Complete pivoting ILU: A multilevel approach [*]

Yousef Saad[†]

December 23, 2003

Abstract

This paper presents a preconditioning method based on combining two-sided permutations with a multilevel approach. The nonsymmetric permutation exploits a greedy strategy to put large entries of the matrix in the diagonal of the upper leading submatrix. The method can be regarded as a complete pivoting version of the incomplete LU factorization. This leads to an effective incomplete factorization preconditioner for general nonsymmetric, irregularly structured, sparse linear systems.

Key words: Incomplete LU factorization, ILUT, multilevel ILU preconditioner, Krylov subspace methods, multi-elimination, complete pivoting

AMS subject classifications: 65F10, 65N06.

1 Introduction

In recent years, preconditioned Krylov subspace methods have made good progress toward their acceptance as general purpose techniques for solving irregularly structured sparse linear systems. One may distinguish between two categories of linear systems that are tackled by such methods. First is the class of linear systems which originate from the discretization of partial differential equations. Initially, preconditioned Krylov methods were only attempted on problems of elliptic type – for which a good body of theory existed. Because of their success, these methods have increasingly been employed on systems arising from various flow problems, Stokes and Navier Stokes equations, Maxwell's equations, Helmhotz equations, and other types of models which lead to indefinite linear systems. The second category of linear systems includes problems which arise from applications which are not governed by Partial Differential Equations. An archetype of this class of problems is one that originated from power networks, see, e.g. [12, 11], and can be considered in some ways the source of irregularly structured sparse matrix computations as we know it today. Nowadays, these problems are considered small and are often best handled by direct sparse solvers. However, there are somewhat related problems in circuit simulation [8, 28], which cause difficulties to linear solvers. One may ask whether this distinction is important for a given system solver. For direct solvers, the origin of the problem has no direct impact on the solution algorithm, except that its 'geometric' nature can influence complexity. The geometry of the underlying graph may matter

*This work was supported in part by the Army Research Office grant DA/DAAD19-00-1-0485, in part by NSF grant ACI-0305120, and in part by the Minnesota Supercomputer Institute.

†Department of Computer Science and Engineering, University of Minnesota, 4-192 EE/CS Building, 200 Union Street S.E., Minneapolis, MN 55455. E-mail: saad@cs.umn.edu.

more than whether the problem is likely to be highly indefinite, because the geometry may have a major impact on fill-in. In general, however, the prevailing consensus so far has been that iterative methods are not as successful for this second class of methods as they have been for the PDE-based problems. As the sizes of the linear systems are becoming larger and more challenging, this issue has resurfaced recently and a few methods have been developed to increase the range of applicability of iterative methods. The work of Olschowka and Neumaier [23] in particular, showed that it is possible to essentially perform 'static' pivoting, prior to Gaussian elimination. Otherwise stated, it is possible to permute the rows of a matrix in such a way that pivoting becomes unnecessary during the elimination process. Olschowka and Neumaier indicated that this could be helpful in an incomplete LU factorization as well. The work by Duff and Koster described in the articles [14, 15], followed this idea further by developing effective codes, and showing that the method can be quite effective in some cases. Other articles have since tested the idea on various problems, and reported excellent success, see for example [28, 2]. On caveat from these reports is that the reorderings do not seem to help much, in fact they may be counter-productive, when the matrix of the linear system is a structured matrix which originates from a discretized Partial Differential Equation (PDE).

2 The quest for robust ILUs

Several distinct paths have been taken to improve the robustness of preconditioners for irregularly structured linear systems. The first [1] is to add partial pivoting to the incomplete LU factorization with threshold (ILUT), see [25]. The standard ILUT technique often fails for irregularly structured matrices. The addition of partial (column) pivoting helps but it does not even guarantee that a factorization will be produced in some cases. One of the most common cases of failure results from the appearance of zero rows during factorization caused by the combination of permutations and dropping. However, ILUTP can work rather well if substantial fill-in is allowed – but the cost of the factorization may become prohibitive.

The second class of methods, which became popular in recent years, avoids ILU factorizations altogether and utilizes instead one of several methods to compute an approximation to the inverse of A directly, see, e.g., [3, 5, 4, 9, 17, 19]. These "approximate inverse methods" have the added advantage of yielding more parallelism, but they tend to be rather expensive to compute.

A third approach is to use a form of preordering whose goal is to put large entries to the diagonal. In this method, the matrix is ordered on one side (e.g., rows only). Then pivoting is obviated during Gaussian elimination [15, 14] The combination of ILU-type preconditioners with such a reordering strategy has been shown to be quite effective [2, 15, 14, 28].

The last class of methods we will mention has been developed following the work by Bollhöfer [6] which addresses the problem of rigorous dropping in ILU-type methods. These methods are founded on the intimate relationships between ILU factorizations and approximate inverse techniques [7]. Small terms are usually dropped in ILU either by considering their location (positional dropping) or their magnitude (threshold dropping) in some relative terms. The first approach works well for regularly structured problems arising from PDEs but it is not designed for the more general case. The second is generally applicable but the greedy criterion on which it is based focusses on

[1] The order is not chronological

making LU close to A instead of making the preconditioned matrix close to the identity. This may have some unpredictable effects. A better strategy was defined in [6] which exploits condition number estimators for dropping terms whose removal is least likely to have negative effects on A. This strategy was combined with a Crout implementation of ILU in [21].

In contrast with similar existing strategies the method discussed in this paper does not attempt to fill the whole diagonal of the original matrix with large entries. Instead, a more progressive, multilevel, approach is taken whereby the unknowns associated with the upper leading submatrix are eliminated and the process is repeated on the reduced system. An alternative view of the process is that of a block complete pivoting ILU. Viewed from this angle, the process consists of finding two permutations P and Q such that

$$PAQ^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix}$$

in which the B matrix has large diagonal entries. The unknowns associated with the B block can now be "safely" eliminated and the process repeated on the reduced system. In the ideal case, B would be of the same size as A. One can think, for example, of an algorithm that would yield a B matrix that is diagonal and contains most of the large entries of the matrix A. However, forcing B to be diagonal could severely limit its size in general, resulting in an expensive option.

Preconditioning techniques are developed based on approximate factorizations derived from this strategy. The method can also be regarded as a nonsymmetric version of the Algebraic Recursive Multilevel Solver, which uses symmetric permutations (i.e., $P = Q$). Numerical experiments indicate a marked improvement in robustness over using symmetric permutations.

We now briefly discuss practical issues surrounding pivoting for incomplete factorization. When a row version of delayed-update Gaussian elimination (the so called IKJ version) is used, it is relatively easy to incorporate column pivoting. Since the pivoting is local, i.e., it is oblivious to the rows below the working (i-th) row, dropping may lead to undesired results in the resulting ILU factorization. For example, we already mentioned the common occurrence of zero rows during the process, especially when low levels of fill are used. When it works, the resulting LU factorization is capable of solving more problems in general, but the loss of structure due to pivoting results in much more expensive algorithm, especially in terms of memory usage.

