ILUM: A MULTI-ELIMINATION ILU PRECONDITIONER FOR GENERAL SPARSE MATRICES [*]

Y. Saad [†]

Revised version, November 5, 1999

Abstract

Standard preconditioning techniques based on incomplete LU (ILU) factorizations offer a limited degree of parallelism, in general. A few of the alternatives advocated so far consist of either using some form of polynomial preconditioning, or applying the usual ILU factorization to a matrix obtained from a multicolor ordering. In this paper we present an incomplete factorization technique based on independent set orderings and multicoloring. We note that in order to improve robustness, it is necessary to allow the preconditioner to have an arbitrarily high accuracy, as is done with ILUs based on threshold techniques. The ILUM factorization described in this paper is in this category. It can be viewed as a multifrontal version a Gaussian elimination procedure with threshold dropping which has a high degree of potential parallelism. The emphasis is on methods that deal specifically with general unstructured sparse matrices such as those arising from finite element methods on unstructured meshes.

Keywords Sparse linear systems; Preconditioned Krylov subspace methods; Incomplete *LU* factorizations; Independent set orderings; Multicoloring; Graph coloring; Threshold dropping strategies.

[*]This research was supported by ARPA under contract number 60NANB2D1272 and by NSF under grant NSF/CCR-9214116

[†]University of Minnesota, Computer Science Department, 4-192 EE/CSci Building, 200 Union Street S.E., Minneapolis, MN 55455

1 Introduction

In this paper we address the problem of developing preconditioners for solving a linear system of the form

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

where A is a general sparse matrix of dimension N. The incomplete LU factorization with no fill-in, or ILU(0) [36], is one of the most popular preconditioners currently available. Its implementation on high-performance computers can be optimized by a technique referred to as 'level-scheduling', or 'wavefront ordering', see, e.g., [3]. A notable disadvantage of ILU(0) is that it is a rather crude approximation and for this reason it is unreliable when used to solve problems arising from certain applications, such as computational fluid dynamics. To improve the efficiency and robustness of ILU factorizations, many alternatives which allow higher levels of fill-in in the ILU factorizations have been developed [24, 38, 13, 12, 53, 52, 45]. Although these alternatives are more robust than the low-accuracy ILU(0) or SSOR, they are often intrinsically sequential.

A number of different approaches have been advocated to remedy the 'sequential nature' of the preconditioners developed in the 1970's and later, see for example the survey papers [14, 42]. The first of these approaches were motivated by vectorization and consisted mainly in replacing occurrences of matrix inverses by polynomials in these matrices. For example, the solutions of bidiagonal systems that arise in the forward and backward solutions on the ILU(0) preconditioning for model problems were replaced by low degree polynomial expansions in the matrices [47]. The paper [27], and subsequently a few others, advocated using polynomials in A as preconditioners. A second class of methods that has been developed consists of introducing parallelism by exploiting 'graph coloring', or *multicoloring* as we will refer to it here. The unknowns are colored in such a way that no two unknowns of the same color are coupled by an equation. In the simplest case of the 5-point matrix arising from the centered difference discretization of the Laplacian in two or three-dimensional spaces, only two colors are needed and they are commonly referred to as "red" and "black". When the unknowns of the same color are numbered consecutively, and a standard ILU factorization is applied to the reordered system, then a large degree of parallelism is available in both the preprocessing phase and in the preconditioning operations, during the iteration phase. A well-known drawback of this approach is that the number of iterations to achieve convergence may increase substantially, when compared with that required for the original system [19, 18, 20]. More generally, preconditioners that allow a large degree of parallelism such as, in the simplest case, diagonal scaling, tend to necessitate a much larger number of iterations to converge when compared with their sequential counterparts, e.g., the standard Incomplete LU factorization (ILU).

However, experience suggests that a good remedy for regaining an acceptable level of convergence rate is to *increase the accuracy of the underlying preconditioning.* For example, ILU preconditioners with more fill-in, or multi-step SSOR or SOR preconditioners, may be used. Experiments in [44] reveal that an approach based on SOR(k) preconditioning, in which each preconditioning operation consists of k steps of SOR sweeps, is superior to the best optimized ILU preconditioning on some problems. Unfortunately, for the standard ILU factorization the use of higher level of fill-in destroys the structure obtained from the multicolor reordering.

In this paper we consider a technique which is based on exploiting the idea of successive independent set orderings [33], a simpler form of multicoloring. This technique is very closely related to multifrontal elimination [15, 40], a classical method that is employed in the parallel implementations of sparse direct methods. Both multifrontal elimination and ILUM rely on the fact that at a given stage of Gaussian elimination, there are many rows that can be eliminated simultanously. The set that consists of such rows is called an independent set. The idea then is to find this set, and then eliminate the unknowns associated with it, to obtain a smaller reduced system. This reduction process is applied recursively a few times to the consecutive reduced systems until the system can be solved by a direct solver (multifrontal) or by an iterative technique (ILUM). In ILUM, we perform the reductions approximately, with the help of a standard threshold strategy, in order to control the sparsity of the L and U factors.

We start by describing the ideas of multicoloring and independent set orderings. Then we briefly show how these ideas can be exploited to develop direct solution methods and, finally, we derive incomplete factorization techniques by introducing numerical dropping strategies.

2 Independent set ordering and multicoloring

Graph theory provides numerous useful tools in sparse matrix computations and can be helpful in unraveling parallelism in standard algorithms [25, 15]. We recall that the adjacency graph of a sparse matrix is a graph $G = (V, E)$, whose N vertices in V represent the N unknowns and whose edges represent the binary relations established by the equations in the following manner: there is an edge from node j to node i when $a_{ij} \neq 0$. This edge will therefore represent the binary relation *equation i involves unknown j*, or equivalently *the unknown x_i depends on unknown x_j*. Note that the graph is directed, except when the matrix has a symmetric pattern ($a_{ij} \neq 0$ iff $a_{ji} \neq 0$ for all $1 \leq i, j \leq N$). Parallelism in the Gaussian elimination process can be obtained by finding unknowns that are independent at a given stage of the elimination, i.e., unknowns that do not depend on each other according to the above binary relation. The rows corresponding to such unknowns can then be used as pivots simultaneously. Thus, in one extreme, when the matrix is diagonal all unknowns are independent. On the other extreme, when a matrix is dense, each unknown will depend on all other unknowns. Sparse matrices lie somewhere in between these two extremes. Multicoloring techniques attempt to find sets of independent equations by coloring the vertices of a graph so that neighboring vertices have different colors. Independent set ordering can be viewed as a less restrictive form of multicoloring, in which a set of vertices in the adjacency graph is found so that no equation in the set involves unknowns from the same set. Multicoloring is best known in the partial differential equation context for 2-dimensional finite difference grid (5-point operator), which is our starting point in this discussion.

2.1 Red-black ordering for finite difference grids

For simple 2-dimensional centered finite difference grids, we can easily separate the grid points in two sets, Red and Black, so that the nodes in one set are adjacent only to nodes from the other set. This 2-color (red-black) ordering is illustrated in Figure 1 for a 6×4 grid.

