BIT 32 (1992), 442–463.

TOWARDS A COST-EFFECTIVE ILU PRECONDITIONER WITH HIGH LEVEL FILL*

E. F. D'AZEVEDO[1], P. A. FORSYTH[2] and WEI-PAI TANG[2]

[1] *Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, USA.*
[2] *Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.*

Abstract.

There has been increased interest in the effect of the ordering of the unknowns on Preconditioned Conjugate Gradient (*PCG*) methods. A recently proposed Minimum Discarded Fill (*MDF*) ordering technique is effective in finding good *ILU*(*l*) preconditioners, especially for problems arising from unstructured finite element grids. This algorithm can identify anisotropy in complicated physical structures and orders the unknowns in an appropriate direction. The *MDF* technique may be viewed as an analogue of the minimum deficiency algorithm in sparse matrix technology, and hence is expensive to compute for high level *ILU*(*l*) preconditioners.

In this work, several less expensive variants of the *MDF* technique are explored to produce cost-effective *ILU* preconditioners. The Threshold *MDF* ordering combines *MDF* ideas with drop tolerance techniques to identify the sparsity pattern in the *ILU* preconditioners. The Minimum Update Matrix (*MUM*) ordering technique is a simplification of the *MDF* ordering and is an analogue of the minimum degree algorithm. The *MUM* ordering method is especially effective for large matrices arising from Navier-Stokes problems.

AMS(MOS) subject classifications: 65F10, 76S05.

Key words: Minimum discarded fill (*MDF*), threshold *MDF*, minimum updating matrix, matrix ordering, preconditioned conjugate gradient.

1. Introduction.

The use of Preconditioned Conjugate Gradient (*PCG*) methods has proven to be a robust and competitive solution for large sparse matrix problems [1, 19, 25]. A vital step for the successful application of *PCG* methods is the computation of a rapidly convergent preconditioner. Many previous studies have explored this

* This work was supported by the Natural Sciences and Engineering Research Council of Canada, by the Information Technology Research Centre, which is funded by the Province of Ontario, and by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc., through an appointment to the U.S. Department of Energy Postgraduate Research Program administered by Oak Ridge Associated Universities.

Received December 1990. Revised October 1991.

topic and their various approaches can be summarized as follows:

- When the incomplete LU (*ILU*) preconditioner was first proposed it was observed that as the fill level l in an *ILU* decomposition increases, the number of iterations decreases [1, 16, 19, 22, 26]. Unfortunately, the resulting reduction in the number of iterations cannot compensate for the increased cost of the factorization, and the forward and backward solves in each iteration. The most efficient fill level is $l = 1, 2$ in most cases. For some problems, the high-level fill entries in the *ILU* decomposition are numerically very small.
- Based on this observation with the high-level-fill approach, a drop tolerance technique was investigated by Munksgaard [27] and Zlatev [38]. The strategy for deciding the sparsity pattern of a preconditioner was to discard all "small" fill entries that are less than a given tolerance. This approach works well for the model Laplace problem. However, this technique can be sensitive to the initial ordering of the unknowns. Consider the following problem,

$$\frac{\partial}{\partial x}\left(K_x \frac{\partial P}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y \frac{\partial P}{\partial y}\right) = -q \tag{1.1}$$

  with a 5-point discretization on the regular grid, where K_x, $K_y > 0$. When $K_x >> K_y$, the drop tolerance approach will produce a dense banded preconditioner, if a natural row (x first, then y) ordering is used. This results in a large amount of work in each iteration. In contrast, very sparse incomplete factors result when a drop tolerance is used when $K_y >> K_x$ [7] and yet the number of iterations is actually smaller than for $K_x >> K_y$. Some results concerning the effects of ordering on the performance of *PCG* were reported recently in [7, 8, 9, 10, 11, 28].
- Greenbaum and Rodrigue [15] addressed the problem of computing an optimal preconditioner for a given sparsity pattern. The values of the nonzero elements in the preconditioner were determined by numerical optimization techniques for a given sparsity pattern. Kolotilina and Yeremin investigated some least squares approximations for block incomplete factorizations [21]. Their results provide insights of a theoretical nature, but have limited practical application.
- Numerous special techniques that produce good preconditioners for elliptic problems and domain decomposition methods have also been reported [3, 4, 17, 20]. These approaches are very successful for applying *PCG* to the solutions of elliptic PDE's. However, they are difficult to generalize to Jacobians arising from systems of PDE's and to general sparse matrix problems.

In summary, the following factors are observed to have a significant impact on the performance of a preconditioner:

1. The ordering of the unknowns in the original matrix A.
2. The sparsity pattern of the preconditioner M.
3. How closely the spectrum of M resembles that of A.

Motivated by the significant effect of ordering on the preconditioner, we have proposed the Minimum Discarded Fill ($MDF(l)$) ordering (or pivoting strategy) [7] for general sparse matrices. This ordering technique is effective in finding $ILU(l)$ preconditioners, especially for problems arising from unstructured finite element grids. This algorithm can identify anisotropy in complicated physical structures and orders the unknowns in an appropriate direction. Numerical testing of the $MDF(l)$ method shows that it produces better convergence performance than some other orderings. The $MDF(l)$ ordering is successful because it takes into account the numerical values of matrix entries during the factorization process, and not just the topology of the mesh. However, the $MDF(l)$ ordering may be costly to produce. A rough estimate of the cost of $MDF(l)$ ordering is Nd^3, where d is the average number of nonzero elements in each row of the fill matrix, $L + L^t$, and N is the size of the original matrix[1].

Our primary interest is in solution of time dependent or non-linear systems of partial differential equations, such as those arising in reservoir simulation [6] and Navier-Stokes flows [5]. In this case, since a number of similar matrix problems need to be solved, the same ordering and sparsity pattern can be used many times. Hence the cost of determining the ordering and sparsity pattern is amortized over several solves. Although we are determining an ordering of the unknowns during the factorization process, and hence this approach is often termed a pivoting strategy, we intend to use the resulting ordering for several matrix solves. Thus, we refer to these ideas as ordering methods.

In this paper, several variants of the MDF ordering algorithm for use with a drop tolerance incomplete factorization are investigated. Section 2 contains a detailed description of these ordering techniques. Test problems are described in Section 3 and the numerical results and discussion are provided in Section 4, with our concluding remarks in the last section.

Efficient use of the MDF algorithm and its variants for the solution of full non-linear Navier-Stokes equations is discussed in [5]. In this work, we consider the effect of ordering and use of a drop tolerance on some Navier-Stokes Jacobians from a single Newton step.

2. Algorithms.

2.1. *Incomplete LU factorization.*

Given the matrix equation $Ax = b$, then the LDU factorization of an $n \times n$ matrix A (assuming pivoting not required for stability) can be described by the following equations:

[1] To simplify notation, a case of symmetric nonzero structure is presented here.

$$A = A_0 = \begin{bmatrix} d_1 & \beta_1^t \\ \alpha_1 & B_1 \end{bmatrix} = L_1 \begin{bmatrix} d_1 & 0 \\ 0 & A_1 \end{bmatrix} U_1, \tag{2.1}$$

where

$$L_1 = \begin{bmatrix} 1 & 0 \\ \alpha_1/d_1 & I_{n-1} \end{bmatrix}, \quad U_1 = \begin{bmatrix} 1 & \beta_1^t/d_1 \\ 0 & I_{n-1} \end{bmatrix}, \quad A_1 = B_1 - \alpha_1\beta_1^t/d_1. \tag{2.2}$$

At the kth step,

$$A_{k-1} = \begin{bmatrix} d_k & \beta_k^t \\ \alpha_k & B_k \end{bmatrix} = L_k \begin{bmatrix} d_k & 0 \\ 0 & A_k \end{bmatrix} U_k, \tag{2.3}$$

where

$$L_k = \begin{bmatrix} 1 & 0 \\ \alpha_k/d_k & I_{n-k} \end{bmatrix}, \quad U_k = \begin{bmatrix} 1 & \beta_k^t/d_k \\ 0 & I_{n-k} \end{bmatrix}, \quad A_k = B_k - \alpha_k\beta_k^t/d_k. \tag{2.4}$$

Here I_k denotes the $k \times k$ identity matrix, d_k a scalar, α_k and β_k are column vectors of length $n - k$. The matrix A_k is the $(n - k) \times (n - k)$ submatrix that remains to be factored after the first k steps of the factorization.

