
ELSEVIER

Journal of Computational and Applied Mathematics 130 (2001) 99–118

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

www.elsevier.nl/locate/cam

Enhanced multi-level block ILU preconditioning strategies for general sparse linear systems ☆

Yousef Saad[a], Jun Zhang[b,*]

[a] *Department of Computer Science and Engineering, University of Minnesota, 4-192 EE/CS Building, 200 Union Street S.E., Minneapolis, MN 55455, USA*

[b] *Department of Computer Science, University of Kentucky, 773 Anderson Hall, Lexington, KY 40506-0046, USA*

Received 2 December 1998; received in revised form 17 September 1999

Abstract

This paper introduces several strategies to deal with pivot blocks in multi-level block incomplete LU factorization (BILUM) preconditioning techniques. These techniques are aimed at increasing the robustness and controlling the amount of fill-ins of BILUM for solving large sparse linear systems when large-size blocks are used to form block-independent set. Techniques proposed in this paper include double-dropping strategies, approximate singular-value decomposition, variable size blocks and use of an arrowhead block submatrix. We point out the advantages and disadvantages of these strategies and discuss their efficient implementations. Numerical experiments are conducted to show the usefulness of the new techniques in dealing with hard-to-solve problems arising from computational fluid dynamics. In addition, we discuss the relation between multi-level ILU preconditioning methods and algebraic multi-level methods. © 2001 Elsevier Science B.V. All rights reserved.

MSC: 65F10; 65F50; 65N55; 65Y05

Keywords: Incomplete LU factorization; Multi-level ILU preconditioner; Krylov subspace methods; Multi-elimination ILU factorization; Algebraic multigrid method

1. Introduction

Central to many scientific and engineering problems is the solution of large sparse linear systems of equations of the form

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

☆ This work was supported in part by NSF under grant CCR-9618827 and in part by the Minnesota Supercomputer Institute.

* Corresponding author. Tel.: +606 257 3892; fax: +606 323 1971.

E-mail addresses: saad@cs.umn.edu (Y. Saad), jzhang@cs.uky.edu (J. Zhang).

PII: S0377-0427(99)00388-X

where A is of dimension n and is usually unsymmetric and unstructured. The most commonly used iterative methods for solving system (1) are multigrid methods and Krylov subspace methods. Each of the two classes of methods contains a rich variety of methods and each has its own advantages and disadvantages. One attractive feature of a multigrid method is its grid-independent convergence and optimal scalability [28]. For certain type of problems, the CPU and memory costs are proportional to the size of the problems. The obvious disadvantage of multigrid methods is their limited applicability. Full multigrid efficiency can only be achieved for problems associated with certain types of partial differential equations defined on regularly structured domains which have sufficient regularity. Though some algebraic multigrid (AMG) methods have been designed to deal with more general problems [9,20], the success of such methods and the type of problems solved are still limited. There seems to exist no multigrid approach that is capable of efficiently solving general unstructured sparse linear systems. On the other hand, the various Krylov subspace methods can be viewed as fairly general-purpose iterative solvers that target general sparse linear systems. However, the simplest Krylov subspace methods are not robust. In addition, their convergence rates depend heavily on problem size, in contrast with multigrid techniques. This lack of scalability puts severe limits on the application of such methods to solving large-scale problems.

The robustness and efficiency of Krylov subspace methods can be improved dramatically by using a suitable preconditioner [23]. The above discussion suggests that multigrid and Krylov subspace methods are complementary in which one method's weakness is a strength of the other. Thus, a judicious combination of both methods may result in a powerful general-purpose iterative solver. This idea has been suggested by several authors [14,22,26] and some results using this type of methods have appeared in the literature, see e.g., [7,8,19,22,26]. Some of these methods [19] take the approach of algebraic multigrid methods, others [7,22,26] are more akin to domain decomposition techniques. As pointed out in [25], the major difference of these two types of methods are the choice of the reduced system (the coarse level system). We note in passing that parallelism is originally emphasized in domain decomposition-type multi-level methods [22]. Promising test results with two-level implementations of similar methods on shared and distributed memory parallel computers have been reported [24,25]. Another research direction in algebraic multi-level preconditioning methods is towards the analysis of such methods for solving regularly structured problems arising from finite element or finite difference discretization of partial differential equations [3,2,5]. Although these results are restricted to structured matrices, they support a theory of near optimality of algebraic multi-level preconditioning methods for certain problems [18]. The multi-level structure of BILUM is more similar to the hierarchical basis multigrid method [4] than to the standard multigrid method. For example, each unknown of the linear system is uniquely associated with exactly one level. Hence, unknowns on the coarse level are not represented on the fine level and vice versa.

Preconditioning techniques based on multi-level block incomplete LU factorization (BILUM) [22,26] have recently been shown to be very effective when solving general large sparse linear systems. BILUM combines the generality of Krylov subspace methods and the robustness of the ILU factorization techniques with the scalability of multigrid methods. The tests show near-grid-independent convergence for certain type of problems [7,26]. Yet, BILUM can be constructed purely algebraically and requires no physical grid information (although such information, if available, may be used to facilitate preconditioner construction and increase robustness of the resulting preconditioner [27]). BILUM is a hybrid method that blends characteristics of multigrid methods, domain decomposition techniques, and ILU factorizations.

For problems arising from practical applications, such as those from computational fluid dynamics, the coefficient matrices are often irregularly structured, ill-conditioned, and of very large size. If the underlying PDEs are discretized by high-order finite elements on unstructured domains, the coefficient matrices may have many nonzeros in each row. These features of the coefficient matrices make the problems harder to solve and the simple strategies used in the standard BILUM technique proposed in [26] may become inefficient. First, small pivoting blocks are no longer suitable for matrices with many nonzeros in each row and blocks of large sizes are preferable. It has been shown that the size of the independent set influences the convergence rate of the preconditioned iterative solver [27] and the use of large-size blocks is advantageous. Second, the inverse of a large sparse block is a full dense matrix in general, so the amount of fill-ins in BILUM increases rapidly as the block size increases. This makes the process more complex and more costly. Third, BILUM can be viewed as a multi-level domain decomposition preconditioner based on the Schur complement approach. As such it also has the drawback that some large blocks (subdomains) may be ill-conditioned or near-singular. Standard techniques used to invert these blocks are likely to produce unstable or inaccurate LU factors and the resulting preconditioner is less efficient.

Several strategies to deal with the above problems are introduced in this paper. First, some background on BILUM and its relation to AMG are discussed in Section 2. Enhanced block preconditioning techniques are introduced in Section 3. In Section 4, robustness and efficiency of these techniques are tested using several large and hard to solve problems. Concluding remarks and comments on future research are included in Section 5.

2. BILUM and AMG

An independent set is defined as a set of unknowns which are not coupled by an equation. A block independent set (BIS) is a set of groups (blocks) of unknowns such that there is no coupling between unknowns of any two different groups (blocks) [26]. Unknowns within the same group may be coupled. The ILUM factorization defined in [22] uses blocks of size one. BILUM in [26] is constructed with blocks of small sizes, e.g., of size two or three. BIS with blocks of large sizes is considered in [25], along with a double-dropping strategy introduced to control the sparsity. Uniform block sizes have always been used in BILUM so far.

Suppose that a (block) independent set ordering has been found by one of the techniques introduced in this paper or in [22,26]. Then the original matrix can be permuted into a 2×2 block matrix of the form (with the unknowns of the independent set listed first) [1]

$$A \sim PAP^{\mathrm{T}} = \begin{pmatrix} D & F \\ E & C \end{pmatrix}, \tag{2}$$

where P is the permutation matrix associated with the BIS ordering, $D=\mathrm{diag}[\tilde{D}_1,\tilde{D}_2,\ldots,\tilde{D}_r]$ is block diagonal and C is a square matrix. The dimensions of the blocks $\tilde{D}_i$, $i=1,\ldots,r$, may not be the same.