If we reorder the unknowns in such a way as to list the red unknowns first together and then the black ones, as is illustrated in Figure 1, we will obtain a system of the form

$$\begin{pmatrix} D & F \\ E & C \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{2}$$

where D and C are diagonal matrices. Matrices that can be permuted into the above form are said to have property A, [51]. Several techniques have traditionally been used to exploit the

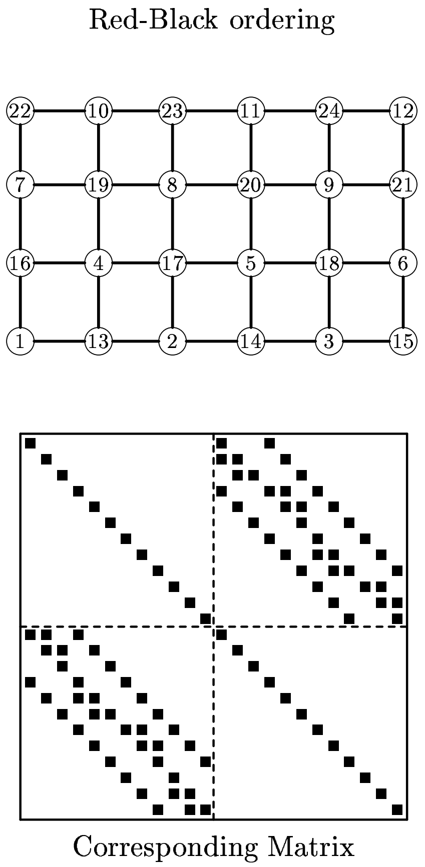

Figure 1: Red-black labeling of a 6×4 rectangular mesh and associated matrix.

above convenient structure see, e.g., [23, 41, 21].

Perhaps the simplest of these approaches is to use a conjugate gradient type technique combined with the standard ILU(0) preconditioner on the block system (2). Here, the degree of parallelism, i.e., the maximum number of arithmetic operations which can be executed in parallel, is of order N. A drawback is that often the number of iterations is higher than with the natural ordering but the approach may still be competitive for problems for which the natural ordering requires a relatively small number of steps, e.g., less than 30, to converge.

We have observed [44] that the number of iterations can be reduced back to a competitive level by using a more accurate ILU factorization on the red-black system, e.g., ILUT [45]. In fact the situation may be improved in that for the same level of fill-in p, the red-black ordering will outperform the natural ordering preconditioner for p large enough, in terms of number of iterations required for convergence. However, the fill-in introduced in the "black" part of the system with such high level ILUs ruin the degree of parallelism achieved from the red-black ordering. A remedy is to resort to similar *high-level SOR or SSOR preconditioners* instead of ILUT [44]. These consist of performing several SOR or SSOR steps in each preconditioning operation, instead of just one, as is traditionally done. A significant advantage of relaxation type preconditioners is that these higher level preconditioners do not lose their degree of parallelism as the accuracy increases.

A second method that has been used in conjunction with the red-black ordering is to eliminate the red unknowns by forming the reduced system which involves only the black unknowns, namely,

$$(C - ED^{-1}F)y = g - ED^{-1}f.$$

This linear system is again sparse and has about half as many unknowns. It has been observed that for easy problems, i.e., problems for which the natural ordering requires a relatively small number of steps to converge, this reduced system can often be efficiently solved with only diagonal preconditioning. The preprocessing to compute the reduced system is highly parallel and inexpensive. In addition, the reduced system is usually well-conditioned and has some interesting properties when the original system is highly nonsymmetric [21].

A general sparse matrix rarely has property A and as a result the application of the red-black ordering is quite limited. Fortunately, many of the above techniques can be generalized by using less restrictive forms of reorderings. For example, to exploit the reduced system approach, all we need is to reorder the original matrix into a matrix of the form,

$$\begin{pmatrix} D & F \\ E & C \end{pmatrix} \tag{3}$$

where D is diagonal but C can be arbitrary. There are three ways of generalizing the standard red-black ordering.

1. Independent set orderings, which lead to the form (3) above;
2. Multicolor orderings, which lead to block matrices in which the diagonal blocks are point-wise diagonal;
3. Full-block versions of the above techniques which allow the diagonal matrices to be block diagonal instead where each block is dense.

In the remainder of this paper we will consider the first and second generalizations. The block versions arise naturally in the solution of partial differential equations (PDEs) when each mesh point involves several unknowns. We also note that a further generalization would consist of allowing these diagonal blocks to be sparse matrices.

2.2 Independent set orderings

We now consider generalizations of the red-black ordering which consist of transforming the matrix into the block form (3), where D is diagonal. Let $G = (V, E)$ denote the adjacency graph of the matrix, and let (x, y) denote an edge from vertex x to vertex y. An *independent set* S is a subset of the vertex set V such that

$$\text{if } x \in S \quad \text{then} \quad \{\ (x,y) \in E \ \text{ or } \ (y,x) \in E\ \} \rightarrow y \notin S\ .$$

In words, elements of S are not allowed to be connected to other elements of S either by incoming or outgoing edges. An *independent set* is maximal if it cannot be augmented by elements in its complement to form a larger independent set. Note that a maximal independent set is by no means the largest possible independent set that can be found. In fact, finding the independent set of maximum cardinality, is NP-hard [32]. Independent set orderings have been used mainly in the context of parallel direct solution techniques for sparse matrices [33, 34, 11], and multifrontal techniques [15] can be viewed as a particular case. One of the goals of this paper is to show how to exploit these ideas in the context of iterative methods.

There are a number of simple and inexpensive heuristics for finding large maximal independent sets. For our purposes simple greedy algorithms such as the ones to be described next are inexpensive and yield enough parallelism in general. In the following we will use the term *independent set* to always refer to *maximal independent set.*

A simple greedy procedure for finding an independent set S is to traverse the graph and accept each visited node as a new member of S, if it is not already marked. We then mark this new member and all of its nearest neighbors. Note that by nearest neighbors of a node x we mean the adjacent nodes, linked to x by incoming or outgoing edges.

ALGORITHM **2.1 Greedy algorithm for independent set ordering**

```
Let $S = \emptyset$.
For $j = 1, 2, \ldots, n$ Do:
        If node $j$ is not marked then
                $S = S \cup \{j\}$.
                Mark $j$ and all its nearest neighbors.
        endif
enddo
```

In the above algorithm, the nodes are visited in the natural order $1, 2, \ldots, n$ but we can also visit them in any permutation $\{i_1, \ldots, i_n\}$ of $\{1, 2, \ldots, n\}$. In what follows we denote by $|S|$ the size of S, i.e., its cardinal. Since the size of the reduced system is $n - |S|$ it is reasonable to attempt to maximize the size of S in order to obtain a small reduced system, although the size is not all that matters.

