Greedy coarsening strategies for non-symmetric problems*

S. MacLachlan† Yousef Saad†

May 26, 2006

Abstract

The solution of large-scale linear systems in computational science and engineering requires efficient solvers and preconditioners. Often the most effective such techniques are those based on multilevel splittings of the problem. In this paper, we consider the problem of partitioning non-symmetric matrices based solely on algebraic criteria. A new algorithm is proposed that combines attractive features of two previous techniques proposed by the authors. The effects of further matrix reorderings within the fine-scale block are also considered. Numerical results show that the new partitioning scheme leads to improved results for a variety of problem.

1 Introduction

Recent advances in the solution of linear systems of equations have been driven by the ever-increasing problem sizes required in computational science and engineering applications. Matrices of interest arise from both discretizations of differential equations that govern physical systems, where accuracy considerations demand fine meshes, and from naturally discrete problems with many degrees of freedom. These systems are typically not only large, but also ill conditioned, requiring advanced techniques for efficient solution.

For many such large-scale linear systems, the most efficient solution techniques utilize multilevel frameworks. For elliptic PDE-based problems, multigrid [11] and algebraic multigrid [7, 24] methods have been demonstrated to have optimal efficiency. For more general problems, however, classical multigrid approaches do not perform as well without more expensive approaches to the multigrid setup phase [8, 9]. While purely algebraic approaches, such as the Algebraic Recursive Multilevel Solver (ARMS) [29] and other multilevel ILU techniques [4, 6] do not typically match the performance of multigrid for discretizations of elliptic PDEs, their robustness across many problems makes them an attractive option when complete knowledge of a problem's origin cannot be guaranteed.

When the system matrix is symmetric and positive definite, theoretical analysis gives insight into the requirements on the partitioning into fine-scale and coarse-scale degrees of freedom. Analysis of multilevel block factorization preconditioners, such as ARMS, shows that it is crucial that the fine-scale submatrix be well approximated, in a spectral sense, by the fine-scale part of the preconditioner in order to achieve effective results [23]. Motivated by this analysis and corresponding theory for multigrid for symmetric and positive-definite matrices [15, 20], we have previously developed a partitioning algorithm for the symmetric case [21]. In this approach, the optimal partition is defined in terms of a diagonal-dominance property of the fine-scale block. Exactly achieving the optimal coarsening was shown to be an NP-complete problem and, so, an $O(N)$ approach to approximately achieve this objective, based on a greedy strategy, was used instead.

In this paper, we consider the extension of the approach from [21] to non-symmetric problems. A direct generalization using symmetric permutations is quite natural, although may not be effective when the matrix is strongly non-symmetric. A more effective approach for such problems is to consider a non-symmetric permutation strategy, similar to that of [27]. We extend our earlier greedy approach to such non-symmetric permutations, seeking a permutation where the fine-scale block is row diagonally dominant. These two approaches are compared on a variety of non-symmetric matrices. Additionally, we consider the

*Work supported by NSF under grant ACI-0305120, and by the Minnesota Supercomputing Institute

†Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455. email: {maclach,saad}@cs.umn.edu

effectiveness of the nonsymmetric approaches discussed here on the symmetric test problems of [21], without looking to exploit symmetry.

Another important question in the ARMS technique is the ordering of the fine-scale block, A_{ff}, so that it may be efficiently approximated by its ILUT factors. Here, we consider various approaches to reordering the fine-scale block based on standard techniques such as those of Sparspak [16] and METIS [19]. The effects of reordering on the performance of incomplete factorization preconditioners has been considered previously [10, 14]. Here, we extend the work of [28] and examine the effects of reordering on multilevel ILU proconditioners, such as ARMS.

This paper is organized as follow. In Section 2, we present the ARMS algorithm and the greedy partitioning strategy. Section 3 details the extension of these approaches to non-symmetric problems, with particular focus on non-symmetric permutations. The important question of ordering the fine-scale block is considered in Section 4. Numerical results, for both symmetric and non-symmetric problems, are presented in Section 5, followed by conclusions in Section 6.

2 Background

2.1 The Algebraic Recursive Multilevel Solver (ARMS) Algorithm

The ARMS algorithm [29] arises from considering the block factorization of a given matrix, A. Partitioning the degrees of freedom of A into two sets, F and C, referred to as the fine-grid and coarse-grid degrees of freedom (respectively), the (reordered) matrix can then be written as a block 2×2 system,

$$A = \begin{bmatrix} A_{ff} & -A_{fc} \\ -A_{cf} & A_{cc} \end{bmatrix}.$$

This block form can then be directly factored, as

$$A = \begin{bmatrix} I & 0 \\ -A_{cf}A_{ff}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{ff} & 0 \\ 0 & \hat{A}_{cc} \end{bmatrix} \begin{bmatrix} I & -A_{ff}^{-1}A_{fc} \\ 0 & I \end{bmatrix}, \tag{1}$$

where $\hat{A}_{cc} = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$ denotes the Schur complement of A. From this factored form, an algorithm for direct solution of $A\mathbf{x} = \mathbf{b}$ is apparent (partitioning the vectors, $\mathbf{x} = \begin{pmatrix} \mathbf{x}_f \\ \mathbf{x}_c \end{pmatrix}$ and $\mathbf{b} = \begin{pmatrix} \mathbf{b}_f \\ \mathbf{b}_c \end{pmatrix}$):

Algorithm 1 (Block Factorization Solve of $A\mathbf{x} = \mathbf{b}$).

1. $\mathbf{y}_f = A_{ff}^{-1}\mathbf{b}_f$
2. $\mathbf{y}_c = \mathbf{b}_c + A_{cf}\mathbf{y}_f$
3. Solve $\hat{A}_{cc}\mathbf{x}_c = \mathbf{y}_c$
4. $\mathbf{x}_f = \mathbf{y}_f + A_{ff}^{-1}A_{fc}\mathbf{x}_c$

For a general F/C partition, there is no advantage in using Algorithm 1 over directly factoring A. The advantage of the block factorization approach occurs when considering preconditioners, however, if the partition is chosen so that the diagonal-block matrices, A_{ff} and $\hat{A}_{cc}$, are easily (approximately) inverted. In this case, solution with an approximate block factorization can be a good preconditioner for a Krylov subspace method. Many variations on this approach have been considered; see, for example, [2, 3, 6].

In the ARMS methodology, the inversion of A_{ff} is approximated through its ILUT factors. That is, we write $A_{ff} \approx LU$, where the L and U factors are truncated based on size and number of non-zeros per row [25] and approximate A as in Equation (1) by

$$B = \begin{bmatrix} I & 0 \\ -A_{cf}U^{-1}L^{-1} & I \end{bmatrix} \begin{bmatrix} LU & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & -U^{-1}L^{-1}A_{fc} \\ 0 & I \end{bmatrix},$$

The application of the preconditioner, B, to a residual, $\mathbf{r}$ is then given by Algorithm 2.

Algorithm 2 (Action of ARMS preconditioner on residual, $B^{-1}\mathbf{r}$).

1. $\mathbf{y}_f = L^{-1}\mathbf{r}_f$
2. $\mathbf{y}_c = \mathbf{r}_c + A_{cf}U^{-1}\mathbf{y}_f$
3. Solve $S\mathbf{x}_c = \mathbf{y}_c$
4. $\mathbf{z}_f = \mathbf{y}_f + L^{-1}A_{fc}\mathbf{x}_c$
5. $\mathbf{x}_f = U^{-1}\mathbf{z}_f$

Here, S is an approximation to $\hat{A}_{cc} = A_{cc} - A_{cf}A_{ff}^{-1}A_{fc}$, computed using $A_{ff}^{-1} \approx U^{-1}L^{-1}$ and similar truncation strategies to those of ILUT. While the solution of $S\mathbf{x}_c = \mathbf{y}_c$ in Step 3 could be considered directly, the set, C, is generally still large enough that this is quite costly. Instead, the ARMS methodology is applied recursively to solve the system with S, stopping only when the dimension of the coarsest-scale problem is small enough that direct solution is practical.

It is apparent that the success of Algorithm 2 as a preconditioner is dependent on the accuracy of the factorization, $A_{ff} \approx LU$. The accuracy of this factorization, in turn, depends on the partition chosen to define F. If the partition allows direct factorization of A_{ff} with no fill (if the graph of A_{ff} is, for example, a tree), then the factorization may be done very accurately at low cost. Unfortunately, this is usually not possible without choosing F to be prohibitively small. In contrast, however, choosing C to be small often leads to an A_{ff} block whose inverse is difficult to approximate in a sparse manner. The contrasting goals of sparse factorization and effective coarsening lead to partitioning algorithms that aim at a compromise.

2.2 Original ARMS Partitioning

The original ARMS algorithm [29] is a natural extension of the ILUM, BILUM, and BILUTM algorithms. The ILUM (Multi-elimination ILU) algorithm [26] is a block-factorization preconditioner, where the partitioning is chosen so that the fine-scale block, A_{ff}, is diagonal (that is, F is an independent set of the degrees of freedom). As A_{ff} is diagonal, it is easily inverted, and the block-factorization solve may be easily implemented (although it is often useful to allow some dropping in the computation of the Schur complement to limit fill). The BILUM (Block ILUM) algorithm [30] extends this idea using small block-independent sets to form F, yielding a block-diagonal matrix, A_{ff}. In BILUTM (Block Multilevel ILUT) [31], the block-independent sets used to form F were allowed to be much larger, leading to inefficiency in computing the exact inverse of the diagonal blocks of A_{ff}. Instead, the ILUT algorithm was used to compute approximate inverses of the large diagonal blocks in A_{ff}, where the block independent sets are formed using a domain decomposition approach.

In [29], the idea of diagonal dominance was first introduced into the partitioning stage. Row-wise diagonal-dominance coefficients of A are initially computed as $\hat{w}(i) = \frac{|a_{ii}|}{\sum_{j=1}^{n}|a_{ij}|}$, then scaled by the maximum dominance ratio, giving $w(i) = \frac{\hat{w}(i)}{\max_j \hat{w}(j)}$. An initial fine-scale block is then chosen using a similar approach as before, either by choosing an independent set of the degrees of freedom, or by choosing block-independent sets (using, for example, nested dissection [16]). Points from this set are, however, rejected based on thresholding of the diagonal-dominance ratios, $w(i)$. If a row, i, chosen to be in F is not sufficiently diagonally dominant (if $w(i) < \theta$, for some predetermined θ), then row i is rejected from the fine set, and switched to a coarse-grid point.

The introduction of a diagonal-dominance measure is motivated by considering the ILUT employed in approximating the inverse of A_{ff} in the ARMS algorithm (Algorithm 2). The accuracy of the preconditioning depends on how accurately A_{ff} is represented by its ILUT factors. The efficiency, however, depends on how sparse these factors are. To best balance these competing concerns, we seek to choose F so that A_{ff} is well approximated by sparse ILUT factors. To achieve this balance, we reject rows from F that are poorly diagonally dominant, as these are rows in which we expect many significant off-diagonal entries in the ILUT factors. Put another way, we can most easily compute accurate, sparse ILUT factors of A_{ff} when it is strongly diagonally dominant; thus, we reject rows from F that are more likely to cause fill.

2.3 Greedy Partitioning

While the use of a diagonal-dominance measure as in [29] is an important improvement in the ARMS algorithm, the approach used there is static; that is, the evolving fine/coarse partition plays no role in the computation or use of these measures. A dynamic approach to diagonal dominance was recently introduced in [21], based on the observation that it is the dominance strictly within A_{ff} that determines how well A_{ff} may be approximated by its ILUT factors, and not just the dominance of the F rows of A (as is measured by the ratios in [29]).

In [21], the measures, $w(i)$, are replaced by dynamic measures, $\hat{\theta}_i = \frac{|a_{ii}|}{\sum_{j \in F \cup U} |a_{ij}|}$, where the degrees of freedom of A are assumed to either have been already partitioned into F and C, or to be in U, the set of undecided points. Thus, $\hat{\theta}_i$ is a measure of the dominance of row i over all points that are either in F or have the potential to be in F. If row $i \in U$ is dominant over this set ($\hat{\theta}_i \geq \theta$ for some predetermined θ), then i is added to set F. If there are no such points, then at least one column from U must be added to C in order to determine if the dominance of the rows in U can be improved to the point where they make good F points, or if they should also be discarded into C. For the case of symmetric and positive-definite matrices, as considered in [21], the partition of rows and columns into F and C was also symmetric (that is, if row $i \in C$, then so is column i); so, the row/column pair associated with the least diagonally dominant row is chosen to be the new C point. Measures for all neighboring U rows are then updated, with any rows whose measures are now large enough added to F.