In the incomplete factorization of matrix A, some of the entries in the factor are discarded to prevent excessive fill and computation. Let matrix F_k contain the discarded entries. Then the incomplete factorization proceeds with the perturbed matrix,

$$B_k - \alpha_k\beta_k^t/d_k - F_k. \tag{2.5}$$

A common choice is to discard the fill that has a "higher fill level" during the incomplete factorization [16]. The simplest strategy is $ILU(0)$ where all new fill is discarded and $ILU(1)$ where only level 1 fill produced by eliminating original nonzeros are retained. The notion of "fill level" will be defined more precisely through the graph model presented in Section 2.2.

2.2. *Graph model.*

In this section we present a graph model [29, 32] for describing the factorization process as a series of node eliminations. The graph model is invaluable in providing an insight into the minimum discarded fill ordering.

To simplify notation, we assume the elimination sequence is $v_1, v_2, \ldots, v_n$. Let graph $G_k = (\mathcal{V}_k, \mathcal{E}_k)$, $k = 0, 1, \ldots, n-1$ be the graph corresponding to matrix $A_k = [a_{ij}^{(k)}]$ of (2.4). The vertex and edge sets are defined as

$$v_k = \{v_{k+1}, v_{k+2}, \ldots, v_n\}, \quad \mathcal{E}_k = \{(v_i, v_j) \mid a_{ij}^{(k)} \neq 0\}. \tag{2.6}$$

We assume each vertex has a self-loop edge $(v_i\, v_i)$ and each edge (v_i, v_j) has a value of $a_{ij}^{(k)}$.

The notion of "fill level" can be defined through reachable sets [14] in the graph G_0. Let $\mathcal{S}$ be a subset of the node set $\mathcal{S} \subset \mathcal{V}_0$ and nodes $u, v \notin \mathcal{S}$. Node u is said to be reachable from a vertex v through $\mathcal{S}$ if there exists a path $(v, u_1, \ldots, u_m, u)$ in graph G_0, such that each $u_i \in \mathcal{S}$, $1 \le i \le m$. Note that m can be zero, so that any adjacent pair of nodes $u, v \notin \mathcal{S}$ is reachable through $\mathcal{S}$. The reachable set of v through $\mathcal{S}$ is denoted by

$$Reach(v, \mathcal{S}) = \{u \mid \text{is reachable from } v \text{ through } \mathcal{S}\}. \tag{2.7}$$

Let $\mathcal{S}_k$ be the set of eliminated nodes so far, $\{v_1, \ldots, v_k\}$, and let $i, j > k$, $v_j \in Reach(v_i, \mathcal{S}_k)$ with the *shortest* path $(v_i, u_1, \ldots, u_m, v_j)$, and nodes u in $\mathcal{S}_k$. We define the fill level for the node pair (v_i, v_j) in G_k to be the length of the shortest path from v_i to v_j minus one, i.e. $\text{level}_{ij}^{(k)} = m$. We define initially

$$\text{level}_{ij}^{(0)} = \begin{cases} 0 & \text{if } a_{ij} \neq 0 \\ \infty & \text{otherwise} \end{cases} \tag{2.8}$$

Since $\text{level}_{ij}^{(k)}$ is defined by reachable sets through $\{v_1, \ldots, v_k\}$, as more nodes are eliminated, there may be a shorter path between v_i and v_j. Thus as the elimination proceeds, the fill levels are given by

$$\text{level}_{ij}^{(k)} := \min(\text{level}_{ik}^{(k-1)} + \text{level}_{kj}^{(k-1)} + 1, \text{level}_{ij}^{(k-1)}). \tag{2.9}$$

It is possible to define a fill level independent of k, if the order of the unknowns is predetermined. In this application, however, the order of the unknowns are dynamically changing during the incomplete factorization. A predetermined level, therefore, is not practical.

The elimination of v_k to form A_k can be modeled as a graph transformation [32],

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} - \dfrac{a_{ik}^{(k-1)} a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}} & \text{if } (v_i, v_k) \text{ and } (v_k, v_j) \in \mathcal{E}_{k-1} \\ \mathbf{a}_{ij}^{(k-1)} & \text{otherwise.} \end{cases}$$

Note that if $a_{ij}^{(k-1)}$ is a zero entry, $(v_i, v_j) \notin \mathcal{E}_{k-1}$, then the elimination of their common neighbor v_k would create at position $a_{ij}^{(k)}$, a new fill of value $0 - a_{ik}^{(k-1)} a_{kj}^{(k-1)} / a_{kk}^{(k-1)}$.

In an $ILU(l)$ factorization, only fill entries with fill-level less than or equal to l are kept. More precisely, the factorization proceeds with a perturbed A_k (2.5),

$$A_k = B_k - C_k - F_k, \qquad C_k = \alpha_k \beta_k^t / d_k = [c_{ij}^{(k)}],$$

where F_k contains the discarded fill entries,

$$F_k = [f_{ij}^{(k)}], \quad f_{ij}^{(k)} = \begin{cases} 0 & \text{if } b_{ij}^{(k)} \neq 0 \\ -c_{ij}^{(k)} & \text{if } \text{level}_{ij}^{(k)} > l. \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

2.3. *MDF(l) algorithm.*

The MDF(l) ordering is motivated by the observation that a small discarded fill matrix F_k in (2.5) would produce a more "authentic" factorization for matrix A. We define the *discard value* for eliminating the kth node as the Frobenius norm of the discarded fill matrix F_k of (2.10),

$$discard(v_k) = \|F_k\|_F = \left(\sum_{i \geq 1} \sum_{j \geq 1} f_{ij}^{(k)^2} \right)^{1/2} \tag{2.11}$$

Note that the discard value for another node v_i can be obtained similarly by first performing a symmetric exchange of nodes v_k and v_i.

The basic idea of the minimum discarded fill algorithm (*MDF(l)*) is to choose the next pivot node that has the minimum discard value. It is an attempt to obtain locally one of the best approximations of the complete decomposition. Since the updating matrix, $C_k = \alpha_k \beta_k^t / d_k$, affects a few rows and columns of B_k, only the discard values of the neighbors of v_k need be recomputed. A description of the *MDF(l)* algorithm is given in Algorithms 2.1 and 2.2. In case of a tie in discard values, we simply picked the first node that achieved the discard value. Other tie breaking strategies are discussed in [7].

During the *MDF(l)* ordering process, the nonzero pattern of L and the number of fill entries in an *ILU(l)* decomposition depend on the ordering and are rather dynamic. Consequently, the complexity analysis of *MDF(l)* is still an open problem. However, a rough estimate can help in estimating the direction for possible improvements in efficiency.