We would like to give a rough idea of the size of S. Recall that the degree of a vertex v is the total number of edges that are adjacent to v. Assume that the maximum degree of each node does not exceed ν. Whenever the above algorithm accepts a node as a new member of S it potentially puts all its nearest neighbors, i.e., at most ν nodes, in the complement of S. Therefore, the size $n - |S|$ of its complement is such that $n - |S| \le \nu |S|$ and as a result,

$$|S| \ge \frac{n}{1 + \nu}.$$

This lower bound can be improved slightly by observing that we can replace ν by the maximum degree ν_S of *all the vertices that constitute S,* resulting in the inequality

$$|S| \ge \frac{n}{1 + \nu_S},$$

which suggests that it may be a good idea to visit first the nodes with smaller degrees. In fact this observation leads to a general heuristic regarding a good order of traversal. We can view the algorithm as follows. Each time a node is visited, we remove it and its nearest neighbors from the graph and then visit a node from the remaining graph and continue in the same manner until exhaustion of all nodes. Every node that is thus visited is a member of S and its nearest neighbors are members of the complement $\bar{S}$ of the set S. As a result, if we call ν_i the degree of the node visited at step i, adjusted for all the edge deletions resulting from the previous visitation steps, then the number n_i of nodes that are left at step i satisfies the relation

$$n_i = n_{i-1} - \nu_i - 1 \ .$$

The process adds a new element to the set S at each step and stops when $n_i = 0$. In order to maximize $|S|$ we need to maximize the number of steps in the procedure. The difficulty in the analysis comes from the fact that the degrees are updated at each step i because of the removal of the edges associated with the removed nodes. All we can say is that if we wish to lengthen the process, a rule of thumb would be to visit the nodes that have the smallest degrees first.

ALGORITHM **2.2 Independent set ordering with increasing degree traversal**

```
Let $S = \emptyset$. Find an ordering $i_1, ..., i_n$ of the nodes by increasing degree.
For $j = 1, 2, \ldots n$ Do:
    If node $i_j$ is not marked then
        $S = S \cup \{i_j\}$.
        Mark $i_j$ and all its nearest neighbors.
    endif
enddo
```

A refinement to the above algorithm would be to update the degrees of all nodes involved in a removal and dynamically select the one with smallest degree as the next node to be visited. This can be efficiently implemented using a min-heap data structure [9]. A different heuristic is to attempt to maximize the number of elements in S by a form of local optimization that determines the order of traversal dynamically. In the following the action of removing a vertex from a graph consists of deleting the vertex and all edges incident to/from this vertex.

ALGORITHM **2.3 Locally optimal algorithm for independent set ordering**

```
Set $S = \emptyset$ and $n_{left} = n$.
While $n_{left} > 0$ do
        Select the vertex with minimum degree in current graph.
        Add this vertex to $S$, then
        Remove it and all of its nearest neighbors from graph
        Update degrees of all vertices
        $n_{left} := n_{left} -$ number_of_removed_vertices.
endwhile
```

There is a striking similarity with the minimal degree ordering algorithm used in sparse Gaussian elimination. The only difference is that the elimination of a node does not introduce what is referred to as fill-in in Gaussian elimination, i.e., the edges of the removed nodes are simply deleted.

There are other similar techniques for generating independent sets. The method described in [33] for an alternative ordering for Gaussian elimination, starts with the observation that constructing a large independent set is equivalent to building a small complement $\bar{S}$ to S. One observes that all the edges of the graph are edges between vertices in $\bar{S}$, i.e., the vertices in $\bar{S}$ form a *vertex cover* of the graph. To find a small vertex cover, Leuze suggests a locally optimal technique which can be viewed as the dual of the previous algorithm. At each step of the procedure, the vertex of *maximum* degree is found and this vertex together with all the incident edges are removed. The process is continued until exhaustion of all edges.

Algorithm **2.4 Vertex Cover Algorithm**

```
Set $\bar{S} = \emptyset$
While (there are still edges left in G) Do
    Select the vertex with maximum degree in current graph.
    Add this vertex to $\bar{S}$, then
    remove it and all its adjacent edges from the graph.
EndWhile
```

One drawback with this type of 'local minimization' techniques illustrated by the last two algorithms is their cost. The fact that the improvements over the simpler procedures 2.1 and 2.2 are likely to be small suggests that it is probably preferable to use these less expensive algorithms in practice. This will be confirmed by our experiments in Section 4.

We now illustrate these algorithms with the help of a little comparison. We consider two test matrices. The first is a 9-point matrix obtained simply by squaring the 5-point discretization of the Laplacian on a 25×25 grid. This matrix of size $N = 625$ has 7629 nonzero elements and it pattern is symmetric. The second matrix is taken from the Harwell-Boeing collection and is called JPWH_991, see [16, 17]. It is of dimension $N = 991$, has a total of 6,027 nonzero elements and its pattern is nonsymmetric.

	First reduction		Second reduction	
Method	N_1	NZ_1	N_2	NZ_2
Alg. 2.1	514	10800	460	15920
Alg. 2.2	523	10414	462	13842
Alg. 2.3	511	10826	454	14868
Alg. 2.4	530	10260	469	13386

Table 1: Performance of 4 different Independent Set Ordering algorithms for first test problem.

Tables 1 and 2 show the results for two reduction steps (see Section 3). In addition, Table 2 illustrates a simple dropping strategy by showing the effect of adding a drop tolerance, called *tol* in the table. When forming the reduced system, a row a_i will be modified by linear combinations of rows corresponding to adjacent nodes in the graph. If $\|a_i\|$ is the 2-norm of a_i, then every element obtained in this transformation is dropped if its magnitude is less than $tol \times \|a_i\|$. In the table N_i is the size of the reduced system obtained at the i-th level reduction, and NZ_i its number of nonzero elements. As can be seen from the tables the difference in the performances is rather small. An interesting observation is that Algorithm 2.3 seems to be best at minimizing the size of the reduced system, i.e., maximizing the size of S, whereas Algorithm 2.4 seems to be better at reducing the amount of fill-in generated in the reduced system. This is somewhat expected because of the nature of the heuristics used. Overall, Algorithm 2.2 seems to perform remarkably well given the simplicity of the underlying heuristic.

2.3 Multicoloring for arbitrary sparse matrices

Multicoloring techniques have often been used in the context of PDEs as a means of introducing parallelism [39, 37, 1, 30, 31, 28]. The goal here is to color the vertices of an arbitrary adjacency

	First reduction		Second reduction	
Method	N_1	NZ_1	N_2	NZ_2
With tol = 0.0				
Alg. 2.1	630	7902	512	12820
Alg. 2.2	603	5640	439	7084
Alg. 2.3	583	6121	433	8381
Alg. 2.4	594	5995	438	7783
With tol = 0.05				
Alg. 2.1	514	8183	455	6481
Alg. 2.2	523	7820	462	6185
Alg. 2.3	511	8209	452	6513
Alg. 2.4	530	7675	465	6734

Table 2: Performance of 4 different Independent Set Ordering algorithms for Matrix JPWH_991.

graph in such a way that two adjacent nodes do not have a common color. In terms of graphs, this means that we must find a partition $S_1, S_2, \ldots, S_k$ of the vertex set V, such that

$$\text{if } (x,y) \in E \text{ with } x \in S_i \text{ and } y \in S_j \quad \text{then } i \neq j \ .$$

Clearly, the red-black ordering is just a particular case with $k = 2$.

