# BILUM: Block Versions of Multi-Elimination and Multi-Level ILU Preconditioner for General Sparse Linear Systems *

Yousef Saad† and Jun Zhang‡

Department of Computer Science and Engineering,
University of Minnesota,
200 Union Street S.E., Minneapolis, MN 55455

August 27, 1997

**Abstract**

We introduce block versions of the multi-elimination incomplete LU (ILUM) factorization preconditioning technique for solving general sparse unstructured linear systems. These preconditioners have a multi-level structure and exhibit properties that are typically enjoyed by multigrid methods. Several heuristic strategies for forming blocks of independent set are introduced and their relative merits are discussed. Advantages of block ILUM over point ILUM include increased robustness and efficiency. We compare several versions of the block ILUM, point ILUM and the dual-threshold-based ILUT preconditioners. In particular, the ILUM preconditioned Krylov subspace solver is tested for some convection-diffusion problems to show convergence that is near Reynolds number independent and near grid independent.

**Key words**: Incomplete LU factorization, ILUM, multi-level preconditioner, GMRES, multi-elimination incomplete LU factorization.
**AMS subject classifications**: 65F10, 65N06.

## 1 Introduction

In this paper, we consider the problem of developing efficient preconditioners for solving general large sparse linear systems of the form

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

*This work was supported in part by ARPA under grant number NIST 60NANB2D1272, in part by NSF under grant CCR-9618827, and in part by the Minnesota Supercomputer Institute.

†e-mail:saad@cs.umn.edu.

‡e-mail:jzhang@cs.umn.edu.

where $A$ is an $n \times n$ matrix that is assumed to be unstructured. Efficient iterative solvers for solving such large problems consist of a combination of an accelerator and a good preconditioner [35]. It is generally recognized that the key to solving a general sparse linear system efficiently by iterative methods is a high quality preconditioner. With a suitable preconditioner, the accelerator plays a secondary role.

The incomplete LU (ILU) factorization with no fill-in, or ILU(0) [26], is one of the best known preconditioners. A disadvantage of ILU(0) is that it is a rather crude approximation of $A$ and therefore it is unreliable when used to solve large problems arising from certain applications, such as computational fluid dynamics. To improve the efficiency and robustness of ILU factorizations, many alternatives which allow higher amounts of fill-in have been developed [10, 11, 21, 27, 37, 41].

The implementation of ILU(0) and other ILU preconditioners on high performance computers can be optimized by a technique called "level scheduling" or "wavefront ordering" [1]. However, the parallelism that can be extracted by level-scheduling techniques is limited. The more accurate ILU factorizations are generally more robust than ILU(0) but their level of parallelism is generally worse. Different alternatives have been considered in the past to improve the degree of parallelism, see for example [35, 36, 30] for references. A particularly interesting group of methods in this category introduce parallelism by exploiting "graph coloring," or multicoloring [24, 33]. The unknowns of the problem are colored in such a way that no two unknowns of the same color are coupled by an equation. When the unknowns of the same color are numbered consecutively, then a large degree of parallelism is usually obtained in both the preprocessing and the preconditioning operations of the associated ILU preconditioning. One notable drawback of this approach is that the efficiency of the preconditioning deteriorates and as a result the number of iterations to achieve convergence may increase substantially, when compared with that required for the original system [16, 18, 19]. Recently, a technique based on exploiting the idea of successive independent sets (a simple form of multicoloring) [25] has been used to develop preconditioners that are akin to multi-level preconditioners [33, 34]. This technique called ILU with Multi-elimination (ILUM) is somewhat related to multifrontal elimination, a standard method used in the parallel solution of sparse linear systems by direct methods [13, 14].

Block versions of ILUM preconditioner are attractive because block ILU preconditioners usually perform better than their point counterparts. In particular, the point ILUM factorization may have difficulties when the diagonal elements of the resulting $U$ factor are small. The problem is not as severe in a block version because of the additional freedom in selecting blocks. Block approaches are also preferred when the matrix has natural blocks that are inherited from the underlying physical problem. In this paper, we introduce some block versions of ILUM, which we call BILUM, to distinguish it to from the already discussed point ILUM preconditioner, or ILUM. BILUM is aimed at alleviating the above shortcomings of ILUM.

Block versions of ILUM have already been suggested (but not implemented) along with point ILUM in [34]. The idea of using pre-selected blocks to deal with convection-diffusion problems, such as the line relaxation methods, has been known for many years.

The use of pre-selected small blocks for diagonal pivoting method was introduced in [6] and the definition of a pivot was extended to $2 \times 2$ blocks when choosing a $1 \times 1$ pivot on the diagonal is no longer stable due to large off-diagonal elements. Such techniques were incorporated in ILU-type preconditioners and tested by Botta and Wubs [5]. In their approach for solving convection-diffusion equations, they select block pivots (defined from pairs of indices) on the basis of the dominant flow direction (similar to the well-known line iteration methods). These pairs are maintained during the whole process. In other words, the blocks are pre-determined by using physical information before the construction of the preconditioner. The use of small blocks constructed dynamically for general sparse matrices does not seem to have been investigated.

In Section 2 some background on ILUM is given. In Section 3, we discuss several heuristic techniques to form independent sets with $k \times k$ blocks. A detailed discussion is given for $2 \times 2$ blocks, with a straightforward extension to larger blocks. Section 4 briefly introduces the construction of ILUM preconditioners. Section 5 is a study of the properties of block independent sets. Extensive numerical experiments with different variants of ILUM and the existing ILUT preconditioners, including comparisons of various blocking strategies, are included in Section 6. Concluding remarks are given in Section 7.

## 2 Background and ILUM

ILUM relies on the fact that many rows can be eliminated simultaneously at a given stage of Gaussian elimination, a consequence of sparsity. A set consisting of such rows is called *an independent set*. This is a set of unknowns that are not coupled by an equation. A *Maximal Independent Set* (MIS) is an independent set which cannot be augmented by one or more elements. These concepts are best described using graph terminology. Let $G = (V, E)$ denote the adjacency graph of the matrix $A$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E$ is the set of edges, and let $(v_i, v_j)$ denote an edge from vertex $v_i$ to $v_j$.

**Definition 2.1** *A vertex independent set $S$ is a subset of the vertex set $V$ such that*

$$\forall\, v_i \in S, \quad \forall\, v_j \in S, \quad (v_i, v_j) \notin E$$

*An independent set $S$ is maximal if there is no independent set which strictly includes $S$, i.e., for any $v \in V$, $S \cup \{v\}$ is not an independent set.*

In the remainder of the paper, we will often use the term independent set to mean *Maximal Independent Set*. The unknowns associated with an independent set can easily be eliminated (in parallel) and yield another sparse linear system. The basic idea is to find such a set and then to eliminate the unknowns associated with it to obtain a smaller reduced system. The process may be combined with a dropping strategy to reduce fill-in and it is repeated a few times until the final reduced system is easily solvable. In [34], several heuristic algorithms have been suggested to find Maximal Independent Sets.

If we reorder the original linear system with an ordering in which the unknowns associated with the independent set are listed first, we obtain a linear system which has

the following block form

$$A \sim \begin{pmatrix} D & F \\ E & C \end{pmatrix}, \tag{2}$$

where $D$ is a diagonal matrix. The reduced system is the Schur complement with respect to $C$ as

$$A_1 = C - ED^{-1}F. \tag{3}$$

Since $D$ is diagonal, $A_1$ is still a sparse matrix in general. The same process may be applied recursively a few times to the consecutive reduced systems, such as $A_1$, until the last reduced system is small enough and can be solved by a direct or iterative solver. In ILUM, the reduction process is performed approximately. Standard threshold strategies are employed to control the sparsity of the L and U factors. One reason for studying this type of preconditioners is that they have a far better parallelism than the traditional incomplete LU factorization type preconditioners. Similar preconditioners have been designed and tested to show the property of near grid-independent convergence in [4].

There are two main reasons to seek block generalizations of ILUM. First, the diagonal elements in $D$ may be close to zero at each reduction step. The resulting reduced systems may be poor and this may lead to inaccurate LU factors. Using blocks instead of single elements will lead to a better control of near-singularity. Second, there are many applications leading to matrices with a fairly large number of nonzero elements per row. In these situations, it is common that the size of the independent set is small. It seems natural to group nodes in clusters of highly coupled subsets of nodes to reduce the size of the vertex-cover (a complement of an independent set is called a vertex cover).

## 3 Block Independent Sets

We first introduce a few straightforward generalizations of maximal independent sets. Consider a collection of non-empty subsets

$$B_j = \{v_{j_1}, v_{j_2}, \ldots, v_{j_k}\} \neq \emptyset$$

of the vertex set $V$ which are disjoint exclusive, i.e., such that

$$B_j \cap B_i = \emptyset, \quad \text{if} \quad j \neq i.$$

Recall that a *quotient graph*, is a graph whose vertices are the subsets $B_i, i = 1, \ldots, m$, see [20, p. 105]. It is defined by coalescing all the nodes in each subset $B_i$ into a *super-vertex* and defining an edge from any super-vertex $B_i$ to another super-vertex $B_j$ if there is an edge from a vertex in $B_i$ to a vertex in $B_j$, i.e.,

$$B_i \rightarrow B_j \quad \text{if} \quad \exists\, k_i \in B_i,\ \exists\, k_j \in B_j \quad \text{such that} \quad a_{k_i,k_j} \neq 0\ .$$

A block-independent set is simply an independent set on this quotient graph.

**Definition 3.1** *Let $B_1, B_2, \ldots, B_m$ be a collection of mutually exclusive nonempty subsets of $V$. The set $S = \{B_1, B_2, \ldots, B_m\}$ is said to be a block independent set if any two distinct subsets $B_j$ and $B_k$ in $S$ are not adjacent in the quotient graph.*

It is not necessary that $B_1, B_2, \ldots, B_m$ have same cardinality (size). However, for simplicity, we will deal mostly with subsets of constant cardinality $k$ in this paper. We will then use the subscript $k$ for $S$ to denote the cardinality of uniform blocks. Given this assumption, Definition (2.1) is a special case of Definition (3.1) with $k = 1$. In the sequel, we may use point version and block version of size 1 interchangeably. A useful and convenient independent set may consist of the union of $S_1$ and $S_k$ with some $k > 1$. Other combination seems to need special treatments that are beyond the scope of the current paper.