Initialization:

$A_0 := A$
for each $a_{ij} \neq 0$
  $\text{level}_{ij}^{(0)} := 0$
end
for each node v_i
  Compute the discarded value $discard(v_i)$
  (see procedure discard_val).
end

for $k = 1 \ldots n-1$

1. Choose as the next pivot node v_m in matrix A_{k-1} which has minimum $discard(v_m)$ (break ties by choosing earlier node).
2. Update the decomposition,

$$A_k := B_k - C_k - F_k, \quad C_k = \alpha_k \beta_k^t / d_k$$

where $F_k = [f_{ij}^{(k)}]$ is given by (2.10) and P_k is the permutation matrix to exchange v_m to first position,

3. Update the fill-level of elements in A_k by (2.9).
   for each v_i a neighbor of v_m, $(v_i, v_m) \in \mathscr{E}_{k-1}$
   for each v_j a neighbor of v_m, $(v_m, v_j) \in \mathscr{E}_{k-1}$

$$\text{level}_{ij}^{(k)} := \min(\text{level}_{im}^{(k-1)} + \text{level}_{mj}^{(k-1)} + 1, \text{level}_{ij}^{(k-1)})$$

   end
   end
4. Update the discard values of v_m's neighbors.
   for each v_i a neighbor of v_m in $\mathscr{E}_{k-1}$

$$discard(v_i) = \|F_{k+1}\|_F, \quad P_{k+1} A_k P_{k+1}^t = \begin{bmatrix} d_{k+1} & \beta_{k+1}^t \\ \alpha_{k+1} & B_{k+1} \end{bmatrix}$$

   where F_{k+1} is given by (2.10), $C_{k+1} = \alpha_{k+1}\beta_{k+1}^t/d_{k+1}$ and P_{k+1} is a permutation matrix to exchange v_i to first position (see procedure discard_val).

end
end

Algorithm 2.1. Description of $MDF(l)$ algorithm.

Procedure discard_val (. .)
   $discard(v_i) := 0$
   for each neighbor v_j of v_i in $\mathscr{E}_k$
   for each p such that $\{a_{i,p}^{(k)} \neq 0, a_{j,p}^{(k)} = 0, \text{level}_{jp}^{(k+1)} < l\}$
   $discard(v_i) = discard(v_i) + (a_{i,p}^{(k)} a_{j,i}^{(k)}/a_{i,i}^{(k)})^2$
   end
   end
end

Algorithm 2.2. Description of procedure for calculating discard value.

Assume matrix A is $n \times n$, the average number of non-zero elements in each row is c, and the average number of non-zero elements in each row in $L + U$ is d. It is clear that $c \leq d$ and d depends on the fill-level l. The complexity of the initialization step is about $O(c^2 n)$. One elimination step costs $O(d^2)$ while updating the discard values of neighbor nodes costs $O(d^3)$. Therefore by a rough estimate, the complexity for $MDF(l)$ is dominated by the cost of updating the discard values, which is about $O(d^3 n)$. If we have a rather large node degree c in the matrix A, d can grow quickly with level l.

2.4. *Effect of anisotropies on drop tolerance ILU.*

Consider the matrix derived from the partial differential equation (1.1). This problem was discretized on a 30 × 30 uniform grid over the unit square with Neumann boundary conditions, using the standard five point molecule. The value of the right hand side of equation (1.1) was zero everywhere, except $q = +1$ at the lower left hand corner, and $q = -1$ at the upper right hand corner. Although this problem is only semi-definite, CG methods still converge [13].

This matrix was solved by a *PCG* method with an *ILU* factorization based on a drop tolerance. New fill entries were ignored if

$$|a_{ij}^{(k)}| \leq 0.001 \min(|a_{ii}|, |a_{jj}|). \tag{2.12}$$

Natural (x-y) row ordering was used, and the convergence tolerance was to reduce the initial l_2 residual by 10^{-6}. Table (2.1) shows the results with two different values of (K_x, K_y). Although the incomplete factor L for $(K_x, K_y) = (100, 1)$ has about four times more fill than for $(K_x, K_y) = (1, 100)$, the preconditioner requires more iterations and computing time for convergence.

Another way of looking at this result is to note that for very anisotropic problems, the ordering of the unknowns has a large effect on the size of the elements of the incomplete factors. This suggests that the ordering of the unknowns is important if a drop tolerance technique is used. Table (2.1) indicates that if (physically) there is a strong coupling in the y-direction, it is best to order along the x-direction first. This is counter-intuitive.

Table 2.1. *Comparison of two problems*

Problem	Nonzeros in L	Iterations	Num. Fact. time	Sol. time
$(K_x, K_y) = (100,1)$	10330	17	0.37	2.19
$(K_x, K_y) = (1,100)$	2705	13	0.13	1.12

In order to explain these results further, the entries of the first row in A_k corresponding to a typical node $k = 435$ (position $(x, y) = (15/29, 15/29)$) in the center of the domain are shown in Table 2.2. Table 2.2 shows that for $(K_x, K_y) = (100, 1)$ the fill that appears adjacent to the outermost band decays slowly as we move towards the main diagonal. On the other hand, for $(K_x, K_y) = (1, 100)$, the entries decay rapidly for high level fill adjacent to the outermost band (right half of Table 2.2).

The situation is a bit more complicated for the fill which appears adjacent to the main diagonal of the matrix (left half of Table 2.2). Note that the first fill which appears in the inner band is level three. For $(K_x, K_y) = (100, 1)$, this fill is much smaller than the adjacent level zero entry. However, the next higher level fill entries decay slowly as we move away from the main diagonal. In comparison, the size of

Table 2.2 *Fill entries in first row of* A_{435} *for ILU* (8). *The grid is* 30×30.

Fills adjacent to main diagonal				Fills adjacent to outer diagonal			
column	fill level	$K_x = 100$ $K_y = 1$	$K_x = 1$ $K_y = 100$	column	fill level	$K_x = 100$ $K_y = 1$	$K_x = 1$ $K_y = 100$
435	0	113.67	112.82	457	8	−0.4110	−0.0011
436	0	−100.10	−5.1987	458	7	−0.4505	−0.0068
437	3	−0.0758	−0.9331	459	6	−0.4974	−0.0186
438	4	−0.0568	−0.3045	460	5	−0.5527	−0.0494
439	5	−0.0404	−0.1062	461	4	−0.6173	−0.1316
440	6	−0.0266	−0.0375	462	3	−0.6927	−0.3559
441	7	−0.0152	−0.0132	463	2	−0.7803	−1.0395
442	8	−0.0063	−0.0045	464	1	−0.8820	−4.6081
				465	0	−1.0000	−100.00

the fill terms for $(K_x, K_y) = (1, 100)$ is larger than those for $(K_x, K_y) = (100, 1)$ until fill level seven is encountered.

It is interesting to observe the mechanics of applying the drop criterion (2.12) to this matrix. Recall that all diagonals are of size 200. For $(K_x, K_y) = (100, 1)$ most of the fill adjacent to the outermost band will be kept, while all the fill adjacent to the main diagonal (even the first level three fill) will be dropped. On the other hand for $(K_x, K_y) = (1, 100)$, fill is kept more evenly in both bands. In particular, much less fill remains due to the fast decay.

From the properties of Gaussian elimination on an M-matrix, the following lower bound for the entries $a_{ij}^{(k)}$ of A_k in $ILU(l)$ decomposition exists.

