ARMS: An Algebraic Recursive Multilevel Solver for general sparse linear systems *

Yousef Saad† Brian Suchomel†

June 16, 1999

Abstract

This paper presents a general preconditioning method based on a multilevel partial solution approach. The basic step in constructing the preconditioner is to separate the initial points into two subsets. The first subset which can be termed "coarse" is obtained by using "block" independent sets, or "aggregates". Two aggregates have no coupling between them, but nodes in the same aggregate may be coupled. The nodes not in the coarse set are part of what might be called the "Fringe" set. The idea of the methods is to form the Schur complement related to the fringe set. This leads to a natural block LU factorization which can be used as a preconditioner for the system. This system is then solver recursively using as preconditioner the factorization that could be obtained from the next level. Unlike other multilevel preconditioners available, iterations between levels are allowed. One interesting aspect of the method is that it provides a common framework for many other techniques. Numerical experiments indicate that the method can be fairly robust.

Key words: Incomplete LU factorization, ILUT, multi-level ILU preconditioner, Krylov subspace methods, multi-elimination, Recursive solution, multigrid

AMS subject classifications: 65F10, 65N06.

1 Introduction

Recent attention of researchers in iterative methods has focussed on preconditioning techniques while research on accelerators is beginning to subside. Among preconditioning methods currently investigated by various groups, a promising class of ILU-type techniques has emerged that possesses many of the attributes of multilevel solvers. Multilevel techniques are often used for problems with regular meshes and perform quite well in those cases. Their main attraction is their excellent scalability with respect to mesh size. Their scope however is limited. A number of methods developed in the last decade have aspired to combine the good intrinsic properties of multigrid techniques and the generality of preconditioned Krylov subspace methods. Among these we cite [2, 4, 7, 8, 17, 19, 20].

Multigrid methods can be extremely efficient when they work. However, their implementation requires multi-level grids and specialized tuning is often needed. The Algebraic MultiGrid (AMG)

*This work was supported in part by NSF under grant CCR-9618827, and in part by the Minnesota Supercomputer Institute.

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

methods were introduced in the seventies – initially by Ruge and Stuben [16] – to remedy these limitations. Their overall success depends on the underlying PDE problem, and has been somewhat mixed. In contrast, preconditioned Krylov methods, using ILU preconditioners, are designed to be 'general-purpose' methods for solving arbitrary sparse linear systems of equations. They can work in many situations where multigrid methods fail but their main drawback is that the convergence rate usually deteriorates as the size of the linear system increases. In years past, where systems of a few tens of thousands unknowns was considered large, preconditioners such as incomplete LU factorizations applied to the original system was an adequate and fairly reliable approach. Although multigrid could be vastly superior for certain problems, users have often sacrificed speed for better robustness and generality. As the problems become larger the advantage of multi-level approaches can be overwhelming. However, multi-level methods are somewhat specialized, i.e., their performance is guaranteed to be optimal only for a specific class of problems. An ideal method would be one whose cost scales well with the size of the problem and which is as general purpose as the ILU-Krylov combination.

Recently, a class of preconditioners that drew much attention is a collection of ILU factorizations which possess certain features of multigrid techniques. ILUM [17] is one such approach and recent work by Botta and co-workers [6, 7], and [19, 20], indicates that this type of approach can be fairly robust and scale well with problem size, unlike standard ILU preconditioners. This method combines the generality of Krylov methods and the scalability of multigrid methods. The idea was recently extended to a block version (BILUTM) with a domain decomposition strategy [19, 20]. The BILUTM preconditioner is a block generalization of ILUM in which the blocks are treated as sparse matrices. Tests in [19] indicate that BILUM is generally more efficient and more robust than a standard ILUT-preconditioned GMRES [18] as well its scalar sibling, ILUM. For certain hard problems, these attributes come with the added benefit of smaller memory usage.

BILUTM provides a general framework for multi-level parallel ILU preconditioning which can accommodate both fine-grain and coarse-grain parallelism. In this paper we extend this approach in several ways. Specifically, the main contributions of this paper relative to [19, 20] are as follows: (1) a fully recursive implementation is introduced; (2) Multi-grid like inter-level iterations are allowed; and (3) new independent set strategies for a better handling of indefinite problems are proposed.

The rest of this paper is arranged as follows. Section 2 gives a general overview of Algebraic Multilevel methods based on Block ILU factorizations. Section 3 discusses the construction and implementation of the Algebraic Recursive Multilevel Solver. The coarsening process, and selection of independent sets is covered in Section 4. Section 5 is a large section devoted to numerical experiments with the ARMS preconditioner. Some conclusions are drawn in Section 7 along with suggestions on how best to use this preconditioner.

2 Multilevel ILU preconditioners: an overview

This section gives an overview of the main ideas used to develop Algebraic Multilevel solution methods. Many of these solvers start by separating the unknowns of the original system in two sets, one of them being labeled "coarse". It is important to mention at the outset that in "algebraic"

methods there is no real mesh and therefore the labeling of vertices into coarse and fine is somewhat arbitrary. There has been some inconsistency in the literature in the way this is done and used. In this paper "coarse" points refer to points that are in an independent set. In the simplest scalar independent set case, this corresponds to a set of points that are not coupled. Once the independent set is obtained the various methods differ in the way they deal with with the reduced system obtained by eliminating the independent set (either exactly or approximately).

2.1 Independent sets and aggregates

The multi-level ILU preconditioners developed in [17, 7, 8, 20, 19] exploit the property that a set of unknowns that are not coupled to each other can be eliminated simultaneously in Gaussian elimination. Such sets are termed 'independent sets', see e.g., [15]. In [19], the ILUM factorization described in [17] was generalized by resorting to "block independent sets". A block independent set is a set of groups (blocks) of unknowns such that there is no coupling between unknowns of any two different groups (blocks) [19]. Unknowns within the same group (block) may be coupled. This is illustrated in Figure 1. The terminology used in the Algebraic Multigrid community for certain types of block independent sets is "aggregates" [21][1]. In this paper this term will be used while the term "block independent sets" will be reserved for the particular case when the aggregates consist of elements that are all coupled with each other (e.g. nodes of a triangle in a triangular finite element mesh). Some simple methods for finding standard and block-independent sets have been considered in [17, 19] and elsewhere. In Section 4 additional strategies will be examined.

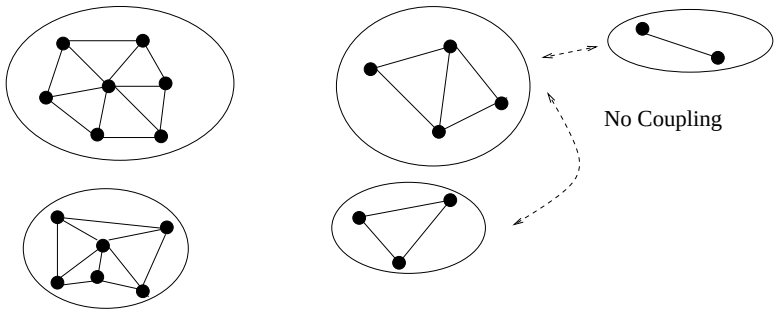

Figure 1: Aggregates, groups or blocks.

2.2 Block LU preconditioners

In various existing forms of multilevel ILU factorizations [3, 17, 7, 6] the unknowns are reordered, listing the nodes associated with the Independent Set first, followed by the other unknowns. After this reordering, the original matrix A_l at the l-th level takes the following form

$$P_l A_l P_l^T = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \tag{1}$$

This is then approximated by

$$\begin{pmatrix} \tilde{L}_l & 0 \\ \tilde{G}_l & I \end{pmatrix} \times \begin{pmatrix} \tilde{U}_l & \tilde{W}_l \\ 0 & \tilde{A}_{l+1} \end{pmatrix} \tag{2}$$

[1] Though "independent aggregates" would have been more appropriate.

where where I is the identity matrix, L_l and U_l are the LU (or ILU) factors of B_l,

$$\tilde{G}_l \approx E_l U_l^{-1}; \qquad \tilde{W}_l \approx L_l^{-1} F_l;$$

and A_{l+1} is an approximation to the Schur complement with respect to C_l,

$$\tilde{A}_{l+1} \approx A_{l+1} = C_l - E_l B_l^{-1} F_l$$

Typically it is inexpensive to solve linear systems with U_l and L_l since these arise from an ILU-type factorization. Therefore observe that all we need for defining a preconditioning for A_l is to provide a way to solve the reduced system (i.e., a system associated with A_{l+1}). It is here that different methods proposed vary.

An obvious option is to use direct solution methods. The reduced system is likely to still be large and a direct method will tend to be expensive both in terms of computation and memory requirement. However, it is important to note that the method of Nested Dissections (ND) of George [13] is a method in this class. The ND reordering exploits independent aggregates obtained by recursively dividing the graph into two disconnected subgraphs using "separators". A number of techniques for stabilizing sparse Gauss Elimination are presented in [12].

In traditional Algebraic Multi-Grid (AMG, [16]), grid transfer operators are defined that restrict and interpolate. In a Galerkin formulation, an interpolation matrix is determined algebraically - and then the restriction matrix is its transpose. In other formulations, as in [9] the grid transfer matrices are not transposes of each other. Interpolation weights are typically defined after some coloring scheme determines fine and coarse grid points. This procedure is repeated until a final coarse grid is reached where the last system is solved using a relaxation scheme or a direct method.

In ILUM [17], B is a diagonal matrix and the above factorization is repeated for A_{l+1} after an independent set ordering is found. A dropping strategy is used to limit fill-in. In BILUM [19], BILUTM [20], and in this paper, B is a block diagonal matrix. One motivation behind this type of factorization is that the diagonal elements or blocks may be used as pivots simultaneously, in parallel. The method being introduced in this paper is a generalization of these methods. The factorization techniques are similar, but the factors are utilized differently in the solve phase.

The AMLI [3, 4] preconditioners are based on a set of nested finite element grids. Here, B_l is associated with basis functions that exist on level l, but not on level $l+1$. The factorization is repeated until the coarsest mesh is reached. In some versions [2], couplings are deleted in order to maintain a sparse Schur complement. Another method based on a series of nested finite element grids is NGILU [8]. In this method, the nodes are reordered so that B contains connections between fine mesh nodes that are not in the coarse mesh. NGILU method does not work for unstructured grids. In a subsequent method, MRILU [7], two of the authors develop a preconditioner similar to ILUM where B is a diagonal matrix.

In MLILU[5], B is constructed by an algorithm that determines an optimal set of parents, or coarse grid nodes based on generating a small amount of fill upon factorization. Alternatively, in [22], B is made diagonal by modifying the right-hand-side.

The AMLI, MLILU and NGILU preconditioners are recursive in nature. Not only is the factorization defined recursively, but the $(l+1)^{st}$ level preconditioner is part of the preconditioner

for the l^{th} level. In AMLI and NGILU, all of the matrices A_l are symmetric positive definite. This allows iterative solution at each level using a preconditioned conjugate gradient algorithm. Results of some non-symmetric test cases are presented for MLILU. The methods work for a wide range of matrices derived from elliptic operators on finite element or finite difference grids. However, their applicability for general sparse matrices has not been documented and remains unclear

3 General Multilevel ILU preconditioners

An algebraic multilevel ILU preconditioner starts with the independent set reordering (1) and the approximate block-factorization (2). There are two immediate questions that need to be addressed. The first is how to obtain the block factorization. The second is how to solve the reduced systems related to the matrix A_{l+1}. We consider these two issues in turn.

3.1 Computing the block factorization

As was observed in [20] the Schur complement A_{l+1} can be computed from a restriction of the Gaussian elimination process. The idea is that if one zeroes out the elements in the F_l matrix by a Gaussian elimination process, then the $(2,2)$ block C_l in the resulting matrix will yield the Schur complement. This is easily seen.

The implementation we use follows very closely that of the ILUT algorithm [18]. The process is illustrated in Figure 2 and described in Algorithm 3.1. The notations $a_{i,\beta}$, $l_{i,\beta}$ and $u_{i,\beta}$ represent the i^{th} rows of A, L and U, respectively. The process is based on the IKJ version of Gaussian elimination [18]. In this version, entries of the i-th row of A that are in the lower part of A are eliminated from left ($j = 1$) to right (up to $j = i - 1$). In the restricted Gaussian elimination, the elimination begins at $j = 1$ and ends at $j = min(i - 1, m)$, where m is the size of the independent set. So when $i \leq m$ the process is the same as in the standard IKJ version of Gaussian elimination. For $i > m$ the elimination is terminated when j reaches m.

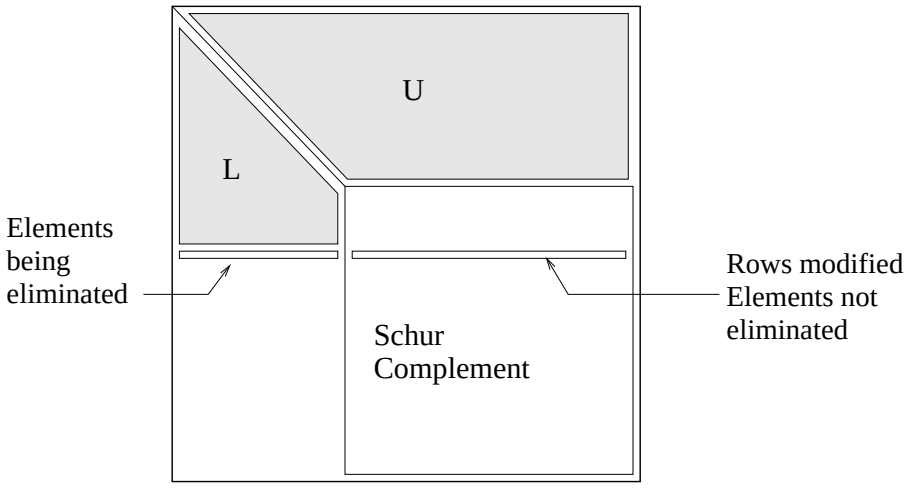

Figure 2: Restricted Gaussian elimination.