[1] This may be viewed as a domain-based multi-level method. Other methods [19] choose to list the unknowns of the independent set last and may be viewed as grid-based multi-level methods.

In [22,26], a block LU factorization of the form

$$\begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ ED^{-1} & I \end{pmatrix} \begin{pmatrix} D & F \\ 0 & A_1 \end{pmatrix} = LU \tag{3}$$

is performed. Here

$$A_1 = C - ED^{-1}F \tag{4}$$

is the Schur complement with respect to C and I is the generic identity matrix. Since C, E, D and F are sparse matrices, A_1 is also sparse in general. In large applications, A_1 may still be too large to be solved inexpensively and further reduction may be needed. In order to maintain sparsity of the LU factors, a dropping strategy based on some given threshold tolerance may be adopted. The BILUM preconditioner is obtained by recursively applying the above procedures (finding a BIS and performing an ILU factorization) to these successive Schur complements up to a certain number of levels, say, *last*. The last reduced system obtained is then solved by a direct method or a preconditioned iterative method.

The solution process (application of BILUM) consists of block forward and backward steps [22,26,25]. At each level j, we partition the vector x_j as

$$x_j = \begin{pmatrix} y_j \\ x_{j+1} \end{pmatrix} \tag{5}$$

corresponding to the 2×2 block matrix (2) and perform the following steps:

Algorithm 2.1 (Application of BILUM Preconditioner)**.**

1. *Copy the right-hand side vector b to* x_0.
2. *For* $j = 0, 1, \ldots, \text{last} - 1$, *do forward sweep:*
3. *Apply permutation* P_j to x_j *to partition it in the form of* (5).
4. $x_{j+1} := -E_j D_j^{-1} y_j + x_{j+1}$.
5. *End do.*
6. *Solve with a relative tolerance* ε:
7. $A_{\text{last}} x_{\text{last}} := x_{\text{last}}$.
8. *For* $j = \text{last} - 1, \ldots, 1, 0$, *do backward sweep:*
9. $y_j := D_j^{-1} y_j$.

9a. $y_j := -D_j^{-1} F_j x_{j+1} + y_j$.

10. *Apply inverse permutation* P_j^{T} *to the solution* y_j.
11. *End do.*

Note that Lines 9 and 9a are written for illustration purposes and are redundant. The computation is actually performed as

9b. $y_j := D_j^{-1}(-F_j x_{j+1} + y_j)$.

The solution on the last level may not need to be exact. In [22,26], the coarsest level solution is obtained by applying several iterations of preconditioned GMRES. However, if A_{last} is close to a dense matrix and is of small dimension, a direct solver may be used. If A_{last} is nearly singular, an

approximate singular value decomposition technique such as the one introduced in Section 3.4 may be used to obtain a stabilized approximate inverse.

BILUM amounts to 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. What distinguishes BILUM from traditional domain decomposition methods [17] is that all subdomains are constructed algebraically and exploit no physical information. In addition, the reduced system is solved by a multi-level recursive process akin to a multigrid technique to avoid expensive solution related to large scales.

2.1. Relationship with algebraic multigrid methods

If we assume that A_j is symmetric and D_j is invertible, $j = 0, 1, \ldots, \text{last} - 1$, with $A_0 = A$. We may define BILUM corresponding to Algorithm 2.1 in the context of algebraic multigrid method (AMG). It is now obvious that $F_j = E_j^{\mathrm{T}}$. In order to conform to multigrid terminology, we define the prolongation operator in Lines 9a and 10 as

$$\mathbf{I}_{j+1}^{j} = P_j^{\mathrm{T}} \begin{pmatrix} -D_j^{-1} F_j \\ I_j \end{pmatrix} \tag{6}$$

and the restriction operator as the transpose of the prolongation operator, i.e.,

$$\mathbf{I}_{j}^{j+1} = \mathbf{I}_{j+1}^{j^{\mathrm{T}}}. \tag{7}$$

Eq. (7) is usually satisfied by classical geometric and algebraic multigrid methods [28]. It can be verified that the restriction operator just defined is actually equivalent to

$$\mathbf{I}_{j}^{j+1} = (-E_j D_j^{-1} \quad I_j) P_j, \tag{8}$$

as in Lines 3 and 4. The smoothing operator may be defined as D_j^{-1}, i.e., exact solve for the fine grid nodes as in Line 9. Similar to AMG, the coarse grid operator may be defined by the so-called Galerkin condition (or Galerkin coarse grid approximation) [28] as

$$A_{j+1} = \mathbf{I}_{j}^{j+1} A_j \mathbf{I}_{j+1}^{j}. \tag{9}$$

Line 9 may be viewed as applying a pre-smoothing operation on the fine grid nodes (the nodes in the independent set).[2] Hence, Algorithm 2.1 may be considered as an algebraic multigrid method with one pre-smoothing and no post-smoothing operation, i.e., a $V(1,0)$ cycle algorithm. In this sense, the smoothing operation on the whole grid is carried out by one Krylov subspace iteration.

Proposition 2.1. *Using the above assumptions and definitions, the coarse grid operator defined in* (9) *is the same as the Schur complement* (4).

[2] The definition of fine grid nodes in AMG is different from that in geometric multigrid method. The algorithm can be written in an equivalent form so that the pre-smoothing operation is performed in the forward sweep, i.e., in the first half-cycle.

Proof. Starting from (9), we have

$$\begin{aligned} A_{j+1} &= (-E_j D_j^{-1} \quad I_j) P_j A_j P_j^{\mathrm{T}} \begin{pmatrix} -D_j^{-1} F_j \\ I_j \end{pmatrix} \\ &= (-E_j D_j^{-1} \quad I_j) \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} \begin{pmatrix} -D_j^{-1} F_j \\ I_j \end{pmatrix} \\ &= C_j - E_j D_j^{-1} F_j. \qquad \square \end{aligned}$$

Note that the symmetry of A_j is not necessary for BILUM. We can still define the prolongation and restriction operators as in (6) and (8), respectively, but relation (7) is no longer valid. However, the fundamental Galerkin condition (9) and the Proposition 2.1 still hold. It is unclear in AMG what are the best inter-grid (inter-level) transfer operators. Recent advances have been reported on improving the quality and accuracy of these operators by proposing different formulas [9]. On the other hand, BILUM idea provides a coherent and natural way to define inter-level transfer operators that are suitable for general sparse matrices.

For example, if we define the prolongation operator according to Line 9b as

$$\mathbf{I}_{j+1}^{j} = P_j^{\mathrm{T}} \begin{pmatrix} -F_j \\ I_j \end{pmatrix}, \tag{10}$$

condition (7) no longer holds. But Galerkin condition (9) still holds and can be used to generate coarse-level operators. In this way, we have a post-smoothing operator D^{-1}. Note that the algebraic multigrid method we just defined is a $V(0,1)$ cycle algorithm with no presmoothing and one post-smoothing sweep. In a different way, we may combine the level by level permutation matrices as a global permutation matrix $P = P_{\mathrm{last}-1} \ldots P_1 P_0$ and only perform permutation and inverse permutation before and after the application of BILUM as in [22]. In this implementation, the definition of inter-level transfer operators is cleaner and contains no permutation matrices. These are just some of the seemingly endless possibilities, the framework of BILUM and successive Schur complements can indeed generate numerous AMG-like or BILUM-like algorithms.

2.2. BIS with large-size blocks

In [22,26], blocks of small sizes were used as pivots. Heuristics based on local optimization arguments were introduced in [26] to find 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 (also see some analyses and comments in [27]). These algorithms were devised for finding independent sets with blocks of small sizes. Extending these heuristic algorithms for extracting BIS with large-size blocks is straightforward. However, these extensions may have some undesirable consequences. First, the cost is not linear with respect to the block size and it can become prohibitive as the block size increases. (Note that many graph optimization problems are NP-hard.) 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 tends to be expensive [26].