THEOREM 2.1. *Let A be a symmetric M-matrix, $a_{ij}^{(k)}$ be a new fill entry in A_k and $l \geq level_{ij}^{(k)} \geq 1$, $i, j > k$. If*

$$(v_i, v_{i_1}, \ldots, v_{i_m}, v_j), \quad v_{i_1}, \ldots, v_{i_m} \in \mathcal{S}_k, \quad \mathcal{S}_k = \{v_1, \ldots, v_k\} \tag{2.13}$$

is a path from v_i to v_j through $\mathcal{S}_k$, then

$$|a_{ij}^{(k)}| \geq \frac{|a_{ii_1} a_{i_1 i_2} \ldots a_{i_m j}|}{d_{i_1} d_{i_2} \ldots d_{i_m}}, \quad \text{where } d_i = a_{ii}.$$

Since A is an M-matrix, all contributions to the fill $a_{ij}^{(k)}$ have the same sign. A simple induction on k is sufficient for the proof of the result above. Some useful properties of Gaussian elimination on an M-matrix are given in [26]. Note that $l_{ij} = a_{ij}^{(j-1)}/a_{jj}^{(j-1)}$, hence for $l = \infty$ this theorem also provides a lower bound for the complete LU decomposition.

If $K_x >> K_y$, all new higher level fill entries adjacent to the main diagonal of the factorization will have a shortest path for which at least two edges lie in the y-direction. The first possible fill is level three. This indicates the cause of the rapid

initial decay. However, the decay of the fill in this direction is very slow after the initial fill. If a drop tolerance is used, then the *ILU* factorization can either drop all these fills or keep most of them. Both scenarios lead to poor *PCG* behavior.

This example shows that for anisotropic problems, it is necessary to consider the ordering of the unknowns if a drop tolerance preconditioner is used. A poor ordering may result in a large number of nonzeros in L and U of the *ILU* factorization, and yet exhibit poor convergence behavior. This appears to be due to a tendency to keep a large amount of some high level fills, while dropping some low level fill, if a poor ordering is selected.

2.5. *Threshold MDF(l) algorithm.*

When level l increases, the average number of nonzeros per row in L of the *ILU* factorization may grow quickly. Fortunately, many of the high-level fill entries are "small". For example, an *ILU*(6) factorization for problem (1.1) with $(K_x, K_y) = (1.100)$ has 9425 nonzeros in L and 3428 entries smaller than 0.001. This observation suggests that there are some higher level fill entries which do make an important contribution to the performance of the preconditioner, but much of the higher level fill entries can be safely ignored. Based on these observations, a new heuristic threshold *MDF(l)* is proposed. As in (2.10), the discard matrix F_k is taken to be

$$f_{ij}^{(k)} = \begin{cases} 0 & \text{if } b_{ij}^{(k)} \neq 0 \\ -c_{ij}^{(k)} & \text{if } |c_{ij}^{(k)}| < \varepsilon \min(R_i, R_j) \quad \text{or } \text{level}_{ij}^{(k)} > l, \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

$$R_i = \max_{m=1\ldots n} (|a_{im}|) = \|a_{i*}\|_\infty \tag{2.15}$$

and ε is a given relative threshold. Note that if A is diagonally dominant and scaled to have unit diagonal, then the absolute threshold criterion $|a_{ij}^{(k)}| \leq \varepsilon$ is equivalent to using in (2.14) $|a_{ij}^{(k)}| \leq \varepsilon(|a_{ii}a_{jj}|)^{1/2}$ and $(|a_{ii}a_{jj}|)^{1/2} \geq \min(|a_{ii}|, |a_{jj}|)$. Munksgaard [27] has used the drop criterion

$$|a_{ij}^{(k)}| \leq \varepsilon(|a_{ii}^{(k-1)}a_{jj}^{(k-1)}|)^{1/2} \tag{2.16}$$

for symmetric positive definite systems. However, matrices arising from the solution of Navier-Stokes equations commonly have zero diagonals and entries with large variations. The question of the best criteria for drop tolerance is still unsettled. We have used the criterion (2.15) because of its simplicity and relationship to scaling.

The combination, $l = \infty$, $10^{-4} \leq \varepsilon \leq 10^{-3}$, is often a good choice in our test problems. If we choose a pivoting strategy based only on numerical values, i.e. $l = \infty$, then there is no overhead for recording the fill-level, and there can be an extra 5–15% saving in ordering time. Of course, a drop tolerance technique requires

dynamic storage, and therefore is more costly than a straightforward fill level *ILU*. As mentioned previously, we intend to use the same ordering and *ILU* sparsity pattern for several solves [5]. Consequently, we have implemented the drop tolerance *ILU* as a two stage process. The first stage uses dynamic storage (linked lists) to determine the ordering and the *ILU* sparsity pattern. This sparsity pattern is converted to static (packed) storage format, and then stored. For future matrix solves, the numerical *ILU* factorization is carried out using the previously determined static structure.

2.6. *Minimum update matrix algorithm.*

The most costly part of an *MDF* type ordering is the traversal of adjacency lists in the determination of sparsity pattern of A_k and B_k in the computation of discard values (see Algorithm 2.2). If the average node degree is large, the high cost of updating discard values renders the *MDF* ordering impractical. We propose a simpler scheme, the *Minimum Update Matrix* (*MUM*) ordering, which is motivated by the Markowitz algorithm [24] in sparse Gaussian elimination. The computation of the discard value is

$$\text{(2.17)} \qquad \textit{discard}(v_k) = \|\tilde{C}_k\|_F, \quad \tilde{c}_{ij}^{(k)} = \begin{cases} 0 & \text{if } i = j \\ c_{ij} & \text{otherwise} \end{cases}, \quad C_k = \alpha_k \beta_k^t / d_k$$

where the matrix $\tilde{C}_k = [\tilde{c}_{ij}^{(k)}]$ contains the off-diagonal entries of $C_k = [c_{ij}^{(k)}]$.

Note that *MDF*(*l*) and threshold *MDF* orderings have discard value computations intimately coupled to the incomplete factorization strategy. The discard value is the norm of actual discarded fill matrices. The *MUM* ordering uses a simpler measure for the discard value and has less coupling to the factorization strategy.

3. Test Problems.

Numerical tests for all the ordering algorithms were carried out on a variety of problems. For problems 2 and 3 below, the matrices are only positive semi-definite. However, the conjugate gradient method still converges to a solution [13].

3.1. *Problem* 1 (*LAPD5*).

The first problem is Laplace's equation on the unit square with Dirichlet boundary conditions, as used in [10]. The usual five-point finite difference discretization was used on a regular 30 × 30 grid.

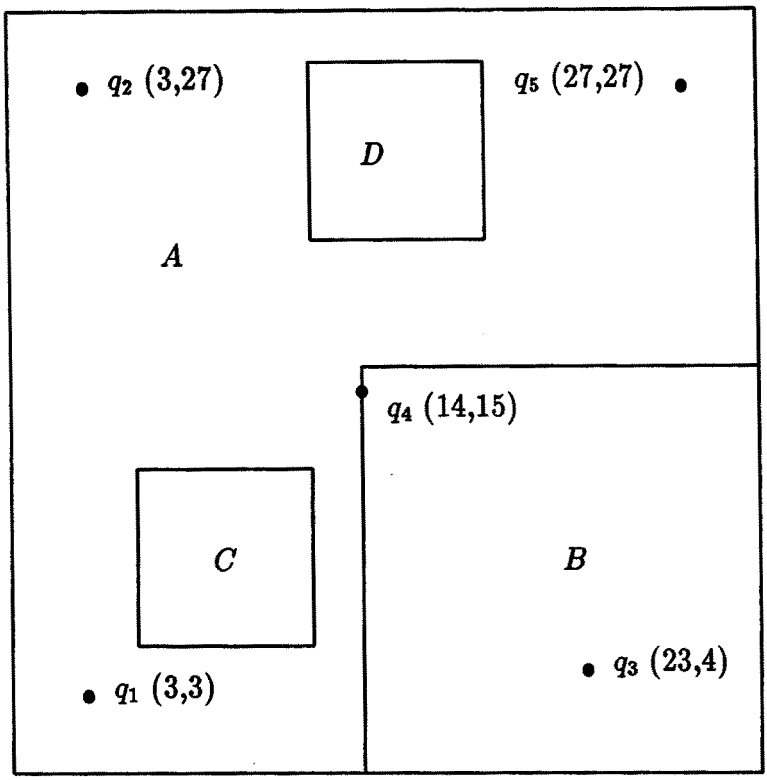

Fig. 3.1. Stone's third problem.