This greedy approach (so named because at each step, all sufficiently-dominant rows are added to F and the least-dominant row is discarded into C) is considered as an approximation to the "ideal" partitioning of the degrees of freedom of A. If diagonal dominance of A_{ff} is all that we need to ensure a good approximation of A_{ff} by its ILUT factors (in terms of both accuracy and sparsity), then the ideal partitioning is the one which maximizes the size of F, given the constraint that for all rows, $i \in F$, $\theta_i = \frac{|a_{ii}|}{\sum_{j \in F} |a_{ij}|} \geq \theta$, for some predetermined θ. In [21], we showed that finding the ideal partitioning for arbitrary A is an NP-complete problem. This greedy approach approximates the ideal partitioning in such a way that we are guaranteed that the A_{ff} block is diagonally dominant and led to effective ARMS performance for a number of symmetric problems.

3 Non-symmetric Partitioning Approaches

While the symmetric partitioning approaches described above work well for many symmetric and some non-symmetric problems, they do not always effectively partition the rows and columns of A. Consider, for example, taking a diagonally dominant tridiagonal matrix and cyclicly permuting all entries two columns to the right. The resulting matrix is no longer diagonally dominant (indeed, the diagonal entries are all zero), and either of the above algorithms would fail to find any F points. A more robust approach is to define separate left and right permutation matrices, allowing large off-diagonal entries to be moved to the diagonal of the reordered system and used as pivots in the ILUT factorization.

3.1 Two-stage algorithm

In [27], a two-stage approach is used to find a non-symmetric permutation that results in an A_{ff} block that is expected to have its largest entries on the diagonal. A preselection stage is used to first identify rows that are good potential rows for A_{ff} and, then, sort these rows based on a measure of their likelihood to produce sparse ILUT factors. In the second stage, the nonsymmetric permutations are defined, by traversing this list (in order), permuting the dominant element for the row to the diagonal, if possible, and discarding rows for which a suitable permutation cannot be defined. In this way, a nonsymmetric permutation pair is constructed that yields an A_{ff} block that is expected to yield sparse ILUT factors based on a simple greedy strategy.

The preselection stage itself first discards rows with no significant dominant element, then orders the remaining rows based on evaluating their potential to yield sparse ILUT factors. For each row, i, the dominant element in row i, $k_i = \text{argmax}_k |a_{ik}|$, is first calculated, along with the row's ℓ_1 norm, $t_i = \sum_j |a_{ij}|$. If t_i is small enough, relative to $|a_{ik_i}|$, measured by a preselected tolerance, then row i is admitted as a candidate

row for A_{ff}. Diagonal dominance alone is not enough, however, to guarantee sparse ILUT factors. Indeed, a row with only two equal-sized nonzero entries is more attractive than a row with one large entry and many small entries. To address this issue, the dominance ratios, $\frac{|a_{ik_i}|}{\sum_j |a_{ij}|}$, are then multiplied by $\frac{1}{|\operatorname{Adj}(i)|}$, where $\operatorname{Adj}(i) = \{j \neq i : a_{ij} \neq 0\}$, and $|S|$ denotes the number of elements in set S. These weighted dominance ratios are then sorted in decreasing order to establish a ranking of the attractiveness of row i to be included in A_{ff}.

Many possible strategies exist for selecting which of the rows passed by the preselection algorithm described above are then included in A_{ff} [27]. A simple greedy approach is to scan these rows in order, building a non-symmetric permutation by accepting row i and column k_i into the F block if neither has already been accepted. As rows appear at most once in the preselection list, this only requires checking if column k_i has already been used as a diagonal by another row already selected for F. If so, row i is discarded to C, otherwise it is accepted, and node (i, k_i) is permuted to the diagonal.

While this approach is successful for many problems, it does not always yield an effective preconditioner. Adding constraints on A_{ff} such as that it is diagonally dominant may yield a smaller block, but a better overall preconditioner (as the ILUT factors of A_{ff} are both sparser and more accurate). A strategy based on a dynamic measure that guarantees row diagonal dominance within the A_{ff} block of the reordered matrix is proposed in [27]. In this approach, for each row, i, accepted by the preselection algorithm, if column k_i has not already been reordered and element $|a_{ik_i}|$ dominates row i over those columns already added to F, row i and column k_i are added to A_{ff}. Additionally, for each remaining column, $j \in \operatorname{Adj}(i)$, that has not already been added to F or C, if a_{ij} is large enough such that row i would fail to be diagonally dominant in A_{ff} if $|\operatorname{Adj}(i) \setminus (F \cup C)|$ entries of size a_{ij} are added to it, column j is rejected from possibly becoming an F column, and added to C. In this way, the A_{ff} block of the reordered system is guaranteed to be row diagonally dominant, but columns that may have been acceptable choices for the A_{ff} block are rejected based on an averaged expectation, not their unique contribution.

3.2 Single-stage greedy non-symmetric partitioning

Here, we propose a technique that combines the nonsymmetric permutation approach of [27] (described in §3.1) with the single-stage greedy algorithm approach of [21] (described in §2.3). The resulting algorithm once again guarantees that the A_{ff} block of the reordered system is diagonally dominant, but is much more reluctant to move columns into C. As such, we expect the resulting algorithm to form similar partitions to those chosen by [27] when that algorithm performs well (i.e., when the chosen F blocks are not too small), but to show improved performance when the average dominance criterion described above is too aggressive at removing columns.

As with the diagonal-dominance measure used in [21], row i of matrix A is defined to be θ-dominated by column k if $\frac{|a_{ik}|}{\sum_j |a_{ij}|} \geq \theta$. As before, we seek to find a partitioning of the rows and columns of A so that each row in A_{ff} is θ-dominated by its diagonal. To do this, consider the three sets of undecided points, U, fine points, F, and coarse points, C but, now, we will maintain different sets for both rows and columns. A point in U_{row} is made into an F-row if, at any point,

$$\frac{|a_{ik_i}|}{\sum_{j \in F_{\text{col}} \cup U_{\text{col}}} |a_{ij}|} \geq \theta,$$

where $k_i = \underset{k \in U_{\text{col}}}{\operatorname{argmax}} |a_{ik}|$. If row i is put into F_{row}, then column k_i must also be put into F_{col}, so that the entry, a_{ik_i}, may be permuted into a diagonal position in A_{ff}. If, at any point, row i is zero over all columns in U_{col}, then it is placed in C_{row}, as it is no longer possible to θ-dominate this row with any valid pivot element.

Rows and columns are always moved into the F sets in pairs, defining a nonsymmetric permutation with a square A_{ff} block. Eliminating rows and columns from the U sets, however, should occur independently, as if row i is no longer sufficiently dominated by an eligible pivot to be put in F_{row}, it does not necessarily mean that any of the columns in $\operatorname{Adj}(i) \cap U_{\text{col}}$ would not make a good pivot for another row in U_{row}. Thus, the elimination of rows and columns from U_{row} and U_{col} into the C sets occurs independently, as follows.

If, at some point in the algorithm, no row in U_{row} is sufficiently dominated by a column in U_{col} such that another row/column pair can be added to F, a single column, $j^\star \in U_{\text{col}}$, that is deemed to be least attractive as a column in A_{ff} is selected to be added to C_{col}. For each $i \in U_{\text{row}}$ such that $a_{ij^\star} \neq 0$, the dominance ratio of row i is updated, by decrementing the ℓ_1 norm by $|a_{ij^\star}|$ and, possibly, recomputing the dominant column, $k_i \in U_{\text{col}}$. If $a_{ij^\star}$ is the last non-zero entry in row i's restriction to U_{col}, then row i is added to C_{row}. Otherwise, if the updated $\hat{\theta}_i = \frac{|a_{ik_i}|}{\sum_{j \in F_{\text{col}} \cup U_{\text{col}}} |a_{ij}|} \geq \theta$, then the pair (i, k_i) is added to the F sets. If, at any point when a column is moved into F_{col} or C_{col}, a row, i, is no longer sufficiently dominated over F_{col} by some $k \in U_{\text{col}}$, that is $\frac{|a_{ik}|}{\sum_{j \in F_{\text{col}}} |a_{ij}|} < \theta$ for all $k \in \text{Adj}(i) \cap U_{\text{col}}$, then there is no possible pivot column for row i that would result in a diagonally dominant A_{ff}, and row i is discarded into C. This pattern continues until either U_{row} or U_{col} is empty.

There are many possible measures of which column, $j^\star$, is chosen as the least attractive potential F column. In the symmetric permutation case considered in [21], $j^\star$ was chosen based on the attractiveness of row $j^\star$ as an F-row. Here, we consider the row diagonal-dominance measures,

$$\hat{\theta}_i = \frac{|a_{ik_i}|}{\sum_{j \in F_{\text{col}} \cup U_{\text{col}}} |a_{ij}|},$$

from which the decrease of $\hat{\theta}_i^{-1}$that occurs when column $j^\star$ is moved from U_{col} to C_{col} is easily quantified as $\frac{|a_{ij^\star}|}{|a_{ik_i}|}$. The column that is most attractive as a C column, then, is the one that brings about the greatest cumulative change in the measures, $\hat{\theta}_i$, for $i \in U_{\text{row}}$ measured, for example, as $\sum_{i \in U_{\text{row}}} \frac{|a_{ij}|}{|a_{ik_i}|}$ for each $j \in U_{\text{col}}$.

The resulting algorithm is then summarized as follows, where we ignore the important question of computational complexity and suppress the updating of k_i for $i \in U_{\text{row}}$, which is assumed to always reference the largest entry, a_{ik}, over $k \in U_{\text{col}}$. In Algorithm 3, all second-order updates are also suppressed, in particular, the elimination of rows to C when new dominating columns, k_i, are computed. For full details of the implementation, see the discussion following, in §3.3.

Algorithm 3 (Single-stage greedy nonsymmetric partitioning algorithm).

1. Initialize $U_{\text{row}} = U_{\text{col}} = \Omega$
2. Initialize $C_{\text{row}} = C_{\text{col}} = \emptyset$
3. Initialize $F_{\text{row}} = F_{\text{col}} = \emptyset$
4. For all $i \in U_{\text{row}}$,
   (a) Compute $k_i = \underset{k \in U_{\text{col}}}{\text{argmax}} |a_{ik}|$
   (b) Compute $l_i = \sum_{j \in F_{\text{col}} \cup U_{\text{col}}} |a_{ij}|$
   (c) Compute $r_i = \sum_{j \in F_{\text{col}}} |a_{ij}|$
   (d) If $a_{ik_i} = 0$, then $C_{\text{row}} = C_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$.
   (e) If $\frac{|a_{ik_i}|}{l_i} \geq \theta$, then make (i, k_i) a diagonal element of A_{ff}:
      - $F_{\text{row}} = F_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$
      - $F_{\text{col}} = F_{\text{col}} \cup \{k_i\}$, $U_{\text{col}} = U_{\text{col}} \setminus \{k_i\}$
   (f) If $\frac{|a_{ik_i}|}{r_i} < \theta$, then $C_{\text{row}} = C_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$.
5. While $U_{\text{col}} \neq \emptyset$ and $U_{\text{row}} \neq \emptyset$,
   (a) For each $j \in U_{\text{col}}$, compute $w_j = \sum_{i \in U_{\text{row}}} \frac{|a_{ij}|}{|a_{ik_i}|}$

(b) Let $j^\star = \underset{j \in U_{\text{col}}}{\text{argmax}}\{w_j\}$

(c) Make $j^\star$ a C-column: $C_{\text{col}} = C_{\text{col}} \cup \{j^\star\}$, $U_{\text{col}} = U_{\text{col}} \setminus \{j^\star\}$

(d) For $i \in U_{\text{row}} \cap \{i : a_{ij^\star} \neq 0\}$,

i. Update $l_i = l_i - |a_{ij^\star}|$

ii. If $a_{ik_i} = 0$, then $C_{\text{row}} = C_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$.

iii. If $\frac{|a_{ik_i}|}{l_i} \geq \theta$, then

- $F_{\text{row}} = F_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$
- $F_{\text{col}} = F_{\text{col}} \cup \{k_i\}$, $U_{\text{col}} = U_{\text{col}} \setminus \{k_i\}$

6. $C_{\text{row}} = C_{\text{row}} \cup U_{\text{row}}$

7. $C_{\text{col}} = C_{\text{col}} \cup U_{\text{col}}$

Steps 1, 2, and 3 in Algorithm 3 simply initialize the sets, U, F, and C, for both columns and rows. Step 4 is an initial pass over the rows, which finds the largest entry in each row and computes the total and fine-grid row sums of each row. If the row is zero over U_{col}, it is immediately discarded into C_{row}. If the row is already θ-dominated by its largest entry, then that entry is permuted to the diagonal and its row and column are moved into the F sets. If no θ-dominance by a column in U_{col} is possible for the row, it is also immediately discarded into C_{row}.

The main elimination loop occurs in Step 5. For each undecided column, j, a measure, w_j, of column j's fitness as an F-column is computed. The larger the measure, the more large entries there are in column j, relative to the largest in the eligible part of each row. Thus, we select the column, $j^\star$, with the largest measure to be eliminated from U_{col} (and added to C_{col}) in Step 5b. For rows, i, such that $j^\star \in \text{Adj}(i)$, the removal of column $j^\star$ increases the dominance of a_{ik_i} over the portion of row i that is in $F_{\text{col}} \cup U_{\text{col}}$; thus, we check each adjacent row to see if it is now sufficiently dominated to be included in F.