Incorporating partial pivoting in the more common rank-one update variant (so-called KIJ variant) or other forms of incomplete LU factorizations does not seem to have been undertaken. However, the more interesting question one might ask is whether or not a form of complete pivoting is practical in the context of ILU. Searching for the largest entry in the working submatrix at each step seems to be exceedingly expensive and impractical. On the other hand, if this were possible, we may potentially be able to drop more terms since dropping may become more rigorous. To cope with the issue of cost, the method presented in this paper essentially resorts to a block pivoting approach that performs only an approximate version of complete pivoting.

3 Multilevel ILU preconditioners

A common method used to obtain a preconditioner, is via a block Incomplete LU factorization, which involves an approximate Gaussian elimination process based on separating the original unknowns into a "coarse" and a "fine" set. These terms are borrowed from the AMG literature. In the class of preconditioners developed in [27, 1, 26] the idea of independent sets or "group-independent" sets [16, 22, 20, 10, 24] is exploited to define this partition. Block Independent set orderings permute the original linear system $Ax = b$ into the form

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{1}$$

in which the submatrix B is block diagonal. The ILUM[24] factorization utilizes standard independent set orderings which result in a matrix B that is diagonal. In this situation, it is easy to eliminate the u variable to obtain a system with only the y variable. The coefficient matrix for this 'reduced system' is the Schur complement $S = C - EB^{-1}F$ which is still sparse. This idea was used recursively, applying dropping to S to limit fill-in, and reordering the resulting reduced system into the above form via independent sets. This is repeated for a few levels until the system is small enough, or a maximum number of levels is reached. Then the system is solved by some other, standard, means such as an ILUT-GMRES combination.

3.1 Two-sided permutations

In this paper we will not require B to have any particular structure. Instead the rows and columns of A are permuted, using two different permutations P (rows) and Q (columns) which will transform A into the form

$$PAQ^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \tag{2}$$

in which the matrix B has the property of being *as diagonally dominant as possible* – in a way that is yet to be specified.

At the l-th level of the procedure, the coefficient matrix is reordered as in (2) and the following block factorization is computed 'approximately' (subscripts corresponding to level numbers are introduced):

$$P_l A_l Q_l^T = \left(\begin{array}{cc} B_l & F_l \\ E_l & C_l \end{array} \right) \approx \left(\begin{array}{cc} L_l & 0 \\ E_l U_l^{-1} & I \end{array} \right) \times \left(\begin{array}{cc} U_l & L_l^{-1} F_l \\ 0 & A_{l+1} \end{array} \right), \tag{3}$$

where

$$B_l \approx L_l U_l \tag{4}$$

$$A_{l+1} \approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l) \ . \tag{5}$$

When described with the help of recursivity, the procedure consists essentially of three steps. First, an ordering of the matrix in the form (2) is obtained and applied; Then, an ILU factorization $B_l \approx L_l U_l$ for B_l is computed. Finally, approximations to the matrices $L_l^{-1} F_l$, $E_l U_l^{-1}$, and A_{l+1} are obtained. The process is repeated recursively on the matrix A_{l+1} until a selected number of levels is reached. At the last level, a simple ILUT factorization, possibly with pivoting, or an approximate inverse method can be applied.

The A_i's remain sparse but become denser as the number of levels increases, so methods based on this approach will drop small terms in various ways in the constructions of L_l, U_l, $L_l^{-1} F_l, E_l U_l^{-1}$, and A_{l+1}, to maintain sparsity. Note that the matrices $E_l U_l^{-1}, L_l^{-1} F_l$ or their approximations

$$G_l \approx E_l U_l^{-1} \ , \qquad W_l \approx L_l^{-1} F_l \tag{6}$$

which are used to obtain the Schur complement via (5) need not be saved. They are computed only for the purpose of obtaining an approximation to A_{l+1}. Once this is accomplished, they are discarded to save storage. Subsequent operations with $L_l^{-1}F_l$ and $E_lU_l^{-1}$ are performed using U_l, L_l and the blocks E_l and F_l.

The rationale of the orderings that will be proposed next is that it is critical to have an accurate and well-conditioned B block in the multilevel factorization discussed above. Alternatively, we can also think in terms of complete pivoting. Assume for a moment that B is a 1×1 block. Then, clearly one can think of selecting the permutations P and Q in such a way as to move the best possible pivot from the matrix to location $(1,1)$. The first row can then be eliminated and the process could then be repeated for the remaining $(n-1) \times (n-1)$ matrix. Unfortunately this procedure is too costly because of the search required to find the best pivot at each step. A more effective alternative is to select the best k pivots at the same time - so B now becomes a block of size k. The next section discusses a few heuristics which use diagonal dominance as a criterion for selecting the B block.

3.2 The block complete pivoting viewpoint

We begin by introducing the notation and terminology used in describing the algorithms. A pair of permutations (P,Q) is completely defined from the corresponding two permutations $\{p_1, p_2, \ldots, p_n\}$ $\{q_1, q_2, \ldots, q_n\}$ of the set $\{1, 2, \ldots, n\}$. Since we use partial permutations (which define the block B), we define a matching set $\mathcal{M}$ as a set of n_M pairs (p_i, q_i) where $n_M \le n$ and

$$1 \le p_i, q_i \le n_M, \text{ for } i = 1, \ldots, n_M \qquad \text{and} \quad p_i \ne p_j, \text{ for } i \ne j \qquad q_i \ne q_j, \text{ for } i \ne j \tag{7}$$

Note that the case when $n_M = n$ corresponds precisely to a (full) permutation pair (P,Q). A partial matching set can be completed into a complete matching set by simply scanning the set of p_i's and adding entries from the set $\{1, 2, \ldots, n\}$ which are missing, and then repeating the process for the q_i's. The function returning the number of nonzero entries in a row or column is denoted by $nz(.)$, so for example $nz(a_{i,:})$ is the number of nonzero elements in the i-th row of A and $nz(a_{:,j})$ is the number of nonzero elements in the j-th column of A.

For lack of a better name we will refer to the class of reordering algorithms to be described in this section as 'ddPQ' orderings (P, Q stand for the permutations used and 'dd' indicates that the ideal permutation would yield a diagonally dominant B block). These are heuristic methods which consist of two stages. In a first stage, candidate entries a_{ij} are 'preselected' as possible diagonal entries. This procedure will also order the candidate entries that are selected by using a certain measure which will attempt to use a criteria of diagonal dominance and the number of nonzero entries. The second stage scans these candidate entries in order given by the preselection stage and decides on accepting or rejecting the entry into the $\mathcal{M}$ set.