3.2. *Problem* 2 (*STONE*).

This is Stone's third problem [34]. The equation

$$\frac{\partial}{\partial x}\left(K_x \frac{\partial P}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y \frac{\partial P}{\partial y}\right) = -q \tag{3.1}$$

was discretized on the unit square using a finite difference technique, with Neumann boundary conditions. If the node spacing is $h = 1/30$, then

$$x_i = ih, \quad y_j = jh, \quad 0 \le i,j \le 30.$$

We refer to the location of the source/sink terms by $q(x_i, y_i) = q(i,j)$ in the following and in Fig. 3.1.

The values of K_x, K_y and q are:

$$(K_x, K_y) = \begin{cases} (1,1) & \text{if } (x_i, y_j) \in A, \quad \text{elsewhere} \\ (1,100) & \text{if } (x_i, y_j) \in B, \quad 14 \le i \le 30, 0 \le j \le 16 \\ (100,1) & \text{if } (x_i, y_j) \in C, \quad 5 \le i \le 12, 5 \le j \le 12 \\ (0,0) & \text{if } (x_i, y_j) \in D, \quad 12 \le i \le 19, 21 \le j \le 28 \end{cases} \tag{3.2}$$

$$q_1(3,3) = 1.0, \quad q_2(3,27) = 0.5, \quad q_3(23,4) = 0.6, \tag{3.3}$$
$$q_4(14,15) = -1.83, \quad q_5(27,27) = -0.27.$$

A 31 × 31 grid was used, and an harmonic average was used for discontinuities in K_x and K_y [2].

3.3. *Problem* 3 (*ANISO*).

This problem uses the same equation as in (3.1) except the values of K_x and K_y:

$$(K_x, K_y) = \begin{cases} (100,1) & \text{if } 0 \le x \le 1/2, \quad 0 \le y \le 1/2 \\ (1,100) & \text{if } 1/2 \le x \le 1, \quad 0 \le y \le 1/2 \\ (100,1) & \text{if } 1/2 \le x \le 1, \quad 1/2 \le y \le 1 \\ (1,100) & \text{if } 0 \le x \le 1/2, \quad 1/2 \le y \le 1 \end{cases}$$

A 30 × 30 grid was used.

Test problems (4–6) are derived from two and three dimensional pressure equations arising in groundwater contamination simulations [12, 18]. The pressure equation is essentially equation (3.1). Since the actual values of K_x, K_y, q and the boundary conditions are quite complicated, only a brief description of these problems will be given.

3.4. *Problem* 4 (*REFINE2D*).

A finite element method using linear triangular basis functions was used to discretize the problem. In this example, K_x and K_y are constant. The triangulation was such that the resulting equation was an M-matrix [12].

3.5. *Problem* 5 (*FE2D*).

A finite element method using linear triangular basis functions was used also on this problem. However, K_x and K_y (equation (3.1)) varied by four orders of magnitude. The grid was defined by constructing a distorted quadrilateral grid, and then triangulating in an obvious manner. A Delaunay-type edge swap [12] was used to produce an M-matrix.

3.6. *Problem* 6 (*FE3D*).

This problem is a three-dimensional version of equation (3.1). A finite element discretization was used, with linear basis functions defined on tetrahedra. The absolute permeabilities (K_x, K_y, K_z) varied by eight orders of magnitude (this model was derived from actual field data). The nodes were defined on a 25 × 13 × 10 grid

(3250 nodes) of distorted hexahedra, which were then divided into tetrahedra. The resulting matrix was *not* an M-matrix, and the average node connectivity was fifteen. In general, it is not possible for a given node placement to obtain an M-matrix in three dimensions if linear tetrahedral elements are used [23].

3.7. *Problem* 7 (*NS2D*).

This problem is derived from a finite volume discretization of the incompressible Navier-Stokes equations [30]. A 40×40 grid was used to model the backward step problem, with a Reynold's number $\mathrm{Re} = 1000$. The matrix was generated from the Jacobian produced for the second Newton iteration. The problem was unsymmetric, so CGSTAB acceleration [36, 37] was used.

3.8. *Problem* 8 (*NS3D*).

This problem was derived from a small three dimensional finite element discretization of the compressible Navier-Stokes equations [31]. Use of a finite element method in three dimensions resulted in a very large computational molecule. The test matrix was generated from a Jacobian produced near the start of a pseudo-time solution of the steady-state problem. CGSTAB acceleration was used here as well.

4. Numerical results.

Computational experiments were performed on a SPARC SLC workstation using double precision. The convergence criterion,

$$\|r^k\|_2 \leq tol\, \|r^0\|_2, \qquad tol = 10^{-6}$$

was used, where r^k was the residual vector after the kth iteration in the conjugate gradient acceleration. In all cases, the initial vector x^0 was chosen to be the zero vector. Some tests were carried out using a random initial vector. The results were qualitatively similar, and hence will not be reported. The right hand sides for the Navier-Stokes problems were the residuals of the non-linear equations. Conjugate gradient acceleration was used for symmetric problems, and CGSTAB [36, 37] was used for unsymmetric problems (for the Navier-Stokes problems, GMRES [33] was uncompetitive with CGSTAB).

Table 5.1 shows the ordering times for $MDF(l)$ orderings. The notation $MDF(l, \varepsilon)$ denotes MDF level l, and drop tolerance ε. Note that for highly anisotropic problems (ANISO), or for the finite element problems, the drop tolerance approach has much less fill in the ILU factors, and takes less time to compute than the high

Table 5.1. *MDF(l) ordering time and number of nonzeros in L.*

Ordering Time (number of nonzeros, in thousands)						
Ordering	LAPD5 (neq = 900)	STONE (neq = 961)	ANISO (neq = 900)	REFINE2D (neq = 1161)	FE2D (neq = 1521)	FE3D (neq = 3250)
MDF(0)	0.86(1.7)	0.84(1.9)	0.87(1.9)	1.19(2.5)	2.06(4.4)	17.27(19.7)
MDF(1)	1.71(3.4)	1.86(3.7)	1.83(3.7)	2.41(4.6)	4.87(7.9)	153.13(49.6)
MDF(2)	1.84(3.6)	1.89(3.8)	1.92(3.8)	3.14(5.3)	8.79(10.6)	1013.3(92.2)
MDF(3)	3.97(5.0)	4.08(5.4)	4.01(5.3)	6.15(7.1)	17.56(13.5)	4729.6(143.1)
MDF(4)	4.35(5.3)	4.48(5.6)	4.45(5.6)	8.05(7.7)	27.81(15.6)	N/A
MDF(5)	6.57(6.1)	6.56(6.5)	6.50(6.4)	11.81(8.7)	43.53(17.5)	N/A
MDF(6)	7.45(6.3)	7.25(6.7)	7.42(6.7)	15.22(9.2)	62.36(19.1)	N/A
$MDF(\infty, 10^{-3})$	12.03(7.6)	8.23(6.7)	3.23(4.9)	4.78(6.5)	7.33(9.4)	125.17(44.7)
$MDF(\infty, 10^{-4})$	32.99(10.8)	19.63(9.1)	4.93(5.9)	9.63(8.6)	23.22(14.7)	469.4(69.4)