As for independent set orderings, finding a multicolor ordering is inexpensive, provided we do not seek to achieve optimality. Multicoloring ideas have been employed in particular to understand the theory of relaxation techniques [51, 49] as well as for deriving efficient alternative formulations of some relaxation algorithms [49, 23]. More recently, they have emerged as useful tools for introducing parallelism in iterative methods, see for example [2, 1, 39, 20, 37]. Multicoloring is also commonly employed in a slightly different form – coloring elements (or edges) as opposed to nodes – in finite elements (or finite volume) techniques especially when using the element-by-element approach [6, 50, 26, 22, 46]. Note that 'multicoloring' is generally based on a graph coloring of some sort and that there are numerous other applications of these techniques which are unrelated to parallel computing. Thus, in a quite different context, a form of multicoloring has also been used in [7] to extract independent columns in a sparse matrix for the purpose of numerically evaluating Jacobians with difference formulas.

There is a rich literature on graph coloring and we refer to [7, 28] for references and a few algorithms that attempt to achieve optimality by some heuristics similar to the ones introduced earlier for independent set orderings. In this paper we will only consider a simple greedy technique for obtaining a multicoloring of an arbitrary graph. Initially, the algorithm assigns a color of zero to each node i. Then, it traverses the graph in the natural order and assigns the smallest positive *admissible* color to each node i visited. Here, an admissible color is a color not already assigned to any of the neighbors of node i. In the following description of the greedy algorithm, we use the notation Adj(i) to represent the set of nodes that are adjacent to node i.

Algorithm **2.5 Greedy multicoloring algorithm**

```
Set $Color(i) = 0$, $i = 1, \dots, N$.
For $i = 1, 2, \dots, N$ Do:
    $Color(i) = \min\{k > 0 \mid k \neq Color(j), \forall\, j \in \mathrm{Adj}(i))\}$.
EndDo
```

At the end of the algorithm, each node i will be assigned the color $Color(i)$. This procedure is illustrated in Figure 2.

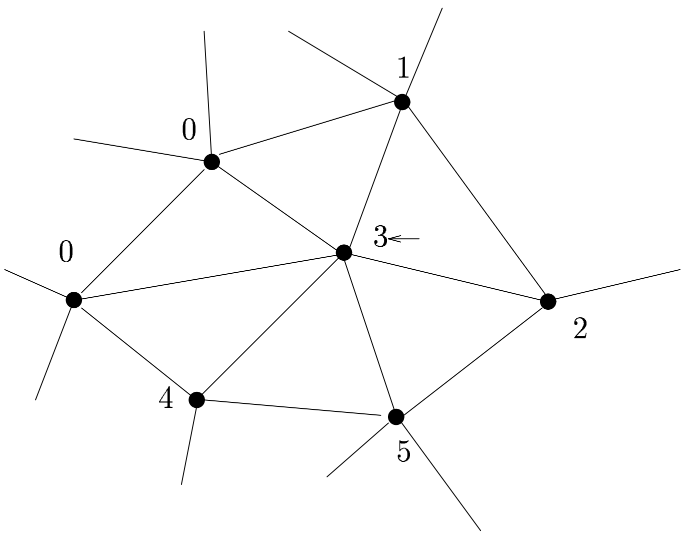

Figure 2: The greedy multicoloring algorithm. The node being colored is indicated by an arrow. It will be assigned color number 3, the smallest positive integer different from 1, 2, 4, 5.

In a manner similar to that for independent set orderings we can traverse the nodes in any order $\{i_1, i_2, \dots i_n\}$ and this initial order of traversal may be important in reducing the number of colors. However, the difference between a poor ordering and a good one is usually small. Here are a few additional properties concerning this greedy algorithm.

- If a graph is bipartite (i.e., two-colorable), the algorithm will find the optimal 2-color (Red-Black) ordering for *breadth-first search* traversals.
- Any chain traversal, i.e., a traversal following a *path through all the nodes* in the graph, will also find the optimal 2-color ordering for any bipartite graph.
- The number of colors needed does not exceed $\nu + 1$ where ν is the maximum degree of all nodes.

Here, we recall that a breadth-first search traversal starts from an arbitrary node and visits its nearest neighbors which form the first level. Then it visits the nearest neighbors of all the nodes in level 1, which have not been visited. These will constitute the the second level. The algorithm continues in this fashion until all nodes have been visited. The proofs of the above properties are straightforward and are omitted. In [7] it is proved that a different ordering referred to as the "Incidence Degree" ordering also achieves optimality for bipartite graphs.

For the natural traversal ordering, we can parallelize this graph coloring algorithm in one of several ways. A simple mechanism implemented in [35] is to impose a local order in which the coloring is to be done. For example, a given node can wait until all its nearest neighbors whose processor numbers are less than its own, have colored their nodes. Once this is done the processor assigns colors to each of its nodes and then sends the color information needed by the nearest neighbors with higher processor number. Several other alternative algorithms exist, see for example [29].

3 ILUM: a Multi-elimination ILU factorization

A parallel direct solver based on performing several successive levels of independent set orderings and reduction was suggested in [33] and in a slightly more general form in [10]. Although the ideas were presented differently, their essence is that when we eliminate the unknowns associated with an independent set we obtain another system, which is a smaller sparse linear system. We can then find an independent set for this reduced system and repeat the process of reduction. We refer to the resulting second reduced system as the second-level reduced system. The process can be repeated recursively a few times. As the level of the reduction increases, the reduced systems gradually lose their sparsity. A direct solution method would consist of continuing the reduction until the reduced system is small enough or dense enough that we can resort to a dense Gaussian elimination to solve it. A sparse direct solution method based on a similar argument was suggested in [33]. In [44] we suggested performing a small number of reduction steps and then switching to a traditional iterative solver, preferably one that has a high level of parallelism such as multicolor SOR, accelerated with GMRES. ILUM is a form of block preconditioning [8, 4] in which reordering is used to ensure that the diagonal blocks to be inverted during the factorization remain diagonal matrices.

After a brief review of the direct solution method based on independent set ordering, we will show how to exploit this approach to derive Incomplete LU factorization strategies based on a drop tolerance.

3.1 Multi-level Reduced Systems

We start by a brief discussion of an *exact* reduction step. Let A_j be the matrix obtained at the j-th step of the reduction, $j = 0, \ldots, nlev$ with $A_0 = A$. Assume that an independent set ordering is applied to A_j and that the matrix is permuted accordingly as follows,

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} \tag{4}$$

where D_j is a diagonal matrix. We can now reduce the system by eliminating the unknowns of the independent set to get the next reduced matrix via the formula,

$$A_{j+1} = C_j - E_j D_j^{-1} F_j \ . \tag{5}$$

Note that we have implicitly performed the block LU factorization

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} = \begin{pmatrix} I & 0 \\ E_j D_j^{-1} & I \end{pmatrix} \times \begin{pmatrix} D_j & F_j \\ 0 & A_{j+1} \end{pmatrix} \tag{6}$$

with A_{j+1} defined above. As a result, in order to solve a system with the matrix A_j we need to perform a forward and backward substitution with the block matrices on the right hand side of (6). The backward solution involves solving a system with the matrix A_{j+1} and the forward solution can be performed with a diagonal scaling and a matrix-vector product.

