Multilevel ILU with reorderings for diagonal dominance *

Yousef Saad†

November 16, 2004

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, Helmholtz 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. [21, 18], and which can be considered in some ways to be at the origin of irregularly structured sparse matrix problems as we know them today. Nowadays, these problems are considered small and are often best handled by sparse direct solvers. However, there are somewhat related problems in circuit simulation [13, 38], which cause difficulties to linear solvers. One may ask whether the distinction between these two classes of problems 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

*This work was supported by NSF grants ACI-0305120 and INT-0003274, and 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.

influence complexity. The geometry of the underlying graph may matter more than whether or not the problem is likely to be highly indefinite, as it 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 [33] 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 in many cases 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 [23, 24], followed this idea further by developing efficient 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 [3, 38].

This paper proposes a reordering method that is similar in spirit to the one described in [23, 24]. While both methods are based on improving diagonal dominance the one presented in this paper is essentially 'dynamic' instead of 'static'. It proceeds by levels whereby each level corresponds to a block which has been extracted from the original matrix by permuting its rows and columns in a nonsymmetric way.

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 [35]. The standard ILUT technique often fails for irregularly structured matrices. Partial column pivoting can be added to ILUT and this yields the ILUTP algorithm, [35]. Pivoting does help but there is no guarantee that a factorization will even 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., [5, 7, 8, 15, 27, 29]. Initially, these "approximate inverse methods" have been primarily developed for their parallelism. However, they are often viewed as robust alternatives to ILU factorization preconditioners, see e.g., comments in [4, 6]. Though the factors produced by these methods do not suffer from the instability which plagues incomplete factorizations, they tend to require more memory and to cost more to build.

A third approach is to use a form of preordering whose goal is to move large entries of the matrix to the diagonal. In this method, the matrix is ordered on one side (e.g., rows only). Then pivoting is obviated during Gaussian elimination [24, 23]. The combination of ILU-type preconditioners with such a reordering strategy has been shown to be quite effective [3, 24, 23, 38].

[1] The order is not chronological.

The last class of methods we will mention has been developed following the work by Bollhöfer [9] 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 [10]. Small terms are usually dropped in ILU either by considering their location (positional dropping) or their magnitude (threshold dropping) in some relative sense. 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 making LU close to A instead of making the preconditioned matrix close to the identity. It is often observed that the resulting factors are very ill-conditioned and this leads to poor performance. A better strategy was defined in [9] which exploits condition number estimators for dropping terms whose removal is least likely to be detrimental to the preconditioner. This strategy was combined with a Crout implementation of ILU in [31].

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. Preconditioning techniques developed in this paper obtain 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.

Note that 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.

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, see [26, p. 99]) 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 (k-th) row, dropping may lead to undesired results in later steps during the 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 a 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, it would potentially enable to drop more terms. This is because the resulting factors are more likely to be stable (the norms of their inverses are not too large) and this will cause a milder effect on the terms dropped, see for example [9, 11]. A simplified argument of this fact is as follows. Let $A = LU + E$ where E represents the matrix of the terms that have been dropped from the factorization. Then the split-preconditioned matrix is such that,

$$L^{-1}AU^{-1} = I + L^{-1}EU^{-1} \ .$$

This shows that the larger $\|L^{-1}\|$ and $\|U^{-1}\|$, the smaller the dropped terms must be in order to obtain a preconditioned matrix that is not too far from the identity. 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 [14, 37, 1, 36] the idea of independent sets or "group-independent" sets [25, 32, 30, 17, 34] 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[34] 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, in general[2]. 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 a direct solver or an ILUT-GMRES combination.

3.1 Two-sided permutations

The algorithms described in this paper do 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.

[2]For example, when B is diagonal, E has no more than nz_E nonzero elements per row, and F has no more than nz_F nonzero elements per row, then $EB^{-1}F$ will have no more than $nz_E \times nz_F$ nonzero entries per row.

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):

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

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 then 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, typically with pivoting and very high fill-in levels can be applied.

The A_i's will no longer remain sparse in general, since B is no longer assumed to be diagonal or block-diagonal. Sparsity is regained by dropping small terms in various ways in the construction of L_l, U_l, $L_l^{-1} F_l, E_l U_l^{-1}$, and A_{l+1}. 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 $E_l U_l^{-1}$ 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 $n_M \leq n$ and

$$1 \leq p_i, q_i \leq n, \text{ for } i = 1, \ldots, n_M \qquad \text{and} \quad p_i \neq p_j, \text{ for } i \neq j \qquad q_i \neq q_j, \text{ for } i \neq j \; . \tag{7}$$

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 criterion 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 [36] 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

$$\frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1} \tag{8}$$

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 (weakly) diagonally dominant row if the above ratio is $\geq 1/2$. In practice, we can screen out all entries for which these ratios are no greater than a certain threshold τ.

In a second step, the preselection algorithm will rank all the pairs $(i, j(i))$ in preparation for the next phase of matching, where the permutations P and Q are actually built. The ranking will simply yield a good order in which to scan the list of pairs when determining which ones are valid or best to keep. The simplest possibility is to sort the selected pairs $(i, j(i)$ by decreasing value of the ratios (8). This would favor diagonally dominant rows to be scanned first. However, this ranking does not take sparsity into account. For example, a dense row may be selected because of a high diagonal dominance ratio, and this will yield a full Schur complement matrix. To remedy this, we multiply the above ratios by a quantity that will encourage sparser rows to be selected first. Several such measures were tested and they lead to similar results. The following measure

$$\frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1} \times \frac{1}{nz(a_{i,:})} \tag{9}$$

gives higher priority to rows with good diagonal dominance characteristics which are at the same time sparse. Other possibilities tested incorporated the value $nz(a_{:,j(i)})$ or both the values $nz(a_{:,j(i)})$ and $nz(a_{i,:})$. The following algorithm uses the ratios (9).