Numerical tests in [26] also indicates that large-size blocks, which yield large independent sets, tend to result in a better BILUM preconditioner. Analytical investigation on the preconditioned errors of ILUM (BILUM with blocks of size one) shows that the Frobenius norms of the factorization and preconditioned errors are proportional to the size of the independent set [27]. Hence, both numerical and analytical results lead to the need for large independent set and the most convenient way to achieve this seems to use blocks of large size.

2.3. Performance measures

A traditional performance measure for iterative methods is the iteration counts. This measure alone is inadequate for comparing different preconditioning techniques, as it does not completely reflect the usage of resources, such as the computational and memory costs in constructing the preconditioner.

In order to describe and compare different preconditioning techniques more accurately, we use several measures defined in [25] to characterize the efficiency of a preconditioning technique. The first one is called the *efficiency ratio* (e-ratio) which is defined as the ratio of the CPU time spent on computing the preconditioner to that on computing the solution by the preconditioned solver. The efficiency ratio determines how expensive it is to compute a preconditioner, relative to the total iteration phase. The second parameter 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.

The third parameter is the *sparsity ratio* (s-ratio) which is the ratio of the number of nonzeros of the preconditioner to that of the matrix A. Note that the number of nonzeros in BILUM includes all the nonzeros of the LU factors at all levels plus the last reduced system and its preconditioner. If a direct method is used to solve the last reduced system, this latter number is to be replaced by the number of nonzero elements of the (exact) LU factorization of this system. If the sparsity ratio is large, a preconditioned iterative solver may lose one of its major advantages over a direct solver. The fourth parameter is the *reduction ratio* (r-ratio) which is the ratio of the number of the total nodes at all levels to that of the original system A. This measures the quality of the independent sets found by an algorithm. (Note that the precise definition of the reduction ratio is slightly different from what we used in [25], both represent similar meaning.) Better performance is usually associated with larger independent sets. Similar measures have been used in some of the earliest algebraic multigrid literature [20] to describe the efficiency of the algorithms, but they usually do not count the storage cost of the inter-grid transfer operators. These four performance measures are more informative than the measure provided by the iteration count alone.

3. Enhanced block preconditioning techniques

Our experience and analysis suggest that larger independent set results in a better BILUM preconditioner [26,27] and this leads to the use of large-size blocks. However, as we mentioned in the introduction, there are some problems associated with using blocks of large sizes. These problems include the potentially near-singular blocks and potentially large amount of fill-ins. Several new techniques are introduced in this section to deal with some of these problems.

3.1. Double dropping strategy

The dropping strategy used in [26,27] is to drop elements in the U factor ED^{-1} and in the reduced system A_1 whenever their absolute value is less than a threshold tolerance τ times the average nonzero value of the current row. For BILUM with large-size BIS formed by the greedy algorithm, this single dropping strategy is not enough to obtain a desired sparsity. Numerical results with the 5-POINT matrices in [25] show that the sparsity ratio is doubled when the block size increases from 1 to 15. The potentially uncontrolled large storage requirement may overflow memory in large-scale applications.

Inspired by the dual threshold dropping strategy of ILUT [23], we proposed a similar dual threshold dropping strategy for BILUM in [25]. We first apply the single-dropping strategy as above to the ED^{-1} and A_1 matrices and keep only the largest p elements (absolute value) in each row of the LU factors at each level.

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

Note that if the sparsity of ED^{-1} and A_1 is not controlled aggressively, the sparsity of D^{-1} may strongly influence the sparsity of ED^{-1} and A_1 and thus the overall sparsity of BILUM.

3.2. Blocks with variable sizes

In some cases, it may be appealing to vary the size of the blocks. It is natural to choose blocks that keep the physical coupling of neighboring nodes. For finite element discretization on unstructured meshes, different nodes may have different number of nearest neighboring nodes. If we block those nodes together, we will have blocks of different sizes in the independent set. Such blocking strategy may also result in relatively dense blocks and the relative increase in the amount of fill-ins caused by the inverse of such blocks is less severe, provided other parts of the factorization are suitably controlled. On the other hand, if the block size is not restricted, the amount of fill-ins may be very large as a result of the inverse of very large-size blocks.

Even if the coefficient matrix is regularly structured, the approximate Schur complement systems are likely to be irregularly structured due to the dropping strategies. (Most dropping strategies currently employed do not keep the structure of the original matrix.) Hence, choosing blocks with variable sizes dynamically may be a viable strategy to accommodate fill-ins as the number of levels increases. In certain cases, the last reduced system is usually very dense and is of small dimension, an exact (or approximate) solution of it may be more efficient. This is done by considering the last reduced system as a single block so an exact or approximate inverse technique may be used to seek an exact or approximate solution (without iteration).

A source of difficulty in dealing with blocks of variable sizes is the resulting programming complexity. An additional array is needed to store the size of each block and an additional pointer array is needed for the starting position of each block. The inverse of each block may be computed

using a LAPACK routine [1], using skyline blocks, or employing the approximate singular-value decomposition technique. The latter two techniques will be described next.

3.3. Skyline blocks

Let G be the set of edges of the adjacency graph of a given matrix. Given a seed node v_j, we search for all of its immediate neighboring nodes such that those neighboring nodes are mutually disconnected from each other. These nodes are grouped together to form a block with the seed node listed as the last one. More formally, let B denote the block with v_j as the seed node,

$v_k \in B$ if and only if $(v_j, v_k) \in G$ or $(v_k, v_j) \in G$,

but $(v_k, v_l) \notin G$ and $(v_l, v_k) \notin G$, $\forall v_l \in B$, $l \neq k$.

The matrix associated with such an ordering has the format [3]

$$B = \begin{pmatrix} b_{1,1} & 0 & 0 & \cdots & b_{1,s} \\ 0 & b_{2,2} & 0 & \cdots & b_{2,s} \\ 0 & 0 & \cdots & \ddots & \cdots \\ b_{s,1} & b_{s,2} & \cdots & \cdots & b_{s,s} \end{pmatrix}. \tag{11}$$

Such a matrix is called skyline matrix or arrowhead matrix. Note that the seed node must be listed as the last one in order to have a downward arrowhead. If it is listed as the first one, we will have an upward arrowhead matrix which is the worst matrix for LU factorization in terms of controlling fill-ins [13, p. 228]. In what follows a skyline or an arrowhead matrix will mean a downward arrowhead matrix.

The advantage of skyline matrices is that they can be factored without fill-in as

$$B = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & \cdots \\ b_{s,1}/b_{1,1} & b_{s,2}/b_{2,2} & \cdots & \cdots & 1 \end{pmatrix} \begin{pmatrix} b_{1,1} & 0 & 0 & \cdots & b_{1,s} \\ 0 & b_{2,2} & 0 & \cdots & b_{2,s} \\ 0 & 0 & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & \cdots & c_{s,s} \end{pmatrix} = LU,$$

where

$$c_{s,s} = b_{s,s} - \sum_{i=1}^{s-1} \frac{b_{i,s}}{b_{i,i}} b_{s,i}.$$

The inverse of B can be computed as

$$B^{-1} = U^{-1}L^{-1} = \begin{pmatrix} 1/b_{1,1} & 0 & 0 & \cdots & -b_{1,s}/(b_{1,1}c_{s,s}) \\ 0 & 1/b_{2,2} & 0 & \cdots & -b_{2,s}/(b_{2,2}c_{s,s}) \\ 0 & 0 & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & \cdots & 1/c_{s,s} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & \ddots & \cdots \\ -b_{s,1}/b_{1,1} & -b_{s,2}/b_{2,2} & \cdots & \cdots & 1 \end{pmatrix}.$$

It is not necessary to store the unit diagonal of L^{-1} and thus the storage space of B^{-1} is the same as that of B.