The simplest preselection algorithm is based on using the ratios

$$\frac{|a_{ij}|}{\|a_{i,:}\|_1} \, .$$

A criterion using ratios of this type for the diagonal entries $(i = j)$ was also used in an early version of ARMS [26] for filtering equations that are least diagonal dominant in a symmetric permutation setting. Let $j(i)$ be a column index of the largest (in absolute value) entry of row i ($|a_{i,j(i)}| \geq |a_{ij}|$ for all j). The ratio

$$\frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1}$$

can be viewed as a measure of rowwise diagonal dominance of row i if the term $(i, j(i))$ is permuted as a diagonal entry. The selection of $a_{i,j(i)}$ as a diagonal term would result in a diagonally dominant row if the above ratio is $> 1/2$. In practice, we can screen out all entries for which these ratios are no greater than a certain threshold τ.

ALGORITHM **3.1** *Preselection*

```
1.  Set $\mathcal{D} = \emptyset$
2.  For $i = 1, \ldots, n$ do
3.      Compute $t_i = \|a_{i,:}\|_1$, and $j(i) = argmax_k |a_{ik}|$
4.      if $|a_{i,j(i)}| > \tau\ t_i$ add $(i, j(i))$ to $\mathcal{D}$
5.  EndDo
6.  For each $(i, j(i)) \in \mathcal{D}$ Do:
7.      Compute $w_i = \frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1} \times \frac{1}{nz(a_{i,:})}$
8.  EndDo
9.  Sort $\mathcal{D}$ by decreasing order of the weight $w_i$: $\mathcal{D} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_{n_D}, j_{n_D})\}$
        with: $w_{i_1} \geq w_{i_2} \geq \ldots \geq w_{i_{n_D}}$.
```

In order to avoid situations in which the preselection algorithm returns with an empty set, we scale the threshold τ relative to the largest of these ratios. So, given an input parameter τ_0, the actual threshold τ used in the algorithm is

$$\tau = \tau_0 \times \max_i \left[\frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1} \right] \tag{8}$$

This ensure that the worst rows on a relative basis will be rejected. The parameter τ_0 should be between 0 and 1 and a typical value is 0.5. If τ_0 is strictly less than one, then at least one row will be preselected.

The algorithm begins by computing the required norms and then in Line 4, it prescreens the rows by rejecting those that do not show enough diagonal dominance. This initial rejection will reduce the cost of the next phase which involves a sort (line 10.) In order to reduce fill-in, the diagonal dominance ratio is modified. It is divided by the number of nonzero elements in row i (see Line 7).

The above algorithm selects good candidates for diagonal entries according to diagonal dominance, and then it sorts them according to a certain criterion. To complete the process we now need to select pairs to keep in $\mathcal{M}$. We could, for example, scan all the pairs yielded by Algorithm 3.1 and keep all the pairs as long as that they do not violate the requirement in the definition (7). Therefore, the simplest strategy is a greedy technique which scans the set $\mathcal{D}$ and accepts any pair (p_i, q_i) such neither p_i nor q_i have already been included in a pair of $\mathcal{M}$.

ALGORITHM **3.2** *Greedy matching set selection*

```
0. Determine the ordered set $\mathcal{D}$ by a preselection algorithm. Set $count = 0$.
1. Set $\mathcal{M} = \emptyset$ ; Set $P(i) := Q(i) = -1$ for $i = 1, \ldots, n$
2. For $k = 1, \ldots, n_D$ Do:
3.     Get $(i_k, j_k)$ the $k$-th pair in $\mathcal{D}$
4.     If $(P(i_k) == -1$ & $Q(j_k) == -1)$ :
5.         Add $(i_k, j_k)$ to $\mathcal{M}$: (a) $count = count + 1$
6.                            (b) set $P(i_k) = count$; $Q(j_k) = count$ ;
7.     EndIf
8. EndFor
```

At the completion of the algorithm, the matching set is the set of all pairs (i, j) such that $P(i)$ and $Q(j)$ are both non-negative. Note also that $P(:)$ represents an incomplete reverse permutation array. If completed it would simply be the 'old-to-new' usual mapping for the row permutations. Similarly, the Q array represents an incomplete 'old-to-new' mapping for the columns. The only remaining step now is to complete these two permutations. This is achieved by a simple greedy procedure such as the following one, described for P.

```
1. For $i = 1 : n$ Do:
2.     if $(P(i) < 0)$
3.             $count = count + 1$
4.             $P(i) = count$;
5.     EndIf
6. EndFor
```

Note that initially count is as output by the matching procedure. The same procedure is applied to complete the permutation Q.

Once the permutation is found, the matrix is reordered into the form (2) and the rows associated with the B block are eliminated. This means that we compute the factorization (3). This entails computing the ILU factorization of B (equation (4)), and then computing an approximation to the Schur complement, as shown in (5). The process is now to be repeated at the next level, i.e., on the Schur complement matrix A_{l+1}. Note that there is no particular structure associated with the block B.

If B were to be a diagonal matrix, then the elimination process could be viewed as a multi-elimination process with complete pivoting. Instead of selecting one pivot at a time, the algorithm selects the largest possible number of pivots allowed by a greedy approach, in which the selection criterion is a combination of diagonal dominance and number of nonzero entries in the columns.

The algorithm stops whenever either a maximum number of levels is exceeded or the Schur complement becomes small enough. Of course, if the matrix is strongly diagonally dominant, then the process will retain many rows in the first one or two levels so the algorithm will require very few levels. In general, the algorithm requires a small number of levels, unless τ_0 is close to one.

4 Matching heuristics

We now examine a few alternatives to the greedy approach of Algorithm 3.2. It may be desirable for example to seek to enforce diagonal dominance in the B part of (1). Having a B matrix in a form that leads to easy solutions may also be appealing. We have tested a good number of strategies but show only two here in addition to the greedy technique of Algorithm 3.2.

The strategy of algorithm 3.2 does not guarantee to produce a B matrix that is diagonally dominant, but it can be modified in two distinct ways for this purpose. First, instructions can be added to the procedure to reject columns and rows of A which fail to satisfy a diagonal dominance criterion. The second modification reaches the same goal but it also forces the matrix B to be triangular. We begin by describing this procedure first as it is somewhat simpler. To obtain a lower triangular matrix, it suffices to mark all column indices of row i so that they will become ineligible for inclusion in the later steps. In addition, before adding $(i, j(i))$ to the set $\mathcal{M}$, we need to verify that the row is diagonally dominant. This gives the following modification of Algorithm 3.2.