Matrices which arise in certain applications have a natural block structure, in which the blocks are small dense matrices, typically of the same size. In such cases, it is natural to use this blocking for defining a block version of ILUM. However, an additional blocking, based on the graph in which each node now represents a physical point, may be mandatory to obtain good performance. The blocking strategies referred to in this paper are related to this graph, or mesh, and are therefore in addition to the original blocking that may be naturally available.

Heuristic algorithms for finding point independent sets have been discussed in [34] and been successfully utilized with ILUM to develop parallelizable preconditioners for solving general sparse matrices. Based on the strategies of [34], we introduce some heuristic algorithms for finding block independent set from a given matrix. The algorithms are described with $k = 2$, but generalization to other block size is straightforward.

### 3.1 Blocking Strategies for Independent Sets of Size 2

Blocks of size 2 will be found by strategies for coupling a given node $j$ with one of its nearest neighbors. Suppose the $j$th row of the matrix has nonzero elements $a_{j,j_1}, a_{j,j_2}, \ldots, a_{j,j_q}$. One way to find this coupling is to examine the absolute value of $a_{j,j_i}, 1 \leq i \leq q, j_i \neq j$, which determines the strength of the link from node $j$ to node $j_i$. A potential benefit of BILUM is that it can handle small or even zero diagonals. The natural choice is the strongest link, i.e., the element with the largest absolute value. In doing so, we will have a $2 \times 2$ submatrix (after permutation) of the form

$$\begin{pmatrix} a_{j,j} & a_{j,j+1} \\ a_{j+1,j} & a_{j+1,j+1} \end{pmatrix}. \tag{4}$$

Notice that in contrast with ILUM small diagonal elements do not necessarily cause difficulties. Even though $a_{j,j}$ or $a_{j+1,j}$ may be small, both singular values of the above matrix can be away from zero provided the co-diagonal elements are not small. This will lead to a stable inversion of the above matrix. In the following description adj($j$) denotes the set of all nearest neighbors of node $j$, i.e., all vertices $l$ such that $a_{j,l} \neq 0$.

**Algorithm 3.2** *Greedy algorithm for blocking elements by strongest links.*

1. *Set* $m = 0$.
2. *For* $j = 1, 2, \ldots, n$, *Do:*
3. *If node* $j$ *is not marked, then*

*4.* $m := m + 1$, $B_m = \{j\}$
*5.* *Choose* $s \in \text{adj}(j)$ *such that* $|a_{j,i}| = \max\{|a_{j,i}|,\ i \in \text{adj}(j)\}$
*6.* $B_m = B_m \cup \{s\}$
*7.* *Mark* $j$ *and all nodes in* adj$(j)$.
*8.* *EndIf.*
*9.* *EndDo.*

The output of the algorithm is a block-independent set $S_2 = \{B_1, B_2, \ldots, B_m\}$ in which each supervertex is a set consisting of two nodes. Clearly, other criteria than the one in Line 5 may be used to select the vertex $s$ to add to $j$ to form a set $B_m$. For example, weakest links may be explored instead of strongest links. We will consider some of these options and study them experimentally. One alternative that is of interest is to use criteria based on the degree of a node. Recall that the degree of a vertex $j$, deg(j), is equal to the total number of edges that are adjacent to $j$. If $\tilde{\eta}$ is the maximum degree of all the vertices that constitute $S_1$, then a lower bound for the size of $S_1$ is given by [34]

$$|S_1| \geq \frac{n}{1 + \tilde{\eta}}. \tag{5}$$

This suggests that it may be a good idea to visit the nodes with the smallest degrees first [34] to form independent sets of size 1. This observation also suggests pairing the current node with the vertex among its nearest neighbors which has the smallest degree, to form independent sets of size 2.

**Algorithm 3.3** *Minimal degree algorithm for blocking elements.*

*1.* *Set* $m = 0$.
*2.* *For* $j = 1, 2, \ldots, n$, *Do:*
*3.* *If node* $j$ *is not marked, then*
*4.* $m := m + 1$, $B_m = \{j\}$
*5.* *Choose* $s \in \text{adj}(j)$ *such that* $\deg(s) = \min\{\deg(i);\ i \in \text{adj}(j)\}$
*6.* $B_m = B_m \cup \{s\}$
*7.* *Mark* $j$ *and all nodes in* adj$(j)$.
*8.* *EndIf.*
*9.* *EndDo.*

We may further improve the stability of the blocks by insisting that the magnitude of the diagonal elements $a_{ss}$ resulting from the strategy in Line 5, are greater than some threshold tolerance $\varepsilon$. However, larger $\varepsilon$ enhances the stability of the factorization, but may reduce the size of the independent set. The best strategy may be to use a threshold parameter $\varepsilon$ to make sure that the diagonals are not zero or too small (this may be applied to ILUM too). Algorithms 3.2 and 3.3 may be modified appropriately to include the threshold strategy. There are other heuristics that may be used to form blocks, such as choosing the node whose row has the most (least) diagonal dominance. We have experimented this and several other possibilities and have not found any of these alternatives performed substantially better than the ones considered here.

At the end of the blocking procedure, there are usually some nodes left which are neither in the independent set nor in its complement because they are not coupled. They are in effect subsets of size one which are independent or *singletons*. We could just add those to the complement set but this would be ineffective. It is best to add these singletons to the set of independent subsets, listing them last. (In the experiments reported in this paper this strategy was not implemented.)

# 4 Block ILUM Factorization

The main lines of a block version of ILUM are similar to those of the scalar version developed in [34]. Here we only recall the main steps.

After an independent set is found and the matrix $A$ is permuted into the form of (2) where $D$ is now a block diagonal matrix

$$D = \mathrm{diag}(\tilde{D}_1, \tilde{D}_2, \ldots, \tilde{D}_l, d_{l+1}, d_{l+2}, \ldots, d_m),$$

and each $\tilde{D}_i$ is a $k \times k$ matrix and each $d_j$ is a single diagonal element. We may eliminate the unknowns of the independent set to obtain a reduced system with the matrix $A_1$ as in (3), and the process may be repeated with $A_1$, We then have a series of reduced systems of the form

$$A_{j+1} = C_j - E_j D_j^{-1} F_j. \tag{6}$$

By doing these reductions, we have implicitly performed a block LU factorization

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} = \begin{pmatrix} I_j & 0 \\ E_j D_j^{-1} & I_j \end{pmatrix} \times \begin{pmatrix} D_j & F_j \\ 0 & A_{j+1} \end{pmatrix} \tag{7}$$

with $A_{j+1}$ defined as in (6). Hence, in order to solve a system with $A$, we may perform a series of forward and backward substitutions with the block matrices defined on the right-hand side of (7).

The inverse of the block diagonal matrix $D_j$ in (6) and (7) can be computed explicitly by inverting each small block exactly. Alternatively, these small matrices can also be factored using Gaussian elimination and the factors will then be kept instead of the explicit inverses. If explicit inversion is employed, it is best to utilize a thresholded pseudo-inverse employing a singular value decomposition as is often done in block ILU factorizations, for details, see [7, 8]. The above reduction process can be continued recursively with the matrix $A_j$ being replaced by $A_{j+1}$ , until we reach a Schur complement matrix $A_{nlev}$ which is small enough to be solved by a direct or a preconditioned iterative method.

The multi-level block factorization described above becomes more expensive as the number of levels increases. This is because of the fill-ins introduced by the elimination process. The matrices $E_j D_j^{-1}$ and $A_{j+1}$ as in (7) become denser as the factorization proceeds. A general practice for avoiding this undesirable result in developing preconditioners is to neglect some of the fill-ins created by using a simple dropping strategies as we form the reduced system [34, 35]. For example, we may drop any fill-in element created whenever its absolute value is less than a given tolerance $\tau$ times the average of the absolute

values of the row in question. A second strategy is to limit the number of fill-ins that are accepted in each row. Thus, an approximate version of the successive reduction steps can be used to provide an approximate solution $M^{-1}v$ to $A^{-1}v$ for any given vector $v$. This approximate solution is usually too crude to approximate the solution of Eq. (1), but it may instead be used to precondition the original linear system. For detailed descriptions, see [34].

Our preconditioner is constructed differently from that of [4, 5]. Specifically, we construct $D_j$ (and $D_j^{-1}$) and $F_j$ exactly whenever this is possible, since these matrices do not have any fill-in (except for $D_j^{-1}$) during the reduction process. Also, there is no dropping for the diagonal elements regardless of their values. However, dropping strategies based on a threshold tolerance are applied to $E_j D_j^{-1}$ and $A_{j+1}$. This construction guarantees some degree of accuracy for the inter-level transfer operators ($F_j$ and $E_j D_j^{-1}$). As a result, we are able to use a uniform dropping tolerance at all the levels. Our numerical experiments suggested that, for most problems, there is no gain in decreasing the dropping tolerance as the number of levels increases.

Based on the algorithms presented in the previous sections, the complete preprocessing phase of BILUM consists of four steps: (1) finding the block independent set ordering, (2) permuting the matrix, (3) computing the inverse (or the exact LU factorization) of the diagonal block matrix $D_j$ by computing the inverse (exact LU factorization) of each $k \times k$ matrix explicitly, and (4) forming the reduced system as well as the $L$ and $U$ parts of the BILUM matrix. Fig. 1 is an illustration of the structure of the processed matrix after two levels of reduction with 4 blocks of size 2 at each level.

The forward-backward solution process is similar to the one described in [34] with the diagonal matrix $D_j$ being replaced by the block diagonal matrix. In our current implementation, we stored the permutation matrices for each level and did the permutation and inverse permutation explicitly at each preconditioning step. A slightly different approach was used in [34] for implementing ILUM, where the matrix is permuted explicitly.

## 5 Size of the Independent Set

In addition to being potentially robust by avoiding instabilities caused by small diagonals in the ILUM factorization, BILUM may also have the advantage of yielding larger independent sets. This will in general produce a smaller Schur complement and the cost of solving the last reduced system is consequently reduced.