ALGORITHM **3.1 Restricted ILUT(τ, p) factorization.**

1. *For* $i = 2, n$, *Do:*
2. $\quad w := a_{i,\beta}$
3. $\quad$ *For* $k = 1, \min(i-1, m)$ *and when* $w_k \neq 0$, *Do:*
4. $\quad\quad w_k := w_k / a_{k,k}$
5. $\quad\quad$ *Set* $w_k := 0$ *if* $w_k < \tau * \text{nzavg}(a_{i,\beta})$
6. $\quad\quad$ *If* $w_k \neq 0$, *then*
7. $\quad\quad\quad w := w - w_k * u_{k,\beta}$
8. $\quad\quad$ *End If*
9. $\quad$ *End Do*
10. $\quad$ *Retain only the* p *largest entries of* $|w_j|$ *for* $j \leq \min(i-1, m)$ *and* $j \geq \min(i, m)$
11. $\quad$ *Set* $l_{i,j} := w_j$ *for* $j = 1, \ldots, \min(i-1, m)$ *whenever* $w_j \neq 0$
12. $\quad$ *Set* $u_{i,j} := w_j$ *for* $j = \min(i, m), \ldots, n$ *whenever* $w_j \neq 0$
13 *End Do*

Algorithm 3.1 yields an ILU factorization of the form

$$A = \mathbf{LU} + R, \tag{3}$$

where R is the residual matrix representing the difference between A and $\mathbf{LU}$. Note that this process is independent of the ordering used for the matrix. In other words it is not necessary to have an independent set or aggregate set ordering. From a practical point of view all that is required is that the resulting Schur complement block be fairly sparse (after dropping is used). This enables a recursive approach in which the Schur complement is treated again as a sparse matrix.

3.2 Recursive Factorization and Solution

As was mentioned above, once the factorization (1), (2) is computed, a preconditioning to the original matrix can be obtained by providing a way of solving linear systems with the matrix A_{l+1}. Several options are available:

(a) Perform an ILU (e.g. ILUT) factorization of A_{l+1} and use this as an approximation when solving linear systems with A_{l+1}.

(b) Solve with an iterative accelerator in combination with the above approximations.

(c) Factor the matrix A_{l+1} recursively. For solving the system with A_{l+1}, use (recursively) the block factorization at the lower levels. For the last level refer to options (a) or (b) above.

Note that any other type of approximation to A_{l+1}, e.g., an approximate inverse, can be used instead of ILUT in (b.) for solving linear systems with A_{l+1}. From the programming point of view Cases (a) and (b) can be considered as particular cases of (c). In fact it can even be said that an ILUT preconditioner for the whole matrix A is a particular case – with the maximum number of levels set equal to zero.

3.3 ARMS Factorization

As described above the main factorization step is as follows

$$P_l^T A_l P_l = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \approx \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix} \times \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix} \tag{4}$$

This process is similar to the BILUTM factorization [20]. One major difference is on the implementation side. ARMS is implemented in C and relies heavily on recursivity. Recursivity is exploited and simplifies the construction of the factorization. The algorithm consists of a sequence of independent set orderings followed by a reduction (PILUT) and a recursive block factorization on the reduced system.

ALGORITHM **3.2** *ARMS(A_{lev}) factorization*

1. *If $lev = last\_lev$ then*
2. *Compute $A_{lev} \approx L_{lev} U_{lev}$ [e.g. ILUT factorization of A_{lev}]*
3. *Else:*
4. *Find an independent set permutation P_{lev}*
5. *Apply permutation $A_{lev} := P_{lev}^T A_{lev} P_{lev}$*
6. *Call Partial ILUT to compute factorization (4)*
7. *Call ARMS(A_{lev+1})*
8. *EndIf*

Possible variants here are in steps 2 and 4. In step 2, different approximations can be used to solve the last level system. In step 4, we can utilize a variety of ordering techniques for finding independent sets.

In this paper we will restrict our attention to solving the last level system by GMRES preconditioned with ILUT. As a special case, if the last system is small enough, a single sweep of an LU solve using the ILUT factorization may be used. Accurate ILU factorizations may be expensive to compute and the L and U factors may be quite large. This can be expensive in a W-cycle where the forward and backward solve operations may need to be performed often. An iterative solution process on the last level using a less accurate factorization may be more efficient. Parameters for the solution of the last system are passed to the solver.

As mentioned above, parts of the code are implemented in C to better exploit recursivity as well as dynamic memory allocation. The block LU restriction and prolongation operators are constructed as a linked list of structs. Each struct contains the grid transfer matrices, a permutation array and pointers to the next and previous levels. The dynamic memory feature in C also allows very efficient use of space during the factorization process.

In section 4 a number of different methods for finding independent sets are discussed.

3.4 Recursive solutions

After permutation, the system at level l can be put in the form

$$P_l^T A_l P_l x_l = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} x_l = b_l; \quad x_l = \begin{pmatrix} y_l \\ z_l \end{pmatrix}; \quad b_l = \begin{pmatrix} f_l \\ g_l \end{pmatrix}$$

or, using the approximate factorization (tildes are dropped):

$$\begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix} \times \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix} \times \begin{pmatrix} y_l \\ z_l \end{pmatrix} = \begin{pmatrix} f_l \\ g_l \end{pmatrix} \tag{5}$$

Hence the following standard V-cycle solution.

ALGORITHM **3.3** *V-cycle_RMLS(A_l, b_l) – Recursive Multi-Level Solution*

1. *Solve* $L_l y = f_l$
2. *Compute* $g_l' := g_l - G_l y$
3. *If* $lev = last\_lev$ *then*
4. *Solve* $A_{l+1} z_l = g_l'$ *using GMRES, preconditioned with ILUT factors*
5. *Else*
6. *Call RMLS(A_{l+1}, g_l')*
7. *Endif*
8. *Back-Substitute to get* y_l, *i.e., solve* $U_l y_l = [y - W_l z_l]$

Note again that in line 6, the recursive incomplete block LU factorization is used for preconditioning the solution of the lower level system (with the matrix A_{l+1}). The heart of the C code for the preconditioning solution step is as follows

```
void arms(double *b, double *x, pmatptr *arm, csptr *schur, csptr *ilusch)
{
   int j;

   for (j=0; j<arm->n; j++)
      wk[j] = b[j];

   forward(x, wk, arm);

   if (arm->next == NULL)
      gmres(&wk[arm->last], &x[arm->last], schur, ilusch);
   else
      arms(&wk[arm->last], &x[arm->last], arm->next, schur, ilusch);

   backward(x, wk, arm);

   return;
}
```

The index of the last unknown grouped in the independent set is stored in `last`.

In the current implementation, the block LU operators are stored in structs called `arm`. The last Schur complement system is stored in `schur` and the ILUT factors of this system are stored in `ilusch`. Note that only the lower portions of the work and solution vectors are sent to the next level.

The usual reasoning behind the success of this type of technique is that low frequency error is eliminated on the coarse grid, and high frequency error is eliminated on the fine grid [14]. Usually, a direct method or a small number of iterations of a relaxation method, such as Gauss-Seidel, is used for the coarse grid. A number of options exist for the lower level solver. In BILUTM a single sweep of a forward and backward solve with the ILUT factorization computed during the factorization phase is utilized. It is possible in some instances to get a more accurate solution on the next

level system and consume less total computing time by using a less accurate ILUT factorization. Computation of an approximate inverse for the last level systems is another option, although finding an approximate inverse of the lower level system through Frobenius norm minimization can be cost prohibitive.

3.5 Intermediate level iteration

As was discussed in the previous section, the solve in step 6 of Algorithm 3.3 involves only a recursive call to the next level. In multigrid terminology this is called a V-cycle. No iteration of any type is performed at the intermediate levels of a V-cycle. The restriction operators are applied until the coarsest system is reached, the last system is solved, and then the prolongation operators are applied up to the top level.

It is possible to include an iteration in step 6 of Algorithm 3.3 leading to the so-called W-cycle, which involves a Krylov acceleration. This is depicted in Algorithm 3.4. In contrast to geometric multi-grid methods where the prolongation operator is actually an interpolation matrix, the successively coarser levels here result from partial ILU factorizations. Therefore, it can be more beneficial to find accurate solutions to the coarse grid systems than it is in other multi-grid methods.

ALGORITHM **3.4** *W-cycle_RMLS(A_l, b_l) – Recursive Multi-Level Solution*

1. *Solve* $L_l y = f_l$
2. *Compute* $g_l' := g_l - G_l y$
3. *If* $lev = last\_lev$ *then*
4. *Solve* $A_{l+1} z_l = g_l'$ *using GMRES, preconditioned with ILUT factors.*
5. *Else*
6. *Solve* $A_{l+1} z_l = g_l'$ *using GMRES preconditioned*
7. *with RMLS(A_{l+1})*
8. *Endif*
9. *Back-Substitute to get* y_l, *i.e., solve* $U_l y_l = [y - W_l z_l]$

Consider first a single level preconditioner. Let A_0 represent the full system and A_1 represent the reduced system. If the block LU operators were exact, and the reduced system, $\tilde{A}_1 \tilde{z}_0 = \tilde{g}_0'$, were solved exactly, convergence on the full system could be achieved in a single outer iteration. In a multiple level preconditioner, A_1 will be referred to as the first reduced system. An exact solution of the first reduced system may be found iteratively by applying Algorithm 3.4. An accurate solution of the first reduced system is obtained iteratively using a Krylov subspace accelerator preconditioned with the next level block factorization for A_1. The second reduced system may be solved the same way, and so on. Each iteration involves a matrix-vector multiply and a preconditioning step. Since the preconditioner may be different at every iteration, a flexible variant of the preconditioned GMRES algorithm[18] must be used.

The solution process for a 3 level ARMS preconditioner can be represented graphically as in Figure 3. The height of each symbol indicates the log of the 2-norm of the residual at a particular level. Horizontal lines divide the scales for the different levels. The full system, level 0, is at the

top of the graph, and the last reduced system, level 3, is at the bottom. Levels 1 and 2 are in between. Note that each time the solve starts "descending" to a level with fewer unknowns it goes all the way to level 3 and does not return to level 2 until the stopping criteria at level 3 has been satisfied. Then it may need more than one iteration at level 2 in order to satisfy the stopping criteria there. If so, it must descend to level 3 again for each iteration. Similarly, the solver does not return up to level 1 or level 0 until either convergence at the next lower level, or the maximum number of iterations at the lower level is reached.

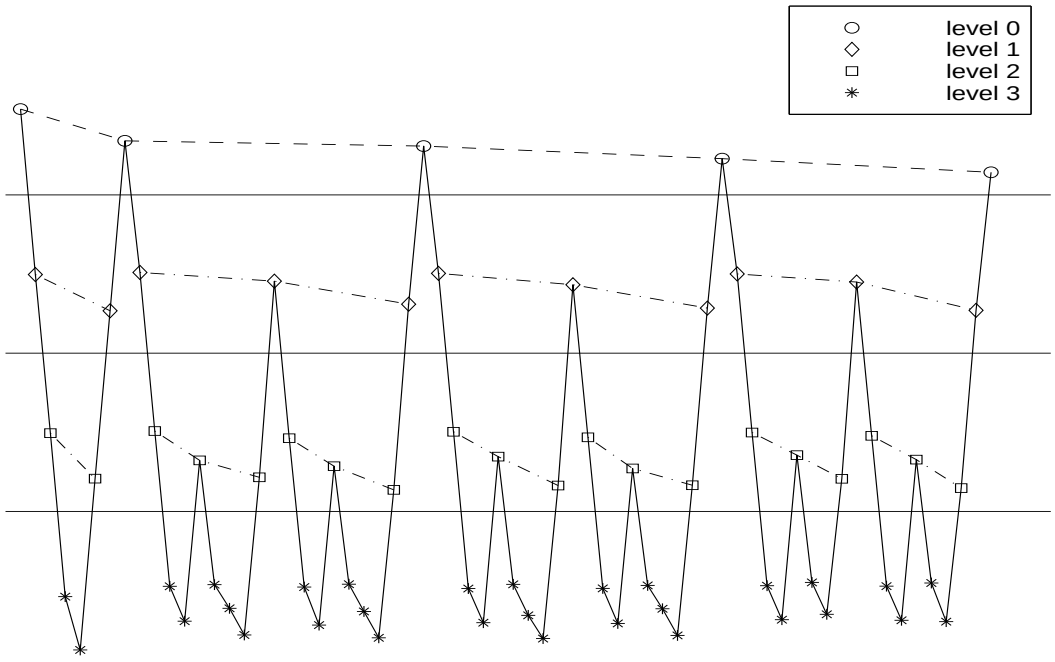

Figure 3: A representation of W-cycle for intermediate level iterations. The preconditioner represented has 3 levels

The code for a W-cycle is similar to the code for a V-cycle. The difference appears after the `else` statement. In a W-cycle the ARMS solver does not call itself, but rather calls GMRES, which calls ARMS in the preconditioning step. The V-cycle can be viewed as a special case of the W-cycle where the preconditioning step is applied once, without any iteration.