ALGORITHM **4.1** *Reordering for a triangular B block*

```
1.  Set $\mathcal{M} = \emptyset$ ; Set $P(i) := Q(i) = -1$ for $i = 1, \ldots, n$
2.  For $i = 1, \ldots, n_D$ Do:
3.      If $(P(i)! = -1$ or $Q((j(i))! = -1)$ Continue (skip to next i)
4.      Compute $t_B = \sum_{k \in S_B(i)} |a_{ik}|$, where $S_B(i) = \{k| \; k \in adj(A, i) \; ; Q(k) \geq 0\}$,
5.      If $t_B \leq |a_{i,j(i)}|$ then
6.          add $(i, j(i))$ to $\mathcal{M}$
7.          Set $Q(k) = -2$ for all $k$'s in $adj(A, i)$ with $Q(k) == -1$.
8.      Endif
9.  EndDo
10. Complete matching set $\mathcal{M}$ arbitrarily.
```

The set $S_B(i)$ introduced in Line 4, represents the set of column indices that have already been accepted into B prior to step i. Step 4 computes the sum of the absolute values of all entries in the strict lower part of the matrix that would become B after reordering. The next line tests if the corresponding row is (weakly) diagonally dominant. If it is, then the candidate pair $(i, j(i))$ is accepted (Line 6) and all column indices in the row which are not yet assigned are marked so that these columns will not be selected in future steps. This ensures that B is lower triangular. Note that Line 6 can be implemented as in steps (a), (b), shown in Lines 5-6 of Algorithm 3.2.

Forcing B to be triangular may be restrictive. It leads to a B block which is typically smaller, and this may lead to a higher number of levels and therefore a more expensive factorization. A compromise is to still allow entries in the U part of B as long as diagonal dominance is preserved. This leads to a modification of the above algorithm which at each step i computes

$$t = |a_{i,j(i)}| - \sum_{k \; \in \; S_B(i)} |a_{ik}| \; .$$

If t is negative then the i-th row is not diagonally dominant since the lower part of B is already not diagonally dominant and the algorithm will skip to the next candidate pair. Otherwise, it will proceed to accept entries that are not larger than $t/(nz(a_{i,:} - n_B(i) - n_F(i))$, where $n_B(i)$ is the number of entries of the row being examined, that are already in B, and, similarly, $n_F(i)$ is the number of entries that are already in F. All other column entries are rejected, i.e., they are tagged as part of F.

ALGORITHM **4.2** *Augmented triangular B*

```
1.  Set $\mathcal{M} = \emptyset$ ; Set $P(i) := Q(i) = -1$ for $i = 1, \ldots, n$
2.  For $i = 1, \ldots, n_D$ Do:
3.      If $(P(i)! = -1$ or $Q((j(i))! = -1)$ Continue (skip to next i)
4.      Let $S_B(i) = \{k |\ k \ \in \ adj(A, i) \ ; Q(k) \geq 0\}$,
5.          $S_F(i) = \{k |\ k \ \in \ adj(A, i) \ ; Q(k) == -2\}$.
6.      Compute $t_B = \sum_{k \in \ S_B(i)} |a_{ik}|, \quad n_B = |S_B(i)|, \quad$ and $n_F = |S_F(i)|$.
7.      If $t_B \leq |a_{i,j(i)}|$ then
8.          add $(i, j(i))$ to $\mathcal{M}$
9.          Compute $\gamma = (|a_{i,j(i)}| - t_B)/(nz(a_{i,:}) - nB - nF)$
10.         For each $k$ in $adj(A, i)$ with $Q(k) == -1$ Do:
11.             If $|a_{ik}| > \gamma$ set $Q(k) = -2$
12.     Endif
13. EndDo
14. Complete matching set $\mathcal{M}$ arbitrarily.
```

The resulting rows of B will be row diagonally dominant. Indeed, let $S'_B(i)$ represents the set of entries that are accepted into the B part at step i, and let $n'_B = |S'_B(i)|$. Note that $n'_B \leq nz(a_{i,:}) - n_B(i) - n_F(i)$. Then,

$$\begin{aligned} |a_{i,j(i)}| - \sum_{k \ \in \ Bpart} |a_{ik}| &= |a_{i,j(i)}| - \sum_{k \ \in \ S_B} |a_{ik}| - \sum_{k \ \in \ S'_B} |a_{ik}| \\ &= t - \sum_{k \ \in \ S'_B} |a_{ik}| \\ &\geq t - n'_B \times \frac{t}{nz(a_{i,:}) - n_B(i) - n_F(i)} \\ &\geq 0. \end{aligned}$$

A slight variation on this procedure consists essentially of adjusting the average γ computing in Line 9 as new entries are accepted. The sum t_B is changed each time a new entry is accepted into B as well as the number n_B. This allows to accept more entries.

This idea can be extended to develop a forward looking approach which uses both A and A^T. The technique begins by assigning the absolute value of the candidate diagonal entry to a measure of diagonal dominance. These measures, which in effect represent the difference between the diagonal entries and the sums of the off-diagonal entries in each row of B, are updated at each step as the permutation which defines B is being constructed. When a row is scanned the columns are accepted when the $|a_{ik}|$ is tested against the current value of $\nu(i)$ divided by the number of entries that may still potentially be candidates for being accepted in B. If it is less, then it is accepted into B and $\nu(i)$ is decreased accordingly. If the test is not satisfied the column k is marked as not accepted. The counter of the number of entries in the row that are not in B is also decremented. The algorithm produces a matrix B that is row diagonally dominant. It is also possible to have an equivalent algorithm which does not require the transpose of A.

ALGORITHM **4.3** *Forward looking matching*

```
 1. Set $\mathcal{M} = \emptyset$ ; Set $P(i) := Q(i) = -1$ for $i = 1, \ldots, n$
 2. Set $\nu(i) = |a(i, j(i))|$ and $nzB(i) = nz(a_{i,:})$ for $i = 1, \ldots, n_D$
 3. For $i = 1, \ldots, n_D$ Do:
 4.     If $(P(i)! = -1$ or $Q((j(i))! = -1)$ Continue (skip to next i)
 5.     Add $(i, j(i))$ to $sM$
 6.     For each $k \in Adj(A, i)$ s.t. $Q(k) == -1$ Do:
 7.         if $|a_{i,k}| * nzB(i) > \nu(i)$
 8.             set $Q(k) = -2$
 9.         else
10.             $\nu(i) = \nu(i) - |a_{i,k}|$
11.         endIf
12.         $nzB(i) = nzB(i) -1$
13.     End
14.     For each $k \in Adj(A^T, j(i))$ s.t. $P(k) == -1$ Do:
15.         $\nu(i) = \nu(i) - |a_{i,k}|$
16.         $nzB(I) = nzB(i) - 1$
17.         if $\nu(k) < |a_{i,k}|$
18.             set $Q(k) = -2$
19.         endIf
20.     End
21. Complete matching set $\mathcal{M}$ arbitrarily.
```

5 Cost analysis of multilevel nonsymmetric orderings

The first concern when considering the use of the algorithms described above is likely to be in regards to its cost, specifically the cost of obtaining the pair of permutations. Obtaining the permutations may be expensive with a poor implementation or a poor choice of parameters.