Finally, note that the termination condition for the main loop (in Step 5) allows the algorithm to stop without a full partitioning of both rows and columns. This is reasonable, as Step 5a could be trivial in two ways: either U_{col} is empty, in which case there are no weights to be calculated, or U_{row} is empty, in which case all the weights are zero. This is possible because columns and rows are moved individually into their respective C blocks, so that while $|F_{\text{col}}| = |F_{\text{row}}|$ must always hold, there is no such constraint on the growth of the C blocks. Thus, iteration of the main loop stops whenever either the rows or the columns are completely partitioned into C and F. Steps 6 and 7 ensure that whatever rows or columns are left are placed into C, so that both the A_{ff} and A_{cc} blocks are square.

The column measures computed in Step 5a are chosen to reflect how much of a roadblock column j poses to finding a diagonally dominant A_{ff} block. If column j has many entries that are large relative to the largest in their row, then it does, indeed, make dominance hard to achieve. One possible downside to this particular measure, however, is the inability to distinguish between good potential pivots (entries in column j that are the largest in their rows) and entries that effectively block dominance (entries that are very close to, but slightly smaller than, the largest in their rows). It may be possible to neglect entries identified as potential pivots in this algorithm, leading to a more effective measure, but we have not experimented with this option. Another option apparent from the discussion above is that this choice of measure is simply one of many possible averages of the changes in the $\hat{\theta}_i$. Other measures (including other ℓ_p norms, or other averages, such as the harmonic average) are also possible, but have not been experimented with.

3.3 Implementation details

While Algorithm 3 describes the important features of the partitioning algorithm, it is intended only to convey the general outline of the scheme. Here, we provide full details of the scheme, as implemented, and as tested in Section 5. In particular, we consider the choices made in order to achieve low computational complexity, as well as the details of updating the row-wise quantities k_i, l_i, and r_i. By tracking when these updates are made, the first opportunities to move a row from U_{row} to F_{row} or C_{row} are more easily identified when the relevant quantities are updated. This detailed implementation described here is presented as Algorithm 4 in the Appendix.

For ease of computation, two preprocessing stages are added. First, the transpose of A is computed (Step 4 of Algorithm 4). While not explicitly necessary, this is convenient because of our use of a compressed sparse row (CSR) storage scheme for the matrix A (and A^T). Thus, for loops such as in Step 5d of Algorithm 3, we have easy access to the set, $\{i : a_{ij^\star} \neq 0\}$, as the adjacency list of row $j^\star$ in A^T. For this reason, we use the notation, $\text{Adj}^T(j) = \{i : a_{ij} \neq 0\}$ in Algorithm 4. Additionally, the adjacency lists for each row, i, in A are sorted in decreasing order by $|a_{ij}|$ (Step 5a). This is done for convenience in updating k_i; instead of needing to search over all entries in $\text{Adj}(i)$ for the next smallest entry that is eligible to dominate row i when the previous k_i is removed from U_{col}, the sorted adjacency lists allow us to simply scan the list from the entry for the old k_i onwards, until we find the next column in $\text{Adj}(i) \cap U_{\text{col}}$. In this manner, the adjacency list for row i is scanned from start to finish at most once over all of the updates to k_i before row i is removed from U_{row}.

The extra operations needed to update k_i, l_i, and r_i first arise in Step 4e of Algorithm 3 (Step 5f of Algorithm 4). Now, instead of just removing row i and column k_i to their respective F sets, the unsorted rows, m, in $\text{Adj}^T(k_i)$ are also examined. If column k_i dominated row m, then a new dominating column for row m is found from those still in U_{col}. If no such column exists, then row m is easily discarded to C_{row}, as there is no pivot that can pair with it to give a diagonally dominant row in A_{ff}. The A_{ff} row-norm for row m, r_m, is incremented by the value in the added column, and if row m is no longer sufficiently dominant relative to its F columns, it is also moved into C_{row}.

The initial computation of the column measures, w_j, for $j \in U_{\text{col}}$, is now broken into a separate loop (Step 6) and these measures are then updated in the main loop whenever a row is sorted. To manage these measures efficiently, and enable easy approximation of the column with maximal measure, a bucket-sort style data structure is used. Within the first loop (Step 5 of Algorithm 4), an upper bound on the measures is calculated as the maximum length of an adjacency list in U_{row}. The interval from 0 to this upper bound is then divided equally into a fixed number of buckets, where each bucket corresponds to a doubly linked list of elements representing columns from U_{col} whose measures lie in the appropriate subinterval. A separate list of pointers is maintained, indexed by column, pointing to the list element corresponding to each column. In this way, columns are easily removed from the data structure when they are moved into C_{col} or F_{col}, by accessing the element through the column-indexed pointer list and updating the bucket from which it came using the doubly linked list properties. Columns are also easily moved when their measure is updated, by removing the corresponding element from its previous list and adding it to the head of its new list. An approximately maximal column is also easily found, as the first item in the non-empty bucket with largest index.

The most significant changes to the algorithm occur the main loop, Step 5 (Step 7 and, in particular, Step 7c of Algorithm 4). Now, when column $j^\star$ is removed, we first check if it dominated any row $i \in U_{\text{row}} \cap \text{Adj}^T(j^\star)$. If so, we update k_i as before, and check if the new k_i is large enough to continue considering row i as a potential F row. If not, row i is moved into C_{row}. If, however, the intersection of $\text{Adj}(i)$ and U_{col} is non-empty, removing row i from U_{row} necessitates an update to the column measures for each $j \in \text{Adj}(i) \cap U_{\text{col}}$. Even if row i is not to be removed, these column weights change whenever the dominating column changes and, so, in Step 7(c)iiD of Algorithm 4, the measures for updated any columns are adjusted. The final major step, Step 7(c)iii, makes row i and column k_i a pivot in A_{ff} if row i is sufficiently dominated by column k_i. Here, we again update the measure of any column in U_{col} that is affected by the removal of row i from U_{row}. Additionally, for any row that is dominated by column k_i, a new dominating column is found, the intersecting column weights adjusted, and the row tested for suitability as a potential F row.

4 Ordering the A_{ff} block

While much effort has been focused on the partitioning of rows and columns within the ARMS algorithm, little consideration has been given to the ordering of the rows within the partitions. While ordering of the coarse-scale block has little practical effect, reordering of the A_{ff} block could result in a significant change in the sparsity of the resulting ILUT factors, as is the case in sparse direct methods (see, for example, [16]). Here, we extend the initial study of [28], where it was seen that the use of standard reordering techniques from sparse direct solvers may also result in improvements to the overall efficiency of the ARMS algorithm.

The efficiency of the ARMS process rests on the accurate approximation of the action of the inverse of

A_{ff} by its ILUT factors. While diagonal dominance of A_{ff} is an important consideration in being able to find sparse, accurate ILUT factors, it is not the only requirement. Indeed, even if the graph of A_{ff} allows exact LU factors to be defined with no fill beyond its nonzeros, this can only occur if A_{ff} is appropriately ordered. As in sparse direct methods, then, the sparsity of the ILUT factors depends on the ordering of the A_{ff} block, and reordering this block may be an effective remedy if the ARMS algorithm performs well but has a high complexity.

In Section 5, experiments with several standard reordering techniques to reorder the A_{ff} block within ARMS are given. As a baseline for comparison, we consider the order in which the F rows are selected by whatever selection procedure is used, as has typically been used in ARMS. There are two possible ways in which to evaluate the success of these reorderings. If the same ILUT tolerances are used for both the ordered and unordered systems, we can compare the combination of the ARMS fill factor (or preconditioner complexity, defined as the total number of nonzeros stored in the ARMS preconditioner on all levels divided by the number of nonzero entries in A) and iteration counts for the resulting solvers. Of particular interest, of course, is the total time to solution, which is, in effect, the product of these two values when computer architecture considerations are neglected. Alternately, the tolerances used to form the ILUT factors could be adjusted so that the ARMS preconditioner complexity using reordering (approximately) matches that of the preconditioner without reordering. In this situation, the relative performance of the two preconditioners can easily be compared based solely on iteration counts, as each iteration has roughly the same cost for both preconditioners. Because of the number of problems and variety of approaches considered, only the first sort of comparison is used here.

Several common reordering techniques are considered. From the METIS toolkit [19], we consider the Nodal Nested Dissection and Multiple Minimum Degree orderings. From the SPARSPAK package [16], we consider both Nested and One-way Dissections, along with Reverse Cuthill-McKee and Quotient Minimum Degree reorderings. Finally, we consider the Approximate Minimum Fill technique of [1].

5 Numerical Results

We consider several sets of test problems for the proposed partitioning scheme, building on problems tested in previous papers. For each set of tests, we compare the performance of ARMS preconditioners based on the new nonsymmetric partitioning scheme to other effective preconditioners, including ARMS with other permutation choices. All tests were run on a dual-processor 3.0GHz Intel Xeon workstation with 2GB of RAM.

5.1 Symmetric PDE-based Problems

To compare the nonsymmetric partitioning schemes with their symmetric counterpart (as described in §2.3), we first consider the test problems from [21]. In [21], the matrices associated with bilinear finite-element discretizations of the second-order elliptic PDE, $-\nabla \cdot K(x,y)\nabla p(x,y)$, on regular rectangular meshes of $[0,1]^2$, with several choices of $K(x,y)$ were considered. The resulting matrices are sparse, symmetric, and positive definite.

We consider four choices for the diffusion coefficient, $K(x,y)$. Constant $K(x,y) = 1$, the Poisson Equation, is a standard test problem for sparse matrix solvers, particularly for multigrid and its algebraic variants. A smoothly varying, but non-constant, isotropic coefficient, $K(x,y) = 10^{-8} + 10(x^2 + y^2)$ is also considered. A more difficult problem, where $K(x,y) = 10^{-8}$ on 20% of the elements of the grid, chosen randomly, and $K(x,y) = 1$ on the remainder, results in degradation of performance of standard geometric multigrid approaches, but is typically solved efficiently by algebraic multigrid and multilevel approaches. Finally, we consider the case of a constant, but anisotropic, diffusion coefficient, $K(x,y) = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0.01 \end{smallmatrix}\right]$, which causes difficulty for many algebraic solvers and preconditioners, including the two-level ARMS preconditioner considered in [21].

Results for the symmetric ARMS preconditioner, based on the symmetric greedy coarsening strategy of [21] are shown in Table 1 (Table 7 of [21]). For each choice of the coefficients, $K(x,y)$, we consider several levels of uniform mesh refinement. For these results, θ is chosen to be 0.55, a symmetrized ILUT algorithm is used where the L factors on all grids but the coarsest are computed with drop tolerance α, as reported in Table 1, and a maximum of twice the average number of nonzeros-per-row of A_{ff} are allowed in each row

Coefficient	Grid	α	# levels	c_B	t_{setup}	t_{solve}	# iters.
$K(x,y) = 1$	128×128	0.01	2	2.59	0.3	0.3	28
	256×256	0.01	2	2.65	1.5	2.5	44
	512×512	0.01	2	2.68	12.7	24.5	82
	1024×1024	0.1	2	1.03	159.1	34.2*	46*
$K(x,y) = 10^{-8} + 10(x^2 + y^2)$	128×128	0.01	2	2.60	0.3	0.4	31
	256×256	0.01	2	2.65	1.5	3.4	56
	512×512	0.01	2	2.68	12.7	31.7	97
	1024×1024	0.1	2	1.03	159.6	40.6*	52*
random $K(x,y)$	128×128	0.01	3	1.40	0.2	0.4	32
	256×256	0.01	3	1.41	0.7	2.5	45
	512×512	0.01	3	1.42	3.1	25.1	83
	1024×1024	0.01	3	1.42	13.5	12.4*	17*
anisotropic $K(x,y)$	128×128	0.01	5	1.61	0.2	0.3	26
	256×256	0.01	5	1.62	0.8	2.3	42
	512×512	0.01	5	1.63	3.3	17.3	65
	1024×1024	0.01	5	1.63	14.9	6.9*	10*

Table 1: Performance of ARMS with symmetric partitioning and symmetrized ILUT on test matrices from discretizations of $-\nabla \cdot K(x,y)\nabla p(x,y)$ for the given $K(x,y)$ and uniform grids on $[0,1]^2$. Results marked with a * indicate that the residual reduction criterion was a relative factor of 10^4 instead of 10^6.

of its L factor. Coarsening continued until either the coarsest-grid operator is θ-dominated by its diagonal or had fewer than 10 degrees of freedom. The coarsest-grid operator is then factored using ILUTP, with a drop tolerance of 10^{-5} and a maximum of twenty times the average number of non-zeros per row of this coarsest Schur complement allowed per row of its L factor. For these examples, a more efficient solver results from not rescaling the A_{ff} matrices prior to computing their ILUT factors, as discussed in [27]. In Table 1, we report the number of levels used by the symmetric ARMS preconditioner, the complexity, c_B, of the preconditioner (defined as the total number of nonzeros stored on all levels of the ARMS preconditioner divided by the number of non-zeros in the original fine-scale operator, A), the setup and solve times, and the number of iterations for preconditioned GMRES to reduce the residual by a relative factor of 10^6 (or, for the problems marked by a *, 10^4).