For large systems, completely accurate calculation of the block LU matrices is cost prohibitive. Parameters may be adjusted in the partial ILUT(τ, p) algorithm to obtain a satisfactory trade-off between factorization time, memory requirements and accuracy. As fill levels are decreased, the factorization time, and time required for individual matrix-vector products go down, while the number of outer iterations required goes up.

3.6 ARMS-2: Avoiding storage of Schur complements

Storing each of the successive Schur complements A_i is expensive. These matrices are not required in the V-cycle approach, except possibly the last reduced system. However, since Krylov acceleration is used for the W-cycle approach, matrix-vector operations with the A_i matrices are required.

In this section we describe a variant which will avoid this by exploiting the multi-level factors to perform matrix-vector product operations $A_i \times$ vector. Algorithm 3.2 produces a factorization at each level of the form

$$A_l = \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix} \times \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix} + R \tag{6}$$

where R is the matrix of elements that have been dropped. Ignoring R, and applying the inverse of the lower triangular matrix on the left yields the following.

$$\begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix}^{-1} \times A_l \times \begin{pmatrix} 0 \\ w \end{pmatrix} \approx \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix} \begin{pmatrix} 0 \\ w \end{pmatrix}$$

The action of A_{l+1} on a vector w can be computed as

$$A_{l+1} w \approx \begin{pmatrix} 0 & I \end{pmatrix} \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix}^{-1} \times A_l \times \begin{pmatrix} 0 \\ w \end{pmatrix} \tag{7}$$

The above formula shows that a matrix-vector product with A_{l+1} can be performed essentially with one block-forward solve and a product with the matrix A_l of the higher level. Therefore, this process may be applied recursively to compute matrix vector products on any level.

Note that the above formula is equivalent to an action by an approximation to $A_{l+1}w$, instead of A_{l+1} itself. As it turns out this is generally a better representation of the local Schur complement S_l than A_{l+1}. Consider first the simple situation when the LU factors L_l, U_l of B_l are exact and no dropping is used when computing G_l, W_l. Then R has the form

$$R = \begin{pmatrix} 0 & 0 \\ 0 & R_{22} \end{pmatrix}$$

and in this case, we have

$$\begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix}^{-1} \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} = \begin{pmatrix} L_l^{-1} B_l & L_l^{-1} F_l \\ 0 & S_l \end{pmatrix}$$

where S_l is the Schur complement associated with C_l, and

$$S_l = C_l - E_l B_l^{-1} F_l.$$

Therefore, the result in the right-hand side of formula (7) is $S_l w$, exactly. On the other hand, equation (6) shows that

$$A_{l+1} = S_l + R_{22}$$

so the result using (7) is actually more accurate in this case, since it uses the Schur complement instead of its approximation A_{l+1}. When ARMS-2 is used, the operator A_{l+1} in Line 6 of Algorithm 3.3 (and Line 7 of 3.4) are replaced by an approximation $\tilde{S}_l$ to S_l.

We have shown one situation when Formula (7) yields the result of the exact Schur complement at level l. One might ask whether there are other situations when this happens. Using (6), we obtain the following general relation:

$$\begin{aligned} \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix}^{-1} A_l &= \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix} + \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix}^{-1} \begin{pmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{pmatrix} \\ &= \begin{pmatrix} U_l + L_l^{-1} R_{11} & W_l + L_l^{-1} R_{12} \\ R_{21} - G_l L_l^{-1} R_{11} & A_{l+1} + R_{22} - G_l L_l^{-1} R_{12} \end{pmatrix} \end{aligned}$$

According to (6) we have $A_{l+1} + R_{22} = C_l - G_l W_l$, so the result of formula (7) is

$$(A_{l+1} + R_{22} - G_l L_l^{-1} R_{12})w = C_l - G_l(W_l + L_l^{-1} R_{12})$$

Now according to (6) again, we note that $L_l W_l + R_{12} = F_l$ so that $W_l + L_l^{-1} R_{12} = L_l^{-1} F_l$ in the above formula. Therefore, the result of formula (7) becomes,

$$(C_l - G_l L_l^{-1} F_l)w$$

which means that the formula is equivalent to replacing the Schur complement by

$$\tilde{S}_l = C_l - G_l L_l^{-1} F_l$$

This is to be contrasted with the regular ARMS which approximates this Schur complement by $A_{l+1} = C_l - G_l W_l + R_{22}$. What the above formula shows is that perturbations (i.e., dropping) has no effect on the computation of the matrix-vector product by formula (7) as long as it is confined to the blocks $(1,2)$ and $(2,2)$. In particular if no dropping is applied when building G_l and L_l, i.e., the lower part of the block factorization, then the matrix-vector product will be the same as that associated with the exact Schur complement.

In fact, there is a less expensive way to avoid spoiling the result of the product with S_l and saving more storage at the same time. Recall that in theory (i.e., when no dropping is used), $G_l \equiv E_l U_l^{-1}$. In (7), we could discard the matrix G_l and perform its action on a vector by invoking the matrices E_l and U_l from which it is theoretically constructed. We have

$$\begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix}^{-1} A_l = \begin{pmatrix} \star & \star \\ \star & C_l - E_l U_l^{-1} L_l^{-1} F_l \end{pmatrix} \tag{8}$$

The action of (7) is therefore equivalent to that of the Schur complement $C_l - E_l U_l^{-1} L_l^{-1} F_l$. This is the exact Schur complement if L_l and U_l are the exact LU factors for B_l, regardless of whether or not dropping is exercised in the rest of the matrix. The action of the matrix W_l can be similarly performed in the backward substitution process. A clear additional advantage of this method is that the storage of the matrices G_l and W_l is obviated. The space for these matrices is only needed temporarily when constructing the matrix A_{l+1} in the restricted ILUT procedure (Algorithm 3.1). Once A_{l+1} is constructed these two parts of the block factorization can be erased.

3.7 Quality of the Multilevel Factorization Preconditioning

We make the simplifying assumption that dropping is allowed only when forming the Schur complement matrix A_{l+1}. An alternative assumption is that ARMS-2 is used with the exact factors L_l and U_l for B_l. In other words, the factorization process does not drop elements in the factors A_l, and in G_l and W_l. This leads to a factorization of the form,

$$A_l = \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix} \times \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & R_{22} \end{pmatrix}$$

where R_{22} is the matrix of elements that have been dropped. The matrices L_l and U_l are the exact LU factors of the matrix B_l. In practice this assumption does not cause any particular problem if

the structure of A_l is carefully selected when obtaining the independent set (for example when A_l is diagonal). Furthermore, it is assumed that the solve with A_{l+1} is exact. Notice that $A_{l+1} = S_l - R_{22}$ where

$$S_l = C_l - E_l B_l^{-1} F_l = C_l - G_l W_l.$$

is the exact Schur complement associated with the matrix C_l. Now consider the preconditioned matrix obtained from the resulting factorization,

$$\hat{A}_l \equiv \begin{pmatrix} L_l & 0 \\ G_l & I \end{pmatrix}^{-1} \quad A_l \quad \begin{pmatrix} U_l & W_l \\ 0 & A_{l+1} \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 \\ 0 & I + R_{22} A_{l+1}^{-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & S_l A_{l+1}^{-1} \end{pmatrix} \tag{9}$$

The spectrum of the preconditioned matrix consists of the eigenvalue one repeated $n_l - n_{l+1}$ times where n_k is the dimension of the matrix A_k and the eigenvalues of $S_l A_{l+1}^{-1}$.

In the case when ARMS-2 is used the solves in lines 6 and 7 of Algorithms 3.3 and 3.4 involve an approximation $\tilde{S}_l$ to the Schur complement S_l rather than the matrix A_{l+1}. Therefore, the matrix A_{l+1} in (9) is to be replaced by this approximation. As was seen in the previous section, under the current assumption of exact solves with S_l and no dropping other than in the (2,2) part, S_l is actually exact. The relation (9) shows that a one -level ARMS-2 preconditioner becomes exact under this assumption. This is not true with the regular ARMS factorization.

It is common when analyzing Block-ILU type preconditioners to make assumptions on the approximation to the Schur complement under consideration [1]. Here we make a similar assumption on the smallness of R_{22} relative to S_l. Specifically, it is assumed that for some vector norm, we have

$$\|R_{22}x\| \leq \gamma \, \|S_l x\| \, , \quad \forall x \tag{10}$$

with $|\gamma| < 1$. Then the following proposition follows immediately.

Proposition 3.1 *Assume that S_l is nonsingular and that (10) holds for some $0 \leq \gamma < 1$ and some vector norm $\|.\|$. Then the eigenvalues of the preconditioned matrix $\hat{A}_l$ are such that*

$$\frac{1}{1+\gamma} \leq |\lambda_i(\hat{A}_l)| \leq \frac{1}{1-\gamma} \tag{11}$$

Proof. From (10) it is seen that the eigenvalues μ_i of the generalized problem $R_{22}x = \mu S_l x$ are bounded in modulus by γ. Since S_l is nonsingular an arbitrary eigenvalue λ_i of $S_l A_{l+1}^{-1} = S_l (S_l + R_{22})^{-1}$ is nonzero and it can be shown that it is the inverse of $1 + \mu_i$, for some eigenvalue μ_i of the generalized problem $R_{22}x = \mu S_l x$. We have

$$1 - \gamma \leq 1 - |\mu_i| \leq |1 + \mu_i| \leq 1 + |\mu_i| \leq 1 + \gamma$$

which gives the result, after inversion, for all eigenvalues of the block $S_l A_{l+1}^{-1}$ in (9). The other eigenvalues are equal to one and satisfy the inequality as well. ■

We note that the condition (10) can be replaced by one involving the transposes of the matrices R_{22} and S_l:

$$\|R_{22}^T x\| \leq \gamma \, \|S_l^T x\| \, , \quad \forall x \tag{12}$$

with $|\gamma| < 1$, and the above result would also hold.

It would now be useful to provide sufficient conditions under which assumption (10) or (12) is satisfied. When constructing the approximate Schur complement A_{l+1}, elements are dropped when they are smaller then a tolerance relative to the norm of the original row. The assumption we make is based on this (though not equivalent to it):

$$\|R_{22}^T e_j\|_1 \leq \tau \|S_l^T e_j\|_1 \quad j = 1, \ldots, n_{l+1} \tag{13}$$

In other words the 1-norm of each row of R_{22} is bounded from above by a multiple of the corresponding row in S_l. We preferred to utilize the row version (12) instead of (10) because our implementations are row oriented. We now have the following proposition.

Proposition 3.2 *Assume that (13) holds. Then*

$$\max_{x \neq 0} \frac{\|R_{22}^T x\|_1}{\|S_l^T x\|_1} \leq \tau \ \kappa_1(S_l^T) \tag{14}$$

where $\kappa_1(S_l^T)$ is the condition number for S_l^T associated with the 1-norm.

Proof. For an arbitrary vector x with components ξ_i we have

$$\begin{aligned} \|R_{22}^T x\|_1 \equiv \| \sum R_{22}^T \xi_j e_j \|_1 &\leq \sum |\xi_j| \|R_{22}^T e_j\|_1 \\ &\leq \tau \sum |\xi_j| \ \|S_l^T e_j\|_1 \\ &\leq \tau \sum |\xi_j| \ \times \max_j \|S_l^T e_j\|_1 \\ \|R_{22}^T x\|_1 &\leq \tau \|x\|_1 \|S_l^T\|_1 \end{aligned} \tag{15}$$

On the other hand

$$\|S_l^T x\|_1 \geq \frac{\|x\|_1}{\|S_l^{-T}\|_1} \tag{16}$$

Dividing (15) by (16) and taking the max yields the desired result. ■

The result of (11) can now be exploited with $\gamma = \tau \kappa_1(S_l^T)$. This result is pessimistic in that γ is only guaranteed to be less than 1 for values of τ that are less than the inverse of the condition number of S_l.

4 Independent Sets and Coarsening

Some strategies for finding independent sets have been discussed in [19]. Relatively inexpensive techniques are presented for finding sets of small blocks which are independent of other blocks in the set. A more sophisticated technique is presented in [5] which does not necessarily compute independent sets, but instead tries to minimize fill-in during factorization.

ARMS employs a simple heuristic which constructs blocks by using a level set approach combined with a heuristic for rejecting elements from the coarse set. A weight array is constructed ahead based on certain criteria for determining when it is acceptable to assign a node to the C set. In the algorithm this array is called w. Details of the procedure employed to construct the block independent sets are shown in Algorithm 4.1.

