173.

[9] M. Benzi and M. Tuma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.

[10] 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, Appl. Numer. Math., 32 (1997), pp. 439–455.

[11] E. F. F. Botta, A. van der Ploeg, and F. W. Wubs, *A fast linear system solver for large unstructured problems on a shared-memory computer*, in Proceedings of the Conference on Algebraic Multilevel Methods with Applications, O. Axelsson and B. Polman, eds., University of Nijmegen, The Netherlands, 1996, pp. 105–116.

[12] E. F. F. Botta and F. W. Wubs, *Matrix renumbering ILU: An effective algebraic multilevel ILU-preconditioner for sparse matrices*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 1007–1026.

[13] D. Braess, *Towards algebraic multigrid for elliptic problems of second order*, Computing, 55 (1995), pp. 379–393.

[14] W. L. Briggs, *A Multigrid Tutorial*, SIAM, Philadelphia, PA, 1987.

[15] T. F. Chan, S. Go, and J. Zou, *Multilevel Domain Decomposition and Multigrid Methods for Unstructured Meshes: Algorithms and Theory*, Technical report CAM 95-24, Department of Mathematics, UCLA, Los Angeles, CA, 1995.

[16] T. F. Chan, W. P. Tang, and W. L. Wan, *Wavelet sparse approximate inverse preconditioners*, BIT, 37 (1997), pp. 644–660.

[17] Q. S. Chang, Y. S. Wong, and L. Z. Feng, *New interpolation formulas of using geometric assumptions in the algebraic multigrid method*, Appl. Math. Comput., 50 (1992), pp. 223–254.

[18] Q. S. Chang, Y. S. Wong, and H. Q. Fu, *On the algebraic multigrid method*, J. Comput. Phys., 125 (1996), pp. 279–292.

[19] A. Chapman, Y. Saad, and L. Wigton, *High-Order ILU Preconditioners for CFD Problems*, Technical report UMSI 96/14, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1996.

[20] E. Chow and M. A. Heroux, *An object-oriented framework for block preconditioning*, ACM Trans. Math. Software, 24 (1998), pp. 159–183.

[21] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM J. Sci. Comput., 18 (1997), pp. 1657–1675.
[22] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.
[23] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.
[24] T. Davis, *University of Florida sparse matrix collection*, NA Digest, 97(23), June 7, 1997.
[25] E. F. D'Azevedo, P. A. Forsyth, and W. P. Tang, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 944–961.
[26] E. F. D'Azevedo, P. A. Forsyth, and W. P. Tang, *Towards a cost effective ILU preconditioner with high level fill*, BIT, 31 (1992), pp. 442–463.
[27] P. M. de Zeeuw, *Matrix-dependent prolongations and restrictions in a blackbox multigrid solver*, J. Comput. Appl. Math., 33 (1990), pp. 1–25.
[28] J. E. Dendy, Jr., *Black box multigrid*, J. Comput. Phys., 48 (1982), pp. 366–386.
[29] I. S. Duff and G. A. Meurant, *The effect of reordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.
[30] L. C. Dutto, *The effect of reordering on the preconditioned GMRES algorithm for solving the compressible Navier-Stokes equations*, Internat. J. Numer. Methods Engrg., 36 (1993), pp. 457–497.
[31] H. C. Elman, *Approximate Schur complement preconditioners on serial and parallel computers*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 581–605.
[32] H. C. Elman and E. Agron, *Ordering techniques for the preconditioned conjugate gradient method on parallel computers*, Comput. Phys. Commun., 53 (1989), pp. 253–269.
[33] K. Gallivan, A. Sameh, and Z. Zlatev, *A parallel hybrid sparse linear system solver*, Comput. Systems Engrg., 1 (1990), pp. 183–195.
[34] J. A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[35] N. I. M. Gould and J. A. Scott, *Sparse approximate-inverse preconditioners using norm-minimization techniques*, SIAM J. Sci. Comput., 19 (1998), pp. 605–625.
[36] M. Griebel and G. Starke, *Multilevel preconditioning based on discrete symmetrization for convection-diffusion equations*, J. Comput. Appl. Math., 83 (1997), pp. 165–183.
[37] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1998), pp. 838–853.
[38] M. T. Jones and P. E. Plassman, *A parallel graph coloring heuristic*, SIAM J. Sci. Comput., 14 (1993), pp. 654–669.
[39] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.
[40] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.
[41] M. Neytcheva, A. Padiy, M. Mellaard, K. Georgiev, and O. Axelsson, *Scalable and Optimal Iterative Solvers for Linear and Nonlinear Problems*, Technical report MRI 9613, Mathematical Research Institute, University of Nijmegen, The Netherlands, 1996.
[42] Y. Notay and Z. Ould Amar, *Incomplete factorization preconditioning may lead to multigrid like speed of convergence*, in Advanced Mathematics: Computation and Applications, A. S. Alekseev and N. S. Bakhvalov, eds., NCC Publisher, Novosibirsk, Russia, 1996, pp. 435–446.
[43] Y. Notay and Z. Ould Amar, *A nearly optimal preconditioning based on recursive red-black orderings*, Numer. Linear Algebra Appl., 4 (1997), pp. 369–391.
[44] O. Orterby and Z. Zlatev, *Direct Methods for Sparse Matrices*, Springer-Verlag, New York, 1983.
[45] A. A. Reusken, *Multigrid with matrix-dependent transfer operators for convection-diffusion problems*, in Multigrid Method IV, Proceedings of the Fourth European Multigrid Conference, P. W. Hemker and P. Wesseling, eds., International Series of Numerical Mathematics 116, Birkhäuser Verlag, Basel, 1994, pp. 269–280.
[46] 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.
[47] J. W. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid Methods, Frontiers Appl. Math. 3, S. McCormick, ed., SIAM, Philadelphia, PA, 1987, pp. 73–130.
[48] Y. Saad, *Krylov subspace methods on supercomputers*, SIAM J. Sci. Stat. Comput., 10 (1989),