Consider the preselection algorithm first. At each level there are $nnz(A_l)$ potential candidates to become diagonal entries. The parameter τ allows to eliminate a large fraction of them with the goal of reducing the cost. The algorithm starts by obtaining the largest entry in each row at the cost of $nnz(A_l)$. In the second phase of the algorithm, weight factors are computed for each of the n_D entries of $\mathcal{D}$ at the cost of $O(n_D)$. Finally, the entries in $\mathcal{D}$ are sorted according to these weights, at the cost of $O(n_D \log n_D)$. Since we do not have an idea of the size of n_D, we need to make an assumption which will enable us to obtain an estimate. We will assume that from one level to the next the size n_l of the matrix A_l is reduced by a constant ratio K. In other words, refering to the splitting (3), we assume that

$$size(A_{l+1})/size(A_l) \approx K \ .$$

On average this is a fairly representative model. It is clear that K will depend on τ as well as on the structure of the matrix, its number of nonzero elements per row, etc.. At each level the size of

$\mathcal{D}$ is a fraction of n_l, the matrix size at level l. Since we limit the fill-in in each row, the number nnz_l of nonzero entries in A_l is a multiple of n_l. With this the cost at each level becomes

$$O(nnz_l) + O(n_D \log n_D) \le \beta n_l + \gamma n_l \log n_l \sim \gamma n_l \log n_l$$

If we have p levels the total cost will be of the form

$$\begin{aligned} T(n) &= \gamma \left[n \log n + \frac{n}{K} \log \frac{n}{K} + \frac{n}{K^2} \log \frac{n}{K^2} + \cdots + \frac{n}{K^p} \log \frac{n}{K^p} \right] \\ &= \gamma \sum_{i=0}^{p} \left[\frac{n}{K^i} \log \frac{n}{K^i} \right] \\ &= \gamma \sum_{i=0}^{p} \left[\frac{n \log n}{K^i} - i \frac{n \log K}{K^i} \right] \\ &= \gamma \left[\frac{K - K^{-p}}{K-1}\, n \log n - n \log(K) \sum_{i=0}^{p} \frac{i}{K^i} \right] \end{aligned}$$

A close look at the second term in the final expression reveals that it is of order n, since the sum of i/K^i is bounded by a constant which depends only on K (and not p). In the end,

$$T(n) \approx \gamma \frac{K}{K-1}\, n \log n \ .$$

As can be seen, when K is close to one, the algorithm will do poorly, reflecting the fact that the diagonal dominance criterion does not reject enough rows at each level. The analysis indicates that the overall cost for obtaining the permutation is dominated by the sorting step in the preselection phase. In fact, it is clear that an exact sort is not necessary and could easily be replaced by an inexact and less expensive sort. For example, it is possible to reduce the cost of the sorting step by grouping the ratios into a small number of nearby representatives and resorting to a bucket sort. This would lead to a cost of $O(n)$.

6 Numerical tests

We have implemented a version of the algorithm described in earlier sections by using the ARMS framework [26]. We refer to the resulting algorithm as ARMS-C (C for complete pivoting). The goal of the experiments in this section is to illustrate the performance of the ARMS-C strategy as well as to give an idea on its cost and the effect of some parameters. The experiments of Sections 6.1 and 6.4 were performed on a Linux platform with two 1.7 Xeon GHz processors, a 256KB cache, and 1GB of main memory. The other experiments were performed on one processor of an IBM/SP computer. The processor is a 375 MHz Power3 (Winterhawk) node with 4 GB of memory.

6.1 Comparing matching strategies

We begin this section with a comparison of a few matching strategies. In the following experiments, we use only one preselection technique, namely the one described by Algorithm 3.1. We run a number of strategies and attempt to solve a sequence of linear systems from the Harewell-Boeing collection [13]. The linear systems are obtained from all those matrices from the RUA collection

BP1000	822	BP1200	822	BP1400	822	BP1600	822
BP200	822	BP400	822	BP600	822	BP800	822
FS5411	541	FS5412	541	FS5413	541	FS5414	541
FS6801	680	FS6802	680	FS6803	680	FS7601	760
FS7602	760	FS7603	760	GEMAT11	4929	GEMAT12	4929
GRE1107	1107	GRE512	512	JPWH991	991	LNS3937	3937
LNS511	511	LNSP3937	3937	LNSP511	511	MAHINDAS	1258
HBMCFE	765	NNC1374	1374	NNC666	666	ORANI678	2529
ORSIRR1	1030	ORSIRR2	886	ORSREG1	2205	PDE9511	961
PORES2	1224	PORES3	532	PSMIGR1	3140	PSMIGR2	3140
PSMIGR3	3140	SAYLR3	1000	SAYLR4	3564	SHERMAN1	1000
SHERMAN2	1080	SHERMAN3	5005	SHERMAN4	1104	SHERMAN5	3312
HBSHL0	663	SHL200	663	SHL400	663	STEAM2	600
WATT1	1856	WATT2	1856	WEST0655	655	WEST0989	989
WEST1505	1505	WEST2021	2021				

Table 1: The 58 matrices from the Harwell-Boeing collection with sizes (shown to their right) ≥ 500

whose size is 500 or higher. There are 58 matrices that satisfy this criterion and their names are listed in Table 1 along with their sizes. If one (or several) right-hand sides is (are) provided then the system with this (the first) righ-hand side is solved. Otherwise an artificial right-hand side is taken, which is formed by taking $b = A * e$, where e is the vector of all ones. The iteration is stopped whenever the residual norm has been reduced by 8 orders of magnitude. This does not necessarily mean that the solution that is computed is accurate, because the matrices involved may be highly ill-conditioned. Since right-preconditioning is used, the tests are on the actual residuals not the preconditioned ones.

In spite of their rather small sizes, some of the matrices in this list cause difficulties to iterative solvers. It is for example not easy to find an iterative procedure which solves all the systems with the same input parameters. Most of the matrices are irregularly structured, and some have a very irregular pattern. The next plots show a comparison of the performance of GMRES preconditioned with ARMS-C with various matching strategies. We compare 5 methods:

- The greedy strategy of Algorithm 3.2,
- The triangular B algorithm described by Algorithm 4.1,
- The augmented triangular B technique as described by Algorithm 4.2
- The forward looking algorithm described by Algorithm 4.3.

The parameters used for the experiment are listed below

		Drop tolerance				Fill-max			
$nlev_{max}$	tol_{DD}	LU-B	GW	S	LU-S	LU-B	GW	S	LU-S
100	0.1	0.001	0.01	0.001	0.01	10	10	10	5

In the table, $nlev_{max}$ is the maximum number of levels allowed, tol_{DD} is the tolerance τ_0 used in the preselection algorithm, see (8). The next 4 parameters define the dropping factors for the factorization algorithms. The drop tolerance for LU-B is the one used in the ILUT factorization

[25] of the B block, while the one denoted by GW is used when computing the matrices G_l and W_l defined by (6) at each level. The drop tolerance used to compute S from G_l and W_l is given next. Finally, the drop tolerance for the ILU factorization at the last level is given. The incomplete factorization with column pivoting (ILUTP) is used for this. The next set of columns define the upper limit for fill-ins for each of these computations in the same order. Note that these numbers are the multiples of the average number of nonzero elements, i.e., at the l-th level the actual amount of nonzero entries allowed in each row is $fill_{max} * nnz(A_l)/n$. We ran ARMS-C with different matching strategies with the *exact same parameters given above for all 58 linear systems.*

