NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2002; **9**:359–378 (DOI: 10.1002/nla.279)

ARMS: an algebraic recursive multilevel solver for general sparse linear systems

Y. Saad[1,*,†] and B. Suchomel[2,‡]

[1] *University of Minnesota, 4-192 EE/CS Building, 200 Union Street, S.E., Minneapolis, MN 55455, U.S.A.*
[2] *252 E. Mountain Ave., Suite D; Ft. Collins, CO 80524, U.S.A.*

SUMMARY

This paper presents a general preconditioning method based on a multilevel partial elimination approach. The basic step in constructing the preconditioner is to separate the initial points into two parts. The first part consists of 'block' independent sets, or 'aggregates'. Unknowns of two different aggregates have no coupling between them, but those in the same aggregate may be coupled. The nodes not in the first part constitute what might be called the 'coarse' set. It is natural to call the nodes in the first part 'fine' nodes. The idea of the methods is to form the Schur complement related to the coarse set. This leads to a natural block LU factorization which can be used as a preconditioner for the system. This system is then solved recursively using as preconditioner the factorization that could be obtained from the next level. Iterations between levels are allowed. One interesting aspect of the method is that it provides a common framework for many other techniques. Numerical experiments are reported which indicate that the method can be fairly robust. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: Incomplete LU factorization; ILUT; multilevel ILU preconditioner; Krylov subspace methods; multi-elimination; Recursive solution; Nested dissection; multigrid

1. INTRODUCTION

Multigrid methods are often the preferred iterative techniques used to solve linear systems arising from problems with regular meshes. 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 References [1–10].

* Correspondence to: Y. Saad, University of Minnesota, 4-192 EE/CS Building, 200 Union Street S.E., Minneapolis, MN 55455, U.S.A.
† E-mail: saad@cs.umn.edu
‡ E-mail: suchomel@numericainc.com

Contract grant/sponsor: NSF; contract grant/number: ACI-0000443
Contract grant/sponsor: Minnesota Supercomputer Institute

Received 22 June 1999
Revised 1 April 2001

Algebraic multigrid (AMG) methods were introduced in the 1970s—initially by Ruge and Stuben [11]—to remedy the limitations of multigrid methods. 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.

Recently, a collection of ILU factorizations was introduced in the literature which drew much attention. These methods possess features of multilevel methods as well as some features of ILU factorizations. ILUM [5] is one such approach and recent work by Botta and co-workers [12, 3], and Saad and Zhang [6, 7], indicates that this type of approach can be fairly robust and scale well with problem size, unlike standard ILU preconditioners. The idea was extended to a block version (BILUM) using dense blocks [6] and then this was further extended into BILUTM which treats the diagonal blocks as sparse [7]. Tests in Reference [6] indicate that BILUM is generally more efficient and more robust than a standard ILUT-preconditioned GMRES [13] as well as 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 multilevel parallel ILU preconditioning which can accommodate both fine- and coarse-grain parallelism. The recursive nature of BILUTM is recognized in Reference [7], though recursivity was not implemented. A recursive version of the preconditioning process is discussed in Reference [10]. The main contributions of this paper relative to References [6, 7, 10] are as follows: (1) a fully recursive implementation is introduced; (2) new independent set strategies are introduced including one that is based on the nested dissection (ND) reordering, and (3) numerical experiments on realistic matrices are reported.

The rest of this paper is organized 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, i.e. the reordering strategies used by ARMS, are covered in Section 4. Section 5 is devoted to numerical experiments. Some conclusions are drawn in Section 6.

2. MULTILEVEL ILU PRECONDITIONERS: AN OVERVIEW

Most of the existing algebraic multilevel solution methods start by separating the unknowns of the original system in two sets, one of which is labelled '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 those that are kept when forming a reduced system. The 'fine' points are those eliminated via the independent sets. 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 how they (approximately) solve the resulting Schur complement system obtained by (approximately) eliminating the independent set.

The multilevel ILU preconditioners developed in References [3–7] exploit the property that a set of unknowns that are not coupled to each other can be eliminated simultaneously

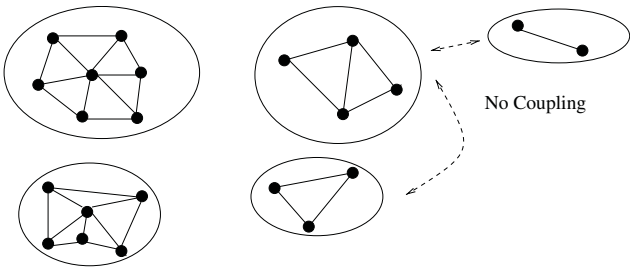

Figure 1. Aggregates, groups or blocks.

in Gaussian elimination. Such sets are termed 'independent sets', see e.g. Reference [14]. In Reference [6], the ILUM factorization described in Reference [5] 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) [6]. 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' [15].[§] Some simple methods for finding standard and block-independent sets have been considered in References [5, 6] and elsewhere. A parallel implementation is described in Reference [16]. In Section 4 we will revisit this issue and other strategies will be examined.

In various existing forms of multilevel ILU factorizations [17, 5, 3, 12] 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 lth level takes the following form:

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

This is then approximated by

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

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

$$\tilde{G}_l \approx E_l U_l^{-1}; \qquad \tilde{W}_l \approx L_l^{-1} F_l \tag{3}$$

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, all 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.

[§]Though 'independent aggregates' would have been more appropriate.

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, the ND method of George and Liu [18] is a method in this class. The ND reordering exploits independent aggregates obtained by recursively dividing the graph into two disconnected subgraphs using 'separators'.

In traditional AMG [11], 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 Reference [19] 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 system is reached, which is then solved either via relaxation or a with direct method.