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

$$\tau = \tau_0 \times \max_i \left[\frac{|a_{i,j(i)}|}{\|a_{i,:}\|_1} \right] . \tag{10}$$

This ensure that the worst rows on a relative basis will be rejected. The parameter τ_0 should be between 0 and 1 and typical values are in the range 0.1–0.3. 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 pre-screens the rows by keeping only those that do show enough diagonal dominance (relative to other rows). This initial rejection of poor candidate rows will reduce the cost of the next phase which involves a sort (Line 9.) As discussed above, in order to reduce fill-in, the diagonal dominance ratios are modified. They are divided by the number of nonzero elements in row i (see Line 7). The last stage of the algorithm sorts the entries in $\mathcal{D}$ in descending order of the resulting ratios.

The above algorithm selects good candidates for diagonal entries according to a simple diagonal dominance criterion. To complete the process we now need to select pairs to keep in $\mathcal{M}$. We could, for example, scan all pairs returned by Algorithm 3.1 and keep all the pairs as long as that they do not violate the requirement in the definition (7). Clearly, the list should be traversed in the same order in which it is provided by Algorithm 3.1. Therefore, the simplest strategy is a greedy technique which scans the set $\mathcal{D}$ and accepts any pair (p_i, q_i) such that 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* $(Q(j_k) == -1)$ *:*
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*

Note that normally we would also require that $P(i_k) == -1$ in Line 4, but this is unnecessary since there is only one (i_k, j_k) selected for each row and therefore $P(i_k)$ can be assigned a positive value only once. 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, applied here to 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* must have the value returned by the matching algorithm. The same procedure as above is applied to complete the permutation Q. We will refer to this procedure as the 'greedy completion' algorithm. It should be mentioned that this simple procedure will allow zero entries to be introduced in the diagonal of the C block. This is not a problem, for two reasons. First, it is the Schur complement matrix that counts and when it is built, fill-ins are likely to be introduced into its diagonal. Second, this block will be further permuted anyway, either by the recursive application of the process to the new level created, or, if the last level is reached, by column pivoting in the ILUTP procedure, or the full Gaussian elimination, when solving the system.

This process is illustrated in Figure 1. In the 8×8 matrix shown on the left of the figure, the circled entries are the ones preselected by the algorithm, with the numbers in the circles indicating the ranks attributed by the sorting step in Line 9 of the algorithm. The entries shown in shaded squares are the other nonzero elements of the matrix.

Observe that, by its nature, the algorithm clearly selects at most one entry per row. However, there may be more than one selected entry per column. If the greedy matching algorithm is applied, it will end up accepting 5 pairs, and the status of the arrays P, Q at the end is as follows:

P	5	-1	3	2	-1	4	1	-1
Q	5	-1	-1	1	3	2	-1	4

These are then completed by 'greedy completion' into

P	5	6	3	2	7	4	1	8
Q	5	6	7	1	3	2	8	4

Once the permutation is found, the matrix is reordered into the form (2). The thick dashed lines in the matrix on the right side of the figure correspond to the partitioning in the right-hand side of (2). Note that there is no particular structure associated with the block B. In the next block-elimination step, equations from 1 to 5 (first level) of the permuted matrix are eliminated and

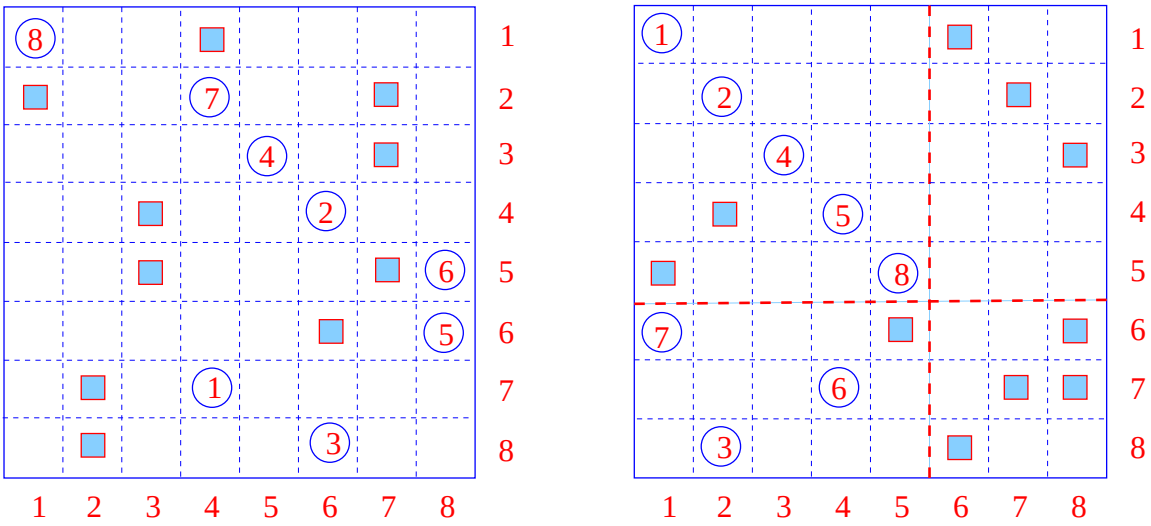

Figure 1: An 8×8 matrix after the preselection algorithm (left side) and the result of the nonsymmetric permutation obtained from the greedy matching procedure. Circled numbers are selected entries and the numbers are the ranks attributed by the preselection algorithm (right).

dropping is applied to the resulting 3×3 Schur complement matrix. 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). If desired, the process can now be repeated on the next level, i.e., on the Schur complement matrix A_{l+1}.