ALGORITHM **4.1** *Indset Ordering with weights*

```
1.  marker(1:n) = 0
2.  For k = nod := 1, n and if marker(nod)== 0 Do:
3.      jcount = 0
4.      If (w(nod) < tol) then Add_to_F (nod)
5.      Else
6.          Add_to_C(nod) ; Level_Set = {nod}
7.          While (jcount < bsize) Do
8.              For each j in current Level_Set Do
9.                  If (w(nod) < tol) then Add_to_F (nod)
10.                 Else: Add_to_C (nod); jcount++; Endif
11.             EndFor
12.         EndWhile
13          For each j in Level_Set set Do
14.             For each k in adj(j) Do
15.                 If marker(k) == 0 Add_to_F(k)
16.             EndFor
17.         EndFor
18.     EndIf
19. EndFor
```

After the independent sets are formed, the matrix is reordered so that C unknowns corresponding to the independent blocks are grouped together at the top followed by the F unknowns. An example of the reordering process is given in Figure 4.

Here are a few additional notes for clarification. The level set *Level_Set* is updated by each of the *Add_to_C* routines. Similarly for the marker array. Each time a node is assigned to the set C or F the routines Add_to_C and Add_to_F update the marker (which in effect carries the ordering information). Another point is that some additional code (not shown) is included to avoid deadlock situations which correspond to cases where *jcount* cannot reach *bsize* because the algorithm is not making progress. There are many possible variants to the above algorithm. For example, based on similar ideas used for the reversed Cuthill McKee ordering, we can reverse the ordering for each level.

An important entry in the above algorithm is the vector w consisting of the weights. The w vector used in our current version is based on relative diagonal dominance. Some raw diagonal dominance coefficients are first computed as:

$$\hat{w}(i) = \frac{|a_{ii}|}{\sum_{j=1}^{n} |a_{ij}|}$$

Note that $0 \leq \hat{w}(i) \leq 1$ and that when $a_{ii} \neq 0$ the inverse of $\hat{w}(i)$ is $\hat{w}(i)^{-1} = 1 + \sum_{j \neq i} |a_{ij}/a_{ii}|$. When the row is very strongly diagonally dominant, $\hat{w}(i)$ is close to one. At the other extreme when the row is very poorly diagonally dominant, $\hat{w}(i)$ is close to zero. However, we found that the use of these absolute ratios was ineffective for some matrices. Some matrices may have all of their rows far from being diagonally dominant and therefore the above algorithm may reject all

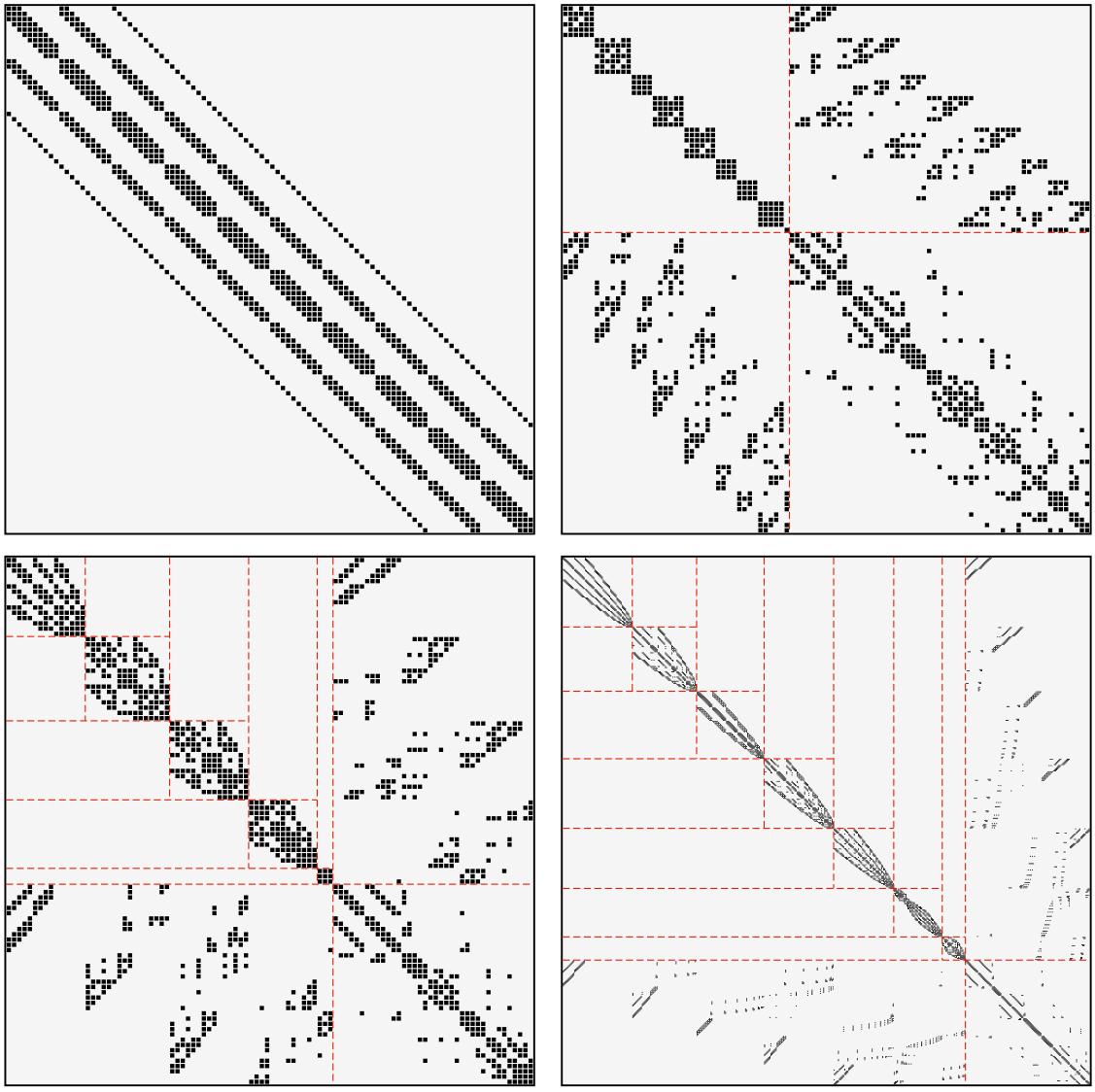

Figure 4: Independent set orderings for 9-point matrices. Top left: original matrix for 10×10 grid. Top right: result using a min block size of 2. Bottom left: result using a min block size of 12. Bottom right: result using a min block size of 50 for a matrix resulting from a 25×20 mesh.

rows to the fringe set. A more effective strategy is to utilize the criterion on a relative basis by normalizing all the $\hat{w}(i)$ ratios by the average or the maximum. For example,

$$w(i) = \frac{\hat{w}(i)}{\max_{j=1,\ldots,n} \hat{w}(i)}$$

5 Numerical Results

All experiments have been conducted on a PC with an Intel Pentium II processor with a clock speed of 450 MHz and a 256k L2 cache (Deschutes Core). The main memory size is 256 MB. For each of the experiments performed, a right hand side vector is generated by assuming a solution vector of all ones. The initial guess is a vector of randomly generated real values between 0 and 1. The convergence history is presented in plots of the norm of the residual versus the elapsed time. The times presented include the factorization and solution times, so the starting point in the curves shows initial residual and factorization time. The outer accelerator is FGMRES, and iteration continues until the norm of the residual has been reduced by a factor of 10^8. The first set of experiments for each matrix compares ARMS to the popular ILUT preconditioner.

Memory requirements are also compared for different preconditioners. The numbers displayed represent the number of non-zero entries in the matrices holding the various factors in the preconditioners. In some cases the level of fill is selected in order to compare the performance of preconditioners that require roughly the same amount of memory.

The are many parameters that can be varied in the ARMS solver and these are described below. The notation $(r_i)_k$ refers to the k^{th} iteration at level i.

- $\epsilon_N,\ k_{max,N}$ – stopping criteria for last level; iteration stops when $\frac{\| (r_N)_i \|}{\| (r_N)_0 \|} < \epsilon_N$ or $k_{max,,N}$ steps are completed on last level.
- $\epsilon_i,\ k_{max,i}$ – stopping criteria for intermediate levels; iteration stops when $\frac{\| (r_i)_j \|}{\| (r_i)_0 \|} < \epsilon_i$ or $k_{max,i}$ iterations are completed on level i, $i = 1, \ldots, N-1$.
- $dim,\ dim_i,\ dim_N$ – Krylov subspace dimensions at outer level, intermediate levels, and last level, respectively.
- N – number of levels.
- $bsize$ – target size of blocks in formation of independent sets.
- $\tau,\ p_N$ – parameters in ILUT(τ, p) factorization of last level.
- $\tau,\ p_i$ – parameters in restricted ILUT(τ, p) factorization of grid transfer matrices.

Results are presented for two different versions of the ARMS preconditioner. The first one, which is referred to as ARMS-1, saves the Schur complement, reduced system matrices for use in iterative solution procedures. The second one, ARMS-2, does not save the Schur complement matrices, but performs reduced system matrix vector product operations using the technique described in Section 3.6.

5.1 Tests with the TOKAMAK Matrices

The first set of experiments solve systems with a relatively small matrix. The TOKAMAK matrices are real unsymmetric and arise from nuclear fusion plasma simulations in a tokamak reactor [2]. These are part of the SPARSKIT collections and have been provided by P. Brown of Lawrence Livermore National Laboratory. The largest of these matrices, named UTM5940, has 5940 unknowns and 83,842 nonzeros. The condition number is estimated at 1.9×10^9.

The first experiment compares single level ARMS preconditioners to a standard ILUT(τ, p) preconditioner with two different levels of fill. The parameters in Table 1 are the same for each simulation. The fill level for the grid transfer matrices is set to 50. The fill level for the ILUT factorization of the last level is set to 50 for both of the ARMS preconditioners and one of the ILUT(τ, p) preconditioners. One of the ILUT(τ, p) preconditioners has $p = 25$ for comparison.

General		Last Level	
$bsize$	32	$k_{max,N}$	10
τ	0	ϵ_N	10^{-15}
p_0	50	p_N	50

Table 1: Parameters used in first set of solutions of the TOKAMAK matrices.

The number of nonzero elements in the various matrices stored for each method is shown in Table 2. The memory requirements for the two ARMS preconditioners are roughly the same as for ILUT(0,50). The memory requirement for ILUT(0,25) is about half. As is expected ARMS-1 requires more memory than ARMS-2 because the Schur complement matrix is stored.

	Unknowns			Memory			
Method	Tot.	Ind. Set	Red. Sys.	Ind. Sets	Schur	ILUT	TOTAL
ILUT(0,25)	5940	0	5940	0	0	271 400	271 400
ILUT(0,50)	5940	0	5940	0	0	537 253	537 253
ARMS-1	5940	3672	2268	265 878	112 115	204 156	582 149
ARMS-1	5940	3672	2268	265 878	0	204 156	470 034

Table 2: Memory used in first set of solutions of the TOKAMAK matrices.

The convergence history for the first set of experiments is displayed in Figure 5. The number of outer GMRES iterations required to reach convergence is not necessarily the best criterion for comparing the quality of a preconditioner. The plot on the left shows the 2-norm of the residual versus the outer iteration count. The plot on the right shows the 2-norm of the residual as a function of elapsed time. Though ARMS preconditioners yield convergence in fewer outer iterations, the results are mixed when the total solution times are compared. ARMS-2 takes the longest time to converge, and ILUT(0,50) takes the least. The elapsed time in Figure 5 includes the time spent creating the preconditioner and the time spent in the solve phase of the algorithm. Note that ILUT(0,25) takes the least time to factor, and ILUT(0,50) takes the longest. The two ARMS

[2] The TOKAMAK matrices available online from the matrix market of the National Institute of Standards Technology at `http://math.nist.gov/MatrixMarket`.

preconditioners are somewhere in between.

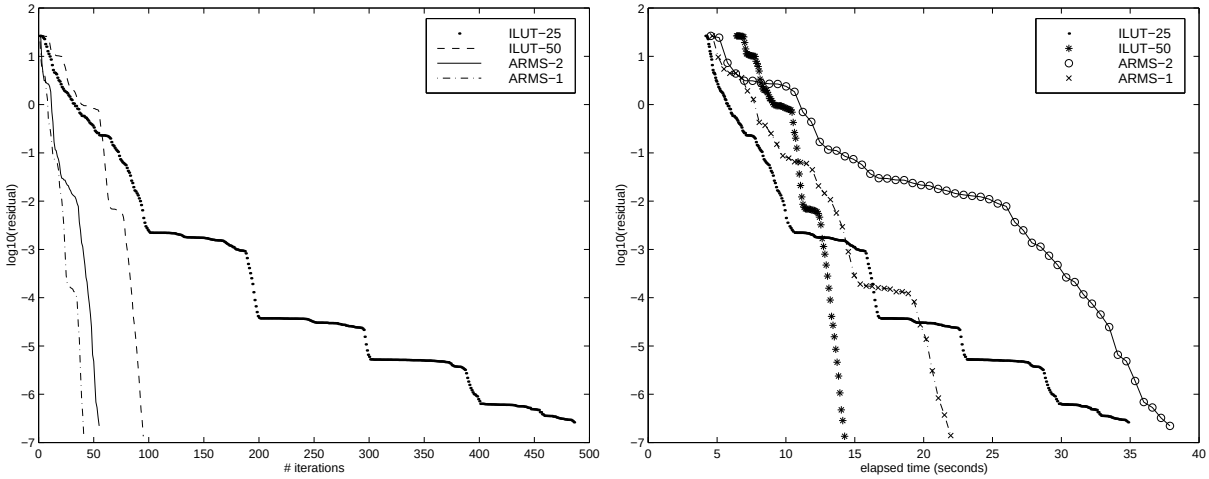

Figure 5: Convergence history of ILUT and 1 level of ARMS for solving TOKAMAK matrix UTM.5940 with small levels of fill.