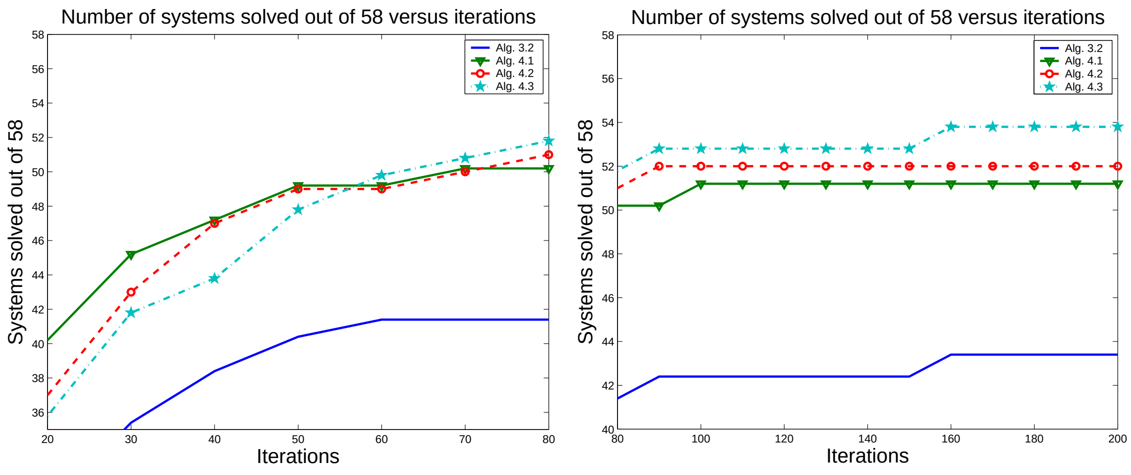

Figure 1: Experiment with 58 HB matrices showing the number of converged systems among 58, versus iteration count.

The incomplete factorizations were computable in all cases. However, the iteration phase was not successful in all cases. An iteration was labeled as a failure when the residual norm was not reduced by 8 orders of magnitude in fewer than 200 GMRES steps. We used GMRES with a restart of 100. This was done in order to measure the quality of the preconditioner alone in distinction from that of the accelerator.

The results are divided in two plots for better clarity. The left side of Figure1 shows the number of successful solves for iteration counts less than 80 and the right side those requiring 80 steps or more. For better visibility, the curves have been slightly shifted in order to avoid superposition. As can be observed the strategy of Algorithm 4.3 handles harder problems quite well (right side of figure) while Algorithm 4.1 yields faster convergence on easier problems (left side of figure).

A standard preconditioner which is often used for comparison purposes is ILUTP (ILU with Threshold and Pivoting [25]). When a good amount of fill-in is allowed the preconditioner can be fairly robust. However, it does not perform well for matrices with highly irregular patterns and when not enough fill-in is allowed. For comparison purposes, we ran ILUTP-GMRES under similar conditions on the 58 problems. Specifically, we selected a drop tolerance of 0.01 and a fill-in limit of $3 * nnz(A)/n$, which means that each row of the factors L and U have at most 3 times the number of nonzero elements per row of A. To give an idea on the overall performance, we show in Table 2 the average fill-factor and average iteration count for each *successful case* for all methods, including

Method	Average Fill-fact	Average Iter. count	Successful Iterations
ILUTP	2.71	12.12	43
Alg. 3.2	1.61	20.58	43
Alg. 4.1	1.87	15.00	51
Alg. 4.2	1.63	17.23	52
Alg. 4.3	1.65	22.11	54

Table 2: Some statistics on the tests with the 58 HB matrices

ILUTP. The fill factor reflects the memory requirement of the preconditioner and is defined as the ratio of the number of nonzero elements of the resulting preconditioner over that of the original matrix.

It is interesting to note that ILUTP required far more fill-in to solve the same number of problems (43) as the worst performer among the ARMS-C algorithms. The average iteration counts are not indicative of robustness. Since the fill-in is rather high, the iteration counts for the easy problems is usually a very small number for the (fewer) successful runs of ILUTP, which leads to a low average among the 43 successful cases. This is in contrast with the ARMS-C strategies which solve more of the harder problems but, as can be expected, at the cost of a larger number of GMRES steps on average.

6.2 Effect of the filtering parameter τ

We vary the parameter τ_0 used in the preselection algorithm and compare the iteration number and the times to compute the factorization for a system associated with the matrix `twotone` from the University of Florida sparse matrix collection[2]. The parameters used for the test are given in the following table

		Drop tolerance				$Fill_{max}$			
$nlev_{max}$	tol_{DD}	LU-B	GW	S	LU-S	LU-B	GW	S	LU-S
200	varies	0.001	0.0001	0.001	0.01	10	10	10	5

Algorithm 4.3 was used to obtain the P, Q permutations.

For this matrix the ARMS-C procedure requires unusually small values for the drop tolerances before the preconditioner starts yielding a converging iteration. On the other hand, and somewhat surprisingly, the resulting fill-factor is not too large, and convergence can be obtained for fill-factors below 2. This phenomenon is not observed too often and is likely related to the structure of the problem. ILUTP did not yield convergence for this problem, even for a fairly large fill-factor.

As τ_0 increases from 0.0 to 0.4 one can obverse a steady improvement in the convergence of the algorithm while the fill-factor remains almost the same. Therefore a better quality factorization is being computed with about the same amount of fill. Then the iteration count sees a sizable decrease, reaching the smallest number of 27 when $\tau = 0.7$. At this point the maximum number of levels, which has been set to 200, is reached and the fill-factor increases. Eventually, the iteration starts to deteriorate slowly reaching 38 when $\tau = 0.9$. For a small value of τ_0 more entries are initially preselected. For example when $\tau = 0$ all the largest entries in each row are candidates.

[2]http://www.cise.ufl.udu/davis/sparse

τ_0	filf	lev	iter	setup sec	iters sec
0.00	1.75	91	81	5.96e+01	6.84e+01
0.10	1.79	90	74	6.05e+01	6.24e+01
0.20	1.75	97	87	5.90e+01	7.45e+01
0.30	1.78	117	71	7.01e+01	6.19e+01
0.40	1.80	145	47	8.70e+01	4.29e+01
0.50	1.89	193	34	1.18e+02	3.60e+01
0.60	1.91	200	28	1.46e+02	3.41e+01
0.70	1.95	200	27	1.64e+02	4.14e+01
0.80	1.95	200	29	1.72e+02	5.45e+01
0.90	1.91	200	35	2.34e+02	9.73e+01

Table 3: Performance of ARMS-C for various values of τ_0 for the Twotone matrix

They are then sorted using a measure which mixes diagonal dominance with the number of nonzero entries. The resulting algorithm is more tolerant of nondiagonal dominance. Selecting a larger τ_0 sets a stricter condition for accepting an entry as a candidate. Excluding more rows in this way results in more robust, often faster, convergence, but this increases the number of levels as can be seen in the table. In fact the number of levels shown in the table for $\tau \geq 0.7$ is the limit allowed.