Table 2 shows results for the new non-symmetric partitioning based ARMS preconditioners on these problems. For the first three problems, the same parameters are used for the nonsymmetric partitioning as are used for the symmetric partitioning. For the anisotropic $K(x,y)$, using the same parameters as the symmetric partitioning resulted in very slow convergence of ARMS with non-symmetric partitioning. This is likely related to the fact that the average size of the nonzero offdiagonal entries for the discrete anisotropic operator is much closer to the size of its diagonal than for the isotropic problems, and only a very particular choice of coarse degrees of freedom leads to the semicoarsened grids that are known to be most effective for this problem. So, in Table 2, we use $\theta = 0.5$ instead of $\theta = 0.55$ for the anisotropic problem, which results in much better performance. Table 2 also includes two extra data points, detailing the performance of the preconditioners for the two cases from Table 1 where memory limitations required the choice of $\alpha = 0.1$. For these problems, we include results for both $\alpha = 0.01$ and $\alpha = 0.1$.

For the first two problems (with smooth isotropic coefficients), using ARMS based on nonsymmetric partitioning is slightly more efficient than the symmetric partitioning scheme. This can be seen, in particular, on the larger meshes, where the solve times for the 512×512 grid examples are slightly better than those of the symmetric approach, but the setup times scale much better. As a result, the overall performance of the nonsymmetric partitioning based approach with $\alpha = 0.01$ is significantly better than the performance of either partitioning scheme with $\alpha = 0.1$. For these two problems with $\alpha = 0.01$, the preconditioner complexity is better across all grids using the nonsymmetric partitioning scheme. For the second two problems, however, the performance of ARMS using the symmetric partitioning scheme is slightly better than that of ARMS using the nonsymmetric partitioning approach. In particular, the lower operator complexities for these two problems lead to notably faster solve times even though the iteration counts are comparable (in fact, the

Coefficient	Grid	α	# levels	c_B	t_{setup}	t_{solve}	# iters.
$K(x,y) = 1$	128×128	0.01	3	2.08	0.3	0.3	24
	256×256	0.01	3	2.17	1.2	2.4	38
	512×512	0.01	3	2.22	5.3	20.3	67
	1024×1024	0.01	3	2.24	25.2	11.9*	14*
	1024×1024	0.1	2	2.31	264.9	22.0*	28*
$K(x,y) = 10^{-8} + 10(x^2 + y^2)$	128×128	0.01	3	1.96	0.3	0.4	28
	256×256	0.01	3	1.96	1.1	3.5	50
	512×512	0.01	3	2.12	5.0	27.9	83
	1024×1024	0.01	3	1.99	26.1	15.8*	18*
	1024×1024	0.1	2	1.76	81.8	25.5*	32*
random $K(x,y)$	128×128	0.01	3	1.72	0.3	0.4	28
	256×256	0.01	4	1.52	1.2	2.8	41
	512×512	0.01	4	1.52	5.3	26.1	72
	1024×1024	0.01	4	1.53	24.1	15.7*	14*
anisotropic $K(x,y)$	128×128	0.01	7	2.19	0.5	0.4	28
	256×256	0.01	6	2.24	2.0	3.6	47
	512×512	0.01	7	2.23	8.2	28.2	76
	1024×1024	0.01	6	2.28	34.6	11.3*	11*

Table 2: Performance of ARMS-preconditioned GMRES based on the nonsymmetric partitioning strategy of §3.2 on test matrices from discretizations of $-\nabla \cdot K(x,y)\nabla p(x,y)$ for given $K(x,y)$ and uniform grids on $[0,1]^2$. Results marked with a * indicate that the residual reduction criterion was a relative factor of 10^4 instead of 10^6.

nonsymmetric partitioning algorithm results in fewer iterations for all grid sizes for the randomly selected coefficient problem).

As an additional comparison, the two-stage nonsymmetric permutation algorithm of [27], as described in Section 3.1 is used. For this method, the parameters are chosen as in [27, §6.1]; namely, a dominance threshold of 0.1 (now relative to the maximum dominance ratio), and drop tolerances of 0.001 in the computation of the ILUT factors, L and U, of A_{ff}, 0.01 in the computation of $L^{-1}A_{fc}$ and $A_{cf}U^{-1}$, 0.001 in the Schur-complement computation, and 0.01 on the coarsest level are used. Fill-in is restricted to ten times the average number of non-zeros per row of A_{ff} in all thresholded computations on the fine scales, and five times the average number of nonzeros per row of the Schur complement on the coarsest scale. Results for the same four model problems and grid sizes are shown in Table 3.

The performance of the two-stage algorithm, shown in Table 3, is interesting because of its contrast to the other results. For each problem, the two-stage algorithm creates hierarchies with more levels, yet results in preconditioner complexities that are, in all cases, bigger than those of either of the other two methods. The setup times clearly do not scale linearly with the increases in problem sizes, yet the iteration counts are similar (and, in some cases, better) than those of the other approaches. Because of the large complexity factors, however, iteration times with this approach are generally larger than those of the others.

While results for preconditioners based on the partitioning scheme of §3.2 are encouraging, timings for the solve stage using these preconditioners may not show any improvement over those of the preconditioners based on the symmetric approach. In order to improve iteration times, we now consider possible reorderings of the A_{ff} block aimed at improving the preconditioner complexities and, thus, cost per iteration, without losing accuracy. In particular, we look to find if using standard reordering techniques from sparse direct methods within ARMS reduces complexity, iteration time, and/or total time to solution. Because of the many possible combinations of partitionings and reorderings, we make use of performance profiles, as in [13], instead of tables to present these results in a more compact manner.

Coefficient	Grid	# levels	c_B	t_{setup}	t_{solve}	# iters.
$K(x,y) = 1$	128×128	6	3.26	0.7	0.4	25
	256×256	7	3.37	9.5	3.8	46
	512×512	7	3.44	155.7	29.9	76
	1024×1024	7	3.48	2546.7	16.3*	14*
$K(x,y) = 10^{-8} + 10(x^2 + y^2)$	128×128	5	2.85	0.5	0.4	25
	256×256	6	2.92	5.6	3.6	44
	512×512	6	2.94	80.2	31.9	76
	1024×1024	7	2.95	1279.1	28.4*	20*
random $K(x,y)$	128×128	7	2.44	0.4	0.4	26
	256×256	7	2.48	4.2	3.3	42
	512×512	7	2.50	55.9	28.5	69
	1024×1024	7	2.51	886.7	22.5*	15*
anisotropic $K(x,y)$	128×128	8	2.84	0.8	0.4	24
	256×256	8	2.82	10.4	3.2	41
	512×512	8	2.82	172.6	24.5	66
	1024×1024	8	2.80	2836.2	8.6*	8*

Table 3: Performance of ARMS-preconditioned GMRES based on the two-stage nonsymmetric partitioning approach of [27] on test matrices from discretizations of $-\nabla \cdot K(x,y)\nabla p(x,y)$ for given $K(x,y)$ and uniform grids on $[0,1]^2$. Results marked with a * indicate that the residual reduction criterion was a relative factor of 10^4 instead of 10^6.

5.1.1 Performance Profiles

Given a measurable solver performance characteristic, such as total time to solution, number of iterations, or preconditioner complexity, performance profiles are a useful tool for comparing different solvers in terms of their measured characteristics for a given set of problems. Let S be the set of solvers, and P be the set of problems. For a solver $i \in S$ and a problem $j \in P$, take s_{ij} to be the performance characteristic measured for solver i on problem j (e.g., the total time taken by solver i to reduce the residual of problem j by a relative factor of 10^6). For a particular problem, j, the best observed performance, in terms of this chosen characteristic, is

$$\hat{s}_j = \min_{i \in S}\{s_{ij}\}.$$

For each solver, i, we can then define

$$p_i(\alpha) = \frac{|\{j : s_{ij} \leq \alpha \hat{s}_j\}|}{|P|},$$

as the fraction of the problems in P for which the measured characteristic of solver i is within a relative factor of α of the optimal. If, for any reason, solver i fails on problem j, we take $s_{ij} = \infty$. Thus, $p_i(1)$ is the fraction of problems for which solver i is the best, and $\lim_{\alpha \to \infty} p_i(\alpha)$ is the fraction of problems for which solver i succeeded. A very good solver (relative to the chosen characteristic) is then one for which $p_i(\alpha)$ is largest (closest to one) for small values of α.

Performance profiles for the preconditioners based on the three partitioning algorithms, whose results are given in Tables 1, 2, and 3, are shown in Figure 1. At left is the profile for total time to solution including setup time plus time required to reduce the residual by a relative factor of 10^6 (10^4 for 1024×1024 grid problems). While ARMS based on symmetric partitioning is the fastest overall in 10 of 16 cases, the new single-stage nonsymmetric partitioning algorithm is seen to be competitive, solving 8 of 16 problems in the fastest time (2 ties in overall time occur). The preconditioner based on this partitioning solvers all problems within a factor of just over 2 of the individual optimal times. The slow setup times of the two-stage nonsymmetric partitioning approach are reflected in the slow growth of its profile. Considering only iteration times (shown on the right of Figure 1), we see that the non-symmetric permutations are much

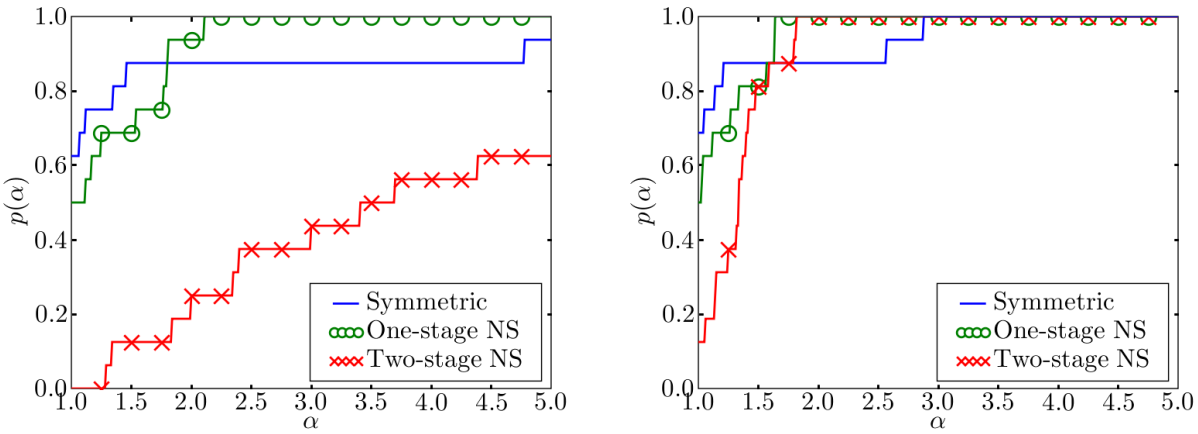

Figure 1: Performance profiles for total time to solution (left) and iteration time (right) for the ARMS preconditioners based on the three partitioning algorithms on the symmetric PDE-based problems. Solution is taken to mean a reduction of the residual by a relative factor of 10^6 on all grids but 1024×1024, where a reduction by a relative factor of 10^4 is used.

closer in performance to the symmetric-permutation based solvers, and that all solvers finish in less than three times the minimal time.

5.1.2 Effects of Reordering

We test the seven reordering schemes described in Section 4 in combination with the single-stage nonsymmetric partitioning approach of Section 3.2. Adding these reorderings to the ARMS setup procedure requires very little change to the overall ARMS algorithm. The partitioning stage in ARMS already defines reorderings of the rows and columns of A whether using the approach described in Algorithms 3 and 4 or not and, implicitly, defines the fine-scale block, A_{ff}. The adjacency graph of A_{ff} is then extracted from that of A, symmetrized (nodes i and j are deemed to be adjacent if either $i \in \text{Adj}(j)$ or $j \in \text{Adj}(i)$), and passed to the reordering algorithm. The ordering computed here is then compounded with that from ARMS over the fine-grid rows and columns, after which the ARMS setup and iterations proceed as usual. In particular, no changes are needed to the partitioning algorithm itself other than the call to compute the reordering, and no changes are needed within the solve phase. The effects of reordering the A_{ff} block on a given level, however, may be significant on coarser scales; reordering the A_{ff} block affects both the sparsity and entries in the ILUT factors, which are then used to compute the approximate Schur complement on the coarse scale. Changes in this operator directly affect decisions made regarding partitioning and reorderings on all coarser scales.