[3] We assume that the matrix B is structurally symmetric. Zero elements may be filled if structural unsymmetry occurs.

The potential saving of storage space by using skyline block format could be substantial for large-size blocks. Suppose a block is of order s, the storage space of the skyline block format is $3s-2$, while that for the direct inverse is s^2.

In general, the dimension of each block is different and we have to deal with the issue of implementing BIS with variable-size blocks. In order to have a stable inverse, we may also restrict the magnitude of the diagonal entries by a threshold tolerance as we did in [27], or perturb the small diagonal values as in the SVD technique. Given a threshold tolerance ε, we need to impose the restriction that $|b_{i,i}| \geqslant \varepsilon$ for all $1 \leqslant i \leqslant s$. If $|b_{i,i}| < \varepsilon$, we replace $|b_{i,i}|$ by ε.

For many practical problems, it is not easy to guarantee that all the neighboring nodes of the seed node are mutually disconnected. This is satisfied, for example, for the 5-point matrices arising from the second-order finite difference discretization of the Laplace operator. It is not satisfied for the 9-point matrices arising from the fourth-order compact discretization of the same operator. In the latter case, we may drop the links between the neighboring nodes to obtain a skyline matrix $\tilde{B}$ of form (11) which will approximate the original block B. The success of such a dropping strategy will depend on the diagonal dominance of the block B.

There is another implementation issue. Since D^{-1} is stored as a factored inverse, it would be advantageous not to compute the L factor ED^{-1}, but to store it as two sparse matrices. However, the approximate Schur complement must be computed and sparsified.

3.4. Singular-value decomposition

Each block submatrix must be inverted in the factorization process. However, it may sometimes happen that a block is singular or nearly singular. This is not uncommon, for example, in traditional domain decomposition approaches [17]. Direct inversion of these near-singular matrices leads to large elements in the inverse and the resulting preconditioner is less efficient. A common strategy to mitigate this problem in developing preconditioners is to perturb the singular-value decomposition (SVD) of the block submatrix [10]. In other words, suppose the SVD of the matrix B is

$$B = U\Sigma V^{\mathrm{T}}, \tag{12}$$

where U and V are two orthogonal matrices and $\Sigma = \mathrm{diag}[\sigma_1, \sigma_2, \ldots, \sigma_s]$, with

$$\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_s \geqslant 0.$$

See Theorem 2.3-1 of Golub and van Loan [15, pp. 16,17] for details on the SVD factorization. It is well known that $\|B\|_2 = \sigma_1$. If B is ill-conditioned or near-singular, some of its small singular values are close to zero. The inverse of B can be expressed as

$$B^{-1} = V\Sigma^{-1}U^{\mathrm{T}},$$

with $\Sigma^{-1} = \mathrm{diag}[\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_s^{-1}]$. Thus, some elements of Σ^{-1} may be very large if some singular values of B are small.

Standard methods for properly 'inverting' a singular or nearly singular matrix are referred to as 'regularization' [6]. The most common method used is to add a constant to all singular values in order to move the smaller ones away from zero. This is referred to as Tychonov regularization [6, p. 101] and is mathematically equivalent to solving a damped least-squares problem with the matrix B. We use a variation of this regularization approach which consists of perturbing only the smallest

singular values. A similar strategy was advocated in [10]. We replace the smallest singular values of Σ by larger values. Specifically, given a threshold parameter $\omega > 0$, the singular values σ_i such that $\sigma_i < \omega$ are replaced by $\omega + \sigma_i$. This may be done dynamically with a threshold strategy.

Hence, the matrix B is perturbed as

$$\tilde{B} = U\tilde{\Sigma}V^{\mathrm{T}},$$

where

$$\tilde{\Sigma} = \mathrm{diag}[\sigma_1, \sigma_2, \ldots, \sigma_{q-1}, \omega + \sigma_q, \ldots, \omega + \sigma_s].$$

The inverse matrix B^{-1} is correspondingly replaced by

$$\tilde{B}^{-1} = V\tilde{\Sigma}^{-1}U^{\mathrm{T}}, \tag{13}$$

where

$$\tilde{\Sigma}^{-1} = \mathrm{diag}[\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_{q-1}^{-1}, 1/(\omega + \sigma_q), \ldots, 1/(\omega + \sigma_s)].$$

Note that the 2-norm condition number of the matrix B is

$$\kappa_2(B) = \|B\|_2\|B^{-1}\|_2 = \sigma_1\sigma_s^{-1},$$

while the 2-norm condition number of $\tilde{B}$ is

$$\kappa_2(\tilde{B}) = \|\tilde{B}\|_2\|\tilde{B}^{-1}\|_2 = \sigma_1 \max\left\{\frac{1}{\sigma_{q-1}}, \frac{1}{\omega + \sigma_s}\right\}.$$

However, the perturbation from B to $\tilde{B}$ introduces errors in the factorization. The 2-norm of the error resulting from this perturbation is governed by the threshold parameter ω and the largest and the smallest singular values σ_1 and σ_s (or σ_{q-1}). The following property is easy to prove.

Proposition 3.1. *Let B be a square matrix with the singular-value decomposition* (12) *and $\tilde{B}^{-1}$ be a square matrix with the singular-value decomposition* (13). *Then $I - B\tilde{B}^{-1}$ is of rank $n - q$ and*

$$\|I - B\tilde{B}^{-1}\|_2 = \begin{cases} 0 & \text{if } \sigma_s \geqslant \omega, \\ \dfrac{\omega}{\omega + \sigma_s} & \text{otherwise.} \end{cases}$$

It may be difficult to select an appropriate ω. Note that any ω such that $0 \leqslant \omega \leqslant \sigma_s$ will yield the exact inverse and in this case we will have $q = n$. A large value of ω will result in an inaccurate approximate inverse. Since the preconditioner is not constructed accurately, due to dropping, a highly accurate inverse will not help. Our numerical experiments suggest that, although optimal ω seems intractable, a suitable value usually can be guessed. Since the SVD technique is costly and inaccurate, it should only be used when it is known in advance that near-singular blocks may arise such as when solving certain indefinite matrices. The SVD technique may be useful when other means fail. In our experiments it enabled us to solve some problems that were otherwise very hard to solve, as will be seen in the next section.

4. Numerical experiments

Additional experiments with BILUM and some implementation details, specifically with small-size blocks and some dropping strategies, have been reported in [22,26,25]. In those papers, the multi-level preconditioned FGMRES is also compared with its single-level counterpart in terms of performance efficiency and storage cost. We used FGMRES(10) [21] as an accelerator for both the inner and outer iterations. The outer iteration process was preconditioned by BILUM with a dropping strategy. The inner iteration process to solve the last reduced system approximately was preconditioned by ILUT(τ, p) [23]. The construction and application of the BILUM preconditioner was similar to those described in [25]. The right-hand side was generated by assuming that the solution is a vector of all ones and the initial guess was random numbers. In all cases, 10 levels of reduction were performed. The inner iteration was stopped when the 2-norm of the residual was reduced by a factor of 10^2 or the number of iterations exceeded 10, whichever occurred first. The outer iteration was terminated when the residual in 2-norm was reduced by a factor of 10^7. 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 arithmetic.

4.1. Experiments with dropping strategies

Experiments with large-size blocks and double-dropping strategy for solving several large and hard-to-solve matrices have been reported in [25]. It has been shown that BIS with large-size blocks is essential for solving some problems and the double-dropping strategy is useful in controlling the amount of fill-ins. We report tests with the block sparsification strategy for solving the 9-POINT and VENKAT01 matrices here. The 9-POINT matrix was first used in [26] and is from a 9-point fourth-order compact finite difference discretization of a convection–diffusion equation with a Reynolds number 10^4 [16]. It has $40,000$ unknowns and $357,604$ nonzeros.