Denote by $V_S$ the union of all subsets $B_1, \ldots, B_m$, and $V_c$ the complement set of vertices. We are interested in lower bounds for $|V_S|$, the total size of the independent set. Let $k$ be the number of nodes in each block and $m$ the number of blocks of uniform size in the independent set $S_k$. Let $\eta_j$ be the degree of vertex $j$. Let $\eta$ be the maximum degree of each node in $V$, i.e.,

$$\eta = \max_{1 \le j \le n} \{\eta_j\},$$

It has been shown in [34] that

$$|V_{S_1}| \ge \frac{n}{1+\eta}. \tag{8}$$

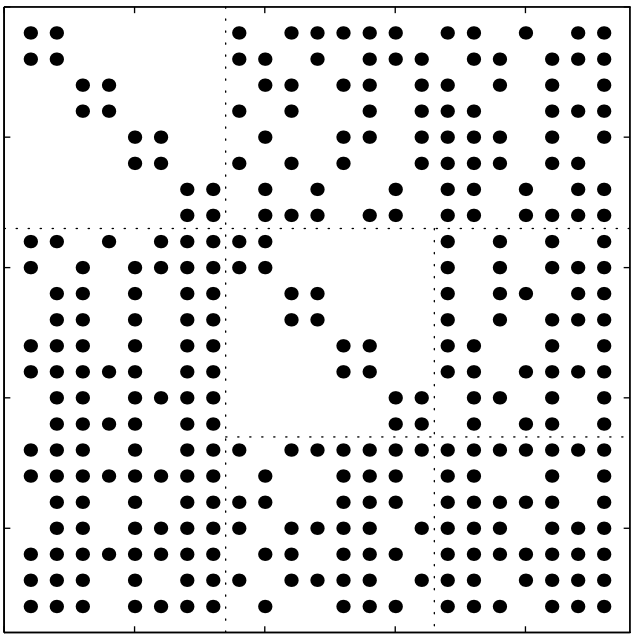

Figure 1: Illustration of the structure of the processed matrix after two levels of reduction with 4 blocks of size 2 at each level.

This is a rough lower bound. For example, when the graph is bipartite, i.e., if it can be colored with two colors, the Breadth-First-Search algorithm will yield an independent set of size $n/2$. For an independent set with blocks, it is hard to estimate the number of nodes in the independent set. However, with some simplifying assumptions, a number of simple results can be established.

We now make the assumption that the independent set is maximal, and that all subsets are of size $\leq k$. In other words, if there is a node or a subset of nodes of size $< k$, then it should be a member of $S$. Algorithmically, this means that some clean-up must be added at the end of the block-forming phase in which all singletons are grouped in independent subsets of size $\leq k$. With this assumption it is clear that any node in the vertex cover must be coupled to at least one subset from $S$.

In the following we will call the peripheral degree (pdeg) of a subset $B$ of vertices the number of vertices not belonging to $B$ which are linked to a vertex in $B$ in the original graph, see Fig. 2. The set of these vertices is usually denoted by adj($B$) [20] and therefore,

$$\mathrm{pdeg}(B) = |\mathrm{adj}(B)|$$

The main parameter which determines the size of $V_S$ is the ratio $\mathrm{pdeg}(B)/|B|$. As the blocks become smaller these ratios become smaller, provided the aspect ratio of each subgraph remains moderate. Let $S = \{B_1, \ldots, B_m\}$ and denote by $\gamma_i$ the ratio

$$\gamma_i = \frac{\mathrm{pdeg}(B_i)}{|B_i|}, \quad 1 \leq i \leq m.$$

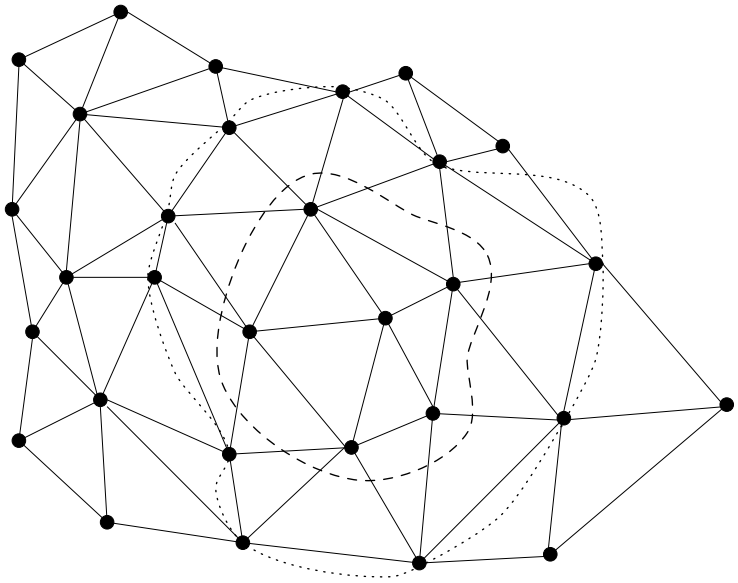

Figure 2: A set $B$ of vertices (surrounded by a dashed line) and its peripheral vertices (joined by a dotted line).

Clearly, the complement of $V_S$ is the set of all peripheral vertices. Therefore,

$$n - |V_S| = \sum_{i=1}^{m} \gamma_i |B_i|$$

Now defining

$$\gamma = \max_{1 \leq i \leq m} \{\gamma_i\},$$

we obtain,

$$n - |V_S| = \sum_{i=1}^{m} \gamma_i |B_i| \leq \gamma \sum_{i=1}^{m} |B_i| = \gamma |V_S|$$

from which we get the following immediate generalization of (8)

**Proposition 5.1** *Let $S = \{B_1, B_2, \ldots, B_m\}$ be a maximal independent set and assume that the peripheral degree of each subset $B_i$ is such that* $\mathrm{pdeg}(B_i) \leq \gamma |B_i|$. *Then,*

$$|V_S| \geq \frac{n}{1+\gamma} \tag{9}$$

As the block-size increases, $\gamma$ will decrease to zero provided the algorithm chooses sets that have 'a round shape'. Typically, for large $k$, one can expect $\gamma$ to be of the order of $k^{-1/2}$ for two-dimensional meshes. A simple algorithm for achieving this effect would be to do a breadth-first search from the next candidate until a satisfactory number of nodes is reached.

**Example.** Consider this algorithm for a 9-point matrix (see Section 6 for definition) on a square $N \times N$ mesh, where $N$ is a multiple of a certain number $s+1$. Let $k = s^2$ be the size of the blocks. We start at node one and take all its neighbors, then the neighbors

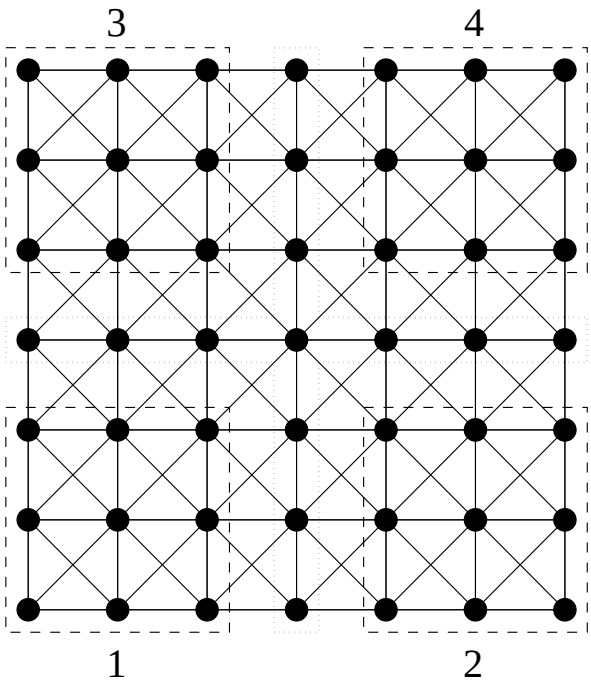


Figure 3: A 9-point $7 \times 7$ mesh. The subsets of nodes surrounded by dashed lines are members of the independent set $S$. Those surrounded by dotted lines are in $V_c$.

of its neighbors until $k$ nodes are found. This means that breadth-first search traverses $s$ level sets to fill one set of $k = s^2$ nodes. Then the next traversal starts at vertex $s + 2$, assuming natural ordering has been used to label the nodes. Hence, the blocks found are square of side $s$, and in this case

$$\mathrm{pdeg}(B_i) \leq 4(s + 1).$$

Therefore,

$$\gamma \leq \frac{4(s+1)}{s^2} = \frac{4}{s} + \frac{4}{s^2} \approx 4k^{-1/2}.$$

Fig. 3 is an illustration with $N = 7$ and $s = 3$. We have $k = 9$ and four blocks with $|B_i| = 9, i = 1, \ldots, 4$. It is easy to see that $\mathrm{pdeg}(B_i) = 2s + 1 = 7$. Hence $\gamma_i = 9/7$ by (5). It follows that $\gamma = 9/7$. Inequality (9) gives the lower bound $|V_S| \geq 343/16 \approx 21.4$. The actual size of the independent set is $|V_S| = 36$ and $|V_c| = 13$. Note that for $k = 1$ $|V_{S_1}| = 16$ and $V_{S_9}$ is significantly larger than $S_1$. In this case, a *properly chosen* larger block size does increase the size of the independent set.

Another simple example can show that an *improperly chosen* large block size does not necessarily increase the size of the independent set. Recall that a square $N \times N$ 5-point mesh is two-colorable and $|V_{S_1}| = N^2/2$, assuming for simplicity that $N$ is even. Then the line red-black ordering which consists of taking $k = N$ and choosing the blocks $B_i$ to be the sets consisting of the vertices forming every other line in the mesh in the $x$ direction satisfies

$$|V_S| \quad = \quad \frac{N^2}{2}, \quad \text{if} \quad N \quad \text{is even},$$

$$|V_S| \quad = \quad \frac{N^2+N}{2}, \quad \text{if} \quad N \quad \text{is odd.}$$

Note that $|V_S|$ is approximately the same as for for $k = 1$ for large $N$. In general blocks formed in a round shape have smaller peripheral degrees and are better than those formed in a thin shape which have larger peripheral degrees.

# 6 Numerical Experiments

Unless otherwise indicated explicitly, all tests used 20 levels of reduction. In all cases, the last reduced system was solved approximately by a GMRES(10) algorithm preconditioned by a dual threshold ILUT($p$,$\tau$) preconditioner [32], which will be referred to as the inner iteration. The inner iteration was terminated when either the number of iterations exceeded 10 or the (inner iteration) residual in 2-norm was reduced by a factor of $10^2$, whichever condition is satisfied first. We chose different values for $\tau$ and $p$ for different problems to demonstrate that there is freedom to adjust these parameters to enhance the performance of the algorithm. The first level was factored exactly, i.e., no dropping in any LU factor or the Schur complement $A_1$ on the first level. Beginning from the second level, elements in the Schur complements $A_j$ and in the U factor $E_j D_j^{-1}$ on each level were dropped when they were smaller than $\tau$ multiplied by the average of the absolute values of the row in question. As was said before, there was no dropping in $D_j$ and $F_j$.