Performance profiles for the total time to solution (setup plus solve times) using reordering are shown in Figure 2. The plot at right shows the performance profile for $1.0 \leq \alpha \leq 3.5$, by which point all methods have $p(\alpha) = 1$. In general, the total times to solution using reordering are slightly larger than those without reordering, but almost all of the methods have total time within a factor of 1.5 of the fastest. The detail at left of Figure 2 shows the performance profiles for $1.0 \leq \alpha \leq 1.5$. Here, notice that while using the standard ARMS ordering is fastest for many problems, using the METIS MMD and Sparspak One-way Dissection and RCM reorderings are each the fastest for some problems. Overall, using the Sparspak One-way Dissection reordering results in times within 20 % of the fastest solver for all problems, while preconditioners based on the AMF, Sparspak RCM, and Sparspak ND reorderings also perform very well. The slowest solvers, in terms of total time to solution, result from using the METIS Nested Dissection and Sparspak QMD reorderings.

Considering only iteration time (the time required by the solution phase), the results are seen to be more mixed, as shown in Figure 3. In the plot at right, all of the resulting preconditioners yield solve times within a factor of 2 of the fastest method for each problem. In the detail at left, notice that the

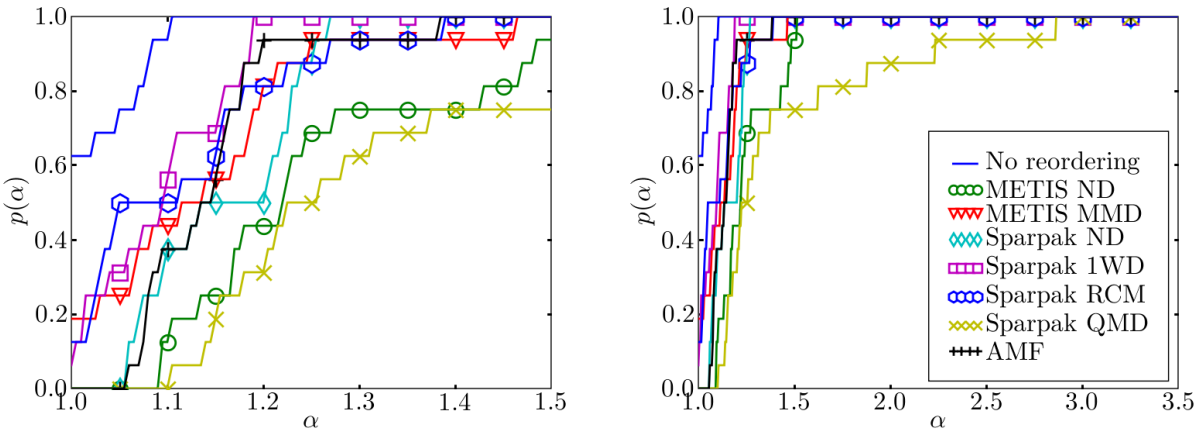

Figure 2: Performance profiles for total time to solution of the symmetric PDE-based problems, using various reordering techniques with the single-stage nonsymmetric partitioning algorithm. Solution is taken to mean a reduction of the residual by a relative factor of 10^6 on all grids but 1024×1024, where a reduction by a relative factor of 10^4 is used. At right are profiles for $\alpha \in [1.0, 3.5]$, the detail at left shows $\alpha \in [1.0, 1.5]$; each reordering algorithm is denoted by the same line color and marker in both figures.

performance curves are quite similar for all of the different methods. Using the standard ARMS ordering is fastest for only half of the problems, while the METIS MMD reordering results in the fastest solver for five problems. The METIS ND reordering is fastest for only two problems, but its performance profile increases most rapidly and, overall, it appears to perform slightly better than using the standard ARMS ordering. Of the remaining reordering approaches, the METIS MMD, AMF, Sparspak Nested and One-way Dissection orderings all perform well, within a factor of 50% of the fastest solver for all problems. Note that considering total time and iteration time separately distinguishes between solvers that have a slow overall time (as shown in Figure 2) because of the costs of the reordering stage, and those that are slow because of some deficiency in the resulting preconditioner. In particular, note that the METIS ND reordering is one of the slowest in terms of total time to solution, yet is among the fastest in iteration time. The added setup costs for this reordering obscure the actual performance gains in the solve phase. In contrast, the solve times for the Sparspak Reverse Cuthill-McKee reordering approach are among the slowest, yet the algorithm is in the middle of the pack for overall time to solution, due to the fast setup time of this approach. Similarly, the performance of the Sparspak One-way Dissection reordering does not distinguish itself in the solve phase, but because of its quick setup time, it is among the best reorderings in terms of overall time to solution.

A more detailed picture of the performance of the various solvers can be seen by considering, in addition to total time to solution and iteration time, the complexities of the resulting preconditioners. Figure 4 shows the performance profiles for preconditioner complexity, a measure of the memory requirements of the preconditioner itself. Overall, the effect of reordering is seen to be small (all preconditioner complexities are within 40% of the minima), but not trivially so. In the detail, we see that the Sparspak Reverse Cuthill-McKee and One-way Dissection algorithms are most effective in reducing the fill, but that all reorderings showed some benefit, in terms of lower complexity, over the original ordering by the ARMS process. In combination with the solve time profiles of Figure 3, a more thorough picture of the performance of the individual preconditioners is seen. For example, while the RCM ordering is most efficient in terms of complexities, the resulting preconditioner required a congruous increase in the number of iterations needed to converge. In contrast, we see that there is no significant benefit, in terms of complexity, in using the METIS Nested Dissection ordering, but the resulting preconditioner is somewhat more effective than that with the original ARMS reordering. These profiles further highlight the overall appeal of the Sparspak One-way Dissection ordering in this setting; not only does it have a low setup cost (close to that of using no reordering) and reasonable solver performance, it also produces preconditioners with relatively low complexities.

These results show that, while the ARMS preconditioners based on symmetric partitioning are the most

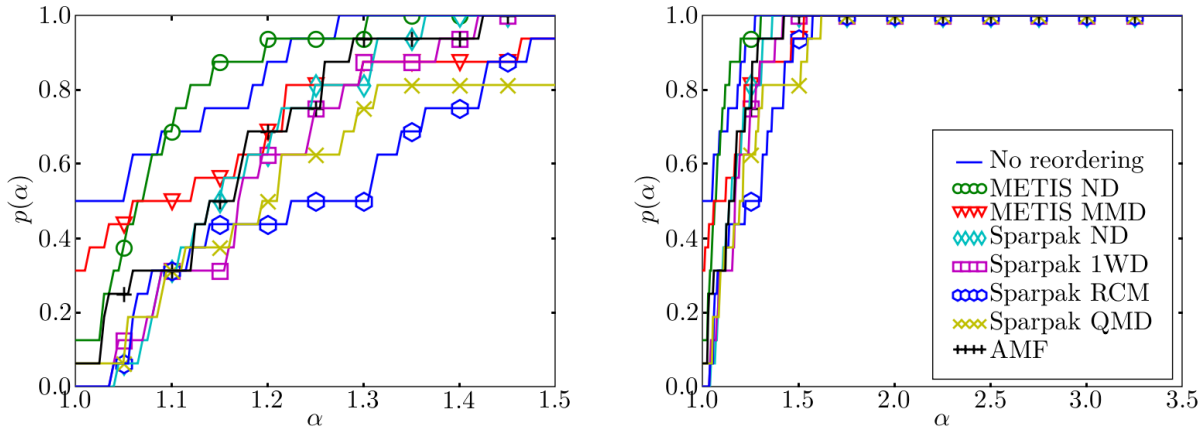

Figure 3: Performance profiles for solution time (excluding setup) for the symmetric PDE-based problems, using various reordering techniques with the single-stage nonsymmetric partitioning algorithm. Solution is taken to mean a reduction of the residual by a relative factor of 10^6 on all grids but 1024×1024, where a reduction by a relative factor of 10^4 is used. At right are profiles for $\alpha \in [1.0, 3.5]$, the detail at left shows $\alpha \in [1.0, 1.5]$; each reordering algorithm is denoted by the same line color and marker in both figures.

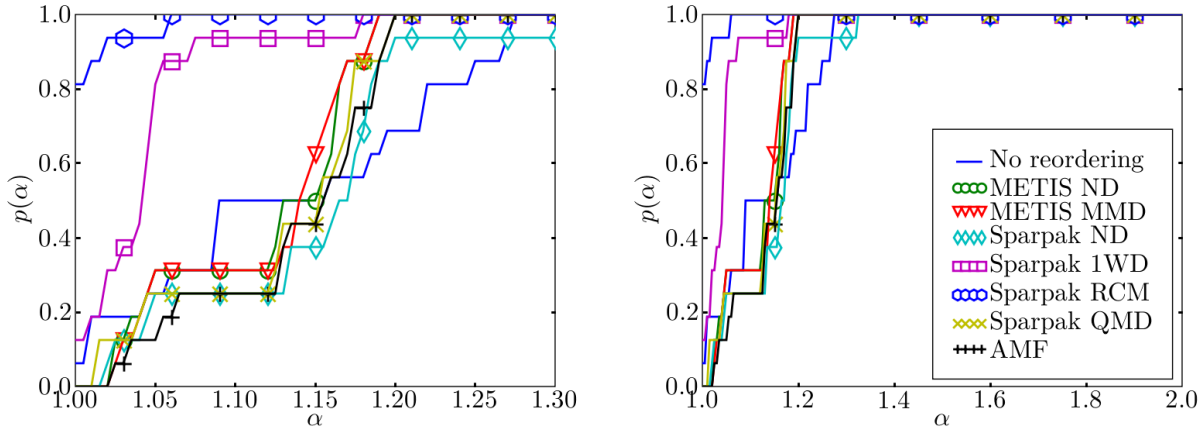

Figure 4: Performance profiles for preconditioner complexities for the symmetric PDE-based problems, using various reordering techniques with the single-stage nonsymmetric partitioning algorithm. At right are profiles for $\alpha \in [1.0, 2.0]$, the detail at left shows $\alpha \in [1.0, 1.3]$; each reordering algorithm is denoted by the same line color and marker in both figures.

Name	n	nnz
bodyy4	17546	121938
bodyy5	18589	129281
bodyy6	19366	134748
bcsstk18	11948	149090
t2dah_a	11445	176117
bcsstk25	15439	252241
OBSTACLE	40000	277600
JNLBRNG1	40000	279200
MINSURFO	40806	285230
bcsstk17	10974	428650
CVXBQP1	50000	449968

Name	n	nnz
t3dl_a	20360	509866
finan512	74752	596992
GRIDGENA	48962	610008
gyro_k	17361	1021159
bcsstk36	23052	1143140
msc23052	23052	1154814
msc10848	10848	1229778
cfd1	70656	1828364
vanbody	47072	2336898
nasasrb	54870	2677324
OILPAN	73752	3597188

Table 4: Names, dimensions, and numbers of nonzero entries for the general sparse symmetric matrix test set selected from the RAL collection [17, 18].

efficient for many of these symmetric PDE-based problems, the new nonsymmetric partitioning approach, possibly in combination with reorderings of the A_{ff} block, results in solvers that are, overall, nearly as effective. In particular, for problems on large grids, where setup times using both the symmetric and two-stage nonsymmetric partitioning approaches grew significantly, setup times using the new nonsymmetric partitioning approach showed much better growth (nearly proportional to that of the problem size). For these problems, the effects of reordering are not significant. For some problems, faster solution is possible (relative to either total solve time or iteration time alone), but the performance improvements are, in general, small. The preconditioners without this secondary reordering required at most 10% more than the minimal total time, and 30% more than the minimal iteration time for all problems. This should not necessarily be surprising, as the preconditioner complexities are not a significant problem for these examples. Fill within the ILUT factors is limited to twice that of A_{ff}, and the operator complexities without reordering are all bounded by 2.5. Reordering would be expected to play a greater role in problems where the ARMS preconditioners are effective in terms of iteration count, but whose preconditioner complexities are significantly larger.

5.2 General Sparse Symmetric Systems

The second set of test problems considered in [21] was drawn from the collection of general, sparse, symmetric and positive-definite matrices considered in [17, 18]. This set of matrices is the subset of the positive definite problems considered in [17] for which full data is available (discounting problems for which only a nonzero pattern is available) with fewer than 3 million nonzero entries (plus the problem, OILPAN, which has 3.5 million nonzeros, but for which a low preconditioner complexity made convergence possible). Here, we consider the same test set; matrix names, dimensions, and numbers of nonzero entries are listed in Table 4.