We can use this block factorization approach recursively until we obtain a system that is small enough to be solved with a standard method, e.g., a dense Gaussian elimination. We must save the transformations used in the elimination process, i.e., the matrices $E_j D_j^{-1}$ and the matrices F_j. The permutation matrices P_j, can also be saved, or we can explicitly permute the matrices involved in the factorization at each new reordering step.

3.2 ILUM

Clearly, the successive reduced systems obtained in the way described above will become more and more expensive to form and store as the number of levels increases. This is due to the fill-in introduced by the elimination process. A common cure for this in developing preconditioners is to neglect some of the fill-in introduced by using a simple dropping strategy as we form the reduced systems. For example, we can drop any fill-in element introduced, whenever its size is less than a given tolerance τ times the 2-norm of the original 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 v. This can be used to precondition the original linear system. Conceptually, the modification leading to an 'incomplete' factorization consists of replacing (5) by

$$A_{j+1} = (C_j - E_j D_j^{-1} F_j) - R_j \tag{7}$$

in which R_j is the matrix of the elements that are dropped in this reduction step. Globally, the algorithm can be viewed as a form of block incomplete LU [8, 5], with permutations.

Thus, we have a succession of incomplete block-LU factorizations of the form

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} = \begin{pmatrix} I & 0 \\ E_j D_j^{-1} & I \end{pmatrix} \times \begin{pmatrix} D_j & F_j \\ 0 & A_{j+1} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & R_j \end{pmatrix} \tag{8}$$

with A_{j+1} defined by (7). We can now recursively find an independent set ordering for the new matrix A_{j+1} and reduce it again in the same manner. Note that we need not save the successive A_j matrices but only the last one that is generated. We must also save the sequence of sparse matrices

$$B_{j+1} = \begin{pmatrix} D_j & F_j \\ E_j D_j^{-1} & 0 \end{pmatrix} \tag{9}$$

which contain the transformation needed at level j of the reduction. We can discard the successive permutation matrices P_j if we apply them to the previous B_i matrices as soon as these permutation matrices are known. We then need only the global permutation which is the product of all these successive permutations. The successive B_j matrices without the diagonal D_j are stored as a succession of sparse matrices in a sparse row format. The diagonals D_j are all stored in an N-dimensional array.

An illustration of the matrices obtained after three reduction steps is shown in Figure 3. The algorithm used for the independent set ordering in the illustration is Algorithm 2.3. The original matrix is a 5-point matrix associated with a 15×15 grid and is therefore of size $N = 225$. Here the matrices B_1, B_2, B_3 (with permutations applied) are shown together with the matrix A_3 occupying the location of the 0 block in (9).

We refer to this incomplete factorization as ILUM (ILU with Multi-elimination). The preprocessing phase consists of a succession of *nlev* applications of the three steps: (1) Finding the Independent Set Ordering, (2) Permuting the matrix, and (3) Reducing it.

ALGORITHM **3.1 ILUM: preprocessing phase**

```
Set $A_0 = A$.
For $j = 0, 1, \ldots, nlev - 1$ Do:
        Find an independent set ordering permutation $P_j$ for $A_j$;
        Apply $P_j$ to $A_j$ to permute it into the form (4);
        Apply $P_j$ to $B_1, \ldots, B_j$;
        Apply $P_j$ to $P_0, \ldots, P_{j-1}$;
        Compute the matrices $A_{j+1}$ and $B_{j+1}$ defined by (7) and (9).
Enddo
```

In the backward and forward solution phases we must solve the last reduced system but we need not solve it accurately. We can for example solve it according to the level of tolerance that we have allowed in the dropping strategy during the preprocessing phase. Observe that since we would like to solve the linear system inaccurately, we should only use an accelerator that allows variations in the preconditioning. Such methods have been developed in [43] and [48]. Alternatively, we can use a fixed number of multi-color SOR or SSOR steps. The implementation of the ILUM preconditioner corresponding to this strategy is rather complicated and involves several parameters.

In order to describe the forward and backward solution we need to introduce some notation. We start by applying the 'global permutation', i.e., the product $P_{nlev-1} P_{nlev-2} \ldots P_0$ to the right hand side. We overwrite the result on the current solution vector, an N-vector which we call x_0. We now partition this vector into

$$x_0 = \begin{pmatrix} y_0 \\ x_1 \end{pmatrix}$$

according to the partitioning (4). The forward step consists of transforming the second component of the right-hand-side as

$$x_1 := x_1 - E_0 D_0^{-1} y_0 .$$

Now x_1 is partitioned in the same manner as x_0 and the forward elimination is continued the same way. Thus, at each step we partition each x_j as

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

A forward elimination step defines the new x_{j+1} using the old x_{j+1} and y_j for $j = 0, \ldots, nlev - 1$ while a backward step defines y_j using the old y_j and x_{j+1}, for $j = nlev - 1, \ldots, 0$. Algorithm 3.2 describes the general structure of the forward and backward solution sweeps. Because we apply the global permutation at the beginning, we need not apply the successive permutations. However, we need to permute the final result obtained back into the original ordering.

ALGORITHM **3.2 ILUM: forward and backward solutions**

```
1.  Apply global permutation to right-hand-side $b$ and copy into $x_0$.
2.  For $j = 0, 1, \ldots, nlev - 1$ do: [Forward sweep]
        $x_{j+1} := x_{j+1} - E_j D_j^{-1} y_j$
    EndDo
3.  Solve with a relative tolerance $\epsilon$:
        $A_{nlev} x_{nlev} := x_{nlev}$.
4.  For $j = nlev - 1, \ldots, 1, 0$ do: [Backward solution]
        $y_j := D_j^{-1}(y_j - F_j x_{j+1})$.
    EndDo
5.  Permute the resulting solution vector back to the original ordering
    to obtain the solution $x$.
```

A major source of difficulty with the use of ILUM lies in its implementation. The implementation issues are similar to those that arise when implementing parallel direct solution methods for sparse linear systems. Some of the issues have been briefly discussed above. Here are some of the important questions that arise, along with some comments or solutions.

1. Should we permute the matrices A_j explicitly or is it preferable to avoid step 3 in Algorithm 3.1? There are two additional burdens if we do not permute the matrices. First, we must store the successive permutations and second, we must apply the permutations during the forward and backward solutions when passing from one level to the other. As was mentioned, in our implementation we explicitly permute the matrices.

2. How to store the successive matrices B_j? In our implementation these are stored in sequence one after the other in a single data structure. As a result the data structure used for storing the successive matrices has two levels of pointers. It is also possible to store all the B_j matrices as a single sparse matrix – the upper part in row format and the lower part in column format. We would then need two sparse data structures and an additional pointer.