For the 9-POINT matrix, we used a single-dropping strategy with $\tau = 0.01$ and 0.1, and blocks of uniform size $s = 20$. The title $\tilde{p}/s$ indicates the number of elements kept in each row of the blocks. The results are shown in Table 1. We see that suitable sparsification of blocks does not hamper the convergence rate but it reduces the amount of fill-in (smaller s-ratio). The results are more striking when $\tau = 0.1$ was used. In this case, keeping more elements were not helpful at all. However, dropping too many elements is not advisable since the gain in the sparsity ratio does not offset the big loss in convergence.

The VENKAT01 matrix was from an unstructured 2D Euler solver at time step 1. It was provided by V. Venkatakrishnan from NASA. The matrix has $62,424$ unknowns and $1,717,792$ nonzeros. We used a double-dropping strategy with $p = 20$ and $\tau = 0.001$, and tested two different uniform block sizes $s = 40$ and 80. The results are given in Table 2. We see that dropping half of the elements in each row of the blocks did not affect the convergence rate significantly. The CPU time and sparsity ratio were reduced. We also see that when the block size was doubled from 40 to 80, the sparsity ratio (without block sparsification) increased substantially. This shows the main drawback of using large-size blocks.

Our conclusion is that the block sparsification strategy does help reduce the amount of overall fill-in. However, the dropping strategy for the blocks should be related to the dropping strategies used

Table 1
Test results with block sparsification and single dropping for the 9-POINT matrix

$s = 20, \tau = 0.01, p = 20$						$s = 20, \tau = 0.1, p = 20$					
$\tilde{p}/s$	Iter	cpu	e-ratio	s-ratio	r-ratio	$\tilde{p}/s$	Iter	cpu	e-ratio	s-ratio	r-ratio
20	9	31.68	2.99	5.96	2.19	20	93	88.52	0.23	4.14	1.89
19	9	31.69	3.03	5.82	2.18	19	183	158.2	0.12	4.02	1.89
18	9	31.01	2.97	5.61	2.14	18	69	69.84	0.32	3.90	1.89
17	11	32.69	2.48	5.47	2.18	17	79	76.83	0.28	3.76	1.90
16	19	39.17	1.44	5.32	2.18	16	66	66.28	0.33	3.61	1.89
15	22	40.77	1.23	5.08	2.14	15	70	69.14	0.31	3.46	1.88
14	49	62.33	0.55	4.86	2.14	14	117	103.9	0.19	3.30	1.86
13	105	105.6	0.26	4.59	2.12	13	156	131.0	0.14	3.13	1.85

Table 2
Test results with block sparsification and double dropping for the VENKAT01 matrix

$s = 40, \tau = 0.001, p = 20$						$s = 80, \tau = 0.001, p = 20$					
$\tilde{p}/s$	Iter	cpu	e-ratio	s-ratio	r-ratio	$\tilde{p}/s$	Iter	cpu	e-ratio	s-ratio	r-ratio
40	17	80.65	1.91	2.67	2.44	80	17	124.7	3.00	3.91	2.27
30	17	79.05	2.06	2.32	2.46	60	17	121.3	3.36	3.19	2.26
20	18	77.80	1.98	1.94	2.42	40	18	118.9	3.33	2.48	2.26
10	23	80.90	1.55	1.49	2.40	30	19	118.7	3.23	2.12	2.26
8	24	80.74	1.48	1.37	2.39	20	21	119.5	3.01	1.76	2.28
6	27	83.69	1.27	1.22	2.35	10	27	122.9	2.40	1.31	2.23
4	32	87.60	1.06	1.04	2.31	6	31	127.0	2.08	1.07	2.21
2	44	99.24	0.76	0.80	2.26	3	45	139.3	1.48	0.82	2.15
1	63	119.3	0.53	0.63	2.16	1	63	160.2	1.02	0.57	2.08

to control other parts of the LU factorization and to set an overall dropping level that is consistent throughout the construction of the BILUM factors. On the other hand, some approximate inverse techniques [10,29,30] may be used to compute sparse approximate inverses of the large-size blocks.

4.2. Approximate SVD technique

We run tests with two matrices from the FIDAP collection [4] using the approximate SVD technique. These matrices were extracted from the test problems provided in the FIDAP package [12]. They model the incompressible Navier–Stokes equations. Several of these matrices contain small or zero diagonal values [11] and have a block structure of the form

$$\begin{pmatrix} A & B \\ C & 0 \end{pmatrix},$$

[4] All FIDAP matrices are available online from the MatrixMarket (`http://math.nist.gov/MatrixMarket`) of the National Institute of Standards and Technology.

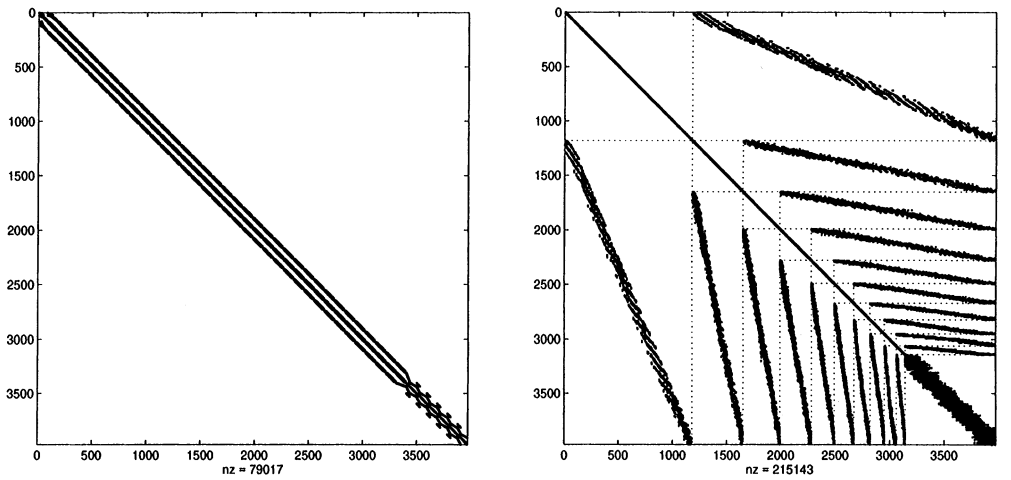

Fig. 1. Nonzero patterns of the original FIDAP012 matrix (left) and the processed matrix (right) using double-dropping strategy.

where 0 is a zero block. The zero diagonals are due to the incompressibility condition of the Navier–Stokes equations [11]. The substantial amount of zero diagonals makes these matrices indefinite. Standard ILU preconditioners may fail to converge for solving these matrices unless a small τ and a large p are used. In the case of ILUM (BILUM with block size 1), the diagonal threshold technique introduced in [27] seems to offer a simple yet effective way of dealing with such indefinite matrices.

The FIDAP012 matrix is from a model of flow in lid-driven wedge. Uniform blocks of size $s=10$ were used. The FIDAP036 matrix is a problem of modeling chemical vapor deposition. Uniform blocks of size $s=20$ were used. We tested several dropping parameter τ and the SVD threshold parameter ω $(0.01, 0.001, 0.0001, 0.00001)$. Fig. 1 shows the nonzero patterns of FIDAP012 and the processed matrix using double-dropping strategies. (The zero diagonals are kept since many ILU-type techniques including ILUT and BILUM cannot deal with matrices with structurally zero diagonals.)