The outer (Gaussian) elimination procedure stops whenever either a maximum number of levels is exceeded or the Schur complement becomes small enough. Assume that for the above example, it is decided that only one level is performed. Then it only remains to solve this last (3×3) linear system. In our code, this is done with a pivoting ILUTP procedure typically with very little dropping (as dictated by a parameter). For all purposes, one can think of this as an accurate solve with Gaussian elimination with column pivoting. ILUTP is selected for the cases when the last block is not small enough, in order to provide more options to the procedure. Note that one can easily obtain a singular Schur complement matrix S after dropping is applied. If no dropping is applied then it is well-known that S will be nonsingular as long as the original matrix and B are both nonsingular. An important point here is that the permutations P, Q that are sought at a given level do not have to take into consideration the quality of the resulting Schur complement A_{l+1}. This matrix will be dealt with at the next level.

Of course, if the original 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.

If B were to be a diagonal matrix, then the elimination process could be viewed as a multi-elimination algorithm 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 attempts to compromise between a diagonal dominance requirement and the number of nonzero entries in the rows.

Note that one alternative to Algorithm 3.2, is to use the MC21 procedure which performs a maximal transversal for bipartite matching, see, e.g., [23, 21, 20, 19]. This algorithm, whose goal is to find a maximal set of matching pairs, is not expensive and is easy to implement. One major difference with Algorithm 3.2 is that MC21 does not take into account numerical values, or any weights assigned to entries. It only finds a row (or column) permutation P such that PA has nonzero entries on the diagonal.

Another observation is that the procedure will fail when a row or column of the Schur complement is zero. It turns out that this rarely happens in practice. Recall that if B is nonsingular, and A is nonsingular then S is nonsingular, so the situation will not happen in this ideal case. The only risk is to make S singular because of dropping, but this is rare with standard threshold-based dropping procedures.

3.3 Diagonal Scaling

Scaling the row and columns of a general sparse matrix prior to attempting to compute its ILU factorization is critical to the success of the preconditioner. The effect of row and column scaling on the quality of the preconditioner is not too well understood, however. When scaling, e.g., the rows of a non-diagonally dominant matrix by, say, their infinity norms, the effect may not necessarily be beneficial to the accelerator. This may be due to the fact that scaling affects the degree of non-normality. Diagonally, or nearly diagonally dominant matrices do not seem to be as much affected by row/column scaling.

Our research code ARMS-C offers some options for scaling the matrices obtained at each level and the last level. If requested, rows of the inter-level matrices A_i are first scaled by their 1-norms. Then, if requested, columns of the resulting matrix are also scaled by their 1-norms. In particular vectors must be allocated to save the scaling factors for the left and right scaling, at each level. Though this strategy works well in many cases, experiments reveal that it is not always the best. As was already mentioned, we have found that the use of scaling may yield a poorer preconditioner than without any scaling at all (see the numerical tests section), though this situation is somewhat uncommon. There are also cases when it is best to start scaling the columns first and then the rows, or scaling rows only or columns only. The issue of scaling in the context of preconditioners is a topic of great importance, and one that has not been given enough attention in the literature. Scaling will be revisited in the numerical experiments section.

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 some form of 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 three here in addition to the greedy technique of Algorithm 3.2.

It is worth it to first examine carefully the problem of enforcing diagonal dominance in B. The strategy of algorithm 3.2 does not guarantee to produce a B matrix that is diagonally dominant, in general. The ideal situation would be to find permutations P, Q so that the B block in (2) is exactly diagonally dominant, as well as of large size. Note that we are restricting the diagonal dominance

requirement to the rows of B not those of A, i.e., entries in F do not matter. We can state that the problem is to find a matching set, satisfying (7), such that the corresponding B matrix, which is defined as

$$\{a_{p(i),q(j)}\}_{i,j=1,\ldots,n_M}$$

is diagonally dominant. As can be easily grasped this is a difficult problem since its requirement is a global one, because the diagonal dominance requirement involves the matching set as a whole.

One possibility for obtaining a diagonally dominant B is simply not to accept any rows that are not diagonally dominant as rows of A in the preselection algorithm. Taking $\tau = 0.5$ in Algorithm 3.1 forces candidate rows scanned in Line 4 of the algorithm to be diagonally dominant. Since the corresponding B rows have fewer (non-diagonal) entries, than the same row of A, then a fortiori the corresponding row in B will be diagonally dominant. However, this does not work in general, simply because in many cases, A may not even have one single row that is diagonally dominant. The sufficient condition used in this strategy is too restrictive. It should be mentioned that in our codes, the scalar τ is determined on a relative basis by a formula such as (10).

Enforcing diagonal dominance in B is easily achieved by restricting B to be triangular. This strategy is presented first in spite of its main limitation, which is that it tends to produce a B block that is small. It will also serve as an introduction to the other matching strategies, which relax the triangularity restriction.

4.1 Matching for a triangular diagonally dominant B

To obtain a lower triangular matrix, it suffices to mark all column indices of a selected 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 $i = 1, \ldots, n$; Set count $= 0$;*
2. *For $i = 1, \ldots, n_D$ Do:*
3. *If ($Q(j(i))$ $!= -1$) 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 $\mathcal{M}$: (a) count $=$ count $+ 1$; (b) $P(i) = Q(j(i)) =$ count;*
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. These will represent the indices of the nonzero entries of row $P(i)$ of the strict lower triangular matrix after permutation. Thus, 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 assigned the value -2, so that these columns will not be selected in future steps. This ensures that B will be lower triangular. A negative value that is different from -1 is used to allow proper completion by the greedy completion algorithm.

4.2 Matching for an augmented triangular B

Forcing B to be triangular may be restrictive. It leads to a B block that is typically much smaller than with the greedy approach of Algorithm 3.2, resulting in a higher number of levels and a more expensive factorization in the end. The algorithm described next can be viewed as a less restrictive version of Algorithm 4.1 whereby some entries are allowed in the upper triangular part of B so long as diagonal dominance is preserved.