Although some of the test matrices were generated on regular grids, the structure of the grids was not utilized and all matrices were considered as general unstructured sparse matrices. In all tests, the right-hand sides were generated by assuming that the exact solution is a vector of all ones. The initial guess was generated randomly. The program was stopped when the initial residual (of the outer iteration) in 2-norm was reduced by a factor of $10^7$. We set an upper bound of 100 for the outer GMRES(20) or GMRES(10) 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.

## 6.1 A Convection-Diffusion Problem

We consider the convection-diffusion equation

$$-\Delta u - \mathrm{Re}\left(\sin x \cos \pi y \frac{\partial u}{\partial x} - \cos \pi x \sin y \frac{\partial u}{\partial y}\right) = f(x,y), \tag{10}$$

on the unit square $(0,1)^2$. Dirichlet boundary values were removed and artificial left-hand side was used.

Here Re roughly reflects the Reynolds number. Eq. (10) is frequently encountered in computational fluid dynamics to model transport problems. Eq. (10) is discretized by the standard 5-point upwind finite difference scheme and the fourth-order 9-point compact scheme [22] with various (uniform) meshsize $h$ and Re. The resulting matrices are conveniently called 5-point or 9-point matrices. In particular, we will refer to a 5-POINT matrix

discretized by the 5-point upwind scheme using Re = 1 and $h = 1/201$. This particular 5-POINT matrix has $40,000$ unknowns and $199,200$ nonzeros. Similarly, a particular 9-POINT matrix is constructed by using the 9-point compact scheme with Re = $10,000$ and $h = 1/201$. This 9-POINT matrix has $40,000$ unknowns and $357,604$ nonzeros. The 5-point matrices are always weakly diagonally dominant as Re increases, due to the large amount of artificial viscosity added. For constant coefficients, it can be shown [38] that the 9-point matrices are weakly diagonally dominant only when the cell Reynolds number ($h$Re/2) is less than 1 and they lose diagonal dominance quickly as Re increases. Consequently, a 9-point matrix is generally more difficult to solve by iterative methods than a 5-point counterpart with the same discretization parameters.

High Reynolds number flow problems are particularly hard to solve with a standard multigrid approach [23, 40]. Techniques based on algebraic multigrid or matrix-dependent grid transfer operators have been used for systems arising from upwind-type discretization schemes [12, 28]. There is strong interest in the multigrid community to develop multi-level algorithms which converge independently of the Reynolds number.

In all tables which follow, "lev." shows the number of levels of reduction used, "it." shows the number of outer GMRES(10) or GMRES(20) iterations, "prec." shows the CPU time in seconds for the preprocessing phase, "solu." shows the CPU time in seconds for the solution phase, "size" shows the dimension of the last reduced system, and "mem." shows the total memory requirement in millions of words to store the real values (excluding integer indices) of the incomplete factorization. In most cases, we focus our attention on the solution process and do not report the preprocessing CPU time. The main reason for this is that our preprocessing routines are currently not optimized. Significant improvements will be achieved by (1) reducing the cost of the search for block-formation and (2) reducing the cost of inverting the small dense blocks.

We first tested our algorithms with the 5-point matrices generated by fixing Re = 1 and varying $h$. The purpose of this experiment is to show how the algorithms perform as the size of the linear system increases. We compared ILUT($10^{-4}, 20$), ILUM(1) ($1 \times 1$ block) and BILUM(2) ($2 \times 2$ block). GMRES(20) was used in all outer iterations. The number of iterations and the (solution) CPU time in seconds per unknown are plotted in Fig. 4.

We note from Fig. 4 that the number of ILUT iterations increased dramatically as the size of the linear system increased. On the other hand, the numbers of ILUM(1) and BILUM(2) iterations were almost independent of the size of the linear system. This suggests a near grid-independent convergence for ILUM preconditioners. The average CPU time (the CPU time in seconds per unknown) was increasing for all preconditioners when the size of the linear system increased. But the increase in ILUM-type preconditioners was moderate, especially for ILUM(1), compared with that due to ILUT. Hence, ILUM are more efficient than ILUT. (We did not have a near grid-independent CPU time because we used a fixed number of 20 level reductions.)

Next, we fixed $h = 1/201$, but varied Re for the 5-point matrices. We again compare ILUT, ILUM(1), BILUM(2) and BILUM(3) ($3 \times 3$ blocks) with 10 levels of reduction, keeping the other parameters the same as above. The number of iterations and the solution

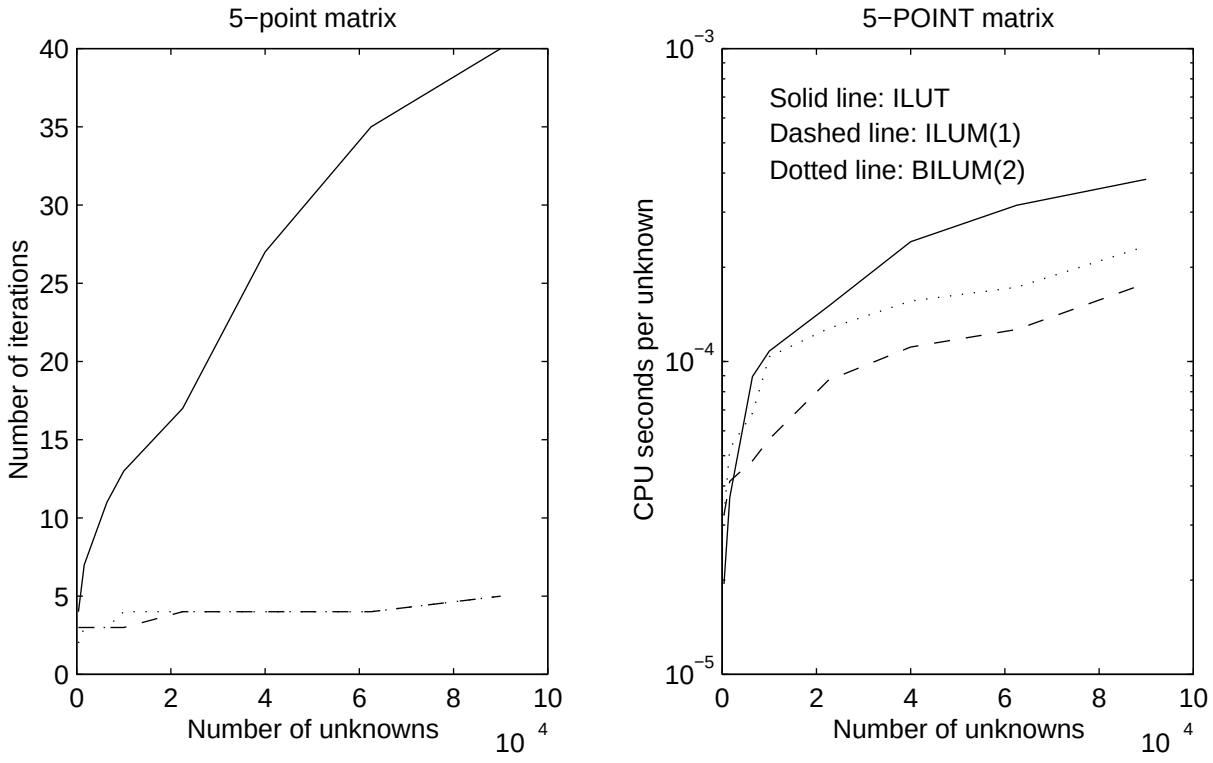


Figure 4: Performance comparison of ILUT($10^{-4}, 20$), ILUM(1) and BILUM(2) with the 5-point matrices with Re = 1 and varying size of linear systems. GMRES(20) was used for all outer iterations.

CPU time in seconds are listed in Table 1. It is seen that ILUM(1), BILUM(2) and BILUM(3) showed convergence that is independent of the Reynolds number Re. But the convergence of ILUT varied with the magnitude of Re. ILUM took less CPU time than BILUM(2). This is somewhat expected since the degree of each node of the 5-point matrices is relatively small and the advantages of using blocks of size 2 are not obvious, i.e., the sizes of the independent sets of ILUM(1) and BILUM(2) are not too different as will be seen later (Fig. 7). (However, BILUM(3) seems to do better than BILUM(2) for this test problem.)

Similarly, we fixed $h = 1/81$ and varied Re to generate some 9-point matrices. This time we used $\tau = 10^{-4}$ and $p = 40$ in ILUT and all the coarsest level of ILUM. For all ILUM-type factorizations 10 levels of reduction were used. GMRES(20) was the outer iteration accelerator for all preconditioners. The number of iterations and the solution CPU time in seconds are tabulated in Table 2. These results show that BILUM with larger size blocks performed better than ILUM. The ILUM preconditioned solvers showed convergence almost independent of the Reynolds number Re even when the matrix is far from diagonally dominant.

We further tested ILUT, ILUM(1) and BILUM(2) preconditioners with $\tau = 10^{-3}, p = 20$ using the 9-POINT matrix. GMRES(20) was used in the outer iterations. Fig. 5

| | ILUT($10^{-4}, 20$) | | ILUM(1) | | BILUM(2) | | BILUM(3) | |
|---|---|---|---|---|---|---|---|---|
| Re | it. | solu. | it. | solu. | it. | solu. | it. | solu. |
| 1 | 27 | 9.66 | 4 | 5.28 | 4 | 6.66 | 4 | 5.27 |
| 10 | 32 | 11.5 | 4 | 5.55 | 4 | 6.95 | 4 | 5.42 |
| $10^2$ | 40 | 14.3 | 4 | 6.84 | 4 | 7.82 | 5 | 7.29 |
| $10^3$ | 20 | 6.65 | 4 | 4.99 | 4 | 5.98 | 4 | 4.77 |
| $10^4$ | 21 | 5.82 | 4 | 3.34 | 4 | 4.53 | 4 | 3.86 |
| $10^5$ | 20 | 5.01 | 4 | 3.24 | 4 | 4.18 | 4 | 3.44 |
| $10^6$ | 14 | 3.11 | 4 | 2.92 | 4 | 4.19 | 4 | 3.41 |

Table 1: Performance comparison of different ILU algorithms with the 5-point matrix ($h = 1/201, n = 40,000, nz = 199,200$). $\tau = 10^{-4}, p = 20$ and 10 levels of reduction are used for all ILUM.

| | ILUT($10^{-4}, 20$) | | ILUM(1) | | BILUM(2) | | BILUM(3) | |
|---|---|---|---|---|---|---|---|---|
| Re | it. | solu. | it. | solu. | it. | solu. | it. | solu. |
| 1 | 7 | 0.58 | 3 | 0.51 | 3 | 0.76 | 3 | 0.58 |
| 10 | 6 | 0.50 | 3 | 0.66 | 3 | 0.76 | 3 | 0.60 |
| $10^2$ | 6 | 0.50 | 3 | 0.69 | 3 | 0.79 | 3 | 0.61 |
| $10^3$ | 7 | 0.47 | 3 | 0.71 | 3 | 0.68 | 3 | 0.57 |
| $10^4$ | 7 | 0.48 | 3 | 0.95 | 3 | 0.78 | 3 | 0.66 |
| $10^5$ | 11 | 0.81 | 4 | 2.40 | 4 | 1.46 | 3 | 0.91 |
| $10^6$ | 14 | 1.02 | 5 | 3.70 | 4 | 1.72 | 5 | 1.70 |

Table 2: Performance comparison of different ILU algorithms with the 9-point matrices ($h = 1/81, n = 6,400, nz = 56,644$). $\tau = 10^{-4}, p = 40$ and 10 levels of reduction are used for all ILUM.