Solver parameters are chosen consistently with previous work. For the symmetric permutation scheme, $\theta = 0.55$, drop tolerances are 0.01 with the number of nonzeros per row of the symmetrized ILUT factor limited to five times the average number of nonzeros per row in A_{ff} on all but the coarsest scale, where the drop tolerance is 0.0001 and twenty times the average number of nonzeros per row of the coarsest Schur complement are allowed in each row of its symmetrized ILUT factor. A maximum of 10 levels (plus the coarsest) are allowed, and the A_{ff} blocks are not rescaled before the ILUT. For the single-stage nonsymmetric partitioning approach, θ is chosen as 0.51, and a maximum of 50 levels are allowed; this partitioning scheme tends to choose smaller coarse grids than the symmetric partitioning and, so, these parameters are chosen to allow a better comparison. Otherwise, parameters for the single-stage nonsymmetric partitioning approach are the same as for the symmetric partitioning scheme. For the two-stage nonsymmetric partitioning scheme, parameters are chosen to be the same as in the previous section and in [27, §6.1]. A maximum of 100 levels are allowed, with a dominance threshold of 0.1 (relative to the most dominant row). Diagonal rescaling before the ILUT of A_{ff} is used, and drop tolerances of 0.01 and 0.001 with maximum fill factors of 10 are used on fine scales. On the coarsest scale, the drop tolerance is 0.01, with five times the average number of nonzeros per row of the coarsest Schur complement matrix allowed in each row of its symmetrized ILT

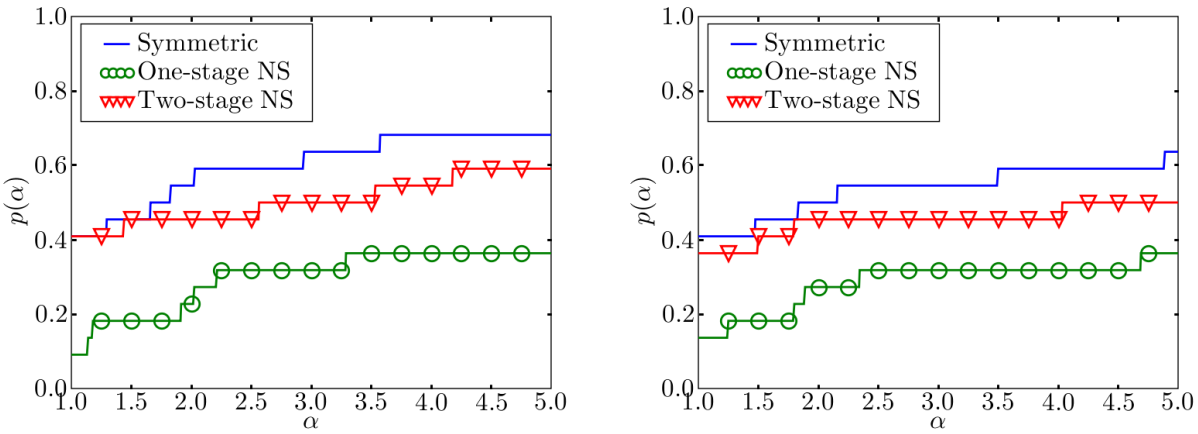

Figure 5: Performance profiles for total time to solution (left) and iteration time (right) for the preconditioners resulting from using the three partitioning algorithms on the general spare symmetric problems. Solution is taken to mean a reduction of the residual by a relative factor of 10^6, within 1000 iterations.

factor.

Performance profiles for total time to solution (at left) and iteration time (at right) for the preconditioners resulting from using the symmetric and one- and two-stage nonsymmetric partitioning algorithms are shown in Figure 5. For these profiles, a problem is deemed to have been solved if the ℓ_2 norm of the residual is reduced by a relative factor of 10^6 within 1000 iterations. For problems that failed to meet this criteria, the total and iteration times are taken to be infinite. For total time to solution, notice that each method is fastest for several problems and that, in general, the symmetric partitioning based approach is most successful. In particular, while the one-stage nonsymmetric partitioning approach is close to the fastest for about 20% of the total problems (noting that none of the solvers is successful on 4 of the problems), the symmetric and two-stage nonsymmetric partitioning schemes showed about equal overall performance. Considering the iteration time alone (at right of Figure 5), we see that the three schemes are much closer in performance. Again, each method is best for several problems, but the distinction between the single-stage nonsymmetric partitioning approach and the others is much less clear. This indicates a drawback of the more complicated setup stage of this approach; while the resulting preconditioners perform well, if a preconditioner with a less expensive setup scheme also performs well, then the advantage of a less expensive setup will, in general, outweigh the benefits of a better preconditioner.

Adding reordering to the nonsymmetric permutation algorithm cannot be expected to improve the total time to solution if the preconditioner complexity does not change significantly, but can yield a real improvement in iteration time. In fact, for these problems, each reordering technique shows some advantage over using the ordering chosen by ARMS in terms of solve time. Among the best performers are the Sparspak RCM, METIS ND, and AMF orderings. Figure 6 shows the performance profiles for the three partitioning schemes without reordering, and for the single-stage nonsymmetric partitioning scheme with these three reorderings. Notice that, for these problems, while the reorderings resulted in improved iteration times for the single-stage nonsymmetric partitioning algorithm, they did not significantly change the overall profile.

For this test set, the symmetric partitioning scheme of [21] is clearly the most effective overall. However, for some problems, using either the single-stage or two-stage nonsymmetric partitioning algorithm results in improved times (either total or iteration). While reordering is helpful for some problems, particularly in terms of iteration time, it does not appear to make a significant difference in the overall performance of the ARMS preconditioners based on the new nonsymmetric permutation approach.

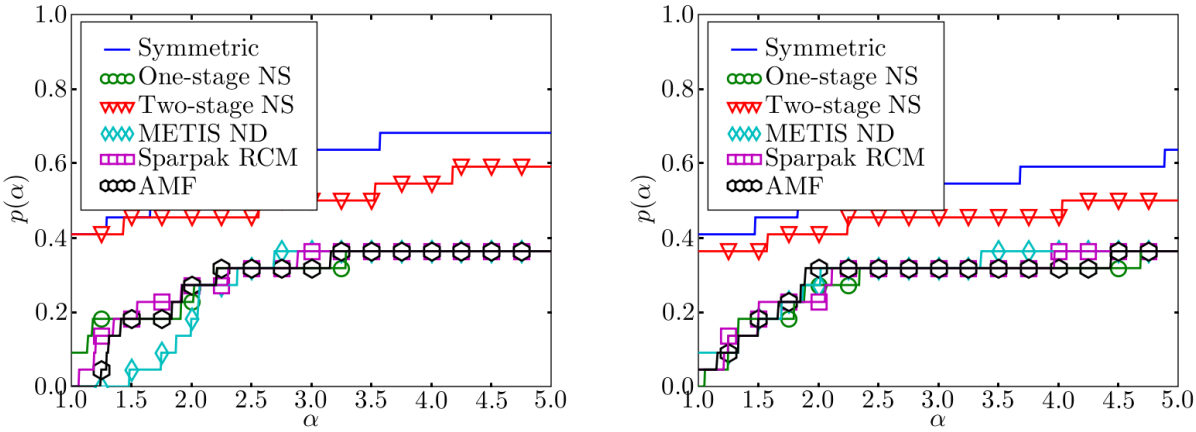

Figure 6: Performance profiles for total time to solution (left) and iteration time (right) for the resulting preconditioners using the three different partitioning algorithms, as well as for three different reorderings in combination with the single-stage nonsymmetric partitioning algorithm on the general sparse symmetric problems. Solution is taken to mean a reduction of the residual by a relative factor of 10^6, within 1000 iterations.

5.3 General Sparse Non-symmetric Systems

The first set of nonsymmetric test problems considered are from [27, §6.1]. This collection of 58 matrices are taken from the Harwell-Boeing collection; those selected are all square matrices from the RUA (real, unsymmetric, assembled) collection that have a dimension of 500 or higher. Matrix names, dimensions, and numbers of nonzero entries are listed in Table 5.

As a comparison for these problems, we consider the performance of a single-level ILUTP preconditioner within GMRES, in addition to the two non-symmetric partitioning-based ARMS preconditioners. For the ILUTP solver, a drop tolerance of 0.0001 is used, with the number of nonzeros per row of L and U^T limited to twenty times the average number of nonzeros per row of A. For the ARMS codes, we use the same parameters as in the previous section, with the exception that A_{ff} is now always diagonally rescaled before it is factored by ILUT.

Figure 7 shows the performance profiles for total time to solution (at left) and iteration time (at right) for the problems of Table 5 using these three preconditioning strategies. It should be noted that for many of the smaller problems in this set, the ARMS preconditioners complete the setup and iteration phases faster than reliable timings can be generated. To address this issue, recorded times that are less than 0.01 s (the minimum granularity of the `clock` command in C on the Intel Xeon system on which these tests were run) are set to 0.01 s. The sharp increase of the iteration time profile for the single-stage partitioning-based ARMS preconditioner at $\alpha = 2$ is due, in part, to these artificial timings, where the two-stage-based preconditioner receives a time of 0.01 s while the single-stage based preconditioner records a time of 0.02 s. This is also reflected in the sharp increase in the total time to solution profile for the single-stage partitioning solver near $\alpha = 1.5$. Quite simply, the recorded timings do not provide the resolution to say that the differences in the recorded timings for these problems are significant. This is also reflected in the significant number of ties recorded for the fastest algorithm; many problems are solved faster than could be accurately quantified.

A significant difference exists between the iteration times between the two ARMS-based preconditioners and the ILUTP results, although the profiles for total time to solution are quite similar. This is due to the significant time required for computing the ILUTP factorization of A. A reflection of this is seen in Figure 8, the profile of preconditioner complexity for the three different approaches. Notice here that the single-stage partitioning strategy results in the least fill for nearly 70% of the problems, compared to roughly 30% for the two-stage partitioning, and only a single problem for ILUTP. While the complexities for the two ARMS-based preconditioners are quite similar, the much higher preconditioner complexities for ILUTP are

Name	n	nnz
BP1000	822	4661
BP1200	822	4726
BP1400	822	4790
BP1600	822	4841
BP200	822	3802
BP400	822	4028
BP600	822	4172
BP800	822	4534
FS5411	541	4285
FS5412	541	4285
FS5413	541	4285
FS5414	541	4285
FS6801	680	2646
FS6802	680	2646
FS6803	680	2646
FS7601	760	5976
FS7602	760	5976
FS7603	760	5976
GEMAT11	4929	33185
GEMAT12	4929	33111

Name	n	nnz
GRE1107	1107	5664
GRE512	512	2192
JPWH991	991	6027
LNS3937	3937	25407
LNS511	511	2796
LNSP3937	3937	25407
LNSP511	511	2796
MAHINDAS	1258	7682
MCFE	765	24382
NNC1374	1374	8606
NNC666	666	4044
ORANI678	2529	90158
ORSIRR1	1030	6858
ORSIRR2	886	5970
ORSREG1	2205	14133
PDE9511	961	4681
PORES2	1224	9613
PORES3	532	3474

Name	n	nnz
PSMIGR1	3140	543162
PSMIGR2	3140	540022
PSMIGR3	3140	543162
SAYLR3	1000	3750
SAYLR4	3564	22316
SHERMAN1	1000	3750
SHERMAN2	1080	23094
SHERMAN3	5005	20033
SHERMAN4	1104	3786
SHERMAN5	3312	20793
SHL0	663	1687
SHL200	663	1726
SHL400	663	1712
STEAM2	600	13760
WATT1	1856	11360
WATT2	1856	11550
WEST0655	655	2854
WEST0989	989	3537
WEST1505	1505	5445
WEST2021	2021	7353

Table 5: Names, dimensions, and numbers of nonzero entries for the nonsymmetric Harwell-Boeing test set

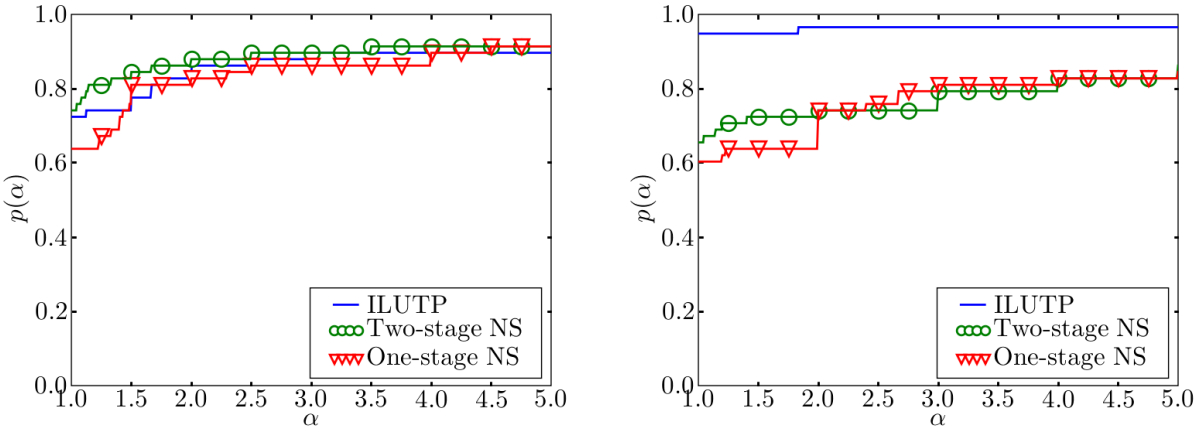

Figure 7: Performance profiles for total time to solution (left) and iteration time (right) for the three nonsymmetric preconditioning strategies on the Harwell-Boeing test problems. Solution is taken to mean a reduction of the residual by a relative factor of 10^6, within 1000 iterations.