At each i-th step, the modified algorithm begins by computing the quantity

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

Recall that $S_B(i)$ is simply the set of indices that correspond to the lower triangular part of row $P(i)$ of the matrix B. Similarly, we denote by $S_F(i)$ the set of column indices that have been rejected prior to step i. We set $n_B(i) = |S_B(i)|$ and, similarly, $n_F(i) = |S_F(i)|$. Thus, there are at most $nz(a_{i,:}) - n_B(i) - n_F(i)$ entries which have not yet been tagged as members of B or F. If t is negative then this row cannot be diagonally dominant since the lower part of B is already not diagonally dominant. In this case the algorithm will skip to the next candidate pair. If, on the other hand, t is positive, then we may accept entries to the right of the diagonal entry of B. In this case, the algorithm will proceed to accept entries that are not larger than

$$\frac{t}{nz(a_{i,:}) - n_B(i) - n_F(i))}$$

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 $i = 1, \ldots, n$; Set count $= 0$.*
2. *For $i = 1, \ldots, n_D$ Do:*
3. *If ($Q(j(i))$!= -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 $t_B = \sum_{k \in S_B(i)} |a_{ik}|$, $n_B = |S_B(i)|$, and $n_F = |S_F(i)|$.*
7. *If $t_B \leq |a_{i,j(i)}|$ then*
8. *add $(i, j(i))$ to $\mathcal{M}$: (a) count $=$ count $+ 1$; (b) $P(i) = Q(j(i)) =$ count;*
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. *EndDo*

13. Endif
14. EndDo
15. Complete matching set $\mathcal{M}$ arbitrarily.

The resulting rows of B will be row diagonally dominant.

Proposition 4.1 *When the permutations P, Q result from Algorithm 4.2 then the matrix B in (2) is row-wise weakly diagonally dominant.*

Proof. Recall that $S_B(i)$ is the subset of culumn indices that have been accepted into B prior to step i and $S_F(i)$ is the set of rejected columns prior to step i. Define also $S'_B(i)$ to be the set of all culumn indices that have been accepted into B *at the completion* of step i, and $S_a(i)$ to be the set of entries that have been accepted at step i. So $S'_B(i) = S_B(i) \cup S_a(i)$. Let $n_a = |S_a(i)|$ and note $n_a + n_B(i) + n_F(i) \leq nz(a_{i,:})$, i.e., $n_a \leq nz(a_{i,:}) - n_B(i) - n_F(i)$. Then,

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

■

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

4.3 Matching based on dynamic averages

This technique assigns a 'dynamic' measure of diagonal dominance of a row when considered as a row of B at a given step. Each step of the algorithm examines the selected pairs $i, j(i)$ from $\mathcal{D}$. It begins by computing $\rho_i = |a_{i,j(i)}|$ which is the entry of the row with largest magnitude. The row is then scanned to determine how much room is left for accepting new entries into B. This is done as follows. Prior to step i the algorithm has accepted a number of entries already. These will show up in the lower triangular part after reordering. The indicator ρ_i will be adjusted by subtracting from it the absolute values of all these entries. It will be updated similarly every time a column is accepted into the B part. Some columns have also been rejected and these will show up in the F part after reordering. The columns that are left are potential candidates for being accepted in the B part. The algorithm will reject columns among these based on a criterion similar to the one in the previous algorithm.

When a row is scanned, an entry $|a_{ik}|$ is tested against the current value of ρ_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 ρ_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.

ALGORITHM **4.3** *Augmented triangular B with dynamic averages*

1. *Set $\mathcal{M} = \emptyset$; Set $P(i) = Q(i) = -1$ for $i = 1, \ldots, n$; Set count $= 0$.*
2. *For $i = 1, \ldots, n_D$ Do:*
3. *If ($Q(j(i))$!= -1) Continue (skip to next i)* *(pair $(i, j(i)$ not eligible)*
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. *$\rho = |a_{i,j(i)}| - \sum_{k \in S_B(i)} |a_{ik}|$*
7. *$nz = |adj(A,i)| - |S_B(i)| - |S_F(i)|$.*
8. *If ($\rho < 0$) Continue (skip to next i)* *(reject pair $(i, j(i))$)*
9. *add $(i, j(i))$ to $\mathcal{M}$: (a) count = count + 1; (b) $P(i) = Q(j(i)) =$ count;*
10. *For each ($j \in Adj(i)$ such that $Q(j) == -1$) Do:*
11. *If ($nz * |a_{i,j}| > \rho$) then*
12. *$Q(j) = -2$* *(Reject j)*
13. *Else* *(Do not reject j)*
14. *$\rho = \rho - |a_{i,j}|$* *(Update ρ)*
15. *EndIf*
16. *$nz = nz - 1$* *(Update nz)*
17. *EndDo*
18. *EndDo*
19. *Complete matching set $\mathcal{M}$ arbitrarily.*

We now briefly discuss the main steps of the algorithm. The variable ρ can be viewed as a diagonal dominance indicator for the running row, as a row of B. As long as it is nonnegative then the current row remains diagonally dominant. Lines 4 to 7 compute the value of ρ at the start, which is the max value $|a_{i,j(i)}|$ minus the absolute values of the entries $a_{i,j}$ for which the column j is already a row of B. The number of columns that are still 'undecided', i.e., that are neither in B nor in F, is also computed (Line 7) and it is called nz. If the computed initial value of ρ is negative then clearly row i must be rejected as is done in Line 8. Otherwise the pair $(i, j(i))$ is added to $\mathcal{M}$ by updating P, Q, and the counter *count* as is done in Line 9. The loop in Lines 10-17, scans row i a second time. Each column j that is still undecided ($Q(j) == -1$), is examined. If $|a_{ij}| > \rho/nz$ then the column is tagged as rejected ($Q(j) == -2$). Otherwise it is accepted (so far), and the B-diagonal dominance indicator ρ is updated accordingly. In either case, the number of undecided columns is reduced by one. It is clear that acceptance into B is only tentative since a column that is accepted at this step may be rejected later.