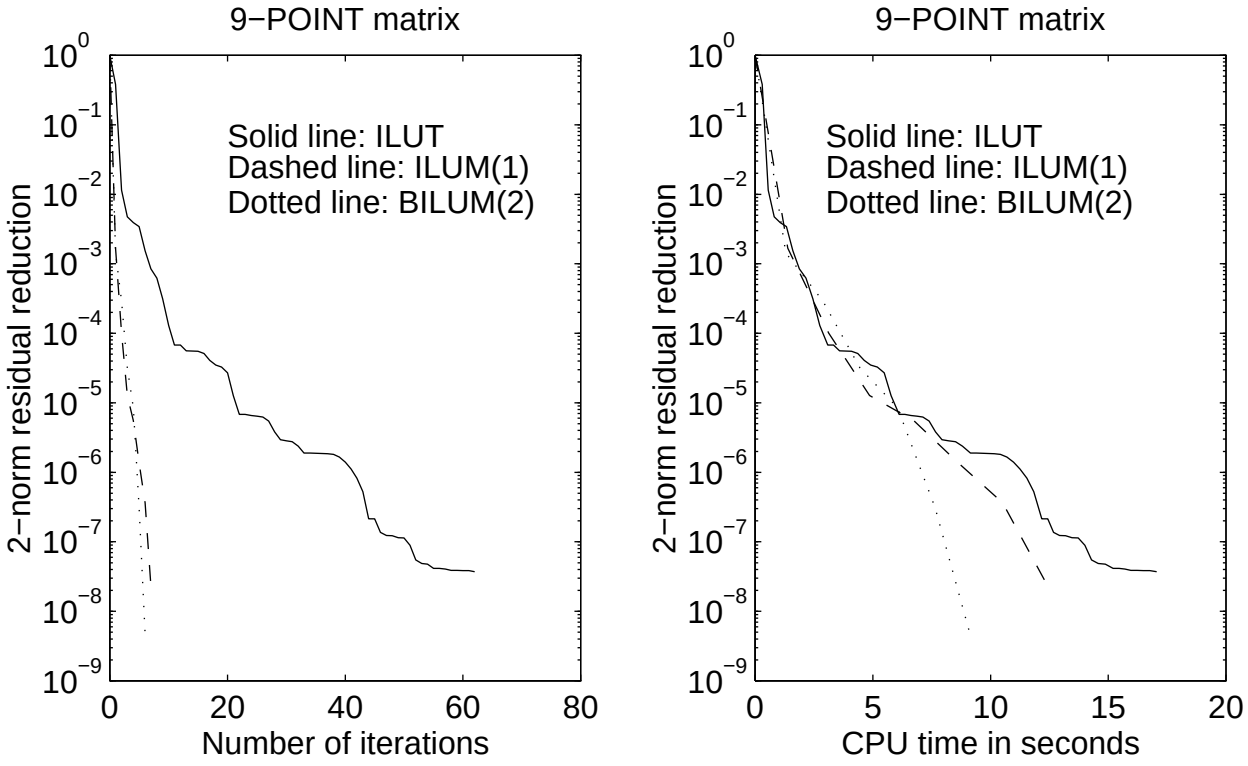


Figure 5: Performance comparison of ILUT($\tau = 10^{-3}, p = 20$), ILUM(1) and BILUM(2) with 9-POINT. GMRES(10) was used for the outer iterations.

depicts the 2-norm residual reduction rate with respect to the number of iterations and the solution CPU time in seconds. In terms of the number of iterations, ILUM(1) and BILUM(2) were again much better than ILUT, with BILUM(2) being the best. In terms of solution time, both ILUM preconditioners were faster than ILUT preconditioner. In contrast with the test results obtained for the 5-point matrices, BILUM(2) was eventually faster than ILUM(1). This suggests that matrices with denser couplings may be better solved with BILUM with larger blocks.

Fig. 6 shows a more dramatic comparison with the Harwell-Boeing matrix SAYLR4 [15, 17]. In this case, ILUM(1) and BILUM(2) were much more faster than ILUT both in terms of iteration counts and CPU times. The performance of the two ILUM algorithms were comparable and BILUM(2) took only slightly fewer iterations than ILUM(1) did.

Fig. 7 illustrates the effect of the number of levels on the size of the independent sets for ILUM(1), BILUM(2) and BILUM(3). The matrices tested here are the 5-POINT and 9-POINT matrices. As is seen larger block methods tend to yield larger independent sets. Furthermore, the difference between these methods becomes more important as the matrix becomes denser, i.e., when the degree of each node is higher.

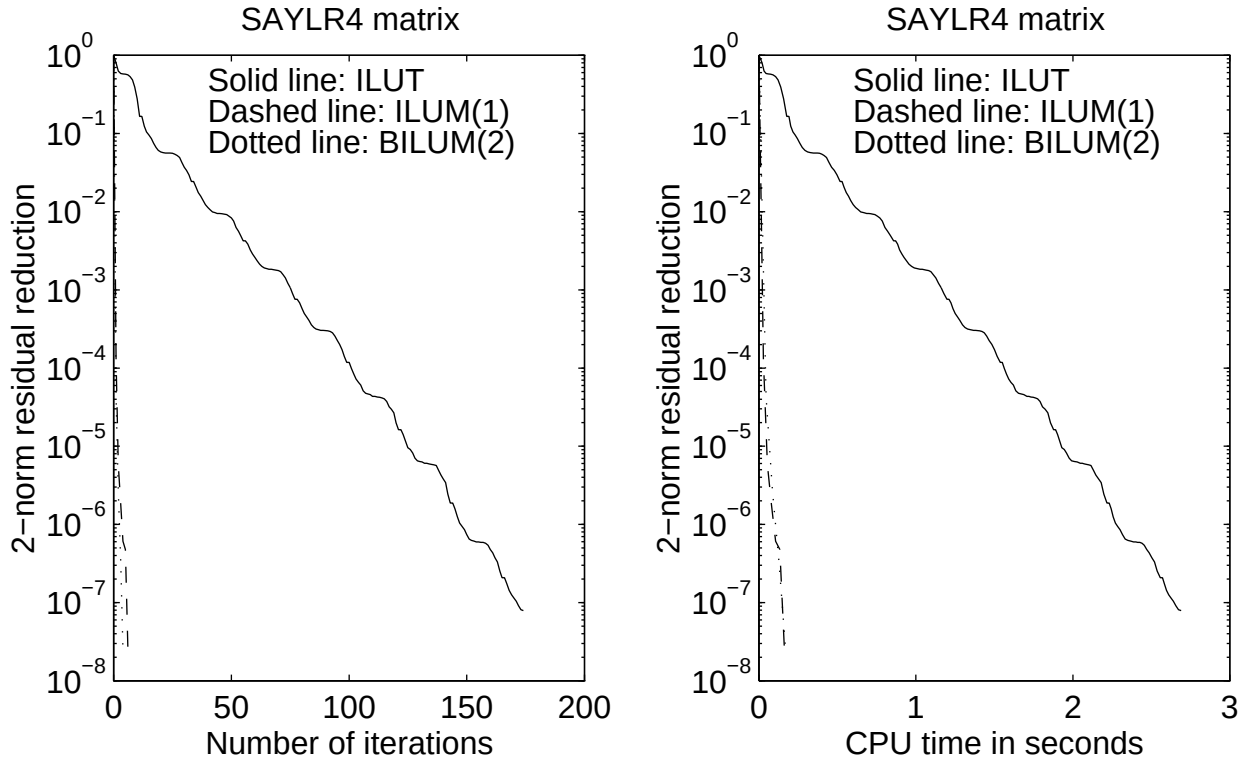


Figure 6: Performance comparison of ILUT($10^{-4}, 20$), ILUM(1) and BILUM(2) with the Harwell-Boeing matrix SAYLR4 ($n = 3,564, nz = 22,316$). GMRES(10) was used for the outer iterations.

### 6.2 Matrices from the Harwell-Boeing Collection

We compare the performance of different BILUM preconditioners with 5 matrices from the Harwell-Boeing collection [15, 17], together with our 5-POINT and 9-POINT matrices. (4 out of the 5 Harwell-Boeing matrices were tested with ILUM in [34] using different independent set ordering strategy.) We used $\tau = 10^{-3}$ and $p = 10$ for the reduction process and the last reduced system. GMRES(10) was used for the outer iterations. Various levels of reduction were tested. The results, including the preprocessing time, are given in Table 3.