One may wonder why the iteration time is not proportional to the amount of fill of the factorization and the iteration count. For example when between $\tau = 0.8$ and $\tau = 0.9$, the iteration time jumps 78% while the iteration count increases only 20%. This is likely due to the fact that the last Schur complement is much larger for the latter case.

6.3 Effect of the number of levels

One important consideration when using ARMS-C, is the number of levels. At one extreme when only one level is used, ARMS-C will yield the ILUTP factorization. Indeed, the procedure considers the matrix at the top level to be the Schur complement matrix and uses ILUTP to factor it. Large Schur complements are likely to lead to expensive factorizations. The current code uses as a parameter the maximum number of levels and the smallest allowable Schur complement which is usually set to 100, i.e., as soon as a matrix A_{l+1} in (3) reaches a size of 100 or smaller the procedure stops generating more levels. In the next experiment we explore this with a goal of demonstrating the beneficial effect of a multilevel approach in contrast with one that is based on one or very few levels.

The matrix used in this test arises from the simulation of a 2-dimensional flow in a driven cavity and is available from the matrix market [3] under the name `E40R0100` in the collection DRIVCAV. Its size is $n = 17,922$ and it has $nnz = 567,467$ nonzero entries. The Reynolds number for the example is 100. The parameters used for the experiment are listed below

		Drop tolerance				Fill-max			
$nlev_{max}$	tol_{DD}	LU-B	GW	S	LU-S	LU-B	GW	S	LU-S
varies	0.2	0.01	0.05	0.05	0.01	6	6	6	6

The stopping criterion is as before, namely the norm of the residual should be reduced by 8 orders of magnitude. The case $nlev_{max} = 0$ corresponds to ILUTP - with the drop tolerance 0.01

[3] `http://math.nist.gov/MatrixMarket/`

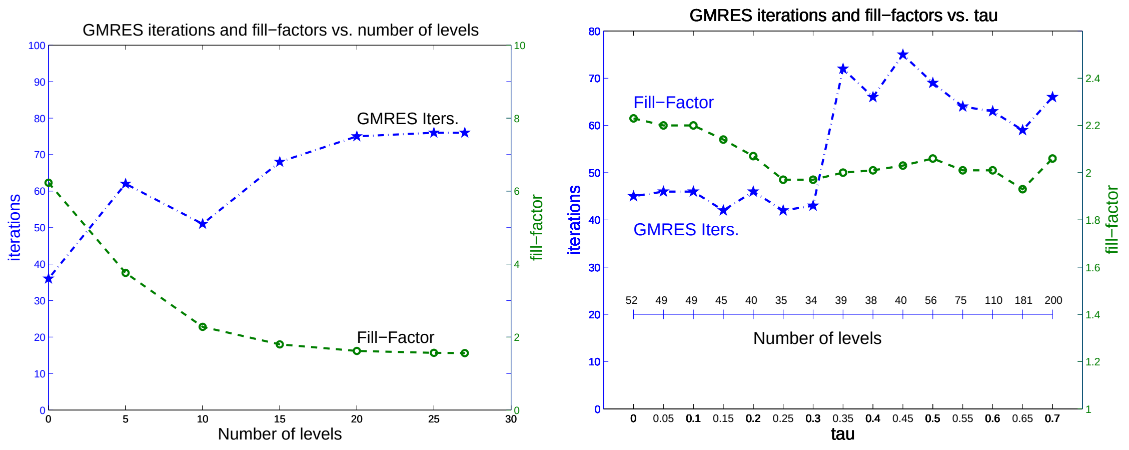

Figure 2: Iteration and fill-factor for the driven cavity problem, versus the number of levels (left-side) and τ_0 parameter (right-side). The right plot uses a slightly different set of parameters.

and a max fill-in of $6 \times nz/n$. Figure 2 shows the number of GMRES iterations as well as the resulting fill-factor when the number of levels varies fom zero to 30. Note that for this case only 27 levels were required before the Schur complement matrix reached the smallest allowable size of 100, so the case $nlev_{max} = 30$ was never reached. The plot shows a fairly regular decrease in the fill-factor as the number of levels increases. At the same time, the iteration count increases and then levels off at around 75.

A different perspective on this experiment would be to vary the parameter τ_0 as was done in the previous section and let the number of levels be as large as required. With the set of parameters used in the previous experiment, the maximum number of levels for the matrix E40R0100 was 27. In order to obtain larger numbers of levels, we tightened the drop tolerance slightly by halving the two drop tolerance values of 0.05 used for the G, W matrices and the Schur complement to 0.025. The parameter τ_0 is varied from 0.0 to 0.7 with increments of 0.05. The plot to the right of Figure 2 shows the result. Essentially, the fill-factor remains in a rather narrow band [1.93 2.23], while the iteration count changes but not substantially. The additional axis shows the number of levels required for each case. These numbers are much bigger than the ones obtained with a drop tolerance of 0.05. This last experiment as well as the one in Section 6.2 and others not reported here, suggest that it is best not to limit the number of levels to a too small number. Limiting the number of levels in this way may lead to a large last Schur complement matrix which is then poorly handled by the ILUTP approach.

6.4 Solution of KKT problems and a comparison

The ARMS-C procedure can be applied to solve saddle-point problems such as those arising from the incompressible Navier-Stokes equations or from Karush-Kuhn-Tucker (KKT) optimality conditions in nonlinear programming. In [18] Haws and Meyer focus on KKT problems and compare a few methods for solving them. Such problems are symmetric and highly indefinite but one of the methods tested in [18] is to ignore symmetry and use ILUT as a preconditioner on a reordered system. The reordering used is MC64, the one-sided reordering developed in [14, 15, 23] whose goal is to put large entries on the diagonal. Two sets of problems are tested in [18]. We only consider the first set corresponding to the results reported in Table I of the paper.

Matrix	n	nnz	nnz_{pre}	lev	Iters	Setup sec.	Iter sec.
stiff4	8496	41318	73332	4	49	0.09	0.39
stiff5	33410	177256	341363	6	112	0.48	6.73
mass04	8496	56818	61799	2	7	0.07	0.04
mass05	33410	241012	259026	5	41	0.35	1.53
mass06	33794	257220	302009	6	20	0.47	0.73

Table 4: Results for five KKT problems.

Matrix	nnz_{pre}	Iters.	Setup sec.	Iter. sec
stiff4	77 118	28	0.129	0.786
stiff5	341417	135	0.575	15.48
mass04	68875	3	0.126	0.186
mass05	323299	32	0.705	3.902
mass06	394821	30	0.806	4.038

Table 5: Results for solving the five KKT systems of Table 4 with ILUT-MC64, as reported in [18].