As can be easily established, the algorithm produces a matrix B that is row (weakly) diagonally dominant.

Proposition 4.2 *When the permutations P, Q result from Algorithm 4.3 then the matrix B in (2) is row-wise weakly diagonally dominant.*

Proof. Throughout step i of the algorithm, the variable ρ represents the absolute value of the difference between $a_{i,j(i)}$ and the entries a_{ij} that are accepted into the B part, i.e., those for which

$Q(j)$ will wind up positive after step $i-1$. After Line 8, the B-part of row i is diagionally dominant since $\rho \geq 0$. When a column is not tagged with -2, then it can potentially become part of B. Consider now the row at the completion of the next loop (Lines 10-17). At each step when a column is not rejected (Lines 13-15), a new ρ is computed, $\rho_{new} = \rho - |a_{ij}|$ and the new total potential entries in the B part of the row is also updated as $nz_{new} = nz - 1$. We have

$$\rho_{new} = \rho - |a_{ij}| \geq nz|a_{ij}| - |a_{ij}| = nz_{new}|a_{ij}|$$

which shows that ρ will remain non-negative throughout the step. In the end there will be fewer entries that are finally accepted (as some columns will be rejected in later steps). Therefore the resulting row of B will remain diagonally dominant. ■

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 and let nnz_l be the number of nonzero elements in A_l. At each level there are nnz_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_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, referring to the splitting (3), we assume that

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

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, (see the parameters used in the experiments) the number nnz_l of nonzero entries in A_l is a multiple of n_l (in worse situations it can possibly be as high as $n_l \log n_l$ but this does not affect the end result). With this the cost at each level becomes

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

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

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

$$
\begin{aligned}
&= \gamma \sum_{i=0}^{p} \left[\frac{n \log n}{K^i} - i \frac{n \log K}{K^i} \right] \\
&= \gamma \left[\frac{K - K^{-p}}{K-1} \; n \log n - n \log(K) \; \sum_{i=0}^{p} \frac{i}{K^i} \right] \; .
\end{aligned}
$$

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 $O(n)$.

The cost of the matching algorithm, for example, Algorithm 4.2, is easier to estimate using a similar model. Once again, the main assumption is that nnz_l is a small multiple, say α, of n_l - due to the dropping strategy. So on average each row of A_l has α nonzero elements. In this case the cost of the i-th step of Algorithm 4.2 is $O(n_D) \times \alpha$, with $n_D \approx n_l/K$. Adding these costs for each level, will yield

$$
n\alpha \left[\frac{1}{K} + \frac{1}{K^2} + \cdots + \frac{1}{K^p} \right] \approx \frac{n\alpha}{K-1}
$$

a cost which is linear in n, and which shows a similar degradation as for the preselection algorithm, when K is close to one.

6 Numerical tests

We have implemented a version of the algorithm described in earlier sections by using the ARMS framework [36]. 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 in Section 6.2 and 6.3 were performed on one processor of an IBM/SP computer. The processor is a 375 MHz Power3 (Winterhawk) node with 4 GB of memory. All other experiments were performed on a Linux platform with two 1.7 Xeon GHz processors, a 256KB cache, and 1GB of main 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 Harwell-Boeing collection [22]. The linear systems are obtained from all those matrices from the RUA collection whose size is 500 or higher. There are 58 matrices that satisfy this criterion and their names are

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

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 four 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 algorithnm of dynamic averages 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 (10). 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 [35] 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 $fill_{max} * nnz(A_l)/n$. Another parameter not shown here is the cut-off dimension for the last level which is set to 100. This means that no further levels are sought when the dimension of the Schur complement is ≤ 100 or when the maximum number of levels allowed is reached. We ran ARMS-C with different matching strategies with the *exact same parameters given above for all 58 linear systems.*

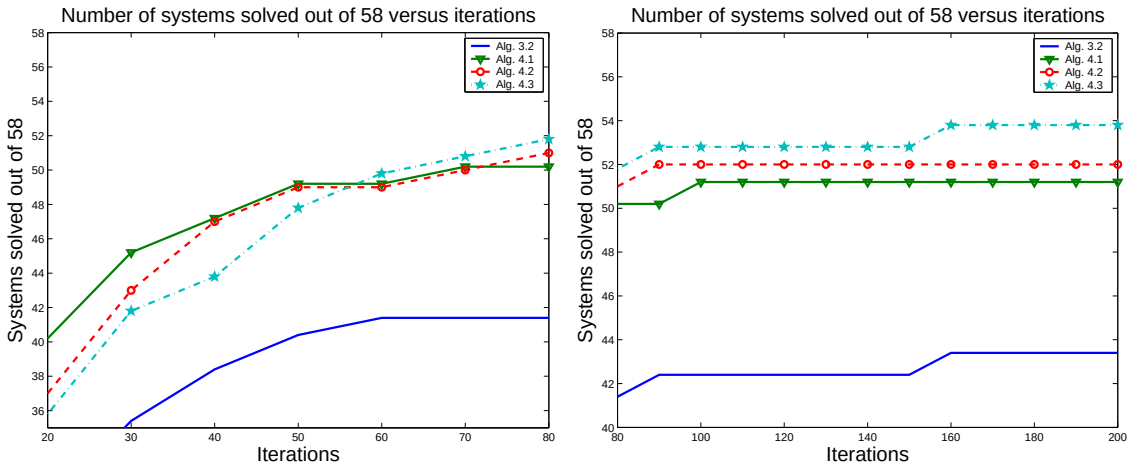

Figure 2: Experiment with 58 HB matrices showing the number of converged systems among 58, versus iteration count.

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 Figure 2 shows the number of successful solves requiring iteration counts of 80 or less and the right side those requiring more than 80 steps. 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 [35]). 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

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

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 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.

ILUTP was able to solve some of the problems for which it failed in the above test. However, this was achieved at the cost of much more memory.