The next set of experiments investigate the effect of varying the number of levels in the ARMS preconditioners. Iterative solves are performed at each level using FGMRES. As is expected, the cost of a multilevel preconditioning step increases exponentially as the number of levels increases. Therefore, it is important that extra work required for each outer iteration results in a more substantial reduction of the residual at each iteration. In this example, the fill levels are relatively low, as they are in the comparison to ILUT. Table 3 shows parameters common to all simulations in this example. The small ϵ values are never realized, and hence the k_{max} values dictate the number of iterations performed at each level.

General		Intermediate Levels		Last Level	
$bsize$	32	$k_{max,i}$	5	$k_{max,N}$	5
τ	0	ϵ_i	10^{-15}	ϵ_N	10^{-18}
$nlev$	varies	p_i	50	p_N	50

Table 3: Parameters used in second set of solutions of the TOKAMAK matrices.

The block LU matrices become smaller as the number of unknowns in the reduced systems decreases. Table 4 shows the sizes of the systems at each level, and how they are divided into independent sets and the reduced system. It also shows the number of nonzeros in the the LU factors of the independent sets, and two other sets of matrices that may or may not be stored at that level, depending on the number of levels in the preconditioner, and the version of ARMS being utilized. The value listed under ILUT is the number of nonzero entries in the incomplete LU factorization of the last level, if that level were indeed the last level. The number of nonzero entries in the Schur complement matrix at a particular level is listed under the heading `Schur`. Recall that the Schur complement matrix is stored for ARMS-1, and not for ARMS-2.

The numbers of nonzero entries in the preconditioning matrices are contained in Table 5. The

	Unknowns			Memory		
Level	Tot.	Ind. Set	Red. Sys.	Ind. Sets	Schur	ILUT
1	5940	3672	2268	265 878	112 115	204 156
2	2269	1064	1204	139 785	60 004	107 782
3	1204	537	667	75 135	33 154	59 022

Table 4: Details of memory requirements in the second set of solutions of the TOKAMAK matrices.

values could be determined from Table 4, but are displayed here explicitly to facilitate comparisons. The Schur complement matrices that need to be stored for ARMS-1 constitute a larger proportion of the total memory requirement as the number of levels grows. The numbers for 3 levels of ARMS-2 are not included because the solution process fails to converge with these low fill levels.

	Individual Factors			Total Memory	
# Levels	Ind. Sets	Schur	ILUT	ARMS-1	ARMS-2
1	265 878	112 115	204 156	582 149	470 034
2	405 663	172 119	107 782	685 564	513 445
3	480 798	205 273	59 022	745 093	

Table 5: Memory used in the second set of solutions of the TOKAMAK matrices.

Convergence is slow for more than one level with these low fill levels. The residual levels as a function of time are shown in Figure 6. The plot on the left shows the 2-norm of the residual as a function of time for ARMS-1, where the Schur complement matrices are stored at each level. The plot on the right is for ARMS-2, where the matrix vector products required at each level for FGMRES are performed using the global matrix and the lower triangular block L matrices. The total time required to converge increases as the number of levels goes up. Note that while time to reach convergence is much greater for 3 levels the outer iteration count for ARMS-1 is lower with 3 levels than with 2 levels. The numbers are 38 and 51, respectively. Comparisons are difficult in ARMS-2 because there is no convergence for more than one level.

The number of iterations on the last level is decreased from 10 to 5 for the second set of tests. It is worth mentioning how this affects the quality of the preconditioners. The number of outer iterations is 40 in each case for ARMS-1, and the total time is practically the same. The situation is different for ARMS-2. The number of outer iterations increases from 54 to 72, but the total solution time drops by more than a third. This illustrates the importance of keeping the number of the inter-level iterations small for ARMS-2. It is also worth noting that ARMS-1 requires fewer outer iterations at this fill level. If the factorization of the original matrix, A, were exact, convergence would be achieved in one iteration. As it is, entries are dropped in the restriction and interpolation matrices, and also in the LU factorization of the last level. When the block LU operators are accurate, a low fill level in the incomplete LU factorization of the last level can be compensated for with an increased inner iteration count. However, when the block LU operators are not accurate, increasing the number of inner iteration is wasteful.

In the next example the fill level, p_0, is increased to provide more accurate block LU operators. The stopping criteria is also adjusted. Table 6 shows parameters common to all trials in this third

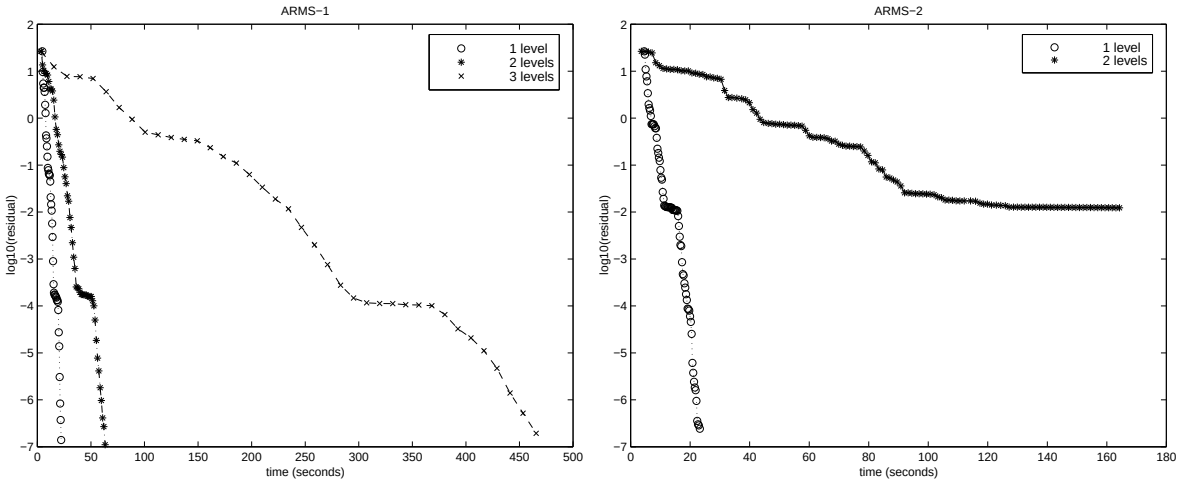

Figure 6: Convergence history for different numbers of levels of ARMS preconditioner solving the matrix UTM.5940 with small levels of fill. The plot on the left is from ARMS-1, where the Schur complement matrices are stored at each level. The plot on the right is from ARMS-2.

set of experiments. There are two main differences between this example and the two previous ones. The fill level, p_0, for these trials is increased to 500 and the number of iterations on the last level is increased to 50. To prevent further iteration when a sufficiently accurate solution of the last system has been computed, GMRES will stop when the 2-norm of the residual has decreased by 10^{-4}.

General		Intermediate Levels		Last Level	
$bsize$	32	$k_{max,i}$	50	$k_{max,N}$	50
τ	0	ϵ_i	10^{-4}	ϵ_N	10^{-4}
$nlev$	varies	p_i	500	p_N	50

Table 6: Parameters used in third set of solutions of the TOKAMAK matrices.

There is a difference in the sizes of the independent sets. Table 7 shows the details. With p_0 10 times as large as for the first two examples, it is expected that the block LU matrices will be larger. The Schur complement matrices are also much denser. This leads to smaller independent sets in the lower levels. The reduced system in the fourth level has 650 unknowns, where in the previous set of experiments, the reduced system in the third level has roughly the same number.

	Unknowns			Memory		
Level	Tot.	Ind. Set	Red. Sys.	Ind. Sets	Schur	ILUT
1	5940	3672	2268	551 589	708 562	209 752
2	2268	590	1678	495 219	665 998	159 131
3	1678	513	1165	798 806	536 849	110 336
4	1165	515	650	667 800	300 254	60 632

Table 7: Memory used in third set of solutions of the TOKAMAK matrices.

The total number of nonzero entries in the preconditioning matrices are contained in Table 8. There is a substantial increase in the memory required at these fill levels. The dense Schur complement matrices also constitute a large portion of the memory for ARMS-1 preconditioners, and make ARMS-2 more attractive if there are memory constraints.

	Individual Factors			Total Memory	
# Levels	Ind. Sets	Schur	ILUT	ARMS-1	ARMS-2
1	551 589	708 562	209 752	1 469 903	761 341
2	1 046 808	1 374 560	159 131	2 580 499	1 205 939
3	1 845 614	1 911 409	110 336	3 867 359	1 955 950
4	2 513 414	2 211 663	60 632	4 785 709	2 574 046

Table 8: Memory used in third set of solutions of the TOKAMAK matrices.

The convergence histories for the third set of experiments are shown in Figure 7. The plot on the left is for ARMS-1 and the plot on the right is for ARMS-2. The different curves are for different numbers of levels. In addition, each symbol corresponds to the conditions at the completion of one outer iteration. The first symbol in each line corresponds to the 2-norm of the initial residual, and also the time required for factorization. Factorization time becomes a factor as the number of levels grows for this high fill level. There are also differences in the relationship between ARMS-1 and ARMS-2. Both versions of the preconditioner are able to promote convergence for up to 4 levels. In practice, however, 4 levels are not practical at this fill level since the factorization time alone is more than total solution time for fewer levels.

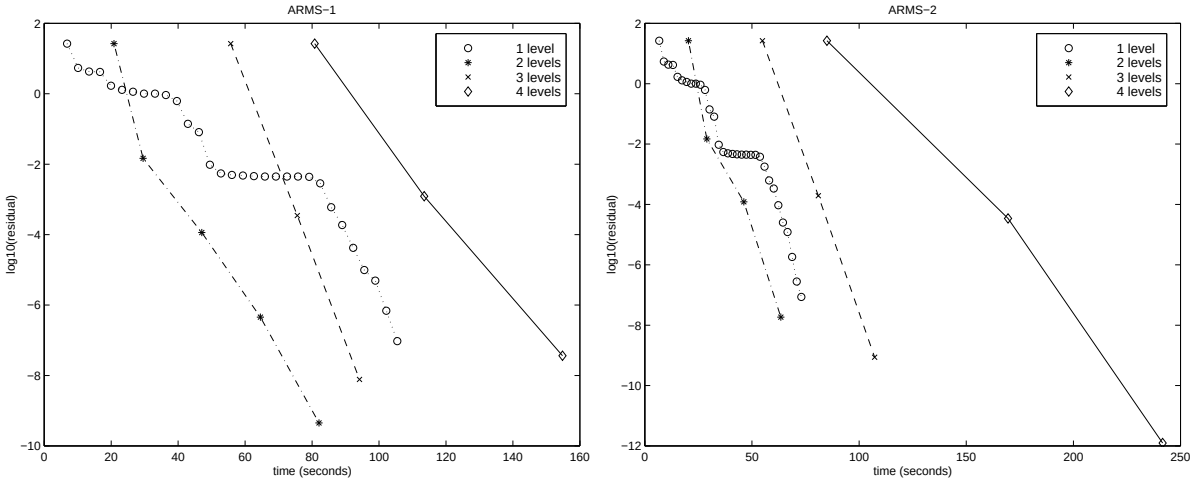

Figure 7: Convergence history for different numbers of levels of ARMS preconditioner solving the matrix UTM.5940 with intermediate fill level, $p_0 = 500$ and low fill, $p_N = 50$ on the last level.

Two levels appears to be optimal for these fill levels. The longer factorization time compared to that for one level is offset by the lower number of outer iterations required to reach convergence. Three and four level preconditioners both require only 3 outer iterations to converge, but the combination of longer factorization times and longer times per outer iteration make for higher total solution times. Also note that in the ARMS-1 tests the 3 level preconditioner is faster than

the single level one, whereas in the ARMS-2 test the one level preconditioner is faster. This is due to the increased cost of matrix vector multiplies in the FGMRES algorithm in ARMS-2. The benefit in the ARMS-2 method is a savings in memory requirements. Similarly, notice that solution times between the two versions do not vary much for 1, 2 or 3 levels, but there is a big difference at 4 levels.

5.2 Tests with the RAEFSKY4 Matrix

The RAEFSKY4 matrix [3] has 19,779 unknowns and 1,328,611 nonzeros. It is from a buckling problem for a container model and was supplied by H. Simon from Lawrence Berkeley National Laboratory (originally created by A. Raefsky from Centric Engineering). This is probably the hardest of the 6 RAEFSKY matrices.

The first set of experiments on the RAEFSKY4 matrix compare a 5 level ARMS preconditioner with no iteration to a couple of ILUT preconditioners. Since there is no iteration within the ARMS preconditioner, no Schur complement matrices are needed and hence, there is no distinction between ARMS-1 and ARMS-2. The parameters used in ARMS are contained in Table 9. The memory requirements are listed in Table 10. The fill levels in ILUT were selected so that the matrices in one of the ILUT preconditioners would occupy more memory than those of the ARMS preconditioner, and those of the other would occupy less.