Table 4 shows the same statistics as those of Table I of [18] plus the number of levels ('lev') required for each case for a similar approach which ignores symmetry and solves the problems by an ARMS-C/GRMRES combination. In the table n and nnz are the size and number of nonzero entries of the matrix, nnz_{pre} is the number of nonzero entries required for the preconditioner, and the rest of the data is self-explanatory. The parameters used for the experiment are listed below.

		Drop tolerance				Fill-max			
$nlev_{max}$	tol_{DD}	LU-B	GW	S	LU-S	LU-B	GW	S	LU-S
100	0.25	0.05	0.01	0.01	0.01	4	4	4	4

The right hand side, initial guess, and convergence criterion are identical with those of [18]. As with earlier tests we do not attempt in any way to tune the parameters for each separate case. These were selected to yield comparable number of nonzero elements for the preconditioners as in [18]. In fact all five problems tested here are relatively easy to solve with the ARMS-C procedure in the sense that almost any parameter selection would yield convergence. This is not the case for some matrices (see, e.g., Section 6.2).

As a comparison we reproduce in Table 5 the statistics for the the MC64-ILUT combination from [18]. As can be seen for similar fill-in, the number of iterations obtained in each case are comparable. Note that the accelerator used in [18] is BCGSTAB instead of GMRES. Execution times are not easily comparable as they depend on many factors in addition to the raw speed of the processors used. The processor used in [18] is an AMD Duron 800 Mhz processor, which is slower than the Xeon 1.7Ghz on which this experiment has been conducted. On the other hand the codes in [18] were optimized with the use of ATLAS BLAS. In addition, the algorithms in [18] are coded in FORTRAN whereas the ARMS-C code is coded mostly in C. The ARMS-C codes have not been optimized in any way. All that can be said is that by taking into account the speeds of the processors, the times obtained in these experiments are within a comparable range.

Note that the MC-64/ILUT procedure was not the best overall method reported in [18]. However, the point of this experiment is not to show that the method discussed in this paper can compete with specialized solvers tailored to a given application or structure. It is only to show that the method can perform reasonably well as a general-purpose solver, in that it can handle KKT problems as easily as a problem arising from the discretization of PDEs.

7 Conclusion

We conclude with a few remarks and observations regarding the performance of ARMS-C. Our first observation is that the method seems to perform in a similar way for problems with a very poor structure as for systems which arise from discretized PDEs. The general consensus has been that it is not advisable to permute matrices arising from discretized PDEs in an unsymmetric way before applying some form of ILU. Another observation is that increasing fill-in does not necessarily improve the quality of the factorization. The situation is similar to that of ILUT and it is related to the difficulty in predicting the effect of dropping small terms in the factors on the conditioning of the preconditioned system.

There are a number questions that remain to be answered and issues to be examined. The first one has to do with the diagonal dominance criterion. This criterion was selected because it is easy to apply and ensures large diagonal entries in a relative sense to the 1-norm of the row. However, one can ask if it possible to find a more elaborate criterion, namely one that takes into account the "stability" of the resulting factors. A related subquestion is the problem of selecting the parameter τ. The choice made in this paper uses a relative measure which keeps the best rows according to a rule based on a parameter input by the user. There remains to find a rigorous way to select the parameter τ, on which performance depends critically.

The second question is related to the dropping strategy. The methods we have tested utilize a simple threshold criterion but it is clear that one based on estimating the condition number of the factors [6] would be ideal in this situation.

Finally, our current codes do not attempt to reduce fill-in by other means than dropping. A simple strategy which can be employed is to use fill-reducing (symmetric) orderings on the B matrices after they have been generated. A forthcoming paper will be examining some of these issues.

Acknowledgements. The code ARMS-C discussed in the experiments has been adapted from ARMS [26], a C code which was developed jointly with Brian Suchomel. I would like to thank John Haws for providing the matrices used in Section 6.4.

References

[1] R. E. Bank and C. Wagner. Multilevel ILU decomposition. *Numerische Mathematik*, 82(4):543–576, 1999.

[2] M. Benzi, J. C. Haws, and M. Tuma. Preconditioning highly indefinite and nonsymmtric matrices. *SIAM Journal on Scientific Computing*, 22(4):1333–1353, 2000.

[3] M. Benzi, C. D. Meyer, and M. Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17:1135–1149, 1996.

[4] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 19(3):968–994, 1998.

[5] M. Benzi and M. Tůma. A robust incomplete factorization preconditioner for positive definite matrices. *Numer. Lin. Alg. w. Appl.*, 2001. special issue, to appear.

[6] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra and its Applications*, 338(1-3):201–213, 2001.

[7] M. Bollhöfer and Y. Saad. On the relations between ILUs and factored approximate inverses. *SIAM Journal on Matrix Analysis and Applications*, 24:219–237, 2003.

[8] C. W. Bomhof and H. A. Van der Vorst. A parallel linear system solver for circuit simulation problems. *Numerical Linear Algebra with Applications*, 7:649–665, 2000.

[9] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19:995–1023, 1998.

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

[11] I. S. Duff. A survey of sparse matrix research. In *Proceedings of the IEEE, 65*, pages 500–535, New York, 1977. Prentice Hall.

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

[13] 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.

[14] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 20:889–901, 1999.

[15] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22(4):973–996, 2001.

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

[17] M. J. Grote and T. Huckle. Parallel preconditionings with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18:838–853, 1997.

[18] J. C. Haws and C. D. Meyer. Preconditioning kkt systems. *Numerical Linear Algebra with Applications*, -, 2004. To appear.

[19] L. Yu. Kolotilina and A. Yu. Yeremin. On a family of two-level preconditionings of the incomplete block factorization type. *Soviet Journal of Numerical Analysis and Mathematical Modeling*, 1:293–320, 1986.

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

[21] N. Li, Y. Saad, and E. Chow. Crout versions of ILU for general sparse matrices. Technical Report umsi-2002-021, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2002. To appear-SISC.

[22] M. Luby. A simple parallel algorithm for the maximum independent set problem. *SIAM J. on Computing*, 14(4):1036–1053, 1986.

[23] M. Olschowka and A. Neumaier. A new pivoting strategy for gaussian elimination. *Linear Algebra and its Applications*, 240:131–151, 1996.

[24] Y. Saad. ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing*, 17(4):830–847, 1996.

[25] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition.* SIAM, Philadelpha, PA, 2003.

[26] Y. Saad and B. Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9, 2002.

[27] Y. Saad and J. Zhang. BILUTM: A domain-based multi-level block ILUT preconditioner for general sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 21, 2000. to appear in SIMAX.

[28] Olaf Schenk, Stefan Röllin, and Anshul Gupta. The effects of nonsymmetric matrix permutations and scalings in semiconductor device and circuit simulation. Technical Report 2003/9, Integrated Systems Laboratory, Swiss Fed. Inst. of Technology (ETH), Zurich, Switzerland, 2003. submitted to IEEE Transactions on Computer-Aided Design.