6.2 Effect of the filtering parameter τ_0

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 [16]. This matrix is known to cause difficulties to standard ILUT and ILUTP preconditioners. 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 observe 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

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

decrease, reaching the smallest number of 27 when $\tau_0 = 0.7$. 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 $\tau_0 = 0.9$. For a small value of τ_0 more entries are initially preselected. For example when $\tau_0 = 0$ all the largest entries in each row are candidates. 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 $\tau_0 \geq 0.7$ 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 $\tau_0 = 0.8$ and $\tau_0 = 0.9$, 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

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

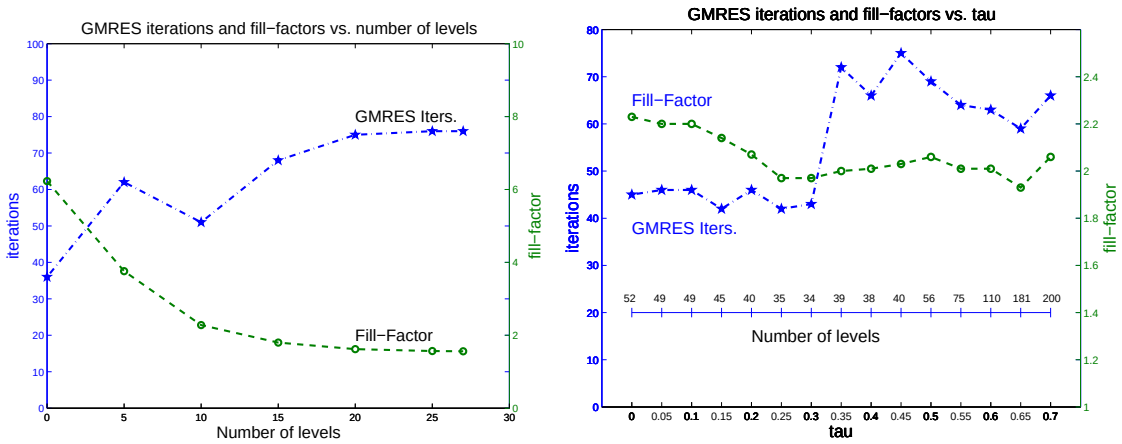

Figure 3: Iteration and fill-factor for the driven cavity problem, versus the number of levels (left-side) and τ_0 parameter (right-side). The right plot uses a slightly different set of parameters.

		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 and a max fill-in of $6 \times nz/n$. Figure 3 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 3 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 [28] 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 [28] 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 [23, 24, 33] whose goal is to put large entries on the diagonal. Two sets of problems are tested in [28]. 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 solving five KKT problems with ARMS-C.

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 [28]. Note that the times are not directly comparable with those of Table 4.

Table 4 shows the same statistics as those of Table I of [28] 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 [28]. As with earlier tests we do not attempt in any way to tune the parameters for each separate case. These were selected to yield a comparable number of nonzero elements for the preconditioners as in [28]. 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 MC64-ILUT combination from [28]. As can be seen for similar fill-in, the number of iterations obtained in each case are comparable. Note that the accelerator used in [28] 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 [28] 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 [28] were optimized with the use of ATLAS BLAS. In addition, the algorithms in [28] 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 [28]. 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.

Matrix	order	nonzeros	Application (Origin)
barrier2-9	115,625	3,897,557	Semiconductor Device (UFL, Schenk set)
matrix_9	103,430	2,121,550	Semiconductor Device (UFL, Schenk set)
matrix-new_3	125,329	2,678,750	Semiconductor Device (UFL, Schenk set)
ohne2	181,343	11,063,545	Semiconductor Device (UFL, Schenk set)
para-4	153,226	5,326,228	Semiconductor Device (UFL, Schenk set)
cir2a	482,969	3,912,413	circuit simulation
scircuit	170998	958936	Digital circuit simulation (UFL, Hamm set)
circuit_4	80209	307604	Circuit simulation (UFL, Bomhof set)
wang3.rua	26064	177168	Device simulation (UFL, Wang set)
wang4.rua	26068	177196	Device simulation (UFL, Wang set)

Table 6: General information on 10 test matrices

6.5 Tests with larger matrices

Apart from the test in Section 6.2 with the matrix `twotone`, the linear systems of the earlier numerical tests are all relatively small. In this section we report on some tests performed with larger, more realistic, matrices arising from applications which are known to cause difficulties to iterative solvers. We selected 10 matrices, most of which have already been used as test problems in the literature. The first 5 matrices were taken from [38]. They are the largest linear systems in size taken from [38], if the matrix `pre2` therein is ignored. For the system `pre2` no method among those reported in [38] worked and a similar conclusion was reached with ARMS-C. These 5 matrices, which originate from device simulation, are available from the `Schenk` sets of the University of Florida collection [16]. Along with these matrices, we also selected the two largest matrices from the paper [2], namely the matrix `circ2a` and the matrix `scircuit`. The matrix `scircuit` can also be obtained from the set `Hamm` of the University of Florida collection, while the larger matrix `circ2a` was sent to us by Achim Basermann [2]. Finally, we added the 2 largest matrices (`wang1,`

wang2) from the Wang set of the University of Florida collection, and the largest matrix (circuit_4) from the set Bomhoff of the same collection. These examples are all from circuit simulation. Some general information on all 10 matrices is shown in Table 6.

We tested ARMS-C, based on the ddPQ ordering given by Algorithm 4.3 on all 10 matrices with the following parameter settings:

		Drop tolerance				Fill-max			
$nlev_{max}$	tol_{DD}	LU-B	GW	S	LU-S	LU-B	GW	S	LU-S
40	0.1	0.01	0.01	0.01	1.e-05	3	3	3	20

The parameters for the ILUTP preconditioner for the last level essentially lead to a direct solve. Another parameter not shown here is the cut-off dimension for the last level which is set to 300. No further levels are sought when the dimension of the Schur complement is ≤ 300 or when the maximum number of levels allowed (40) is reached.