3. What number of levels should we use? How do we relate the number of levels to the tolerance used in the factorization? These are difficult questions to answer in a rigorous way. The number of levels is left as a parameter in our implementation.

4 Numerical tests

In this section we provide a few experimental results to: help (i) give an idea of the performance of the ILUM preconditioner when compared with similar techniques and (ii) examine how these performances vary with respect to the type of independent set ordering used. All experiments have been performed on a Cray-2 in single precision (64–bit arithmetic) and the times are in seconds. Although parallelism is not exploited our implementation of the iterative phase, which

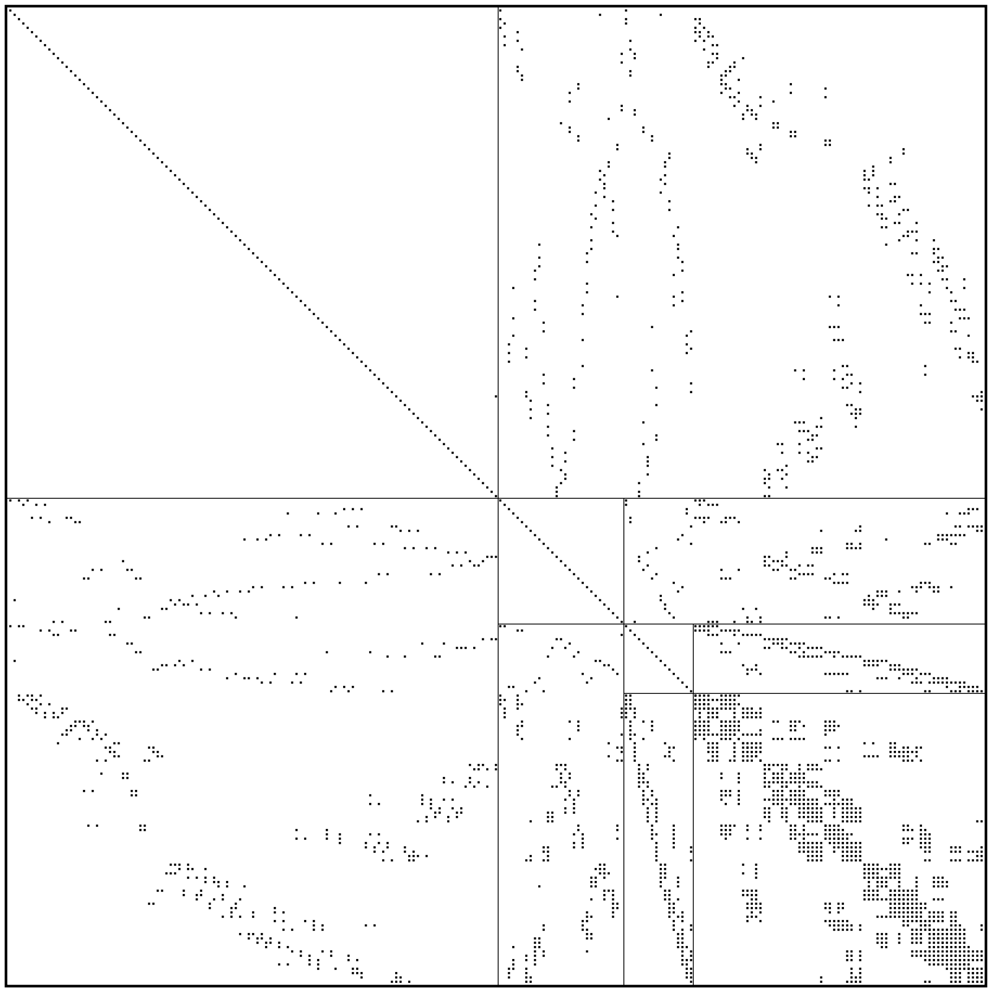

Figure 3: Illustration of the processed matrices obtained from three steps of independent set ordering and reductions.

excludes the preprocessing required to compute the ILU factorization itself, takes advantage of vectorization.

In all tests, we construct the right-hand sides artificially to be of the form $b = Ae$, where the solution e is the vector whose components are all ones. The initial guess is taken to be a random vector. Although some of the test matrices used in these experiments are associated with regular grids, this was not exploited, i.e., the matrices are considered as general sparse. The iterations were stopped as soon as the 2-norm of the residual was reduced by a factor of $\epsilon = 10^{-7}$.

4.1 An elliptic problem

We consider the partial differential equation:

$$-\Delta u + \gamma \left(\frac{\partial e^{xy} u}{\partial x} + \frac{\partial e^{-xy} u}{\partial y} \right) + \alpha u = f \ ,$$

with Dirichlet boundary conditions and $\gamma = 10, \alpha = -60$; discretized with centered differences on a $27 \times 27 \times 27$ grid. This leads to a linear system with $N = 25^3 = 15,625$ unknowns. We refer to the linear system associated with this matrix as Problem 1.

Method		Performance					
Algorithm	*nlev*	tot_T	its_T	Memory	its-out	its-tot
Multicolor	1	3.52	1.54	7.9	4	56
	2	7.30	3.69	142.6	4	52
	3	11.41	5.58	166.5	5	65
Alg-2.1	1	3.26	1.49	7.9	4	56
	2	7.41	4.00	142.9	4	55
	3	10.94	5.64	173.5	5	68
Alg-2.2	1	3.91	1.78	9.3	4	60
	2	7.56	3.77	127.4	4	55
	3	11.29	5.61	161.1	5	68
Alg-2.3	1	4.46	1.52	7.9	4	57
	2	8.95	3.59	143.9	4	51
	3	14.04	5.56	176.4	5	68
Alg-2.4	1	4.05	1.59	5.4	4	55
	2	8.28	3.53	138.1	4	51
	3	14.00	6.17	165.1	5	67

Table 3: Performance of GMRES(10)-ILUM preconditioning for first problem, using different independent set ordering algorithms.

The results obtained with a drop tolerance of $\tau = 0.0001$ in ILUM are shown in Table 3. The last reduced system is solved with GMRES(20) preconditioned by SOR(1) and using a tolerance of $\epsilon = 0.01$ for the stopping criterion. Note that in all the SOR preconditioning operations we use $\omega = 1$ as relaxation parameter. The reordering algorithms tested are indicated on the left column, in which Multicolor refers to taking the first set (color) obtained from the greedy multicoloring algorithm 2.5. The column 'its_T' shows the time required to solve the system excluding the preprocessing phase required to compute the incomplete factorization. The total time, including preprocessing, is shown in the column 'tot_T'. The column 'memory' shows the total memory requirement in thousands of words to store the real values (excluding integer indices) of the incomplete factorization. The numbers 'its-out' refer to the number of outer iterations required, i.e., the number of ILUM preconditioned GMRES steps needed to achieve convergence. The numbers 'its-tot' refer to the total number of inner iterations required, i.e., the overall total number of matrix vector products in the calls to GMRES(20)-SOR(1) needed to solve all the reduced systems occurring throughout the solution of the linear system under consideration.