General		Intermediate Levels		Last Level	
$bsize$	200	$k_{max,i}$	0	$k_{max,N}$	0
τ	0	p_i	100	p_N	100

Table 9: Parameters used in first set of solutions of the RAEFSKY4 matrix.

	Unknowns			Memory		
Method	Tot.	Ind. Set	Red. Sys.	Ind. Sets	ILUT	TOTAL
ILUT(0,100)	19779	0	19779	0	3 632 937	3 632 937
ILUT(0,200)	19779	0	19779	0	6 934 486	6 934 486
ARMS-2	19779	12237	7542	2 488 896	0	
	7542	3716	3826	1 046 361	0	
	3826	2158	1668	559 183	0	
	1668	1123	545	251 411	0	
	545	403	142	83 114	17 146	4 446 111

Table 10: Memory used in first set of solutions of the RAEFSKY4 matrix.

The convergence histories are shown in Figure 8. ARMS is clearly a better preconditioner for this matrix. A GMRES accelerator using the ARMS preconditioner requires 48 outer iterations to converge. The ILUT(0,200) preconditioner requires only 38 iterations. However, it takes more time to create the the ILUT(0,200) preconditioner than the total solution time for the ARMS preconditioner. The ILUT(0,100) preconditioner takes less time than ILUT(0,200) to factor, but

[3] The RAEFSKY matrices are available online from the University of Florida sparse matrix collection [11] at `http://www.cise.ufl.edu/~davis/sparse`.

still more than the total solution time using ARMS. In addition, ILUT(0,100) requires over 600 iterations to reach convergence.

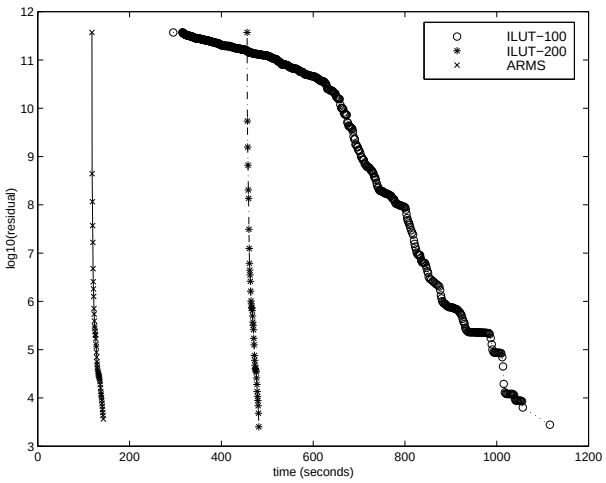

Figure 8: Convergence history of two ILUT and one 5 level ARMS preconditioner for solving the matrix RAEFSKY4.

The second set of experiments on the RAEFSKY4 matrix are designed to investigate the effect of using different numbers of levels of ARMS with iteration on the last level, but not at intermediate levels. Experiments showed that intermediate level iteration is inefficient at these low fill levels for the RAEFSKY4 matrix. ARMS-1, where the reduced systems are stored for each level, is compared with ARMS-2, where the reduced systems are not stored. The preconditioner parameters may be found in Table 11.

General		Intermediate Levels		Last Level	
$bsize$	200	$k_{max,i}$	0	$k_{max,N}$	5
τ	0	p_i	100	p_N	100

Table 11: Parameters used in second set of solutions of the RAEFSKY4 matrix.

The memory requirements for the preconditioners are shown in Table 12. Since there is no iteration at intermediate levels, the only Schur complement matrix stored is the last one. Hence, the difference in storage required for ARMS-1 and ARMS-2 decreases as the number of levels increases. Note that the size of the preconditioner actually decreases for ARMS-1 as the number of levels increases. This is due to a large initial decrease in the number of unknowns in the reduced systems.

The convergence histories of the second set of tests on the RAEFSKY4 matrix are plotted in Figure 9. The 2-norms of the residuals for ARMS-1 are on the left and those for ARMS-2 are on the right. The different symbols correspond to different numbers of levels. At least two levels are needed to yield convergence for ARMS-1. However, a total of five levels are required for ARMS-2. This indicates that the last system, which is solved iteratively, is not constructed adequately by the block LU operators. This problem could be overcome with higher fill levels in the block LU

	Memory Requirements			Total Memory	
# Levels	Ind. Sets	ILUT	Schur Comp.	ARMS-1	ARMS-2
1	2 489 535	1 432 167	753 600	4 675 302	3 921 702
2	3 533 877	728 844	382 900	4 645 621	4 262 721
3	4 065 601	356 211	191 800	4 613 612	4 421 812
4	4 365 036	103 572	62 400	4 531 008	4 468 608
5	4 456 169	26 827	18 600	4 501 596	4 482 996

Table 12: Memory used in second set of solutions of the RAEFSKY4 matrix.

matrices, but that would also increase the factorization time, and the time required for each outer iteration.

The plots in Figure 9 also show that there is very little difference in the time required to construct a preconditioner with 1, 2, or more levels, and that when the solution process converges, it does so quickly, compared with the time required to perform the factorization. The 5 level ARMS-2 preconditioner converges in only 47 iterations, compared with 49 iterations required for the 5 level ARMS-1 preconditioner. However, since each iteration of ARMS-2 is more expensive, due to the Schur Product routine used for the matrix vector product operations on the last level, it takes much longer overall.

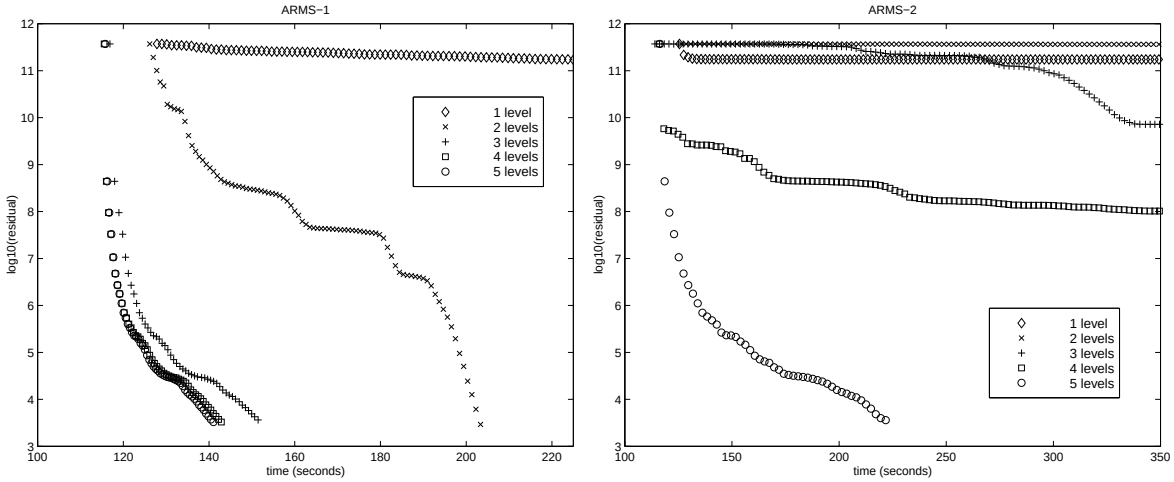

Figure 9: Convergence history of ARMS-1 and ARMS-2 with different numbers of levels for solving the matrix RAEFSKY4.

5.3 Tests with the BARTH Matrices

These matrices, which originate from a 2D high Reynolds number airfoil problem, have been supplied by Tim Barth of NASA Ames [4]. The mesh for the matrices investigated in this paper has a concentration of elements with poor aspect ratios close to the airfoil resulting in ill-conditioned matrices. They both have 14,075 unknowns. The BARTHT1A matrix has 481,125 nonzeros and

[4]The BARTH matrices are available from the authors.

the BARTHT2A matrix investigated in the next section has 1,311,725 nonzeros. These matrices are discussed in [10]. The idea in [10] is to use the ILUT factors of the first order discretiation matrix BARTHT1A to precondition the second order matrix BARTHT2A. This is not done in this paper.

BARTHT1A. The first set of tests compare a number of ARMS preconditioners with no iteration, to ILUT preconditioners with various levels of fill. They show that even a crude ARMS preconditioner performs well on this matrix compared with some other preconditioners. The solution parameters and total memory for each preconditioner are listed in Tables 13 and 14. The ARMS preconditioners have a smaller memory requirement than ILUT. It is not of interest to test ILUT with smaller fill levels because with ILUT(0,100) the solution process requires 381 outer iterations to achieve a reduction in the 2-norm of the residual of 10^{-8}. This number drops to 81 for ILUT(0,200).

General		Intermediate Levels		Last Level	
$bsize$	200	p_i	100	p_N	100

Table 13: Parameters used in first set of solutions of the BARTHT1A matrix.

All three of the ILUT preconditioners tested require more memory than all of the ARMS preconditioners in this test. The memory requirements listed in Table 14 include the sizes of the block LU matrices and the ILUT factorization of the last level.

Method	Total Memory Requirement
ILUT(0,100)	2 763 711
ILUT(0,200)	5 352 233
ILUT(0,300)	7 805 162
1-level ARMS	2 188 913
2-level ARMS	2 300 116
3-level ARMS	2 367 746
4-level ARMS	2 396 588
5-level ARMS	2 408 088

Table 14: Memory used in second set of solutions of the BARTHT1A matrix.

Figure 10 compares the convergence history of the two methods. The plot on the left is the ILUT preconditioner with different levels of fill. Note that ARMS has a much shorter factorization time, and takes fewer iterations to converge. Best performance is achieved for a single level of ARMS. However, performance for higher level ARMS preconditioners are still better than for ILUT. The main advantage in ARMS is the shorter factorization time. All of the ARMS preconditioners converge before any of the ILUT preconditioners are even factored.

The first set of experiments on the BARTHT1A matrix do not involve any intermediate level iteration. The next set of experiments investigate whether iteration on reduced systems can speed up convergence. The same fill levels are used. The second set of tests compare ARMS-1 and ARMS-2 in a V-cycle, where there is iteration at the last level but not at intermediate levels.

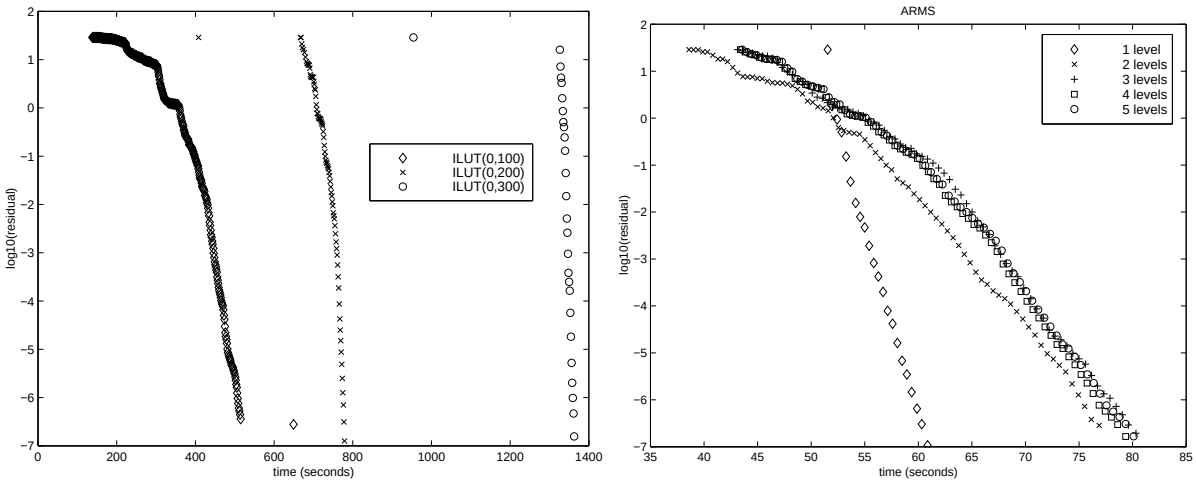

Figure 10: Convergence history of ILUT$(0, p)$ and ARMS with no iteration for solving the matrix BARTHT1A.

The third set of tests compares them when implemented in a W-cycle, where intermediate level iteration is also included in the solver. Solver parameters and memory requirements for the second set of experiments are shown in Tables 15 and 16. When there is no iteration at intermediate levels, the only reduced system that needs to be stored is the last one. This matrix shrinks as the number of levels increases.

General		Intermediate Levels		Last Level	
$bsize$	200	$k_{max,i}$	0	$k_{max,N}$	5
τ	0	p_i	100	p_N	100

Table 15: Parameters used in second set of solutions of the BARTHT1A matrix.

	Memory Requirements			Total Memory	
# Levels	Ind. Sets	ILUT	Schur Comp.	ARMS-1	ARMS-2
1	1 410 769	778 144	427 060	2 615 973	2 189 913
2	1 999 912	300 204	172 800	2 472 916	2 300 116
3	2 238 557	129 189	75 300	2 443 046	2 367 746
4	2 346 530	50 058	32 600	2 429 188	2 396 588
5	2 391 218	16 870	14 700	2 422 788	2 408 088

Table 16: Memory used in second set of solutions of the BARTHT1A matrix.

The convergence histories are shown in Figure 11. The fastest method is ARMS-2 with one level which converges in 8 outer iterations and requires a total time of about 63.7 seconds. The next best time is with ARMS-1 with one level. Each iteration takes less time, but 17 outer iterations are needed for convergence leading to a total time of about 68.5 seconds.