We ran two tests with the GMRES accelerator, one with a restart dimension of 60, the other with a restart dimension of 100. Note that memory is allocated dynamically for the basis vectors required by GMRES(k). In both cases, the iteration was stopped whenever either a maximum number of iterations of 300 was reached or the initial residual was reduced by 10^{-8}. The times are in seconds on the 1.7Ghz Xeon processor mentioned earlier. The iteration times are repeated for the case when GMRES(60) and GMRES(100) take the same number of iterations (these correspond to the exact same operations and should be the same under ideal conditions).

Of all the matrices tested, only scircuit failed to reduce the residual by 8 orders of magnitude in 300 steps with the set of parameters selected above. In 300 steps the residual norm was reduced by $\approx 1.8 \times 10^{-07}$ by GMRES(100) and by 3.13×10^{-7} by GMRES(60). This particular matrix is a good representative of those situations mentioned earlier in which row/column scaling does not necessarily help. If the exact same parameters are used, but no row or column scaling is performed (set by parameters in ARMS-C) then, as is shown in Table 8, GMRES(100) does succeed in reducing the residual norm by 8 orders of magnitude in 297 steps, while GMRES(60) also achieves the same result in 400 steps. Table 8 also shows an additional test for the same matrix when no scaling is used. Here the drop tolerance parameters labeled "LU-B", "GW", and "S", which were set to 0.01

	Fill		Set-up	GMRES(60)		GMRES(100)	
Matrix	Factor	Levels	Time	Its.	Time	Its.	Time
barrier2-9	0.62	5	4.01e+00	113	3.29e+01	93	3.02e+01
matrix-new_3	0.89	8	7.53e+00	40	1.02e+01	40	1.00e+01
matrix_9	1.77	9	5.53e+00	160	4.94e+01	82	2.70e+01
ohne2	0.62	6	4.34e+01	99	6.35e+01	80	5.43e+01
para-4	0.62	5	5.70e+00	49	1.94e+01	49	1.93e+01
wang3	2.33	4	8.90e-01	45	2.09e+00	45	1.95e+00
wang4	1.86	4	5.10e-01	31	1.25e+00	31	1.20e+00
scircuit	0.90	4	1.86e+00	Fail	7.08e+01	Fail	8.80e+01
circuit_4	0.75	2	1.60e+00	199	1.69e+01	96	1.07e+01
circ2a	0.76	3	2.19e+02	18	1.08e+01	18	1.03e+01

Table 7: Solution of the 10 systems in Table 6 with GMRES(60) and GMRES(100) preconditioned with ARMS-C.

	Fill Factor	Levels	Set-up Time	GMRES(60)		GMRES(100)	
				Its.	Time	Its.	Time
Same parameters	0.89	4	1.81e+00	400	9.13e+01	297	8.79e+01
Drop-tol = 0.001	1.00	5	1.89e+00	98	2.23e+01	82	2.27e+01

Table 8: Solution of the system `scircuit` when no scaling is used, using two different sets of parameters.

in this experiment, are changed to 0.001. The resulting convergence is much faster. It should be noted that convergence is also achieved with these values of the drop tolerance for the situation when row/column scaling is used, although it is much slower.

Observe that the fill-factors, the memory required for the preconditioner normalized by that required for the original matrix A, are generally quite modest. It is less than one in 7 of the 10 cases. Not seen in the reported results is a peculiar behavior of the solver with the matrix `circuit_4` as well as with the matrix `circ2a`. In contrast with most other systems, results with `circuit_4` were exceedingly sensitive to parameters and even to the initial guess (which is always a random vector).

Regarding the matrix `circ2a`, it can be observed in the table that the time to compute the preconditioner appears to be abnormally high, relative to the other test cases. This matrix is the largest in size among the 10 tested, but it is not the largest in terms of the number of nonzero entries. The first reaction to this might be that the algorithms have some hidden cost associated with the size n rather than the nonzero entries. A test with the matrix `pre2` mentioned earlier disproved this. This matrix is of size $n = 659,033$ and has $nz = 5,959,282$ nonzero entries. Yet, the same parameters produced the factorization in almost an order of magnitude less time (24.2 seconds to be exact) on the same platform. The fill-factor was 0.99 which was even higher than that for `circ2a`. We examined the sizes and numbers of nonzero entries of the matrices obtained at each level. The method stopped prematurely with a C block of size zero yielding the previous level (3rd one in this case) as the last level. This level had a size of 4,679 which is somewhat large. Indeed, no fill-minimizing ordering is performed prior to factoring the last Schur complement, because in most cases the last Schur complement is a small matrix of size a few hundreds at most. However, further examination revealed that, quite remarkably, this last Schur complement is a diagonal matrix. This in fact explains why the algorithm stopped prematurely. In the end, we reached the only plausible explanation that some intermediate matrices that were generated were quite dense, before they were sparsified by dropping. Selecting a different filter parameter τ_0 can force a larger number of levels and reduce the set-up substantially in this case. For example, with $\tau_0 = 0.7$, 18 levels were used, with a fill factor of 0.77, quite close to the previous case. The set-up time was reduced to 43 seconds. The number of GMRES steps was also reduced from 18 to 12 (but the iteration time increased slightly to 12.1 seconds due to the larger number of levels).

6.6 Available software and other tests

The algorithms described in this paper are part of a recent package, called ILUPACK, developed primarily by Matthias Bollhöfer [11]. The package is available from Bollhöfer's web-site, see [12]. It offers a large selection of preconditioners, many of which utilize the inverse-based dropping technique

mentioned in the introduction, see also [9]. Along with software, the site offers also a repository of statistics on a sizable number of numerical experiments with various methods. These experiments show in particular a number of comparisons with the similar MC-64 strategy.