Table 5.2. *Total solution time and number of iterations.*

Solution Time (number of iterations)						
Ordering	LAPD5	STONE	ANISO	REFINE2D	FE2D	FE3D
ILU(0)	3.05(44)	4.84(66)	5.42(74)	5.84(63)	7.41(54)	21.59(52)
ILU(1)	2.16(28)	3.09(38)	4.37(55)	3.80(38)	5.55(38)	17.04(31)
MDF(0)	3.04(44)	4.68(64)	5.32(73)	5.68(61)	5.42(39)	13.93(32)
MDF(1)	1.74(21)	2.26(26)	2.50(29)	2.99(28)	4.86(30)	16.44(24)
MDF(2)	1.77(21)	2.20(25)	2.52(29)	3.03(27)	4.10(22)	22.58(18)
MDF(3)	1.31(13)	1.67(16)	2.11(21)	2.56(20)	3.99(18)	40.69(15)
MDF(4)	1.32(13)	1.70(16)	2.07(20)	2.54(19)	3.47(14)	N/A
MDF(5)	1.37(11)	1.53(13)	1.72(15)	2.39(16)	3.54(13)	N/A
MDF(6)	1.29(10)	1.58(13)	1.68(14)	1.96(12)	3.49(11)	N/A
$MDF(\infty, 10^{-3})$	1.15(8)	1.37(11)	1.94(20)	1.95(15)	3.10(17)	9.61(13)
$MDF(\infty, 10^{-4})$	1.20(5)	1.28(7)	1.46(13)	1.60(10)	2.61(10)	11.76(9)

level *ILU* factorizations without drop tolerance. For the $MDF(l, \varepsilon)$ methods, the ordering time includes the time for determining the sparsity patterns of the incomplete factors.

Table 5.2 shows the total solution time (numerical factorization, forward and back solve and acceleration time) for the symmetric problems. *ILU*(*l*) refers to a level *l* incomplete factorization, using the original ordering. Natural *x*-*y* ordering was used for LAPD5, STONE, ANISO, while Reverse Cuthill-McKee (RCM) ordering was used for REFINE2D, FE2D, and FE3D.

For the five point operator problems (LAPD5, STONE, ANISO) and the two dimensional finite element problem (REFINE2D, FE2D), the total solution time keeps decreasing as *l* increases for the *MDF*(*l*) orderings. On the other hand, the

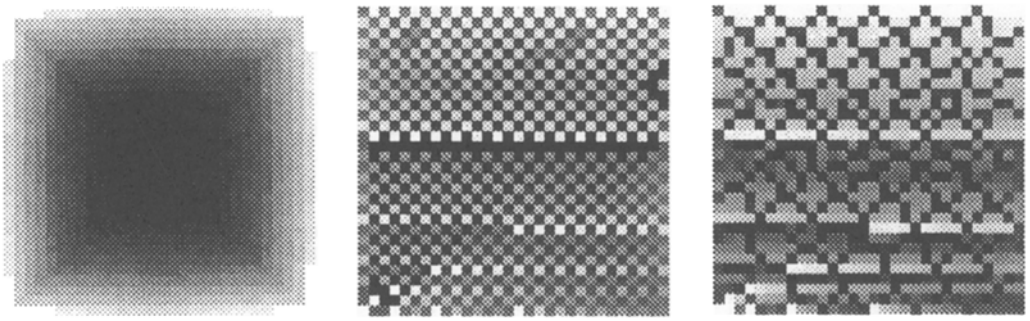

Fig. 3.2. Ordering pictures for Laplace problem (LAPD5). Lightest, first; darkest, last

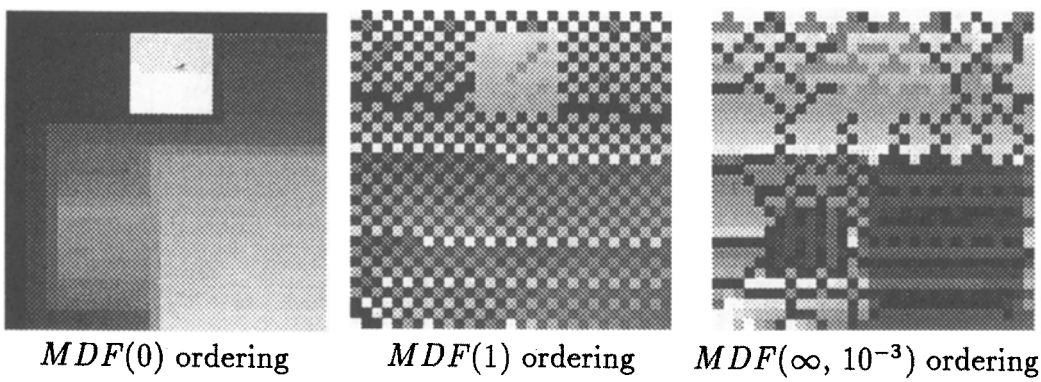

Fig. 3.3 Ordering pictures for Stone's problem (STONE). Lightest, first; darkest, last.

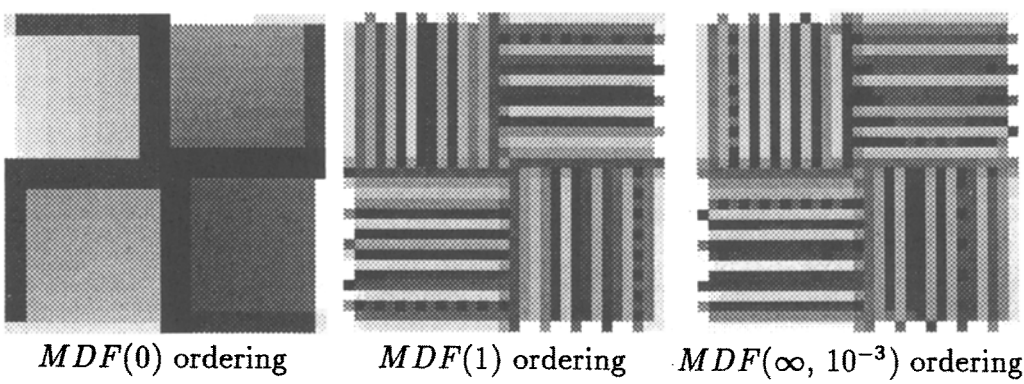

Fig. 3.4 Ordering pictures for Anisotropic problem (ANISO). Lightest, first; darkest, last.

Table 5.3. *Summary of the test results for Minimum Updating Matrix (MUM) ordering.*

		LAPD5			STONE		
Algorithm	(l, ε)	Ordering time	Nonzeros in L	Solution time	Ordering time	Nonzeros in L	Solution time
MUM	(0, 0.0)	0.71	1740	3.04(44)	0.74	1860	4.69(64)
MUM	(1, 0.0)	0.99	2581	2.10(27)	1.00	2832	3.48(43)
MUM	$(\infty, 10^{-3})$	2.58	7481	1.37(11)	2.95	8045	2.50(21)
T-MDF	$(\infty, 10^{-3})$	12.03	7616	1.15(8)	8.23	6666	1.37(11)
ILU(1)	(1, 0.0)	0	2581	2.16(28)	0	2760	3.09(38)
		ANISO			REFINED		
Algorithm	(l, ε)	Ordering time	Nonzeros in L	Solution time	Ordering time	Nonzeros in L	Solution time
MUM	(0, 0.0)	0.77	1860	5.32(73)	1.01	2480	5.86(63)
MUM	(1, 0.0)	0.96	2762	5.21(66)	1.23	3607	5.23(53)
MUM	$(\infty, 10^{-3})$	2.02	6334	3.72(37)	4.01	11094	4.01(27)
T-*MDF*	$(\infty, 10^{-3})$	3.23	4872	1.94(20)	4.78	6535	1.95(16)
ILU(1)	(1, 0.0)	0	2760	4.37(55)	0	3571	3.80(38)
		FE2D			FE3D		
Algorithm	(l, ε) time	Ordering time	Nonzeros in L	Solution time	Ordering time	Nonzeros in L	Solution time
MUM	(0, 0.0)	1.60	4373	9.99(73)	8.12	19725	22.38(53)
MUM	(1, 0.0)	1.95	5989	8.48(59)	16.11	35963	16.20(30)
MUM	$(\infty, 10^{-3})$	4.97	14223	5.92(30)	20.67	42222	10.91(17)
T-*MDF*	$(\infty, 10^{-3})$	7.33	93559	3.10(17)	125.17	44715	9.61(13)
ILU(1)	(1, 0.0)	0	5835	5.55(38)	0	39066	17.04(31)

total solution time increases with increasing l for FE3D. However, the best methods for FE3D are the pure drop tolerance *MDF* orderings, i.e. with infinite l, and the sparsity pattern determined solely by a drop tolerance. However, from Table 5.1, it is clear that the combined ordering and determination of the *ILU* pattern must be used for many matrix solves in order to result in a net saving.