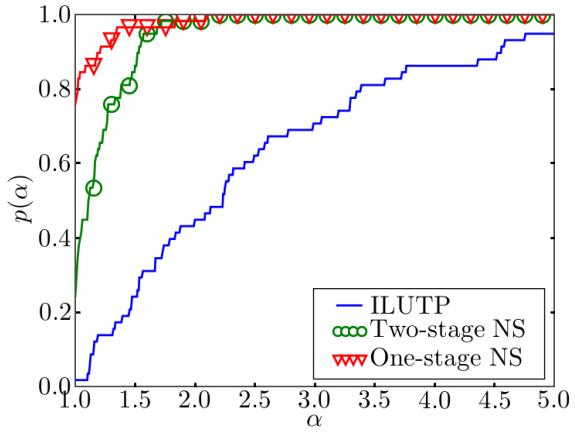

Figure 8: Performance profiles for preconditioner complexities for the three nonsymmetric preconditioning strategies on the Harwell-Boeing test problems.

clearly shown by its much lower, and slower to increase, profile. These larger complexities not only impact solver time, but also reflect a significant increase in the memory requirements of the ILUTP preconditioner; ILUTP required more than twice of the minimum memory on more than half of the problems in this set.

5.4 Circuit and Semiconductor Simulation Matrices

As a final test set, we consider matrices arising in semiconductor and circuit simulation, obtained from the University of Florida Sparse Matrix collection [12]. The matrices are taken from five sets within this collection: Bomhof [5], Hamm, Schenk_IBMSDS [32], Schenk_ISEA [32], and Wang [22]. The dimensions and numbers of nonzero entries for these problems are listed in Table 6; the matrices range in dimension from 2395 to 181343, and in number of nonzero entries from 17 thousand to 11 million.

For these matrices, as in the previous section, we compare the preconditioners produced by the ARMS methodology using both the new single-stage nonsymmetric partitioning approach and the two-stage nonsymmetric partitioning approach of [27], along with an ILUTP approach. For the single-stage partitioning, $\theta = 0.5$ is chosen, along with a drop tolerance of 0.01 and maximum fill per row in the ILUT factors of five times the average number of nonzeros per row of A_{ff} on all but the coarsest level, where a drop tolerance of 0.0001 and a fill factor of ten are used. For the two-stage partitioning approach, we use the parameters from [27, §6.5], with a relative diagonal dominance threshold of 0.1, drop tolerance of 0.01 and allowable fill factor of three on all but the coarsest grid, where a drop tolerance of 10^{-5} and fill factor of twenty is used. For both preconditioners, we allow rescaling of A_{ff} on all grids before the ILUT factors are computed. For the ILUTP preconditioner, a drop tolerance of 0.0001 and a maximum fill factor of twenty are used.

The performance profiles for these problems, shown in Figure 9, indicate a much bigger gap between the ARMS approaches and the ILUTP preconditioner than is seen for the smaller problems of Section 5.3. In terms of total time, the performance profiles for the two ARMS-based approaches are quite similar, with the single-stage approach being fastest for roughly 40% of the test problems, while the two-stage partitioning yields the fastest solver for approximately 55% of the test problems. The ILUTP preconditioner is notably slower in terms of total time to solution but, as before, shows a much better profile of iteration time.

Looking at these results in more detail than is shown in the performance profiles, we see that the difference between the two ARMS-based approaches is significant for some of the problems in this test set. On the matrix, circuit_3, from the Bomhof set, the preconditioner based on the single-stage approach failed to converge within 1000 iterations, while the two-stage preconditioner converged, but slowly, in 723 iterations. For the matrix, circuit_4, both approaches converged in 9 iteration (about 0.3 seconds), but the single-stage approach needed only 0.5 seconds for setup, compared to 1.2 seconds for the two-stage approach. For the matrix, 3D_28984_Tetra, from the Schenk_IBMSDS set, the two-stage preconditioner did not converge within

Name	n	nnz
circuit_1	2624	35823
circuit_2	4510	21199
circuit_3	12127	48137
circuit_4	80209	307604
add20	2395	17319
add32	4960	23884
bcircuit	68902	375558
hcircuit	105676	513072
memplus	17758	126150
scircuit	170998	958936

Name	n	nnz
barrier2-1	113076	3805068
barrier2-2	113076	3805068
barrier2-3	113076	3805068
barrier2-4	113076	3805068
barrier2-9	115625	3897557
barrier2-10	115625	3897557
barrier2-11	115625	3897557
barrier2-12	115625	3897557
igbt3	10938	234006
nmos3	18588	386594
ohne2	181343	11063545
para-4	153226	5326228
para-5	155924	5416358
para-6	155924	5416358
para-7	155924	5416358
para-8	155924	5416358
para-9	155924	5416358
para-10	155924	5416358

Name	n	nnz
2D_27628_bjtcai	27628	442898
2D_54019_highK	54019	996414
3D_28984_Tetra	28984	599170
3D_51448_3D	51448	1056610
ibm_matrix_2	51448	1056610
matrix_9	103430	2121550
matrix-new_3	125329	2678750
swang1	3169	20841
swang2	3169	20841
wang1	2903	19093
wang2	2903	19093
wang3	26064	177168
wang4	26068	177196

Table 6: Names, dimensions, and numbers of nonzero entries for the circuit and semiconductor device simulation matrices. In the left column are the Bomhof and Hamm collections. At center, the matrices from the Schenk_ISEI set. At right, the Schenk_IBMSDS and Wang sets.

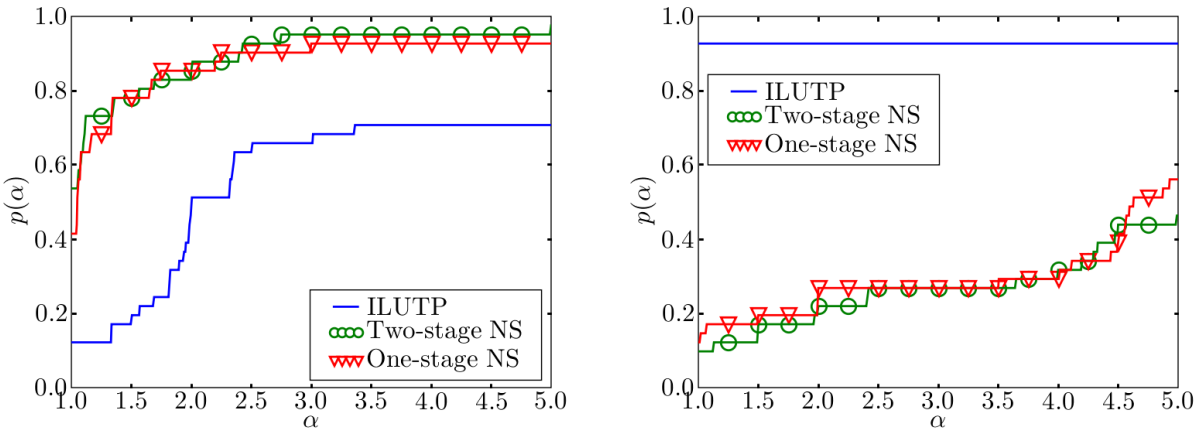

Figure 9: Performance profiles for total time to solution (left) and iteration time (right) for the three nonsymmetric preconditioning strategies with the circuit and semiconductor device simulation matrices. Solution is taken to mean a reduction of the residual by a relative factor of 10^6, within 1000 iterations.

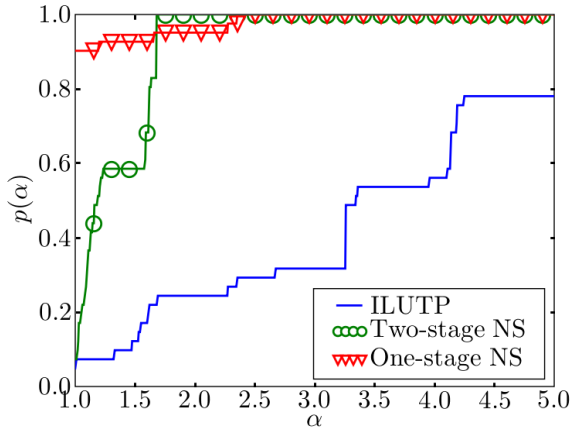

Figure 10: Performance profiles for preconditioner complexities for the three nonsymmetric preconditioning strategies on the circuit and semiconductor device simulation test problems.

1000 iterations, yet the single-stage preconditioner converged in only 1 iteration, requiring 0.5 seconds to set up a successful preconditioner, compared to 0.4 seconds for the unsuccessful approach. On the matrix, scircuit, from the Hamm set, the new partitioning approach yielded a preconditioner that converged in 32 iterations (and 3.8 seconds) compared to 60 iterations (and 9.3 seconds) for the two-stage approach, yet required only slightly more setup time, 1.4 seconds versus 1.2 seconds. For the wang3 and wang4 matrices, the single-stage partitioning approach yielded preconditioners that are faster both in setup and iteration, yielding a total speedup of over 25%. For many of the problems, however, the differences are less significant, with the single-stage preconditioner being approximately 10% faster on the barrier2 problems, and the two-stage preconditioner approximately 10% faster on the para problems, both from the Schenk_ISEI set.

Some of the speedup seen in iteration times for the single-stage partitioning preconditioners is due to improved preconditioner complexities, as seen in Figure 10. For both ARMS-based approaches, the preconditioner complexities are quite good, below one for many problems, but the new partitioning scheme leads, in general, to lower complexities. Once again, the memory requirements of the ILUTP preconditioners are much higher; for this test set, ILUTP required over three times the amount of memory of the optimal solver on over two-thirds of the problems. On 20% of the problems, it required more than five times the storage than the optimal multilevel approach. These increased memory requirements do not result in large iteration times, as the constructed preconditioners require only a few iterations to sufficiently reduce the residual, but are reflected in the large gap in setup times between the ILUTP and ARMS approaches shown in Figure 9. While overall time to solution is an important measure of these preconditioners, the increased memory requirements of the ILUTP approach should not be ignored, especially for large matrices such as these, where the feasibility of storing such large preconditioners must be considered.

The single-stage partitioning scheme results in low ARMS preconditioner complexities (below one for all but 10 of these problems, below one-half for 17 of them); this is likely responsible for the very slight performance improvements realized by any of the reorderings considered. For these problems, the SPARSPAK One-Way Dissection and Reverse Cuthill-McKee reorderings are most effective in terms of reducing total time to solution, and time-based performance profiles for these two reorderings are shown in Figure 11. These results again highlight the importance of a fast reordering algorithm; nearly all preconditioner complexities were within 10% of the optimal approach for each problem, resulting in only a small possible savings in iteration time. The most effective approaches were those with low overhead in terms of setup costs.

For these circuit and semiconductor device simulation matrices, preconditioners based on the new, single-stage nonsymmetric partitioning approach yield faster total time to solution for many problems and, generally, lower preconditioner complexities. Using a single set of parameters, fast solution is obtained for all but one of the problems in this set. Perhaps because of the efficiency already seen in the obtained preconditioners, reordering the A_{ff} block using standard techniques from sparse direct solvers produces only slightly

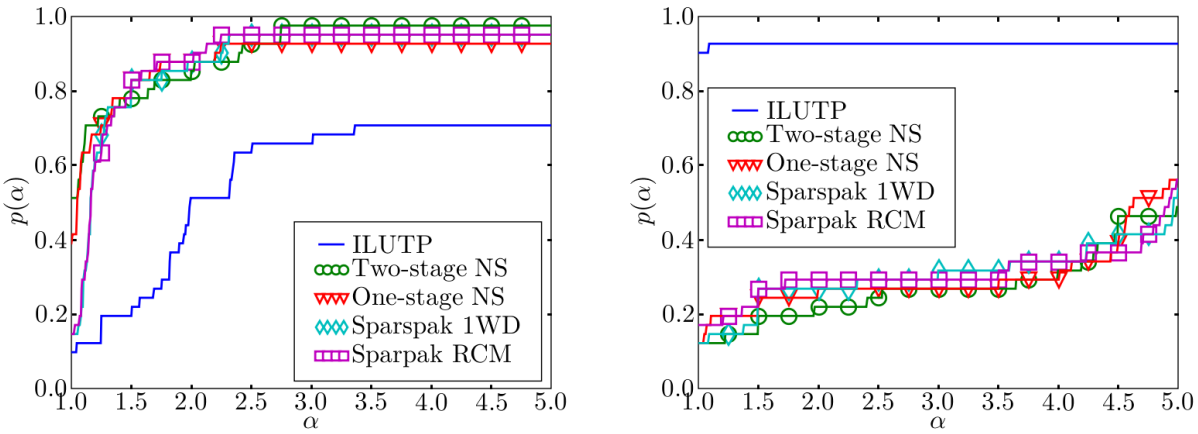

Figure 11: Performance profiles for total time to solution (left) and iteration time (right) for the ILUTP-preconditioned GMRES and preconditioners using the nonsymmetric partitioning schemes for ARMS, possibly with reordering, on the circuit and semiconductor device simulation problems. Solution is taken to mean a reduction of the residual by a relative factor of 10^6, within 1000 iterations.

faster solvers but, overall, does not show a significant benefit.