For ORSREG1 and PORES2, ILUM(1) seems to be slightly faster. They performed comparably with 5-POINT. For the other four matrices BILUM is more efficient. Note that ILUM(1) did not converge in all cases with SHERMAN3. The efficiency usually increased as the size of the blocks became larger. Here, the preprocessing cost is of the same magnitude of the solution cost, except for the two largest problems (5-POINT and 9-POINT matrices). This high cost is due mainly to the unoptimized code segments used to find the independent sets.

| | | ILUM(1) | | | BILUM(2) | | | BILUM(3) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Matrices | lev. | it. | prec. | solu. | it. | prec. | solu. | it. | prec. | solu. |
| ORSREG1 | 2 | 4 | 0.156 | 0.084 | 4 | 0.322 | 0.123 | 4 | 0.294 | 0.150 |
| ($n = 2,205$) | 8 | 3 | 0.182 | 0.042 | 4 | 0.413 | 0.085 | 4 | 0.381 | 0.057 |
| ($nz = 14,133$) | 14 | 3 | 0.216 | 0.042 | 4 | 0.445 | 0.089 | 4 | 0.402 | 0.055 |
| | 20 | 3 | 0.232 | 0.041 | 4 | 0.460 | 0.089 | 4 | 0.407 | 0.054 |
| PORES2 | 2 | – | – | – | – | – | – | – | – | – |
| ($n = 1,224$) | 8 | 40 | 0.119 | 0.398 | 27 | 0.179 | 0.339 | 65 | 0.150 | 0.467 |
| ($nz = 9,613$) | 14 | 19 | 0.144 | 0.152 | 18 | 0.205 | 0.204 | 58 | 0.162 | 0.352 |
| | 20 | 19 | 0.160 | 0.143 | 20 | 0.223 | 0.229 | 59 | 0.165 | 0.355 |
| ORSIRR1 | 2 | 5 | 0.076 | 0.057 | 5 | 0.121 | 0.072 | 5 | 0.138 | 0.068 |
| ($n = 1,030$) | 8 | 7 | 0.080 | 0.044 | 5 | 0.139 | 0.042 | 5 | 0.133 | 0.033 |
| ($nz = 6,858$) | 14 | 7 | 0.090 | 0.034 | 5 | 0.150 | 0.044 | 5 | 0.139 | 0.031 |
| | 20 | 7 | 0.092 | 0.034 | 5 | 0.154 | 0.046 | 5 | 0.147 | 0.030 |
| SHERMAN3 | 2 | – | – | – | 9 | 0.748 | 1.144 | 9 | 0.709 | 1.058 |
| ($n = 5,005$) | 8 | – | – | – | 6 | 2.327 | 0.836 | 7 | 2.149 | 0.754 |
| ($nz = 20,033$) | 14 | – | – | – | 7 | 3.822 | 0.983 | 6 | 3.528 | 0.603 |
| | 20 | – | – | – | 5 | 5.113 | 0.734 | 8 | 4.700 | 0.772 |
| SHERMAN5 | 2 | 11 | 0.195 | 0.384 | 7 | 0.398 | 0.501 | 7 | 0.403 | 0.457 |
| ($n = 3,312$) | 8 | 12 | 0.412 | 0.382 | 6 | 0.797 | 0.410 | 6 | 0.754 | 0.308 |
| ($nz = 20,793$) | 14 | 13 | 0.766 | 0.430 | 4 | 1.193 | 0.297 | 5 | 1.158 | 0.258 |
| | 20 | 13 | 1.214 | 0.459 | 4 | 1.589 | 0.327 | 6 | 1.520 | 0.288 |
| 5-POINT | 2 | 5 | 2.781 | 5.344 | 5 | 47.94 | 6.717 | 5 | 35.39 | 6.236 |
| ($n = 40,000$) | 8 | 6 | 12.03 | 6.008 | 6 | 66.63 | 7.699 | 6 | 53.24 | 6.421 |
| ($nz = 199,200$) | 14 | 6 | 22.87 | 5.483 | 6 | 80.42 | 7.496 | 6 | 66.94 | 5.581 |
| | 20 | 6 | 32.59 | 4.843 | 7 | 91.47 | 8.202 | 6 | 77.43 | 4.694 |
| 9-POINT | 2 | 40 | 5.555 | 81.21 | – | – | – | – | – | – |
| ($n = 40,000$) | 8 | 10 | 25.07 | 20.73 | 12 | 68.42 | 21.27 | 6 | 53.05 | 7.648 |
| ($nz = 357,604$) | 14 | 8 | 46.20 | 15.27 | 5 | 84.07 | 8.099 | 7 | 64.80 | 7.433 |
| | 20 | 8 | 64.70 | 13.84 | 7 | 95.42 | 10.93 | 6 | 73.11 | 5.588 |

Table 3: Performance comparison of different ILUM algorithms with $\tau = 10^{-3}$ and $p = 10$. GMRES(10) is used in the outer iterations. "–" indicates that convergence was not reached within 100 outer iterations.

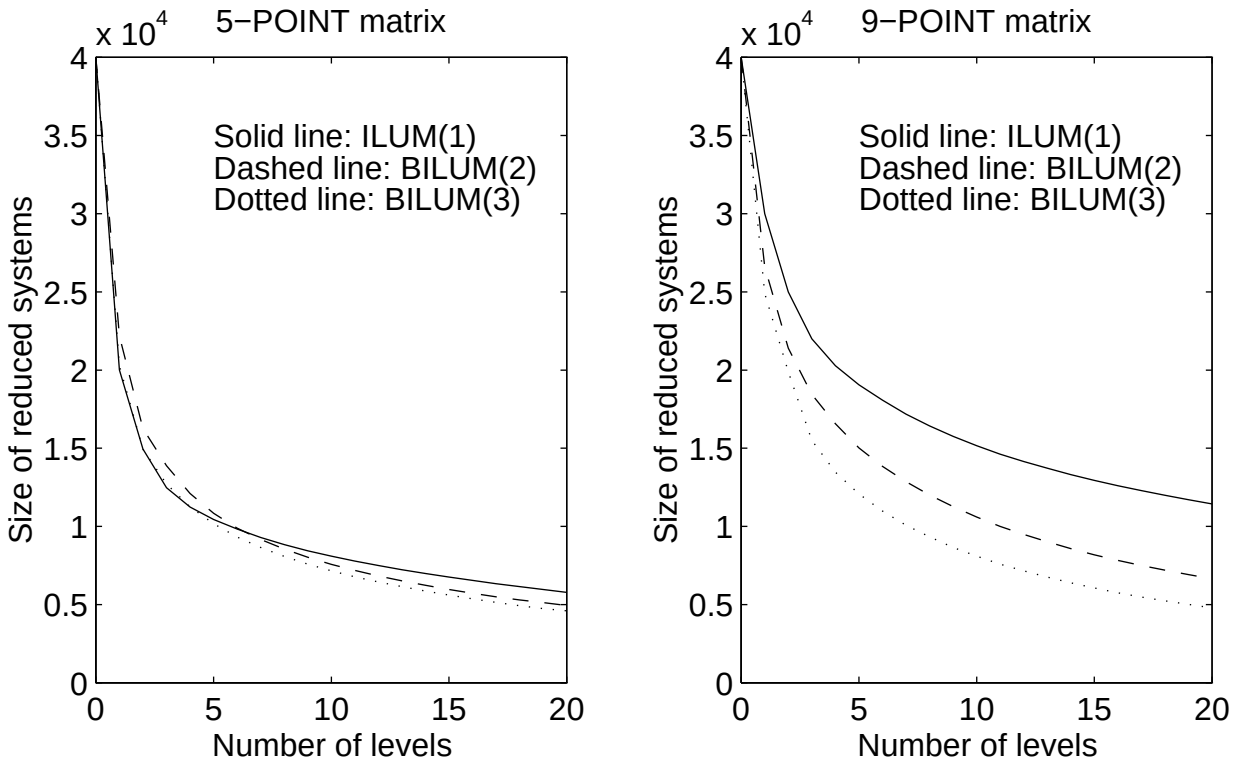


Figure 7: The size of the reduced systems after each level of approximate factorizations with single dropping tolerance ($\tau = 10^{-4}$).

### 6.3 Comparison of Different Heuristics

Since our algorithms for finding independent set are entirely based on heuristic arguments, we would like to test how they perform on real test problems. We tested three heuristics with BILUM(2), keeping other conditions the same as those used for the tests of Table 3. The heuristic algorithms we tested are:

1. Forming block pairs using strongest link as in Algorithm 3.2. This is denoted by BILUM(2) (Strong);
2. Forming block pairs by the weakest link, denoted by BILUM(2) (Weak); and
3. Forming block pairs by the minimal degree (Algorithm 3.3), denoted by BILUM(2) (Minimal).

Table 4 shows the number of iterations, the solution CPU time in seconds, and the size of the last reduced system. These results indicate that the heuristic which uses the minimal degree is the most efficient overall. It is followed by the strongest links strategy and finally the weakest link strategy is the least efficient. In most cases, efficiency seems to correlate with a small reduced system, i.e., a heuristic which generates larger independent set tends to be more efficient.

| | | BILUM(2) (Strong) | | | BILUM(2) (Weak) | | | BILUM(2) (Minimal) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Matrices | lev. | it. | solu. | size | it. | solu. | size | it. | solu. | size. |
| ORSIREG1 | 2 | 5 | 0.155 | 1153 | 4 | 0.120 | 1205 | 5 | 0.125 | 1161 |
| ($n = 2,205$) | 8 | 4 | 0.088 | 221 | 5 | 0.111 | 297 | 4 | 0.084 | 235 |
| ($nz = 14,133$) | 14 | 4 | 0.089 | 85 | 5 | 0.116 | 125 | 4 | 0.088 | 85 |
| | 20 | 4 | 0.086 | 33 | 5 | 0.116 | 57 | 4 | 0.089 | 31 |
| PORES2 | 2 | – | – | 792 | – | – | 792 | – | – | 708 |
| ($n = 1,224$) | 8 | 30 | 0.361 | 190 | 30 | 0.361 | 190 | 27 | 0.341 | 218 |
| ($nz = 9,613$) | 14 | 33 | 0.363 | 82 | 33 | 0.363 | 82 | 18 | 0.209 | 106 |
| | 20 | 37 | 0.405 | 34 | 37 | 0.405 | 34 | 20 | 0.232 | 48 |
| ORSIRR1 | 2 | 5 | 0.070 | 578 | 5 | 0.072 | 550 | 5 | 0.072 | 550 |
| ($n = 1,030$) | 8 | 4 | 0.036 | 102 | 5 | 0.042 | 104 | 5 | 0.043 | 104 |
| ($nz = 6,858$) | 14 | 4 | 0.035 | 30 | 5 | 0.044 | 34 | 5 | 0.045 | 34 |
| | 20 | 4 | 0.036 | 10 | 5 | 0.045 | 12 | 5 | 0.045 | 12 |
| SHERMAN3 | 2 | 9 | 1.180 | 3609 | 9 | 1.136 | 3613 | 9 | 1.150 | 3589 |
| ($n = 5,005$) | 8 | 9 | 1.426 | 3023 | 9 | 1.342 | 3027 | 6 | 0.839 | 2977 |
| ($nz = 20,033$) | 14 | 9 | 1.346 | 2811 | 7 | 1.100 | 2803 | 7 | 0.990 | 2693 |
| | 20 | 7 | 1.187 | 2679 | 8 | 1.267 | 2657 | 5 | 0.741 | 2567 |
| SHERMAN5 | 2 | 6 | 0.441 | 2936 | 7 | 0.531 | 2946 | 7 | 0.491 | 2844 |
| ($n = 3,312$) | 8 | 5 | 0.336 | 2158 | 8 | 0.644 | 2428 | 6 | 0.404 | 2164 |
| ($nz = 20,793$) | 14 | 4 | 0.295 | 1986 | 7 | 0.643 | 2230 | 4 | 0.295 | 2012 |
| | 20 | 4 | 0.316 | 1906 | 7 | 0.706 | 2126 | 4 | 0.325 | 1928 |
| 5-POINT | 2 | 5 | 7.00 | 16984 | 5 | 6.888 | 15994 | 5 | 6.726 | 16258 |
| ($n = 40,000$) | 8 | 6 | 8.683 | 8998 | 6 | 8.922 | 9990 | 6 | 7.701 | 7770 |
| ($nz = 199,200$) | 14 | 6 | 8.176 | 6078 | 7 | 10.22 | 7204 | 6 | 7.498 | 5174 |
| | 20 | 7 | 8.899 | 4300 | 8 | 10.99 | 5408 | 7 | 8.201 | 3742 |
| 9-POINT | 2 | – | – | 24244 | 36 | 79.14 | 24352 | – | – | 21396 |
| ($n = 40,000$) | 8 | 8 | 18.20 | 14150 | 8 | 17.87 | 14662 | 12 | 21.21 | 11636 |
| ($nz = 357,604$) | 14 | 7 | 15.08 | 9782 | 7 | 15.56 | 10634 | 5 | 8.103 | 7488 |
| | 20 | 7 | 14.67 | 7208 | 7 | 15.24 | 8118 | 7 | 10.93 | 5038 |