pp. 1200–1232.
[49] Y. Saad, *Highly parallel preconditioners for general sparse matrices*, in Recent Advances in Iterative Methods, IMA Volumes in Mathematics and Its Applications 60, G. Golub, M. Luskin, and A. Greenbaum, eds., Springer Verlag, New York, NY, 1994, pp. 165–199.
[50] Y. Saad, *ILUT: A dual threshold incomplete ILU preconditioner*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.
[51] Y. Saad, *ILUM: A multi-elimination ILU preconditioner for general sparse matrices*, SIAM J. Sci. Comput., 17 (1996), pp. 830–847.
[52] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, MA, 1996.
[53] 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.
[54] Y. Saad, M. Sosonkina, and J. Zhang, *Domain decomposition and multi-level type techniques for general sparse linear systems*, in Domain Decomposition Methods 10, Contemporary Mathematics 218, J. Mandel, C. Farhat, and X.-C. Cai, eds., AMS, Providence, RI, 1998, pp. 174–190.
[55] Y. Saad and J. Zhang, *BILUM: Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., 20 (1999), pp. 2103–2121.
[56] Y. Saad and J. Zhang, *Diagonal Threshold Techniques in Robust Multi-Level ILU Preconditioners for General Sparse Linear Systems*, Technical report UMSI 98/7, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1998.
[57] Y. Saad and J. Zhang, *Enhanced Multi-Level Block ILU Preconditioning Strategies for General Sparse Linear Systems*, Technical report UMSI 98/98, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1998.
[58] H. A. van der Vorst, *A vectorizable version of some ICCG methods*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 350–356.
[59] P. Vanek, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.
[60] D. P. Young, R. G. Melvin, F. T. Johnson, J. E. Bussoletti, L. B. Wigton, and S. S. Samant, *Application of sparse matrix solvers as effective preconditioners*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 1186–1199.
[61] J. Zhang, *Sparse Approximate Inverse and Multi-Level Block ILU Preconditioning Techniques for General Sparse Matrices*, Technical report 279-98, Department of Computer Science, University of Kentucky, Lexington, KY, 1998.
[62] J. Zhang, *A Sparse Approximate Inverse Technique for Parallel Preconditioning of General Sparse Matrices*, Technical report 281-98, Department of Computer Science, University of Kentucky, Lexington, KY, 1998.
[63] J. Zhang, *Preconditioned iterative methods and evaluation of finite difference schemes for convection-diffusion*, Appl. Math. Comput., to appear.
[64] Z. Zlatev, *Use of iterative refinement in the solution of sparse linear systems*, SIAM J. Numer. Anal., 19 (1982), pp. 381–399.