Similar phenomena for the ARMS-2 preconditioner appear in Figures 9 and 11. Both sets of experiments involve preconditioners with iteration at the last level, but not any intermediate level

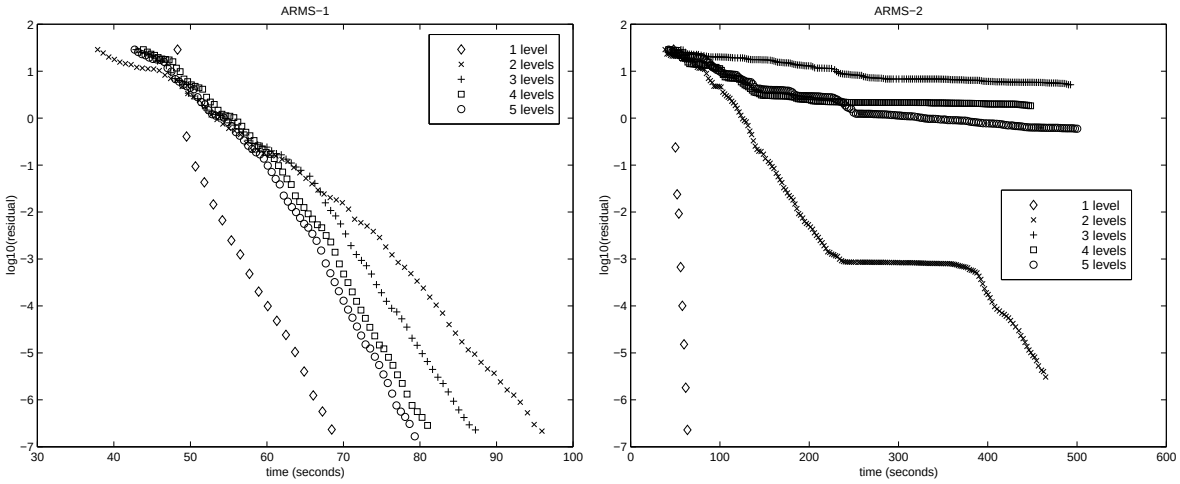

Figure 11: Convergence history of ARMS-1 and ARMS-2 with different numbers of levels for solving the matrix BARTHT1A. Iteration at the last level, but not at any intermediate levels.

iteration. There is a difference, though, in that for the RAEFSKY4 matrix, a high number of levels tends to work, while a low number does not, and the situation is reversed for the BARTHT1A matrix. We do not attempt to explain the phenomena here.

For this matrix, there is no advantage in iterating at intermediate levels, because convergence is reached quickly with only one level of ARMS. In fact, it was found that for a level of fill $p_0 = 100$, intermediate level iteration is quite slow. Intermediate level iteration does decrease the number of outer iterations if the fill level for the grid transfer matrices is increased. The purpose of the third set of experiments on the BARTHT1A matrix is to show that intermediate level iteration can be helpful if the parameter p_0 is set high enough. The full set of parameters used for the third set of tests on the BARTHT1A matrix is displayed in Table 17.

General		Intermediate level		Last Level	
$bsize$	100	$k_{max,i}$	5	$k_{max,N}$	5
τ	0	ϵ_i	10^{-4}	ϵ_N	10^{-4}
p_0	300	p_i	300	p_N	100

Table 17: Parameters used in third set of solutions of the BARTHT1A matrix.

The preconditioners require slightly more memory, but convergence is attained in far fewer outer iterations, sometimes in less time. The memory requirements are shown in Table 16. Note that since iterative solves are performed at each level, the memory requirement for Schur complement systems is higher for ARMS-1 than in the previous set of tests.

It takes very few outer iterations to converge with this relatively high fill level when intermediate level iteration is included in the solution algorithm. The quickest convergence is still for a single level of ARMS, though, because of the extra work involved in intermediate level iteration. The lowest total solution times are lower with this higher fill level, while preconditioners with higher fill levels are not as quick due to increased factorization times. Fine tuning an ARMS

	Memory Requirements			Total Memory	
# Levels	Ind. Sets	ILUT	Schur Comp.	ARMS-1	ARMS-2
1	1 767 389	784 288	798 045	3 349 722	2 551 677
2	2 987 222	391 094	1 398 678	4 776 994	3 378 316
3	3 676 705	182 967	1 688 780	5 548 452	3 859 672
4	4 053 399	71 919	1 812 680	5 937 998	4 125 318

Table 18: Memory used in the third set of solutions of the BARTHT1A matrix.

preconditioner consists of weighing the trade-offs involved between iteration on full and reduced systems.

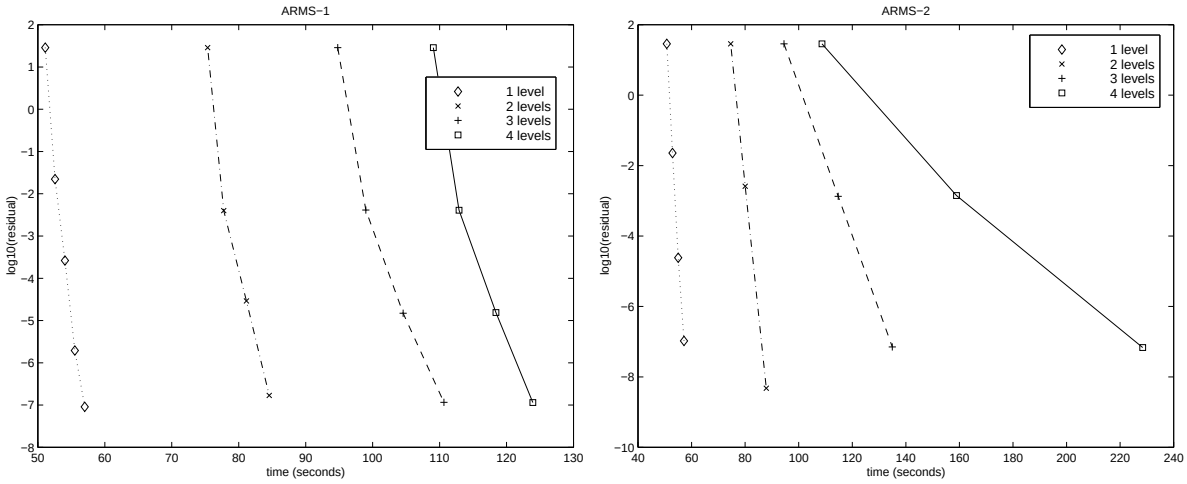

Figure 12: Convergence history of ARMS-1 (left), and ARMS-2 (right) with different numbers of levels for solving the matrix BARTHT1A. GMRES iteration at all levels.

A few remarks are in order for this set of tests. Iterating at intermediate levels is able to decrease the number of outer iterations required. In fact, for a 2, 3 or 4 level ARMS-2 preconditioner, convergence is reached in only two outer iterations. A single level preconditioner is fastest for each method, and both take about the same amount of time, but ARMS-1 takes 4 outer iterations, while ARMS-2 requires only 3. For this higher level of fill, the L U block factors are able to produce a better approximation of the reduced systems actually being solved than the Schur complement systems saved after each level of factorization can. Lastly, note that as few as 5 iterations per level are able to greatly reduce the number of outer iterations needed.

BARTHT2A. The second Barth matrix, BARTHT2A is more difficult to solve than the first one. Higher fill levels are required to get the method to converge in a reasonable number of outer iterations. The first set of experiments compare some ARMS preconditioners using parameters that are not carefully selected, with an ILUT preconditioner with very high fill. Later examples show how performance is enhanced when the parameters are more carefully selected. The parameters used to create the ARMS preconditioners are shown in Table 19. In this first set of tests, the parameter, tol_{ind}, which determines whether or not a node is diagonally dominant enough to be

included in a particular independent set, is set to 0.1. This means that node i may be rejected from an independent set and grouped with the reduced system if the diagonal entry is not large enough compared to the rest of the entries on that row, see Section 4 for details.

General		Last Level	
tol_{ind}	0.1	dim_N	10
$bsize$	400	$k_{max,N}$	10
τ	0	ϵ_N	10^{-15}
p_0	400	p_N	400

Table 19: Parameters used in first set of solutions of the BARTHT2A matrices.

Memory usage is tabulated in Table 20. The ILUT preconditioner needs a high fill level in order to work at all. Details of the sizes of various parts of the ARMS preconditioners are shown in lines 2–5, and total sizes are shown in the last two lines. Note that both of them, while large, are still smaller than what is required in a marginally effective ILUT preconditioner. For these tests, there is no iteration on intermediate levels.

	Unknowns			Memory			
Method	Tot.	Ind. Set	Red. Sys.	Ind. Sets	Schur	ILUT	TOTAL
ILUT(0,1000)	14075	0	14075	0	0	23 210 849	23 210 849
ARMS, level 1	14075	4931	9144	4 620 905	0	0	
ARMS, level 2	9144	2311	6833	3 880 962	0	0	
ARMS, level 3	6833	1358	5475	2 928 035	0	0	
ARMS, level 4	5475	1227	4248	2 483 697	1 699 200	3 190 549	
ARMS-1							18 803 348
ARMS-2							17 104 148

Table 20: Memory used in first set of solutions of the BARTHT2A matrix.

The convergence history is shown in Figure 13. The ILUT preconditioned solver was stopped after 100 iterations. The residual had decreased to close to 10^{-4} after about 20,000 seconds. The system is solved with the two ARMS preconditioners by the time the ILUT preconditioner has been created. The ARMS-1 preconditioner converges in 57 outer iterations and the ARMS-2 preconditioner converges in 76 outer iterations. Total time to convergence is less for the ARMS-1 preconditioner.

In the next set of tests, the parameter tol_{ind} is set to zero. This leads to larger independent sets, or, equivalently, smaller reduced systems. Experimentation leads to the conclusion that fill levels of $p_0 = 400$ and $p_N = 400$, with a block size of $bsize = 400$ form a fairly efficient preconditioner. The preconditioner parameters are tabulated in Table 19. The number of iterations on the last level is 5, while it is 10 in the first set of experiments. Otherwise, there is little change from the previous experiments except for the parameter tol_{ind}.

The memory requirements are also different. Details concerning the sizes of the independent sets are in Table 22 and totals are in Table 23. The numbers of unknowns in the independent sets and reduced systems are almost exactly reversed in the first level.

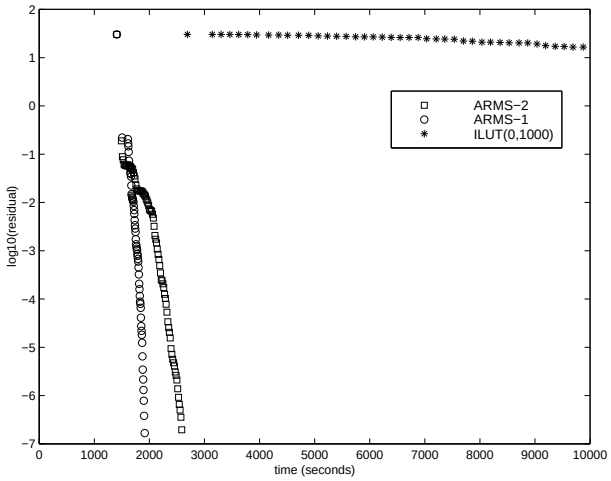

Figure 13: Comparison of convergence history for ARMS and ILUT preconditioners for solving the matrix BARTHT2A.

General		Last Level	
tol_{ind}	0	dim_N	5
$bsize$	400	$k_{max,N}$	5
τ	0	ϵ_N	10^{-15}
p_0	400	p_N	400

Table 21: Parameters used in the second set of solutions of the BARTHT2A matrices.

Comparisons of the convergence histories appear in Figure 14. The parameter tol_{ind} has a tremendous effect on the performance of the preconditioner for this matrix. Factorization times are reduced to about a quarter of those in the first set of tests. The ARMS-2 preconditioner behaves similar to the way it does on the BARTHT1A matrix. Convergence is achieved with few levels, but performance deteriorates as the number of levels increases.

A single level ARMS preconditioner appears to perform best at these fill levels. The last set of tests are performed to try to fine tune the ARMS preconditioners to this particular matrix. Combinations of parameters were tested to find a set that performed well for both ARMS-1 and ARMS-2 with one level. The reasoning behind the selection of parameters that were tested is discussed below.

For many of the more effective preconditioners, the majority of the time is spent in the

	Unknowns			Memory
Level	Tot.	Ind. Set	Red. Sys.	Ind. Sets
1	14075	9845	4230	5 270 399
2	4230	2574	1656	2 319 362
3	1656	922	734	928 495
4	734	407	327	327 487

Table 22: Memory used in second set of solutions of the BARTHT2A matrix.

	Memory Requirements			Total Memory	
# Levels	Ind. Sets	ILUT	Schur Comp.	ARMS-1	ARMS-2
1	5 270 399	2 858 080	1 578 273	9 706 752	8 128 479
2	7 589 761	1 016 846	662 400	9 269 007	8 606 607
3	8 518 256	403 273	293 600	9 215 129	8 921 529
4	8 890 743	106 930	106 929	9 104 602	8 997 673

Table 23: Memory used in the second set of solutions of the BARTHT2A matrix.