Table 4: Performance comparison of BILUM(2) with $\tau = 10^{-3}, p = 10$ and different heuristics for finding pair of blocks. GMRES(10) was used in the outer iterations. "–" indicates that convergence was not reached within 100 outer iterations.

### 6.4 Memory Cost

A major advantage of iterative methods over direct methods is that they may require far less memory. Previous experiments have shown that BILUM factorizations yield better performance than ILUT. One may ask whether this is achieved at the cost of using more memory. The amount of memory required by ILUM is highly dependent on the dropping threshold $\tau$ used in the reduction. Note also that the last reduced system needs to be solved with the help of ILUT. Depending on the number of levels, this last reduced system and the corresponding ILUT preconditioner require another $5-30\%$ more memory. Even with our current (unoptimized) code, ILUM and BILUM tend to use less memory space than ILUT does for similar performance with many matrices. Memory costs seem to increase as the block sizes increase, depending on the density of the matrix. The increase in memory costs is usually coupled with increase in performance. Table 5 lists the memory requirement (storing incomplete factorization) and performance for each preconditioner with different parameters to solve the 5-POINT matrix with GMRES(10). (The data for ILUM and BILUM include the storage of the last reduced systems and their ILUT preconditioners.) The best results for each preconditioner are highlighted. The following conclusions can be drawn for the 5-POINT matrix:

- As the block size increases, so does the memory cost;
- As the number of levels increases, the memory cost increases slightly. However, the substantial performance improvement outweighs the increase in memory cost;
- Both ILUM and BILUM outperform and use much less memory than ILUT;
- ILUM and BILUM(3) are to some extent comparable if we trade memory cost for CPU time, while BILUM(2) seems less efficient.

Table 6 lists similar information with the 9-POINT matrix. In this case, ILUT seems to perform slightly better than ILUM and BILUM with slightly larger memory cost. The interesting observation is that BILUM is much better than ILUM in terms of both memory and CPU time, and BILUM(3) is the best among these three.

Our current code relies on a single criterion (threshold tolerance) to control fill-in in the reduction and it works less efficiently when the matrix is denser. More subtle dropping strategies for ILUM can reduce the memory cost significantly. For example, dropping strategy based on the number of fill-ins may be used. Furthermore, it is not necessary to use ILUT to solve the last reduced system. Other direct or iterative solvers may be incorporated. In addition the benefit of added parallelism for ILUM are an obvious added advantage.

### 6.5 Condition Number of the Reduced Systems

One incentive for using ILUM preconditioning is to reduce the system to a smaller one which can be solved by a direct or preconditioned iterative method. In doing the reduction, we expect that the reduced systems become easier to solve as the number of levels increases.

| | ILUM(1) | | | BILUM(2) | | | BILUM(3) | | | ILUT | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\tau = 10^{-3}, p = 10$ | | | | | | | | | $p = 20$ | | | |
| lev. | mem. | it. | solu. | mem. | it. | solu. | mem. | it. | solu. | $\tau$ | mem. | it. | solu. |
| 17 | 1.26 | 6 | 5.47 | 1.48 | 7 | 8.86 | 1.65 | 6 | 4.89 | $10^{-3}$ | 1.35 | 35 | 11.5 |
| 23 | 1.31 | 7 | 5.53 | 1.53 | 7 | 8.11 | 1.71 | 7 | 4.90 | $10^{-4}$ | 1.58 | 27 | 9.84 |
| 29 | 1.35 | 7 | 5.02 | 1.57 | 7 | 7.69 | 1.74 | 7 | 4.48 | $10^{-5}$ | 1.59 | 27 | 9.94 |
| | $\tau = 10^{-3}, p = 15$ | | | | | | | | | $p = 30$ | | | |
| 17 | 1.31 | 6 | 5.19 | 1.52 | 7 | 8.49 | 1.69 | 6 | 4.64 | $10^{-3}$ | 1.35 | 35 | 11.5 |
| 23 | 1.35 | 6 | 4.55 | 1.56 | 7 | 7.99 | 1.73 | 6 | 4.06 | $10^{-4}$ | 2.32 | 16 | 7.68 |
| 29 | 1.38 | 7 | 4.84 | 1.59 | 7 | 7.54 | 1.76 | 7 | 4.37 | $10^{-5}$ | 2.37 | 17 | 8.09 |
| | $\tau = 1.5 \times 10^{-4}, p = 10$ | | | | | | | | | $p = 40$ | | | |
| 17 | 1.51 | 5 | 5.75 | 1.76 | 5 | 7.81 | 1.99 | 5 | 5.73 | $10^{-3}$ | 1.35 | 35 | 11.5 |
| 23 | 1.61 | 5 | 5.49 | 1.87 | 5 | 7.66 | 2.07 | 5 | 5.29 | $10^{-4}$ | **2.96** | **11** | **6.09** |
| 29 | 1.69 | 5 | 5.23 | 1.93 | 5 | 7.48 | 2.12 | 5 | 4.94 | $10^{-5}$ | 3.14 | 11 | 6.45 |
| | $\tau = 1.5 \times 10^{-4}, p = 15$ | | | | | | | | | $p = 50$ | | | |
| 17 | 1.57 | 4 | 4.74 | 1.82 | 4 | 6.43 | 2.04 | 4 | 4.55 | $10^{-3}$ | 1.35 | 35 | 11.5 |
| 23 | 1.66 | 4 | 4.40 | 1.91 | 4 | 6.11 | 2.11 | 4 | 4.00 | $10^{-4}$ | 3.21 | 11 | 6.39 |
| 29 | **1.73** | **4** | **3.98** | **1.97** | **4** | **5.93** | **2.15** | **4** | **3.74** | $10^{-5}$ | 3.87 | 10 | 6.82 |

Table 5: Performance and memory cost comparison of different ILUM algorithms with the 5-POINT matrix. GMRES(10) is used in the outer iterations.

| | ILUM(1) | | | BILUM(2) | | | BILUM(3) | | | ILUT | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\tau = 10^{-3}, p = 10$ | | | | | | | | | $p = 20$ | | | |
| lev. | mem. | it. | solu. | mem. | it. | solu. | mem. | it. | solu. | $\tau$ | mem. | it. | solu. |
| 17 | 2.03 | 8 | 15.4 | 1.74 | 7 | 11.5 | 1.96 | 6 | 6.76 | $10^{-3}$ | 0.98 | 62 | 17.2 |
| 23 | 2.10 | 8 | 14.4 | 1.80 | 7 | 11.1 | 2.02 | 7 | 6.99 | $10^{-4}$ | 1.27 | 22 | 7.26 |
| 29 | 2.15 | 8 | 13.6 | 1.84 | 7 | 9.03 | 2.06 | 7 | 6.58 | $10^{-5}$ | 1.40 | 22 | 7.73 |
| | $\tau = 10^{-3}, p = 15$ | | | | | | | | | $p = 30$ | | | |
| 17 | 2.11 | 7 | 14.0 | 1.79 | 6 | 10.2 | 1.99 | 6 | 6.95 | $10^{-3}$ | 1.14 | 35 | 10.6 |
| 23 | 2.17 | 7 | 13.0 | 1.83 | 6 | 9.58 | 2.04 | 6 | 6.18 | $10^{-4}$ | 16.4 | 20 | 7.61 |
| 29 | **2.21** | **7** | **12.4** | **1.86** | **5** | **7.63** | **2.08** | **6** | **5.63** | $10^{-5}$ | 1.92 | 20 | 8.46 |
| | $\tau = 1.5 \times 10^{-4}, p = 10$ | | | | | | | | | $p = 40$ | | | |
| 17 | 2.58 | 7 | 17.7 | 2.08 | 7 | 14.5 | 2.39 | 6 | 9.22 | $10^{-3}$ | 1.29 | 24 | 7.76 |
| 23 | 2.71 | 7 | 16.9 | 2.18 | 6 | 11.9 | 2.49 | 6 | 8.38 | $10^{-4}$ | 1.94 | 11 | 4.70 |
| 29 | 2.81 | 7 | 16.8 | 2.25 | 6 | 11.7 | 2.57 | 6 | 7.85 | $10^{-5}$ | 2.32 | 11 | 5.30 |
| | $\tau = 1.5 \times 10^{-4}, p = 15$ | | | | | | | | | $p = 50$ | | | |
| 17 | 2.69 | 5 | 13.0 | 2.15 | 5 | 10.6 | 2.45 | 5 | 8.02 | $10^{-3}$ | 1.41 | 24 | 8.12 |
| 23 | 2.81 | 7 | 17.8 | 2.23 | 5 | 10.1 | 2.53 | 6 | 8.69 | $10^{-4}$ | **2.19** | **10** | **4.58** |
| 29 | 2.89 | 6 | 14.8 | 2.29 | 5 | 9.81 | 2.60 | 5 | 6.60 | $10^{-5}$ | 2.64 | 10 | 5.21 |

Table 6: Performance and memory cost comparison of different ILUM algorithms with the 9-POINT matrix. GMRES(10) is used in the outer iterations.