Method		Performance				
Algorithm	*nlev*	tot_T	its_T	memory	its-out	its-tot
Multicolor	1	1.44	1.17	19.4	6	232
	2	2.68	2.21	25.6	7	194
	3	2.94	2.25	30.3	8	171
	10	3.64	1.56	53.3	11	90
	20	4.71	1.03	71.8	10	43
Alg-2.1	1	1.57	1.33	19.5	6	261
	2	2.70	2.27	25.4	7	185
	3	2.73	2.13	30.3	8	167
	10	3.22	1.31	55.9	10	73
	20	4.41	1.03	75.2	11	39
Alg-2.2	1	2.54	2.22	15.7	7	394
	2	3.70	3.16	22.3	7	318
	3	2.90	2.16	27.9	9	190
	10	3.14	1.09	51.8	9	68
	20	4.43	0.89	70.5	9	35
Alg-2.3	1	2.06	1.60	20.5	7	329
	2	3.25	2.52	25.7	8	232
	3	3.36	2.29	30.5	8	190
	10	4.13	1.14	54.2	10	67
	20	5.762	0.76	72.4	9	31
Alg-2.4	1	2.17	1.84	19.7	7	363
	2	3.37	2.74	24.9	7	241
	3	3.69	2.73	29.6	8	218
	10	4.47	1.24	52.3	9	78
	20	6.49	0.80	70.3	8	33

Table 4: Performance of GMRES(10)-ILUM preconditioning for matrix Sherman 3, using different independent set ordering algorithms.

4.2 Experiments with Harwell-Boeing matrices

We first consider a linear system made up from the matrix Sherman-3 of the Harwell-Boeing collection [16, 17]. This matrix is of dimension $N = 5,005$ and has $nz = 20,033$ nonzero elements. It arises in a IMPES (IMplicit Pressure, Explicit Saturation) simulation of a black-oil reservoir on a $35 \times 11 \times 13$ grid. We refer to the linear system associated with this matrix as Problem 2. Results similar to those of the previous example are shown in Table 4.

	ILUT time	GMRES time	tot. time	its
Problem 1	12.1	2.70	14.8	25
Problem 2	1.69	0.761	2.45	20

Table 5: Performance of GMRES(10)-ILUT(p,τ), with $p = 10$ and $\tau = 0.0001$ using level scheduling for the triangular system solutions.

The number of outer iterations is roughly the same in all cases but the number of inner iterations varies rather substantially. We should point out that the preprocessing to build the incomplete factorization has not been optimized. It is possible to improve performance by exploiting parallelism in the elimination since the main operation in forming the reduced systems consists of a sparse matrix-matrix product. In addition, even if the preprocessing costs remain high, this technique may be appealing for solving linear systems with several right-hand sides on parallel or vector computers. It is worth noting that if we ignore the preprocessing times then for Problem 2, some of the best performances, in term of execution time, are obtained with larger numbers of reductions, a situation which is opposite to that of Problem 1. This depends largely on the parameters used in the factorization.

For comparison, we show in Table 5 typical execution times using an optimized ILUT preconditioned GMRES approach. The ILUT(p, τ) preconditioner described in [45] is a dual-threshold based incomplete LU factorization which performs numerical dropping based on a relative tolerance τ and which retains at most the p largest fill-in elements in L and in U. The optimization of ILUT on the CRAY consists of using level scheduling [3] as well as jagged diagonal data structures to optimize matrix-vector products. Note that if we ignore preprocessing times, the best times achieved with ILUT and ILUM are comparable but the degree of parallelism in ILUM is much higher that in ILUT.

Finally, we show the performance of ILUM and an optimized ILUT preconditioned GMRES approach, both with various parameters, on eight matrices from the Harwell-Boeing collection. The two methods are not comparable for similar values of their parameters. However, the results will give an indication of how the two methods may compare for reasonable choices of the parameters and for unstructured matrices. The sizes (N) of these matrices and their number of nonzero elements (Nz) are shown in the first column of Table 6. Of these matrices, only ORSREG1, SHERMAN1, and SHERMAN5 are regularly structured.

Here, the ILUT(p, τ) preconditioner was used with $\tau = 10^{-4}$ and the level-of-fill parameter p takes the values 0, 5, 10, 15. For ILUM we took the same value for τ and retained at most 20 elements in each row of the reduced system. Since the size of the reduced matrices decreases at each level, it is difficult to compare the number of nonzero elements obtained with a given value of p for ILUM and ILUT. The reduced systems are solved with GMRES(10) with diagonal preconditioning. Only one outer GMRES(10) iteration is performed for each different system. The algorithm used for obtaining the independent sets was Algorithm 2.4. The results are shown in Table 6. In most cases the best iteration times with ILUM are achieved for larger values of the level number. In addition, these times are often better than the level-scheduling implementation of ILUT. The preprocessing times are not optimized for both algorithms so it is difficult to give a comparison with the current implementations of the overall process. These experiments do indicate, however, that a good implementation of ILUM may be a competitive approach on vector supercomputers.

5 Conclusion

The ideas of graph coloring and independent set ordering can be exploited to derive highly parallel incomplete LU factorizations. We have developed such incomplete factorizations and tested them on a Cray computer, exploiting only vectorization. However, the implementation of these techniques on massively parallel computers is likely to be complex. In most cases a good compromise may well be to perform a small number of ILUM reductions, perhaps one or two, and then solve the last reduced system with a multicolor multi-step SOR preconditioned Krylov subspace iteration [44]. Nevertheless, the general idea of independent set orderings is powerful and can certainly be exploited in many other ways than those described in this paper.

Acknowledgements. The author would like to thank the anonymous referees for their help in improving the quality of this paper, and the Minnesota Supercomputer Institute for providing the computer facilities and an excellent research environment to conduct this research.

References

[1] L. Adams and J. Ortega. A multi-color SOR Method for Parallel Computers. In *Proceedings of the 1982 International Conference on Pararallel Processing*, pages 53–56, 1982.

[2] L. M. Adams. *Iterative algorithms for large sparse linear systems on parallel computers.* PhD thesis, Applied Mathematics, University of Virginia, Charlottsville, VA, 22904, 1982. Also NASA Contractor Report 166027.

[3] E. C. Anderson and Y. Saad. Solving sparse triangular systems on parallel computers. *International Journal of High Speed Computing*, 1:73–96, 1989.

[4] O. Axelsson, V. Eijkout, B. Polman, and P. Vassilevski. Incomplete block-matrix factorization iterative methods for convection-diffusion problems. *BIT*, 29:4, 1989.

[5] O. Axelsson and B. Polman. On approximate factorization methods for block matrices suitable for vector and parallel processors. *Lin. Alg. and its Appl.*, 77:3–26, 1986.

[6] M. Benantar and J. E. Flaherty. A Six color procedure for the parallel solution of Elliptic systems using the finite Quadtree structure. In J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, editors, *Proceedings of the fourth SIAM conference on parallel processing for scientific computing*, pages 230–236, 1990.