7 Conclusion

We conclude with a few remarks and observations regarding the performance of ARMS-C and on issues remaining to be examined. Our first observation is that the method seems to perform well as a general purpose alternative to a direct solver. Though the algorithms do not compete with the robustness of direct solvers, they can often tackle problems that were inaccessible to iterative solvers so far, at a small fraction of the memory and computational costs required by direct solvers. Note also that the methods perform in a similar way for problems with a very poor structure as for systems which arise from discretized PDEs. There are, however, a number of questions that remain to be answered and issues to be examined for a better understanding of the general approach proposed in this paper.

One of the primary concerns with a strategy such as the one described in this paper is the large number of tunable parameters available. With such variety it may be indeed difficult to find the best fit for a given linear system. However, in a software package, one may simply fix most of these parameters without causing serious harm to performance. The primary candidates for this 'hard-wiring' are the maximum number of levels allowed (along with the minimum size of the Schur complement before switching to ILUTP), and the diagonal dominance filtration parameter τ_0. Indeed, when the better matching algorithms are used (Alg. 4.3 or Alg. 4.2) performance does not vary too much with these parameters for *most* problems. The other parameters left are those that control fill-in, i.e., the accuracy of the factorization. Although our research code provides separate fill-in parameters for various parts of the factorization at a given level, these can be taken to be the same as was done in most of our experiments. The drop tolerance and fill-in parameters can be fixed to provide a very accurate factorization for the last level Schur complement matrix. In the end all that remains to select are the drop tolerances and the max fill-in allowed for the interlevel matrices (those for LU-B and GW in the various tables). This would not be too different from the parameters required by ILUT [35].

We observed that fill-in does not necessarily improve the quality of the factorization. In fact the contrary is often seen. While the same observation can be made for ILUT, we note that it is not as common. This behavior underscores the difficulty in predicting the effect of dropping small terms in the factors, on the conditioning of the preconditioned system. The dropping strategies which were tested utilize a simple threshold criterion but it is clear that one based on estimating the condition number of the factors [9] would be ideal in this situation.

A diagonal dominance criterion was selected as a means of building a good B block, 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 is possible to find a more elaborate criterion, for example 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.

Another important issue that deserves further investigation is that of scaling. As was reported, the performance of the preconditioners is significantly affected by scaling. Yet, it still remains difficult to devise scaling strategies whose effects on the preconditioner are well understood and predictable.

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.

Acknowledgements. The code ARMS-C discussed in the experiments has been adapted from ARMS [36], a C code which was developed jointly with Brian Suchomel. I am grateful to Matthias Bollhöfer for numerous insightful discussions on the topic of this paper. The numerical experiments would not have been possible without the availability of a sizable collection of test matrices. I would like to thank John Haws for providing the matrices used in Section 6.4, Achim Basermann for sending us his circuit simulation matrices, and Olaf Schenk for making available the large device and circuit matrices discussed in Section 6.5. Special thanks to Tim Davis for making these and other linear systems available in his excellent repository of test matrices. Finally, I am grateful to the referees for their numerous suggestions which helped improve the quality of this paper.

References

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

[2] A. Basermann, U. Jaekel, and K. Hachiya. Preconditioning parallel sparse iterative solvers for circuit simulation. In *SIAM conference on Applied Linear Algebra, July 15-19, 2003, Williamsburg, VA*, pages CP–2, 2004.

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

[4] M. Benzi, R. Kouhia, and Tůma. Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics. *Comput. Methods Appl. Mech. Engrg.*, 190:6533–6554, 2001.

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

[6] M. Benzi and Tůma. Orderings for factorized sparse approximate inverse preconditioners. *SIAM Journal on Scientific Computing*, 21:1851–1868, 2000.

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

[8] M. Benzi and M. Tůma. A robust incomplete factorization preconditioner for positive definite matrices. *Numer. Lin. Alg. w. Appl.*, 10:385–400, 2003.

[9] 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.

[10] M. Bollhöfer and Y. Saad. On the relations between ILUs and factored approximate inverses. *SIAM Journal on Matrix Analysis and Applications*, 24:219–237, 2002.

[11] M. Bollhöfer and Y. Saad. Multilevel preconditioners constructed from inverse–based ILUs. Technical Report umsi-2004-75, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2004. submitted.

[12] Matthias Bollhöfer and Yousef Saad. ILUPACK - preconditioning software package, release v1.0, may 14, 2004. Available online at http://www.tu-berlin.de/ilupack/.

[13] 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.

[14] E.F.F. Botta and F.W. Wubs. Matrix Renumbering ILU: an effective algebraic multilevel ILU. *SIAM Journal on Matrix Analysis and Applications*, 20:1007–1026, 1999.

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

[16] T. Davis. University of Florida sparse matrix collection, http://www.cise.ufl.edu/research/sparse, June 1997.

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

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

[19] I. S. Duff. Algorithm 575: Permutations for a zero-free diagonal. *ACM Transactions on Mathematical Software*, 7:387–390, 1981.

[20] I. S. Duff. On algorithms for obtaining a maximal transversal. *ACM Transactions on Mathematical Software*, 7:315–330, 1981.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] G. H. Golub and C. Van Loan. *Matrix Computations, 3rd edn.* The John Hopkins University Press, Baltimore, 1996.

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

[28] J. C. Haws and C. D. Meyer. Preconditioning KKT systems. Technical Report M&CT-TECH-02-002, Boeing, 2002.

[29] 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.

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

[31] N. Li, Y. Saad, and E. Chow. Crout versions of ILU for general sparse matrices. *SIAM Journal on Scientific Computing*, 25(2):716–728, 2003.

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

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

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

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

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

[37] 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*, 21:279–299, 2000.

[38] O. Schenk, S. Röllin, and A.Gupta. The effects of nonsymmetric matrix permutations and scalings in semiconductor device and circuit simulation. *IEEE trans. on Computer Aided design of Integrated Circuits and systems*, 23(3):400–411, 2004.