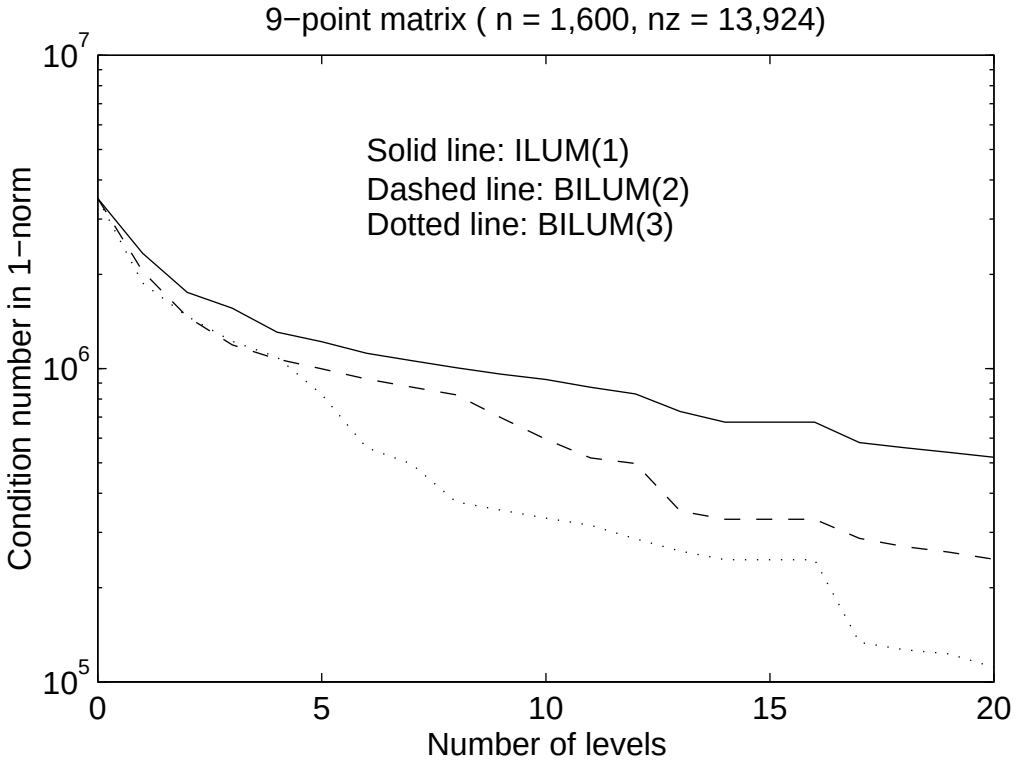


Figure 8: Condition numbers of a 9-point matrix and the reduced systems from ILUM and BILUM with $\tau = 10^{-4}$.

In other words, we expect the condition number of the reduced systems to be smaller than that of the full system. Fig. 8 depicts a relation between the condition numbers in 1-norm of a 9-point matrix ($\mathrm{Re} = 10,000, n = 1,600, nz = 13,924$) and the reduced systems from ILUM(1), BILUM(2) and BILUM(3) with $\tau = 10^4$. It is seen that the condition numbers are decreasing as the number of levels increases. It seems that the decreasing rate of the condition numbers slows down after several levels of reduction, especially for ILUM(1). This can be explained as the ILUM factorizations become less efficient (due to single dropping strategy) as the reduced systems become dense. Strategies should be ultimately designed to determine the number of levels and the degree of dropping dynamically. Ideally, it is best to stop when the final resulting reduced system is *easy enough* to solve. Heuristic criteria based on degree of diagonal dominance, number of nonzero elements, etc., might be developed.

# 7 Concluding Remarks

Numerical experiments indicate that the block ILUM preconditioning strategy presented here is generally more robust, and often more efficient, than both the point ILUM strategy and the dual-threshold ILUT preconditioning techniques. Our tests with convection-diffusion problems showed convergence that is nearly independent of the Reynolds number.

Convergence was also found to be nearly independent of the mesh-size of the underlying system. The results compared favorably with similar problems tested with the geometric or matrix-dependent multigrid methods [12, 28, 39]. Our algorithms, however, are purely algebraic and do not make any assumptions on properties of the linear system.

Thus, ILUM-type preconditioners can be viewed as multi-level variants of Incomplete LU factorizations which combine the benefits of generality and robustness of ILU techniques with those of grid-independent convergence enjoyed by multi-grid methods.

The numerical experiments that we conducted in this paper used small blocks. Use of larger blocks may be more efficient for certain types of problems and for some parallel computers. However, as the block-sizes become larger, sparsity should be exploited again. Ultimately, this general approach will provide a general framework that encapsulates the coarse-grain domain decomposition-type approach as well as the fine-grain point-ILUM approach which is more akin to the multigrid viewpoint. Further studies are planned in this direction.

## References


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

[2] O. Axelsson and B. Polman, On approximate factorization methods for block matrices suitable for vector and parallel processors, *Linear Algebra Appl.* **77**, 867–889 (1989).

[3] O. Axelsson, V. Eijkout, B. Polman and P. Vassilevski, Incomplete block-matrix factorization iterative methods for convection-diffusion problems, *BIT*, **29**, 867–889 (1989).

[4] 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., pp. 105–116, 1996.

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

[6] J. R. Bunch and B. N. Parlett, Direct methods for solving symmetric indefinite systems of linear equations, *SIAM J. Numer. Anal.* **8**, 639–655 (1971).

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

[8] E. Chow and Y. Saad, Experimental study of ILU preconditioners for indefinite matrices, *J. Comput. Appl. Math.* (to appear).

[9] P. Concus, G. H. Golub and G. Meurant, Block preconditioning for the conjugate gradient method, *SIAM J. Sci. Stat. Comput.* **6**, 187–209 (1983).

[10] E. F. D'Azevedo, F. A. Forsyth and W. P. Tang, Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems, *SIAM J. Matrix Anal. Appl.* **13** 944–961 (1992).

[11] E. F. D'Azevedo, F. A. Forsyth and W. P. Tang, Towards a cost effective ILU preconditioner with high level fill, *BIT* **31**, 442–463 (1992).

[12] P. M. de Zeeuw, Matrix-dependent prolongations and restrictions in a blackbox multigrid solver, *J. Comput. Appl. Math.* **3**, 1–27 (1990).

[13] I. S. Duff and J. Reid, The multifrontal solution of unsymmetric sets of linear equations, *SIAM J. Sci. Stat. Comput.* **5**, 633–641 (1984).

[14] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct Methods for Sparse Matrices,* Clarendon Press, Oxford, UK, 1986.

[15] I. S. Duff, R. G. Grimes and J. G. Lewis, Sparse matrix test problems, *ACM Trans. Math. Software* **15**, 1–14 (1989).

[16] I.S. Duff and G. A. Meurant, The effect of reordering on preconditioned conjugate gradients, *BIT* **29**, 635–657 (1989).

[17] I. S. Duff, R. G. Grimes and J. G. Lewis, *User's Guide for the Harwell-Boeing Sparse Matrix Collection,* Technical Report TR/PA/92/86, CERFACS, Toulouse, France, 1992.

[18] 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**, 457–497 (1993).

[19] H. C. Elman and E. Agron, Ordering techniques for the preconditioned conjugate gradient method on parallel computers, *Comput. Phys. Comm.* **53**, 253–269 (1989).

[20] J. A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, N. J., 1981.

[21] K. Gallivan, A. Sameh and Z. Zlatev, A parallel hybrid sparse linear system solver, *Comput. Systems Engrg.* **1**, 183–195 (1990).

[22] M. M. Gupta, R. P. Manohar and J. W. Stephenson, A single cell high order scheme for the convection-diffusion equation with variable coefficients, *Internat. J. Numer. Methods Fluids* **4**, 641–651 (1984).

[23] M. M. Gupta, J. Kouatchou and J. Zhang, A compact multigrid solver for convection-diffusion equations, *J. Comput. Phys.* **132**, 123–129 (1997).

[24] O. G. Johnson, C. A. Micchelli and G. Paul, Polynomial preconditionings for conjugate gradient calculations, *SIAM J. Numer. Anal.* **20**, 362–376 (1983).

[25] R. Leuze, Independent set orderings for parallel matrix factorizations by Gaussian elimination, *Parallel Comput.* **10**, 177–191 (1989).

[26] 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**, 148–162 (1977).

[27] O. Orterby and Z. Zlatev, *Direct Methods for Sparse Matrices,* Springer-Verlag, New York, 1983.

[28] A. Reusken, Fourier analysis of a robust multigrid method for convection-diffusion equation, *Numer. Math.* **71**, 365–398 (1995).

[29] Y. Saad and M. H. Schultz, GMRES: a generalized minimal residual method for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **7**, 856–869 (1986).

[30] Y. Saad, Krylov subspace methods on supercomputers, *SIAM J. Sci. Stat. Comput.* **10**, 1200–1232 (1989).

[31] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, *SIAM J. Sci. Comput.* **14**, 461–469 (1993).

[32] Y. Saad, ILUT: a dual threshold incomplete ILU preconditioner, *Numer. Linear Algebra Appl.* **1**, 387–402 (1994).

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

[34] Y. Saad, ILUM: a multi-elimination ILU preconditioner for general sparse matrices, *SIAM J. Sci. Comput.* **17**, 830–847 (1996).

[35] Y. Saad, *Iterative Methods for Sparse Linear Systems,* PWS Pub. Co., Boston, 1996.

[36] H. A. van der Vorst, A vectorizable version of some ICCG methods, *SIAM J. Sci. Stat. Comput.* **3**, 350–356 (1982).

[37] D. M. Young, R. G. Melvin, F. T. Johnson, J.B. Bussoletti, L. B. Wigton and S. S. Samant, Application of sparse matrix solvers as effective preconditioners, *SIAM J. Sci. Stat. Comput.* **10**, 1186–1199 (1989).

[38] J. Zhang, On convergence of iterative methods for a fourth-order discretization scheme, *Appl. Math. Lett.* **10**, 49–55 (1997).

[39] J. Zhang, Accelerated multigrid high accuracy solution of the convection-diffusion equations with high Reynolds number, *Numer. Methods for PDEs,* **13**, 77–92 (1997).

[40] J. Zhang, On convergence and performance of iterative methods with fourth-order compact schemes, *Numer. Methods for PDEs* (to appear).

[41] Z. Zlatev, Use of iterative refinement in the solution of sparse linear systems, *SIAM J. Numer. Anal.* **19**, 381–399 (1982).