[7] T. F. Coleman and J. J. More. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20:187–209, 1983.

[8] P. Concus, G. H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Stat. Comput.*, 6:309–332, 1985.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* Mc Graw Hill, New-York, 1990.

[10] P. J. Davis. *Interpolation and Approximation.* Blaisdell, Waltham, Mass., 1963.

[11] T. A. Davis. *A parallel algorithm for sparse unsymmetric LU factorizations.* PhD thesis, University of Illinois at Urbana Champaign, Urbana, IL., 1989.

[12] 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. Math. Anal. Appl.*, 13:944–961, 1992.

[13] 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.

[14] J. J. Dongarra, I. S. Duff, D. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, Philadelphia, 1991.

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

[16] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM trans. Math. Soft.*, 15:1–14, 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] I. S. Duff and G. A. Meurant. The effect of reordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989.

[19] L. C. Dutto. The effect of reordering on the preconditioned GMRES algorithm for solving the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Engineering*, 36:457–497, 1993.

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

[21] H. C. Elman and G. H. Golub. Iterative methods for cyclically reduced non-self-adjoint linear systems. *Math. Comp.*, 54:671–700, 1990.

[22] R. M. Ferencz. *Element-by-element preconditioning techniques for large scale vectorized finite element analysis in nonlinear solid and structural mechanics.* PhD thesis, Applied Mathematics, Stanford, CA, 1989.

[23] R. S. Varga G. H. Golub. Chebyshev semi iterative methods successive overrelaxation iterative methods and second order Richardson iterative methods. *Numer. Math.*, 3:147–168, 1961.

[24] K. Gallivan, A. Sameh, and Z. Zlatev. A parallel hybrid sparse linear system solver. *Computing Systems in Engineering*, 1(2-4):183–195, June 1990.

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

[26] T. J. R. Hughes, R. M. Ferencz, and J. O. Hallquist. Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients. *Computer Methods in Applied Mechanics and Engineering*, 61:215–248, 1987.

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

[28] M. T. Jones and P. E. Plassmann. Parallel iterative solution of sparse linear systems using ordering from graph coloring heuristics. Technical Report MCS-P198-1290, Argonne National Lab., Argonne, IL, 1990.

[29] M. T. Jones and P. E. Plassmann. A parallel graph coloring heuristic. Technical Report MCS-P246-0691, Argonne National Lab., Argonne, IL, 1991.

[30] M. T. Jones and P. E. Plassmann. The effect of many-color orderings on the convergence of methods. Technical Report MCS-P292-0292, Argonne National Lab., Argonne, IL, 1992.

[31] M. T. Jones and P. E. Plassmann. Scalable iterative solution of sparse linear systems. Technical Report MCS-P277-1191, Argonne National Lab., Argonne, IL, 1992.

[32] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–104, New York, 1972. Plenum Press.

[33] R. Leuze. Independent set orderings for parallel matrix factorizations by Gaussian elimination. *Parallel Computing*, 10:177–191, 1989.

[34] J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorizations. *SIAM J. Sci. Stat. Comput.*, 6:1146–1173, 1989.

[35] S. Ma and Y. Saad. Distributed ILU(0) and SOR preconditioners for unstructured sparse linear systems. Technical Report 94–027, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, 1994.

[36] 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(137):148–162, 1977.

[37] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York, 1988.

[38] O. Osterby and Z. Zlatev. *Direct methods for sparse matrices.* Springer Verlag, New York, 1983.

[39] E. L Poole and J. M. Ortega. Multicolor ICCG methods for vector computers. *SIAM J. Numer. Anal.*, 24:1394–1418, 1987.

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

[41] G. Rodrigue and D. Wolitzer. Preconditioning by incomplete block cyclic reduction. *Math. Comp.*, 42:549–565, 1984.

[42] Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. Scient. Stat. Comput.*, 10:1200–1232, 1989.

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

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

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

[46] F. Shakib. *Finite element analysis of the compressible Euler and Navier Stokes Equations.* PhD thesis, Aeronautics Dept., Stanford, CA, 1989.

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

[48] H. A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Numerical Linear Algebra with Applications*, 1:369–386, 1994.

[49] R. S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1962.

[50] V. Venkatakrishnan, H. D. Simon, and T. J. Barth. A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids. Technical Report RNR-91-024, NASA Ames Research Center, Moffett Field, CA, 1991.

[51] D. M. Young. *Iterative solution of large linear systems.* Academic Press, New-York, 1971.

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

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

	ILUM				ILUT			
Matrix	*nlev*	tot_T	its_T	its	p	tot_T	its_T	its
FS7602	2	1.84	1.73	85	0	0.58	0.47	95
$N = 760$	8	0.43	0.17	12	5	0.27	0.10	17
$Nz = 5976$	14	0.40	0.06	6	10	0.29	0.06	9
	20	0.45	0.05	5	15	0.33	0.07	9
ORSIRR1	2	0.66	0.53	26	0	0.19	0.08	15
N= 1030	8	0.41	0.13	9	5	0.18	0.06	8
Nz= 6858	14	0.45	0.09	7	10	0.18	0.05	7
	20	0.50	0.09	7	15	0.18	0.06	7
ORSIRR2	2	0.59	0.48	28	0	0.17	0.07	15
N= 886	8	0.38	0.14	11	5	0.17	0.05	8
Nz= 5970	14	0.39	0.08	7	10	0.17	0.05	7
	20	0.44	0.08	7	15	0.17	0.05	7
ORSREG1	2	0.91	0.62	16	0	0.42	0.17	16
N= 2205	8	0.78	0.19	7	5	0.36	0.08	6
Nz= 14133	14	0.89	0.18	7	10	0.35	0.08	6
	20	1.03	0.18	7	15	0.35	0.08	6
PORES2	2	4.08	3.93	200	0	1.17	1.06	171
N= 1224	8	1.19	0.92	59	5	0.62	0.47	47
Nz= 9613	14	1.11	0.77	52	10	0.55	0.39	33
	20	1.14	0.73	51	15	0.52	0.36	27
PORES3	2	0.54	0.48	46	0	0.12	0.07	22
N= 532	8	0.20	0.06	8	5	0.14	0.05	10
Nz=3474	14	0.21	0.04	6	10	0.18	0.05	8
	20	0.24	0.04	6	15	0.22	0.06	7
SHERMAN1	2	0.21	0.12	7	0	0.22	0.15	31
N=1000	8	0.34	0.07	4	5	0.24	0.09	8
Nz= 3750	14	0.48	0.06	4	10	0.32	0.11	6
	20	0.57	0.05	4	15	0.38	0.15	6
SHERMAN5	2	1.24	0.85	15	0	1.25	0.74	45
N=3312	8	1.62	0.61	11	5	1.40	0.44	19
Nz=20793	14	1.86	0.33	7	10	1.81	0.46	15
	20	2.17	0.25	6	15	2.22	0.52	14

Table 6: Comparison of ILUM and ILUT on a few matrices from the Harwell-Boeing collection