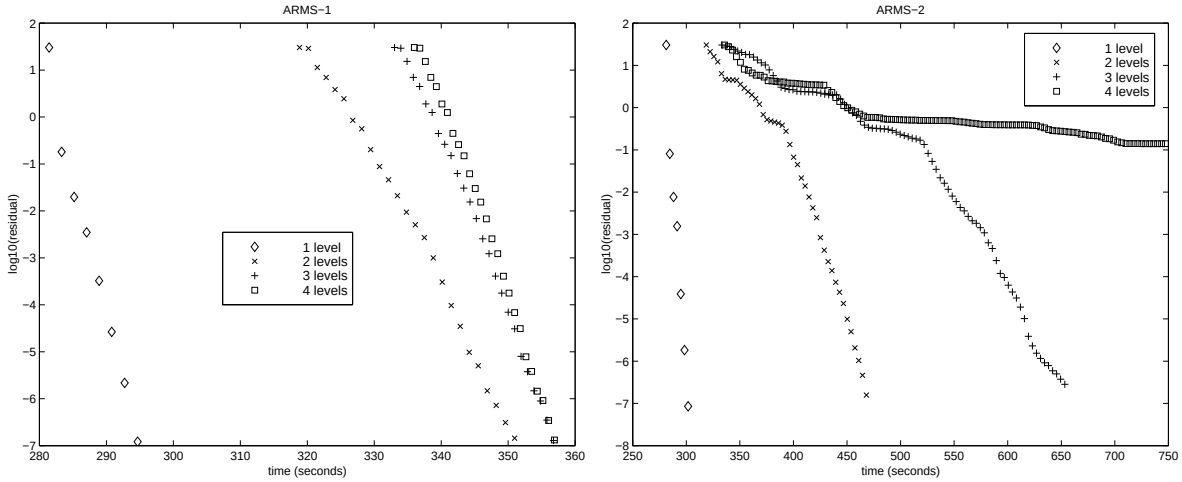

Figure 14: Comparison of convergence history for ARMS-1 and ARMS-2 preconditioners with different numbers of levels for solving the matrix BARTHT2A. There is iteration at the last level, but not at intermediate levels.

factorization phase, with relatively little time spent in the acceleration phase. This can be exploited when the same system must be solved repeatedly, with different right hand sides. However, in situations where the system is solved only once, the area in which there is greatest room for improvement is in the factorization of the preconditioner. The ILUT factorization of the last level is expensive. The cost can be reduced by decreasing the fill on the last level, p_N, while retaining a relatively high fill level for the grid transfer operators, p_0. Some experimentation reveals that the parameters in Table 24 perform quite well.

General		Intermediate Levels		Last Level	
$bsize$	400	$k_{max,i}$	NA	$k_{max,N}$	0–20
τ	0	tol_{ind}	0	ϵ_N	10^{-4}
$nlev$	1	p_i	300	p_N	100

Table 24: Parameters used in third set of solutions of the BARTHT2 matrix.

Factorization yields preconditioners with the sizes listed in Table 25. Note the large number of unknowns lumped into the independent set on account of tol_{ind} being set to 0. The preconditioners are constructed with one level, so there is no intermediate level iteration. The only remaining variable in the solution process is the number of iterations performed on the last level.

	Unknowns			Memory			
Method	Tot.	Ind. Set	Red. Sys.	Ind. Sets	Schur	ILUT	TOTAL
ARMS-1	14075	9845	4230	4 539 490	1 222 417	791 229	6 553 136
ARMS-2	14075	9845	4230	4 539 490	0	791 229	5 330 719

Table 25: Memory used in third set of solutions of the BARTHT2A matrix.

Convergence histories appear in Figure 15. Note that the horizontal axis begins at 150 seconds rather than 0. This is to highlight the differences, since factorization time is the same for all. There are some general differences between the ARMS-1 and ARMS-2 preconditioners. The ARMS-1 method is less expensive per outer iteration due to the way the matrix vector product on the lower level is performed. There is a penalty in memory requirements to be paid for this computational savings, as is seen in Table 25. It is not reflected in the plots, but the minimum number of outer iterations required drops to 9 for ARMS-2, but only down to 11 for ARMS-1, as the number of inner iterations is increased. However, total solution time increases for ARMS-2 more as the number of inner iterations increases. The effects of varying the number of inter-level iterations in each of the two methods is discussed below.

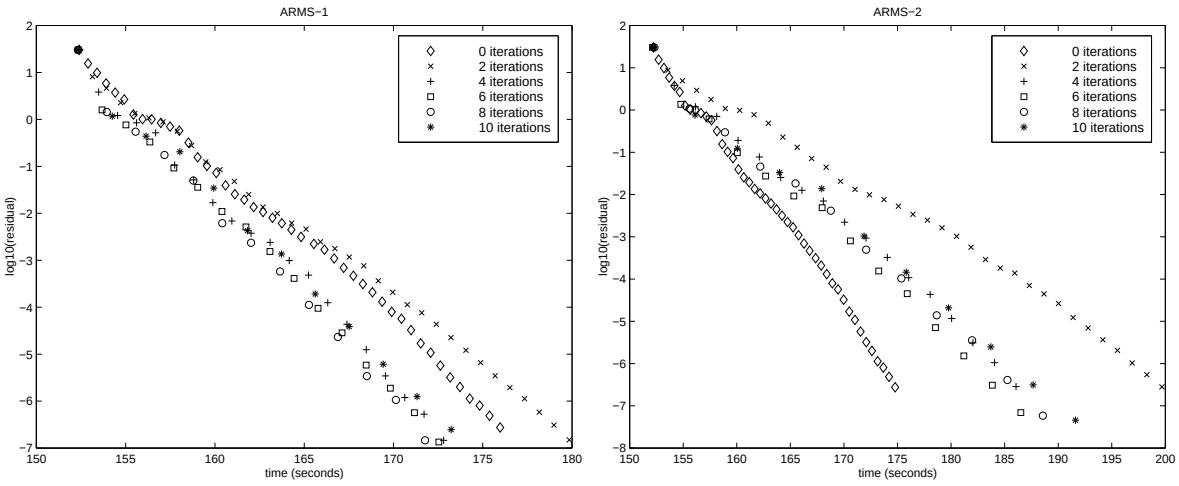

Figure 15: Comparison of convergence history for single level ARMS-1 (left) and ARMS-2 (right) preconditioners with different numbers of iterations on the inter level for solving the matrix BARTHT2A.

The cost of a matrix vector product operation on the reduced system for the ARMS-1 preconditioner is proportional to the size of the Schur complement matrix, $1.2M$. The total cost of a reduced system iteration also includes the preconditioning step, which is an LU forward-backward step using the incomplete LU factorization, for a total of around $2M$. This is much lower than the cost of an ARMS-2 inner iteration. A conservative estimate for the block L factor matrix is $2.4M$. Combined with a cost of $1.3M$ for a full system matrix vector product and the cost of an LU forward-backward step, the total cost of an ARMS-2 inner iteration is more than $4.5M$. The number of iterations performed on the last reduced system is an important consideration. There

is a trade-off between work done towards an accurate solution of the last system, and work done on the outer iterations.

For the BARTHT2A matrix we found that ARMS-1 performed better than ARMS-2. With no iteration on the lower level, 44 GMRES iterations are required on the outer level. Two inner level iterations reduce the number of outer iterations to 34, but increase the total solution time. Four inner iterations reduce the number of outer iterations to 19, and reduce the total solution time to below that required for 0 iterations. Eight inner iterations is optimal, leading to a drop of the number of iterations to 12. More iterations on the lower level only reduces the outer iteration count to 11, but this pushes up the total solution time. Twelve inner level iterations in an ARMS-2 preconditioner can reduce the number of outer iterations to 9, but tests reveal that no number of inner level iterations in an ARMS-2 preconditioner can reduce the total solution time below that for 0 iterations. However, four through fourteen inner level iterations all perform better than two inner level iterations.

6 Conclusion

The ILU multilevel solver presented in this paper works for general sparse matrices and is often able to solve large, unsymmetric matrices when other methods fail. It is competitive with ILUT(τ, p) on relatively small matrices, but is far superior on larger matrices.

The method presented here generalizes BILUTM [20] by exploiting more fully the recursive nature of multilevel block ILU factorizations. In particular, ARMS allows intermediate level iterations. The main benefit of iterating at these levels is to make it possible to obtain an accurate solution to the Schur complement systems with a less accurate factorization. This leads to a less costly factorization, in terms of memory and computation time.

Fine tuning of parameters is required to get the most out of the method. As is often done in typical applications, this is best done by intensive testing on samples of problems from the application and determining what parameters work well. A few general guidelines follow. For some matrices, ARMS appears to work best as a multi-grid type method, with no Krylov subspace iteration at any level. The system is restricted until a small enough problem is reached, then solved approximately on the last level and prolonged. This is the case for the RAEFSKY4 matrix. For other matrices, it appears that using only 1 or 2 levels of ARMS works best, and that it is important to have accurate restriction and prolongation operators - especially if ARMS-2 is being used. In general, it can pay to iterate at last level, but it is not always helpful at other levels.

The method is parallelizable being a form of domain decomposition method and its parallel implementation is currently under way.

7 Conclusion

The ILU multilevel solver presented in this paper works for general sparse matrices and is often able to solve large, unsymmetric matrices when other methods fail. It is competitive with ILUT(τ, p) on relatively small matrices, but is far superior on larger matrices.

The method presented here generalizes BILUTM [20] by exploiting more fully the recursive nature of multilevel block ILU factorizations. In particular, ARMS allows intermediate level iterations. The main benefit of iterating at these levels is to make it possible to obtain an accurate solution to the Schur complement systems with a less accurate factorization. This leads to a less costly factorization, in terms of memory and computational time.

Fine tuning of parameters is required to get the most of the method. A common practice in typical applications is to perform intensive tests on samples of problems from the application and determine what set of parameters work well. A few general guidelines follow. For some matrices, ARMS appears to work best as a multi-grid type method, with no Krylov subspace iteration at any level. The system is restricted until a small enough problem is reached, then solved approximately on the last level and prolonged. This is the case for the RAEFSKY4 matrix. For other matrices, it appears that using only 1 or 2 levels of ARMS works best, and that it is important to have accurate restriction and prolongation operators - especially if ARMS-2 is being used. In general, it can pay to iterate at the last level, but it is not always helpful at other levels.

The method is parallelizable being a form of domain decomposition method and its parallel implementation is currently under way.

References

[1] O. Axelsson. *Iterative Solution Methods.* Cambridge University Press, New York, 1994.

[2] O. Axelsson and M. Larin. An algebraic multilevel iteration method for finite element matrices. *J. Comput. Appl. Math.*, 89:135–153, 1997.

[3] O. Axelsson and P. Vassilevski. Algebraic multilevel preconditioning methods. I. *Numer. Math.*, 56:157–177, 1989.

[4] O. Axelsson and P. Vassilevski. Algebraic multilevel preconditioning methods. II. *SIAM J. Numer. Anal.*, 27(6):1569–1590, December 1990.

[5] R. Bank and C. Wagner. Multilevel ILU decomposition. *Numer. Math.*, 1999. to appear.

[6] E. F. F. Botta, A. van der Ploeg, and F. W. Wubs. A fast linear-system solver for large unstructured problems on a shared-memory computer. In O. Axelsson and B. Polman, editors, *Proceedings of the Conference on Algebraic Multilevel Methods with Applications*, pages 105–116, 1996.

[7] E. F. F. Botta and W. Wubs. MRILU: it's the preconditioning that counts. Technical Report W-9703, Department of Mathematics, University of Groningen, The Netherlands, 1997.

[8] E.F.F. Botta, A. van der Ploeg, and F.W. Wubs. Nested grids ILU-decomposition (NGILU). *J. Comp. Appl. Math.*, 66:515–526, 1996.

[9] Q. Chang, Y.S. Wong, and H. Fu. On the algebraic multigrid method. *J. Comp. Phys.*, 125:279–292, 1996.

[10] A. Chapman, Y. Saad, and L. Wigton. High-order ILU preconditioners for CFD problems. Technical Report UMSI 96/14, Minnesota Supercomputer Institute, 1996.

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

[12] J. Demmel and Li X. Making sparse gaussian elimination scalable by static pivoting. In *Supercomputing 98*, 19998. to appear.

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

[14] W. Hackbusch. *Multi-Grid Methods and Applications*, volume 4 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1985.

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

[16] A. Ruge and K. Stüben. Algebraic multigrid. In S. McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in Applied Mathematics*, chapter 4. SIAM, 1987.

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

[18] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS publishing, New York, 1996.

[19] Y. Saad and J. Zhang. BILUM: Block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems. Technical Report UMSI 97/126, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1997. To appear.

[20] Y. Saad and J. Zhang. BILUTM: A domain-based multi-level block ILUT preconditioner for general sparse matrices. Technical Report UMSI 98/118, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1998. to appear.

[21] C. Wagner. *Introduction to Algebraic Multigrid - Course Notes of an Algebraic Multigrid Course at the University of Heidelberg in the Wintersemester 1998/99.*

[22] C. Wagner, W. Kinzelbach, and G. Wittum. Schur-complement multigrid, a robust method for groundwater flow and transport problems. *Numer. Math.*, 75:523–545, 1997.