SVD with single dropping: The single dropping and the SVD technique were tested with different parameters for solving the FIDAP012 matrix. The last reduced systems were solved by FGMRES preconditioned by ILUT$(\tau, 20)$. The convergence results are presented in Fig. 2. It is found that the SVD parameter ω affected the convergence rate of this problem significantly. $\omega=0.0001$ seems to yield the best result for $\tau=0.001$ and 0.0001. Larger or smaller values of ω deteriorated convergence rate markedly. On the other hand, too large and too small τ also made the convergence rate worse. The optimal ω is problem dependent, but $\omega=\tau=0.0001$ seems to be a good choice for many problems.

SVD with double dropping: We tested FIDAP036 matrix for the SVD technique with the double-dropping strategy. We kept at most 30 elements in each row of the L and U factors on all levels. For the rest tests in this paper, the last reduced systems were solved by FGMRES preconditioned by ILUT$(\tau, 30)$. Fig. 3 shows the results. It can be seen that choosing $\omega=0.0001$ yielded the best convergence rate almost in all cases, the second best was $\omega=0.001$.

SVD with block sparsification: Our next test is to see how the SVD technique works with the double-dropping strategy and the block sparsification strategy. For FIDAP036, we only kept 10

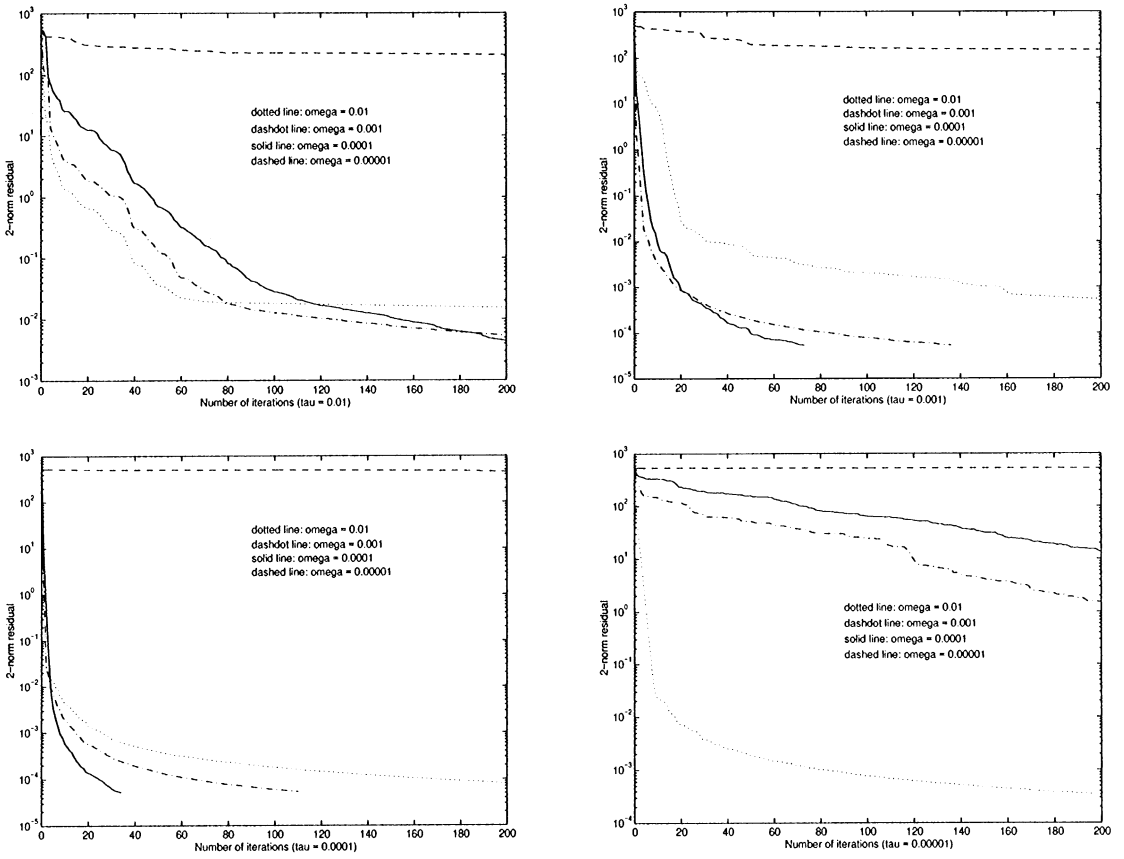

Fig. 2. Convergence behavior of BILUM using different dropping parameter τ and SVD parameter ω for solving the FIDAP012 matrix with single-dropping rule.

elements (out of 20) with the largest absolute values in each row of the inverse of each block of the BIS. The results are shown in Fig. 4. Note that, because of the aggressive dropping, convergence is usually slower than the best results in Fig. 3. In the current test conditions, the best results were achieved with $\omega = 0.001$ and the parameter τ did not make a significant difference.

From the above tests, we conclude that the SVD technique is useful in dealing with those indefinite matrices with small or zero diagonals. Aggressive control of sparsity usually impedes convergence. However, the effectiveness of the various dropping strategies cannot be appreciated without showing how much storage space is saved. Thus we show in Table 3 the performance measures and the iteration counts for some selected test parameters with the FIDAP036 matrix. We point out that even though the double-dropping strategy usually yielded a worse convergence rate (iteration counts) than the single-dropping strategy, it required much less CPU time and storage space. In other words, BILUM with the double-dropping strategy is a more efficient implementation for this test problem. Additional block sparsification reduced storage costs further, but also reduced convergence rate.

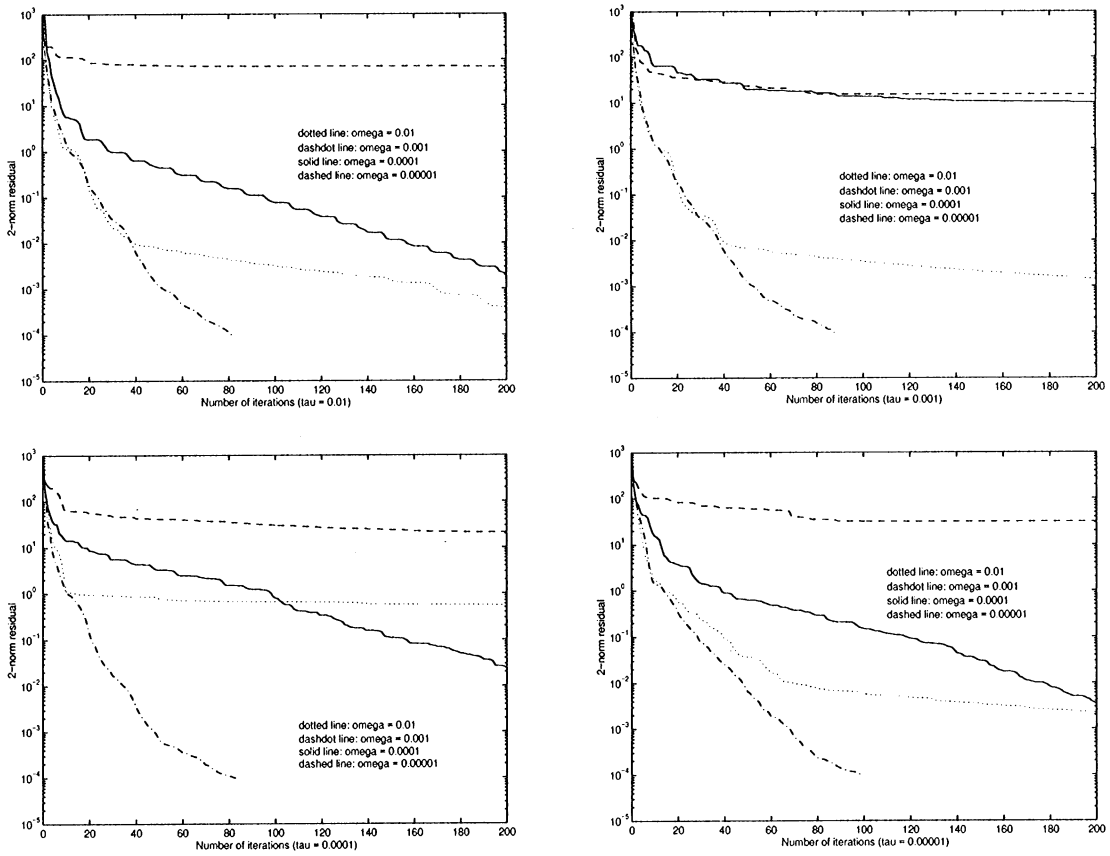

Fig. 3. Convergence behavior of BILUM using different dropping parameter τ and different SVD parameter ω for solving the FIDAP036 matrix with double-dropping rule.