From these tables, we observe:

- Much of the high-level fill is small and can be safely ignored.
- Some high-level fill is important for obtaining a good *ILU* preconditioner.

Table 5.4. *Results for NS2D.*

NS2D (neq = 2369, nz = 20619)						
Level	Threshold	Ordering	Ordering time	Nonzeros in L	Nonzeros in U	Solution time
0	0.0	*ORG*		9125	9125	39.64(110)
		MDF	5.35	9125	9125	100.76(276)
		MUM	3,.42	9125	9125	24.24(66)
1	0.0	*ORG*		14268	14268	36.74(89)
		MDF	23.07	19372	19372	22.64(47)
		MUM	8.12	20281	20281	23.04(49)
1	1.0^{-3}	*ORG*	2.34	14193	14234	63.44(155)
		MDF	27.73	18705	17970	18.02(38)
		MUM	6.36	16348	18138	19.15(42)
2	0.0	*ORG*		20656	20656	26.69(55)
		MDF	39.64	25101	25101	18.88(34)
		MUM	30.03	43630	43630	26.95(35)
2	1.0^{-3}	*ORG*	3.38	18734	20422	***[a]
		MDF	44.38	23280	23530	15.21(28)
		MUM	15.11	28673	31151	18.82(31)

[a] Here "***" indicates that the computation did not converge after 300 iterations.

Table 5.5. *Results for NS3D.*

NS3D (neq = 684, nz = 52930)						
Level	Threshold	Ordering	Ordering time	Nonzeros in L	Nonzeros in U	Solution time
0	0.0	*ORG*		26123	26123	***[a]
		MDF	452.17	26123	26123	133.69(221)
		MUM	41.45	26123	26123	68.92(111)
1	0.0	*ORG*		46610	46610	***
		MDF	6899.95	61633	61633	274.21(278)
		MUM	316.76	74790	74790	178.32(153)
1	1.0^{-3}	*ORG*	50.53	45708	45726	***
		MDF	6907.34	57739	58710	99.18(92)
		MUM	165.90	53233	60632	144.03(147)
2	0.0	*ORG*		60360	60360	***
		MDF	15883.50	83131	83131	95.44(49)
		MUM	873.72	121392	121393	161.87(59)
2	1.0^{-3}	*ORG*	81.11	59248	59238	***
		MDF	13629.1	80675	81784	41.90(38)
		MUM	355.20	76666	92216	31.59(28)

[a] Here "***" indicates that the computation did not converge after 300 iterations.

- For about the same number of nonzeros in L the choice of a higher fill level and larger ε is more effective than that of a lower fill level and smaller ε.
- A good choice of l and ε for most of our test problems is:

$$l = \infty, \quad \varepsilon = 10^{-3} \text{ or } 10^{-4}.$$

It is also instructive to visualize the ordering produced by the different techniques. Figures 3.2, 3.3 and 3.4 show a visialization of the orderings [35]. For LAPD5, both $MDF(0)$ and $MDF(1)$ orderings are given. Here the nodes with lightest shading were ordered first and the darkest last. We can see that the $MDF(0)$ ordering for the Laplace problem is very similar to the spiral ordering, which was one of the best orderings for $ILU(0)$ [10]. $MDF(1)$ ordering gives a generalized red-black ordering, which is again known to be effective for $ILU(1)$ preconditioning. The actual timings supported the same conclusions.

For Stone's problem, $MDF(0)$ ordering did identify the physical structure (regions B, C, and D), but failed to detect the fast decay orientations in different physical regions. Numerical results confirm this observation. When a higher level MDF ordering was considered, both physical structure and fast decay orientation were successfully identified in the ordering.

We designed the anisotropic test problem to emphasize the importance of the orientation of the ordering to the convergence. Again, $MDF(0)$ recognized the physical regions but failed to determine the "preferred" orientation, while $MDF(1)$ effectively identified both.

Table 5.3 compares MUM ordering, threshold MDF (T-MDF), and $ILU(1)$ based on the original ordering. Note that the MUM ordering time is much smaller than the MDF ordering times, but that the MDF solution times are always faster than the MUM solution times.

The simple MUM variant has difficulty identifying the "preferred" direction in strongly anisotropic problems. For problem STONE and ANISO (Table 5.3), it would appear that MUM is not any improvement over natural ordering.

However, MUM is quite effective for the three dimensional problem FE3D (Table 5.3). For FE3D, $MUM(\infty, 10^{-3})$ ordering took approximately the same time as two solutions but the solution time was almost half the cost of $ILU(1)$ with RCM ordering. The MUM ordering failed to identify a rapidly convergent ordering for two of our anisotropic test problems. For example, if an $ILU(1)$ factorization is used, then with the MUM simplification, after the elimination of an initial corner node for problem (1.1), the next elimination of the neighbor node in the x- or y-direction produces the identical discard value. Hence without considering the actual fill pattern, the preferred direction cannot be identified, at least for simple anisotropic problems.

Tables 5.4 and 5.5 show the results for the unsymmetric Navier-Stokes Jacobians. For the incompressible problem NS2D, the Jacobian is indefinite (some of the diagonals are identically zero). The ORG ordering refers to ordering cells in the x-y

order, and then within each cell, ordering the momentum before the pressure. Note that in Table 5.4 and 5.5, the ordering time includes the time for determination of the sparsity pattern for the drop tolerance *ILU*. Consequently, this time is relevant for *ORG* ordering, if a drop tolerance is used. If the pivot element becomes too small during the course of the incomplete factorization, it was replaced by the maximum in the row (this did not occur for this problem). The asterisks in Tables 5.4 and 5.5 indicate that the particular method did not converge in 300 iterations.

The original orderings in Tables 5.4 and 5.5 often failed to converge for these problems. On the other hand, *MUM* ordering converges with all types of *ILU* preconditioning. The *MDF* ordering was beneficial but the ordering cost was very large. For NS3D, *MUM* ordering was the only reliable, practical way to obtain an iterative solution.

Some interesting but puzzling observations about the unsymmetric test problems are:

- The threshold ordering for both *MDF* and *MUM* converges faster than the no-threshold ordering in most of the cases.
- The level 0 *MUM* ordering converges even faster (fewer iterations) than level 0 *MDF* ordering.

5. Conclusion.

In [7], it has been demonstrated that *MDF*(*l*) ordering is effective for matrix problems arising from complicated partial differential equation discretizations, and in particular, for problems with anisotropic and heterogeneous physical parameters. However, the cost of *MDF*(*l*) ordering is high if the average degree in the original matrix is large. When solving partial differential equations, this situation occurs if the discretization uses a large molecule or stencil.