6 Conclusions

We present a new greedy algorithm for partitioning matrices within a multilevel preconditioner such as ARMS. This algorithm combines the attractive features of two previous works, in that it is based on a direct greedy approach to finding the largest diagonally dominant A_{ff} block (as in [21]), but allows for non-symmetric permutations, as are known to be appropriate for problems with significant non-symmetry in the operator [27]. We also consider the important question of ordering of the fine-scale block of the partitioned system to achieve a better approximation of this block by its ILUT factors while emphasizing sparsity.

The new partitioning approach generalizs the symmetric approach introduced previously, but does not rely on relative diagonal dominance measures as a previous nonsymmetric partitioning approach does. The setup algorithm is somewhat more complicated, resulting in typically longer setup times for the preconditioner construction, but generally lower operator complexities. In many cases, the improved performance of the preconditioner compensates for the added expense in forming it, but this is not always so. The similarity in performance between these two approaches suggests that the expected improvement from further adjustments of diagonal-dominance-based coarsening approaches may not be a fruitful area for further research.

Using reordering techniques from sparse direct solvers on the A_{ff} block before computing its ILUT factors does result in lower preconditioner complexities and may improve total time to solution (or iteration time), but did not lead to significant improvements in our tests. The already low preconditioner complexities observed within the ARMS framwork may preclude significant improvements due to these reorderings; we have not observed a case where the ARMS preconditioner has a large complexity, but converges in only a few iterations. In such a situation, reordering the fine-scale degrees of freedom may be an effective tool to reduce preconditioner complexity and, as a result, iteration time. Alternately, the use of truncated incomplete factorizations may already be a sufficient control on the complexity of the preconditioners, and little improvement may be possible by using a more sophisticated ordering. This would be consistent with the results of [14], where the naive row ordering was found to be as effective as any other ordering tried within an incomplete factorization setting. In either case, while the effects of reordering are clearly seen in reduced preconditioner complexities, the cost of finding these orderings is seen to generally outweigh any improvements in iteration times.

References

[1] P. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 886–905.

[2] O. Axelsson and P. S. Vassilevski, *Algebraic multilevel preconditioning methods, I*, Numer. Math., 56 (1989), pp. 157–177.

[3] ——, *Algebraic multilevel preconditioning methods. Part II*, SIAM J. Numer. Anal., 27 (1990), pp. 1569–1590.

[4] R. E. Bank and C. Wagner, *Multilevel ILU decomposition*, Numer. Math., 82 (1999), pp. 543–576.

[5] W. Bomhof and H. van der Vorst, *A parallel linear system solver for circuit simulation problems*, Numer. Linear Algebra Appl., 7 (2000), pp. 649–665.

[6] E. F. F. Botta and F. W. Wubs, *Matrix renumbering ILU: An effective algebraic multilevel ILU preconditioner for sparse matrices*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 1007–1026.

[7] A. Brandt, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and Its Applications, D. J. Evans, ed., Cambridge University Press, Cambridge, 1984.

[8] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive smoothed aggregation (αSA)*, SIAM J. Sci. Comp., 25 (2004), pp. 1896–1920.

[9] ——, *Adaptive algebraic multigrid*, SIAM J. Sci. Comp., 27 (2006), pp. 1261–1286.

[10] R. Bridson and W.-P. Tang, *A structural diagnosis of some IC orderings*, SIAM J. Sci. Comp., 22 (2000), pp. 1527–1532.

[11] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, SIAM Books, Philadelphia, 2000. Second edition.

[12] T. Davis, *University of Florida sparse matrix collection.* http://www.cise.ufl.edu/research/sparse/matrices, NA Digest, vol. 92, no. 42, October 16, 1994, NA Digest, vol. 96, no. 28, July 23, 1996, and NA Digest, vol. 97, no. 23, June 7, 1997.

[13] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., Ser. A, 91 (2002), pp. 201–213.

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

[15] R. D. Falgout and P. S. Vassilevski, *On generalizing the AMG framework*, SIAM J. Numer. Anal., 42 (2004), pp. 1669–1693. Also available as LLNL technical report UCRL-JC-150807.

[16] J. A. George and J. W.-H. Liu, *Computer solution of large sparse positive-definite systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[17] N. I. M. Gould and J. A. Scott, *Complete results for a numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations*, Numerical Analysis Internal Report 2003-2, Rutherford Appleton Laboratory, 2003.

[18] ——, *A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations*, ACM Transactions on Mathematical Software, 30 (2004), pp. 300–325.

[19] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.

[20] S. MacLachlan, T. Manteuffel, and S. McCormick, *Adaptive reduction-based AMG*, Numerical Linear Algebra with Applications, (2006). To Appear.

[21] S. MacLachlan and Y. Saad, *A greedy strategy for coarse-grid selection*, Tech. Rep. umsi-2006-17, Minnesota Supercomputing Institute, University of Minnesota, 2006. Submitted.

[22] J. J. H. Miller and S. Wang, *An exponentially fitted finite element method for a stationary convection-diffusion problem*, in Computational methods for boundary and interior layers in several dimensions, J. J. H. Miller, ed., Boole Press, Dublin, 1991, pp. 120–137.

[23] Y. Notay, *Algebraic multigrid and algebraic multilevel methods: a theoretical comparison*, Num. Lin. Alg. Appl., 12 (2005), pp. 419–451.

[24] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1987, pp. 73–130.

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

[26] ——, *ILUM: a multi-elimination ILU preconditioner for general sparse matrices*, SIAM J. Sci. Comp., 17 (1996), pp. 830–847.

[27] ——, *Multilevel ILU with reorderings for diagonal dominance*, SIAM J. Sci. Comput., 27 (2006), pp. 1032–1057.

[28] Y. Saad, A. Soulaimani, and R. Touihri, *Adapting algebraic recursive multilevel solvers (ARMS) for solving CFD problems*, Tech. Rep. umsi-2002-105, Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN, USA, 2002.

[29] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Num. Lin. Alg. App., 9 (2002), pp. 359–378.

[30] Y. Saad and J. Zhang, *BILUM: Block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems*, SIAM J. Sci. Comp., 20 (1999), pp. 2103–2121.

[31] ——, *BILUTM: A domain-based multi-level block ILUT preconditioner for general sparse matrices*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 279–299.

[32] O. Schenk, S. Roellin, and A. Gupta, *The effects of nonsymmetric matrix permutations and scalings in semiconductor device and circuit simulation*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 23 (2004), pp. 400–411.

Appendix: Detailed implementation of Algorithm 3

The following is a detailed description of the implementation of the single-stage greedy coarsening algorithm sketched in Algorithm 3. These details are discussed in Section 3.3.

Algorithm 4 (Detailed single-stage greedy nonsymmetric partitioning algorithm).

1. Initialize $U_{\text{row}} = U_{\text{col}} = \Omega$
2. Initialize $C_{\text{row}} = C_{\text{col}} = \emptyset$
3. Initialize $F_{\text{row}} = F_{\text{col}} = \emptyset$
4. Compute A^T
5. For all $i \in U_{\text{row}}$, ,
   (a) Sort Adj(i) by decreasing $|a_{ij}|$

(b) Compute $k_i = \underset{k \in U_{\text{col}}}{\text{argmax}} |a_{ik}|$

(c) Compute $l_i = \sum_{j \in F_{\text{col}} \cup U_{\text{col}}} |a_{ij}|$

(d) Compute $r_i = \sum_{j \in F_{\text{col}}} |a_{ij}|$

(e) If $a_{ik_i} = 0$, then $C_{\text{row}} = C_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$.

(f) If $\frac{|a_{ik_i}|}{l_i} \geq \theta$, then make (i, k_i) a diagonal element of A_{ff}:

i. Make i an F row: $F_{\text{row}} = F_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$

ii. Make k_i an F column: $F_{\text{col}} = F_{\text{col}} \cup \{k_i\}$, $U_{\text{col}} = U_{\text{col}} \setminus \{k_i\}$

iii. For $m \in U_{\text{row}} \cap \text{Adj}^T(k_i)$,

- If k_m has already been defined and $k_m = k_i$,
  - Compute new $k_m = \underset{k \in U_{\text{col}}}{\text{argmax}} |a_{mk}|$
  - If $a_{mk_m} = 0$, then $C_{\text{row}} = C_{\text{row}} \cup \{m\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{m\}$.
- Update $r_m = r_m + |a_{mk_i}|$
- If $\frac{|a_{mk_m}|}{r_m} < \theta$, then $C_{\text{row}} = C_{\text{row}} \cup \{m\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{m\}$.

(g) If $\frac{|a_{ik_i}|}{r_i} < \theta$, then $C_{\text{row}} = C_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$.

6. For $j \in U_{\text{col}}$, compute $w_j = \sum_{i \in U_{\text{row}}} \frac{|a_{ij}|}{|a_{ik_i}|}$

7. While $U_{\text{col}} \neq \emptyset$ and $U_{\text{row}} \neq \emptyset$,

(a) Let $j^\star \approx \underset{j \in U_{\text{col}}}{\text{argmax}} \{w_j\}$

(b) Remove $j^\star$ from U_{col}: $C_{\text{col}} = C_{\text{col}} \cup \{j^\star\}$, $U_{\text{col}} = U_{\text{col}} \setminus \{j^\star\}$

(c) For $i \in U_{\text{row}} \cap \text{Adj}^T(j^\star)$,

i. Update $l_i = l_i - |a_{ij^\star}|$

ii. If $k_i = j^\star$,

A. Compute new $k_i = \underset{k \in U_{\text{col}}}{\text{argmax}} |a_{ik}|$

B. If $a_{ik_i} = 0$, then make i a C row: $C_{\text{row}} = C_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$.

C. If $a_{ik_i} \neq 0$ and $\frac{|a_{ik_i}|}{r_i} < \theta$ then,

- For each $j \in U_{\text{col}} \cap \text{Adj}(i)$, update column weight, $w_j = w_j - \frac{|a_{ij}|}{|a_{ik_i^{\text{old}}}|}$
- Make i a C row: $C_{\text{row}} = C_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$

D. If $a_{ik_i} \neq 0$ and $\frac{|a_{ik_i}|}{r_i} \geq \theta$ then, for $j \in U_{\text{col}} \cap \text{Adj}(i)$, update column weight,

$$w_j = w_j - \frac{|a_{ij}|}{|a_{ik_i^{\text{old}}}|} + \frac{|a_{ij}|}{|a_{ik_i^{\text{new}}}|}.$$

iii. If $i \in U_{\text{row}}$ and $\frac{|a_{ik_i}|}{l_i} \geq \theta$, then make (i, k_i) a diagonal element of A_{ff}:

A. Make i an F row: $F_{\text{row}} = F_{\text{row}} \cup \{i\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{i\}$

B. Make k_i an F column: $F_{\text{col}} = F_{\text{col}} \cup \{k_i\}$, $U_{\text{col}} = U_{\text{col}} \setminus \{k_i\}$

C. For $j \in U_{\text{col}} \cap \text{Adj}(i)$, update column weight, $w_j = w_j - \frac{|a_{ij}|}{|a_{ik_i}|}$

D. For $m \in U_{\text{row}} \cap \text{Adj}^T(k_i)$

- If k_m has already been defined and $k_m = k_i$,

- Compute new $k_m = \underset{k \in U_{\text{col}}}{\text{argmax}} |a_{mk}|$
- If $a_{mk_m} = 0$, then make i a C row: $C_{\text{row}} = C_{\text{row}} \cup \{m\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{m\}$.
- If $a_{mk_m} \neq 0$, then for each $j \in U_{\text{col}} \cap \text{Adj}(m)$, update column weight,

$$w_j = w_j - \frac{|a_{ij}|}{|a_{ik_i^{\text{old}}}|} + \frac{|a_{ij}|}{|a_{ik_i^{\text{new}}}|}.$$

- Update $r_m = r_m + |a_{mk_i}|$
- If $m \in U_{\text{row}}$ and $\frac{|a_{mk_m}|}{r_m} < \theta$,
  - Make m a C row, $C_{\text{row}} = C_{\text{row}} \cup \{m\}$, $U_{\text{row}} = U_{\text{row}} \setminus \{m\}$.
  - For $j \in U_{\text{col}} \cap \text{Adj}(m)$, update column weight, $w_j = w_j - \frac{|a_{mj}|}{|a_{mk_m}|}$

8. $C_{\text{row}} = C_{\text{row}} \cup U_{\text{row}}$
9. $C_{\text{col}} = C_{\text{col}} \cup U_{\text{col}}$