Table 3
Characteristic measures with SVD parameter $\omega = 0.001$ for FIDAP036 matrix

	$s = 20, \tau = 0.001, p = 30$					$s = 20, \tau = 0.0001, p = 30$				
Strategy	Iter	cpu	e-ratio	s-ratio	r-ratio	Iter	cpu	e-ratio	s-ratio	r-ratio
S. Drop	28	6.79	2.72	7.50	3.57	34	9.90	2.51	9.02	3.76
D. Drop	43	3.96	1.01	4.81	3.32	50	4.23	0.95	4.83	3.15
Block S.	88	5.23	0.55	3.65	3.27	83	5.08	0.60	3.72	3.16

Recall that the value of ω determines the condition of the blocks and the accuracy of the resulting factorization. These two factors are in conflict. The ω which provides the best trade-off between them is difficult to obtain. As a rule, the SVD technique should only be used as a last resort. Without the SVD threshold technique (or equivalently by setting $\omega = 0$) BILUM did not converge for these two

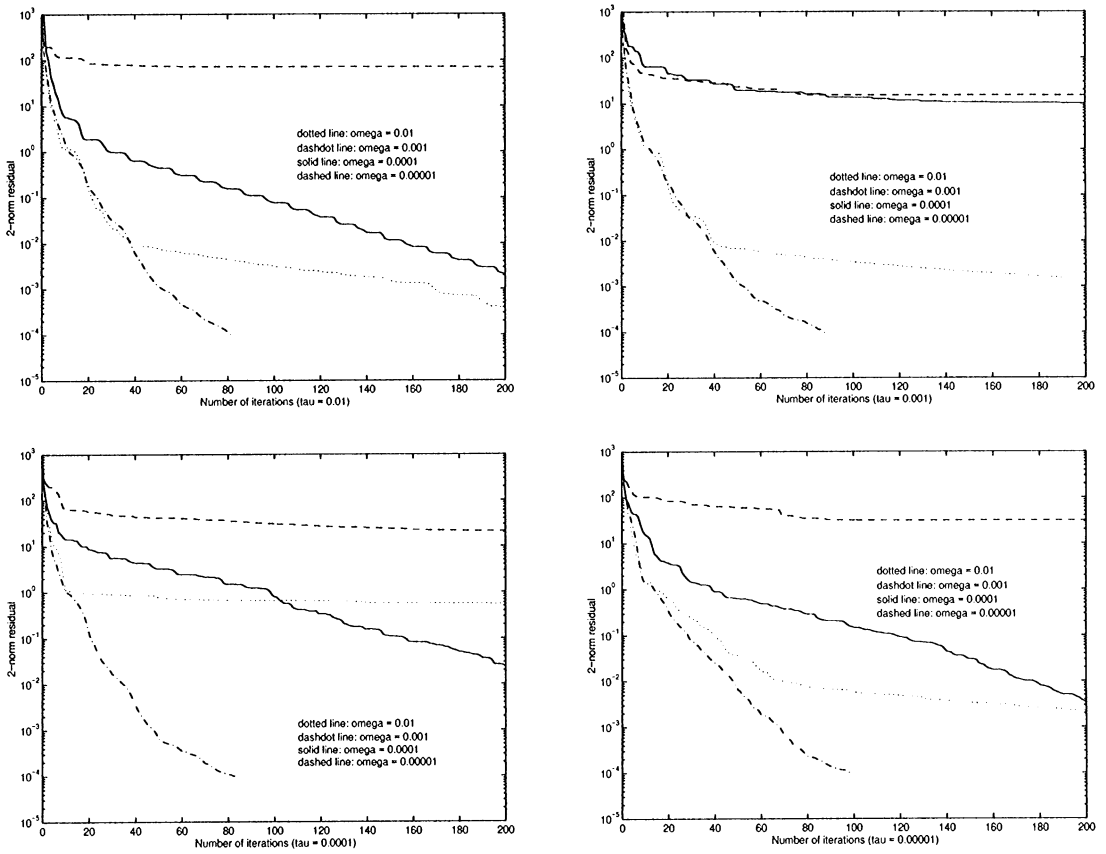

Fig. 4. Convergence behavior of BILUM using different dropping parameter τ and SVD parameter ω for solving the FIDAP036 matrix with double-dropping rule and block sparsification.

matrices. Even the construction of the BILUM factors failed in some cases, due to the occurrence of some very ill-conditioned blocks.

4.3. Variable-size blocks

We tested BILUM with variable-size blocks for some of the largest FIDAP matrices. To guarantee a stable construction of BILUM, the approximate SVD technique with some values of ω and the single-dropping strategy with a fixed $\tau = 10^{-5}$ were also used. No attempt was made to find the optimal ω. For the variable-size block implementation, we limited the maximum block size to $s=200$. The maximum number of blocks at each level was also limited to 2000. The test results are given in Table 4. We point out as reference that BILUM with some uniform-size blocks did not converge for several FIDAP matrices tested.

Generally speaking, Table 4 shows our best results so far for solving the FIDAP matrices from the point of view of successful convergence. Some undesirable features of the results are the larger values of the efficiency ratio and the sparsity ratio for some matrices. These large values indicate that,

Table 4
Test results with variable-size blocks and SVD techniques for solving FIDAP matrices

Matrices	Unknowns	Nonzeros	SVD (ω)	Iter	cpu	e-ratio	s-ratio	r-ratio
FIDAP008	3096	106302	10^{-6}	3	16.9	33.1	9.77	4.42
FIDAP009	3363	99397	5.00	12	14.0	7.23	8.78	3.53
FIDAP010	2410	54816	1.00	5	2.64	10.4	4.45	2.86
FIDAP012	3973	80151	10^{-5}	2	27.3	62.4	16.9	3.83
FIDAP013	2568	75628	10^{-5}	10	6.27	7.25	5.98	3.14
FIDAP015	6867	96421	1.00	2	11.0	27.1	8.40	2.61
FIDAP018	5773	69335	1.00	3	9.26	21.3	8.85	2.52
FIDAP019	12005	259863	3.00	5	27.2	11.9	6.29	3.09
FIDAP020	2203	69579	10^{-6}	5	7.44	16.7	7.40	3.12
FIDAP024	2283	48733	10^{-6}	2	7.91	44.4	10.7	3.11
FIDAP028	2603	77653	10^{-6}	2	10.5	48.9	8.10	2.90
FIDAP029	2870	23754	10^{-6}	1	1.10	24.8	5.04	2.41
FIDAP035	19716	218308	10^{-1}	2	32.8	55.6	8.87	2.60
FIDAP036	3079	53851	10^{-3}	3	9.73	31.3	10.8	3.30
FIDAP037	3565	67591	10^{-3}	2	7.20	41.3	6.53	2.48
FIDAPM07	2065	53533	10^{-5}	4	13.2	30.7	12.7	4.44
FIDAPM08	3876	103076	10^{-5}	3	35.1	46.5	15.5	4.65
FIDAPM10	3046	53842	10^{-7}	2	8.63	44.5	10.3	3.11
FIDAPM13	3549	71975	10^{-7}	3	19.6	38.3	14.7	3.39
FIDAPM29	13668	186294	10^{-5}	2	50.2	46.4	12.8	2.99

for these test implementations, the preconditioned iterative solver spent most of the time computing the preconditioners and the amounts of fill-ins were large. These less-desirable side effects were due to inverting some very large-size blocks in the factorization process. The computation of the SVD approximations is time-consuming for large-size blocks. Furthermore, since we did not sparsify the large blocks, it is not surprising that the amounts of fill-in were large. These drawbacks may be fixed to some extent by employing more aggressive dropping strategy and block sparsification techniques as was discussed in this paper.