In order to reduce both ordering and iteration cost, the threshold *MDF* ordering has been developed. This combined ordering/factorization method drops any fill entries that are less than a prescribed tolerance. It is important to combine an ordering technique with drop tolerance *ILU* preconditioning. Without a good ordering strategy, an initial poor ordering can result in slow decay of the fill entries, and consequently an unacceptably dense *ILU* factorization. The high cost of the forward and back solve can then lead to a slow *PCG* method.

Numerical tests show that a high-level threshold *MDF*(*l*) ordering combined with a drop tolerance produces excellent results for partial differential equation problems having a relatively small molecule. This is because most of the high level fill is small and can be ignored, but there are a few high-level fill entries that improve the quality of the preconditioner. The testing for anisotropic problems confirms this conclusion.

Discretization of systems of partial differential equations, for example the Navier-Stokes equations, typically gives rise to large molecules. For these problems, even

threshold *MDF* ordering is too expensive. Consequently, we have developed an approximate *MDF* ordering based on the concept of the minimum update matrix (*MUM*). The *MUM* ordering is much less expensive to compute. However, *MUM* ordering can fail to select a good ordering for anisotropic problems having small molecules. Of course, this is precisely the situation where threshold *MDF* works well. On the other hand, partial differential equation discretizations with large molecules are less affected by anisotropy. For large molecules, numerical tests indicate that *MUM* ordering is very effective. In particular, we have applied *MUM* ordering to Jacobians from Navier-Stokes problems. In some cases, the original ordering did not converge at all, while *MUM* ordering resulted in fast convergence.

To summarize, it appears from our numerical tests that threshold *MDF* is an effective ordering for partial differential equation discretizations with small molecules, while threshold *MUM* ordering is a good choice for discretizations having large molecules. We have also demonstrated by a simple counter example, that an effective ordering strategy must consider the value of the matrix entries, not just the graph of a matrix. This is particularly important for incomplete factorizations with a drop tolerance.

REFERENCES

1. O. Axelsson, *Solution of Linear Systems of Equations: Iterative Methods*, Springer-Verlag, Berlin, Heidelberg, New York, 1977, pp. 2–51.
2. A. Behie and P. A. Forsyth, *Comparison of fast iterative methods for symmetric systems*, IMA J. Num. Anal., 3 (1983), pp. 41–63.
3. T. F. Chan, *Analysis of preconditioners for domain decomposition*, SIAM J. Num. Anal., 24 (1987).
4. T. F. Chan and D. C. Resasco, *Analysis of domain decomposition preconditioners on irregular regions*, in *Advances on Computer Methods for Partial Differential Equations* VI, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, 1987, pp. 317–322.
5. P. Chin, E. F. D'Azevedo, P. A. Forsyth and W.–P. Tang, *Iterative methods for solution of full Newton Jacobians in incompressible viscous flow*, International Journal on Numerical Methods in Fluids, (1991). (submitted).
6. E. F. D'Azevedo, P. A. Forsyth and W.-P. Tang, *An automatic ordering method for incomplete factorization iterative solvers*, in Proceedings of the 1991 Reservoir Simulation Symposium, Anaheim, 1991. Paper SPE 21226.
7. E. F. D'Azevedo, P. A. Forsyth and W.-P. Tang, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM Journal On Matrix Analysis and Applications, (1992). (to appear).
8. S. Doi, *A graph-theory approach for analyzing the effects of ordering on ILU preconditioning*, SIAM J. Sci. Statist. Comp. (submitted).
9. S. Doi and A. Lichnewsky, *An ordering theory for incomplete LU factorizations on regular grids*, SIAM J. Sci. Statist. Comp. (submitted).
10. I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, (1989), pp. 635–657.
11. V. Eijkhout, *Analysis of parallel incomplete point factorizations*, Tech. Rep. CSRD Report No. 1045, Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois, 1990.
12. P. A. Forsyth, *A control volume finite element approach to NAPL groundwater contamination*, SIAM J. Sci. Statist. Comp., 12 (1991), pp. 1029–1057.

13. G. Forsythe and W. Wasow, *Finite Difference Methods for Partial Differential Equations*, Wiley, New York, 1960.
14. A. George and J. W. H. Liu, *Computer Solution of large Sparse Positive-definite Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
15. A. Greenbaum and G. Rodrigue, *Optimal preconditioners of a given sparsity pattern*, BIT, 29 (1989), pp. 610–634.
16. I. Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142–156.
17. T. Hughes, I. Levit and J. Winget, *An element by element solution algorithm for problems of structural and solid mechanics*, Computer Methods in Applied Mechanics and Engineering, 36 (1983), pp. 241–254.
18. P. S. Huyakorn and G. F. Pinder, *Computational Methods in Subsurface Flow*, Academic Press, New York, 1983.
19. D. S. Kershaw, *The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations*, Journal of Computational Physics, 26 (1978), pp. 43–65.
20. D. Keyes and W. Gropp, *A comparison of domain decomposition techniques for elliptic partial differential equations*, SIAM J. Sci. Statist. Comp., 8 (March (1987)), pp. 166–202.
21. L. Y. Kolotilina and A. Y. Yeremin, *On a family of two-level preconditionings of the incomplete block factorization type*, Sov. Journal of Numerical Analysis and Mathematical Modelings, 1 (1986), pp. 293–320.
22. H. P. Langtangen, *Conjugate gradient methods and ILU preconditioning of non-symmetric matrix systems with arbitrary sparsity patterns*, International Journal on Numerical Methods in Fluids, 9 (1989), pp. 213–233.
23. F. W. Letniowski, *Three dimensional Delaunay triangulations for finite element approximations to a second order diffusion operator*, SIAM J. Sci. Statist. Comp., (accepted) (1989).
24. H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, 3 (1957), pp. 255–269.
25. J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a M-matrix*, Mathematics of Computation, 31 (1977), pp. 148–162.
26. J. A. Meyerink and H. A. van der Vorst, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, Journal of Computational Physics, 44 (1981), pp. 134–15.
27. N. Munksgaard, *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Software, 6 (June 1980), pp. 206–219.
28. J. M. Ortega, *Orderings for conjugate gradient preconditionings*, Tech. Rep. TR-90-24, Department of Computer Science, University of Virginia, Charlottesville, Virginia, August 14, 1990.
29. S. V. Parter, *The use of linear graphs in Gaussian elimination*, SIAM Review, 3 (1961), pp. 364–369.
30. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere, New York, 1980.
31. O. Pironneau, *Finite Element Methods for Fluids*, John Wiley & Sons, New York, 1989.
32. D. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in *Graph Theory and Computing*, R. C. Read, ed., Academic Press, 1972, pp. 183–217.
33. Y. Saad and M. Schultz, *GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM J. Sci. Statist. Comp., 7 (1986), pp. 856–869.
34. H. L. Stone, *Iterative solution of implicit approximations of multidimensional partial differential equations*, SIAM J. Num. Anal., 5 (1968), pp. 530–558.
35. G. Townsend and W.-P. Tang, *Matview 1.1 – A two dimensional matrix visualization tool*, Tech. Rep. CS89-36, Department of Computer Science, University of Waterloo, Waterloo, Ontario, 1989.
36. H. A. van der Vorst, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comp., (1991). (to appear).
37. H. A. van der Vorst and P. Sonneveld, *CGSTAB: A more smoothly converging variant of CG-S*, Tech. Rep. Report 90-50, Delft University of Technology, Delft, the Netherlands, May 21, 1990.
38. Z. Zlatev, *Use of iterative refinement in the solution of sparse linear systems*, SIAM J. Num. Anal., 19 (1982), pp. 381–399.