5. Concluding remarks

We discussed several new strategies to deal with the problems associated with using blocks of large sizes in forming block independent set during the construction of multi-level block ILU factorization (BILUM) preconditioner. The multi-level domain-decomposition-type algorithm (BILUM) for solving general sparse linear systems is based on a recursive application of Schur complement techniques using large subdomains. We proposed a dual-dropping strategy, block sparsification, variable-size blocks, and skyline blocks, to enhance sparsity in the BILUM factorization. An approximate singular-value decomposition technique is suggested to treat ill-conditioned blocks. Several parameters were introduced to characterize the efficiency of a preconditioner. Numerical results showed that the proposed strategies work well for reducing the storage cost and stabilizing the factorization of BILUM with large-size blocks.

We also showed the connection between BILUM and the algebraic multigrid method. We showed that the reduced system of BILUM is the same as the coarse grid operator that could be generated by using Galerkin coarse grid approximation. We showed that BILUM idea could generate a new class of multi-level algorithms that define the inter-level transfer operators based on the matrix instead of heuristic formulas.

Different techniques introduced in this paper are aimed at dealing with different problems that may be encountered in constructing BILUM preconditioner. Their efficient use may improve the robustness and efficiency of standard BILUM technique, especially when extracting parallelism associated with large-size blocks.

Our numerical experiments were primarily done with the approximate SVD technique with several sparsification techniques. We have shown that the SVD technique is very useful for solving those indefinite matrices when BILUM was used with large-size blocks, especially with aggressive sparsification. We demonstrated the usefulness of constructing BIS with variable-size blocks to potentially capture the underlying physical information. We also experienced usefulness of the defined performance measures.

Each technique introduced in this paper deserves more detailed tests and their efficient implementation may not be trivial.[5] The combination of these techniques for constructing efficient stable BILUM preconditioner is certainly worth investigating.

The blocking strategies used in this paper do not consider the values of the matrix and thus are blind to physical information that may be contained in the matrix. Those used in [26] extract full information and are expensive to implement for large-size blocks. A middle ground would be to use limited information on the values of the matrix, such as the diagonal values as we did for ILUM in [27]. This may offer an informative yet inexpensive way to construct robust and efficient preconditioners.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide, 2nd Edition, SIAM, Philadelphia, PA, 1995.
[2] O. Axelsson, M. Neytcheva, Algebraic multilevel iteration method for Stieltjes matrices, Numer. Linear Algebra Appl. 1 (3) (1994) 216–236.
[3] O. Axelsson, P.S. Vassilevski, Algebraic multilevel preconditioning methods II, SIAM J. Numer. Anal. 27 (6) (1990) 1569–1590.
[4] R.E. Bank, T.F. Dupont, H. Yserentant, The hierarchical basis multigrid method, Numer. Math. 52 (1988) 427–458.
[5] R.E. Bank, C. Wagner, Multilevel ILU decomposition, Technical Report, Department of Mathematics, University of California at San Diego, La Jolla, CA, 1997.
[6] A. Bjork, Numerical Methods for Least-Squares Problems, SIAM Publications, Philadelphia, PA, 1996.
[7] E.F.F. Botta, F.W. Wubs, MRILU: an effective algebraic multi-level ILU-preconditioner for sparse matrices, SIAM J. Matrix Anal. Appl. 20 (1999) 1007–1026.
[8] T.F. Chan, V. Eijkhout, ParPre: a parallel preconditioners package reference manual for version 2.0.17, Technical Report CAM 97-24, Department of Mathematics, UCLA, Los Angeles, CA, 1997.
[9] Q.S. Chang, Y.S. Wong, H.Q. Fu, On the algebraic multigrid method, J. Comput. Phys. 125 (1996) 279–292.

[5] We also experimented with many other test matrices and tested different SVD threshold strategies and combinations of dropping strategies. For reasons of space, those results are not reported here, but they support the conclusions made in this paper.

[10] E. Chow, Y. Saad, Approximate inverse techniques for block-partitioned matrices, SIAM J. Sci. Comput. 18 (1997) 1657–1675.
[11] E. Chow, Y. Saad, Experimental study of ILU preconditioners for indefinite matrices, J. Comput. Appl. Math. 86 (2) (1997) 387–414.
[12] M. Engelman, FIDAP: Examples Manual, Revision 6.0, Technical Report, Fluid Dynamics International, Evanston, IL, 1991.
[13] G.H. Golub, J.M. Ortega, Scientific Computing: An Introduction with Parallel Computing, Academic Press, Boston, MA, 1993.
[14] G.H. Golub, H.A. van der Vorst, Closer to the solution: iterative linear solvers, in: I.S. Duff, G.A. Watson (Eds.), The State of the Art in Numerical Analysis, Clarendon Press, Oxford, 1997, pp. 63–92.
[15] G.H. Golub, C.F. van Loan, Matrix Computations, The Johns Hopkins University Press, Baltimore, MD, 1983.
[16] M.M. Gupta, R.P. Manohar, J.W. Stephenson, A single cell high order scheme for the convection-diffusion equation with variable coefficients, Int. J. Numer. Methods Fluids 4 (1984) 614–615.
[17] J. Mandel, Balancing domain decomposition, Commun. Appl. Numer. Methods 9 (1993) 233–241.
[18] Y. Notay, Z. Ould Amar, Incomplete factorization preconditioning may lead to multigrid like speed of convergence, in: A.S. Alekseev, N.S. Bakhvalov (Eds.), Advanced Mathematics: Computation and Applications, NCC Publisher, Novosibirsk, Russia, 1996, pp. 435–446.
[19] A.A. Reusken, New block. Approximate cyclic reduction preconditioning, Technical Report RANA 97-02, Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands, 1997.
[20] J.W. Ruge, K. Stüben, Algebraic multigrid, in: S. McCormick (Eds.), Multigrid Methods, Frontiers in Appl. Math., SIAM, Philadelphia, PA, 1987, pp. 73–130 (Chapter 4).
[21] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM J. Sci. Statist. Comput. 14 (2) (1993) 461–469.
[22] Y. Saad, ILUM: a multi-elimination ILU preconditioner for general sparse matrices, SIAM J. Sci. Comput. 17 (4) (1996) 830–847.
[23] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing, New York, 1996.
[24] Y. Saad, 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.
[25] Y. Saad, M. Sosonkina, J. Zhang, Domain decomposition and multi-level type techniques for general sparse linear systems, in: J. Mandel, C. Farhat, X.-C. Cai (Eds.), Domain Decomposition Methods 10, Contemporary Mathematics, Vol. 218, AMS, Providence, RI, 1998, pp. 174–190.
[26] Y. Saad, J. Zhang, BILUM: block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems, SIAM J. Sci. Comput. 20 (1999) 2103–2121.
[27] Y. Saad, J. Zhang, Diagonal threshold techniques in robust multi-level ILU preconditioners for general sparse linear systems, Numer. Linear Algebra Appl. 6 (1999) 257–280.
[28] P. Wesseling, An Introduction to Multigrid Methods, Wiley, Chichester, 1992.
[29] 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.
[30] 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.