In ILUM [5], 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 [6], BILUTM [7], 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 [17, 2] 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 [1], couplings are deleted in order to maintain a sparse Schur complement. Another method based on a series of nested finite element grids is NGILU [4]. In this method, the nodes are reordered so that B contains connections between fine mesh nodes that are not in the coarse mesh. The NGILU method breaks down on unstructured grids. In a subsequent method, MRILU [3], two of the authors develop a preconditioner similar to ILUM where B is a diagonal matrix.

In MLILU [20], 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 Reference [21], 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 lth 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. THE ARMS FACTORIZATION

As described above the main factorization step is as follows

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

There are a number of major differences with the BILUTM factorization described in Reference [7]. ARMS relies heavily on recursivity which is exploited to simplify the construction of the factorization as is done in other multilevel codes. This could not be done for BILUTM which was implemented in Fortran77. BILUTM keeps the approximations $\tilde{G}_l$ and $\tilde{W}_l$ in (3) and the intermediate Schur complements. ARMS does not save any of these matrices resulting in substantial memory savings. This is explained in Section 3.2. ARMS allows inter-level iteration. BILUTM can, in principle, also allow inter-level interaction—though this would have required a completely different implementation. Most important, ARMS includes much more elaborate strategies for block independent sets. For example, a ND method has been added as a means of computing block independent sets. In addition, other techniques borrowed from sparse direct solution methods have been included to reorder the nodes of the block-independent set. As always in sparse linear systems solutions, implementation details are crucial and for the reason we devote Section 3.3 to this issue.

The ARMS factorization algorithm consists of a sequence of independent set orderings followed by a reduction and a recursive block factorization on the reduced system.

Algorithm 3.1
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. *Compute the block factorization* (4)
7. *Call ARMS*(A_{lev+1})
8. *EndIf*

A number of additional details will be provided in Section 3.3. In particular, the matrices $\tilde{G}_l$, $\tilde{W}_l$ in (3) which approximate $E_l U_l^{-1}$ and $L_l^{-1}F$, respectively, are approximately computed in order to obtain A_{l+1} but they are not kept. 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. Pivoting is not performed during the block factorization. Some amount of diagonal dominance can be assured as the block independent sets are being selected so small pivots are avoided. Any pivoting that was performed would need to be within individual block independent sets so that the block diagonal structure is not destroyed. This is not a problem on the last level, where a method which includes pivoting, such as ILUTP [13], may be used. The last level system is solved by either a single forward–backward ILUT sweep, or by GMRES preconditioned with ILUT.

3.1. Preconditioning operations

At level l, the linear system $A_l x_l = b_l$ to solve is stored in the permuted form

$$P_l^{\mathrm{T}} \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} P_l \times x_l = b_l$$

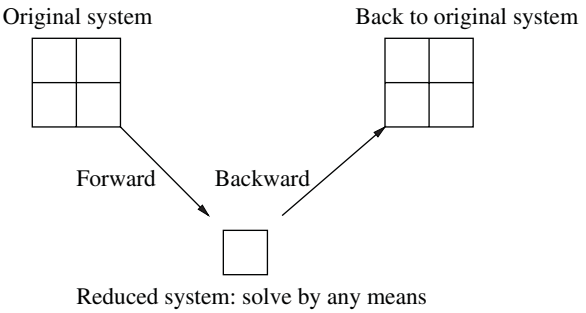

Figure 2. Illustration of a preconditioning operation.

Apart from permutations, the factored form of the system can be written as

$$\begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \times \begin{pmatrix} I & 0 \\ 0 & S_l \end{pmatrix} \times \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & I \end{pmatrix} \times \begin{pmatrix} y_l \\ z_l \end{pmatrix} = \begin{pmatrix} f_l \\ h_l \end{pmatrix} \tag{5}$$

The solution of (5) requires (1) a forward solve followed by (2) a solve with the coarser level matrix S_l, followed by (3) a backward solve. The first and third of these operations move from one level to another and are similar to a restriction or prolongation operation in multigrid techniques. It is helpful to illustrate the process in order to understand the variations that can be obtained by an abstract recursive thinking. The diagram in Figure 2 simply illustrates the point that at any given step the box at the lower level can be any approximate or exact solution technique for solving the system at the next level, i.e., the system

$$S_l \times z_l = h'_l \tag{6}$$

from the forward (restriction) operation. The variations that arise are related to the ways in which this coarser level system is solved. Below are three simple options that come to mind and which we have implemented and tested. There are many other options which we have not explored.

($VARMS$). If the current level is not the last, continue to descend by using the level-structure. When the last level is reached solve with GMRES-ILUT and then ascend back to the current level.
($WARMS$). If the current level is not the last, use a few steps of GMRES to solve the reduced system—utilizing VARMS as a preconditioner. When the last level is reached solve with GMRES-ILUT.
(WARMS*). If the current level is not the last, use a few steps of FGMRES to solve the reduced system—using WARMS* (recursively) as a preconditioner. At the last level ILUT-GMRES is again used.

WARMS* was discussed in the technical report [22]. This algorithm turns out to be fairly expensive when the number of levels exceeds two. Note that in a recursive implementation, WARMS will resort to iterations only at the top level and the last level. This is because WARMS is called only at the top level. At other levels VARMS is called and will descend to the last level where it will iterate as prescribed by the parameters. The VARMS preconditioning step is shown in the following algorithm.

Algorithm 3.2
VARMS-solve(A_l, b_l)*—Recursive Multi-Level Solution*

1. *Solve* $L_l f_l' = f_l$
2. *Descend, i.e., compute* $h_l' := h_l - E_l U_l^{-1} f_l'$
3. *If* $l = last\_lev$ *then*
4. *Solve* $A_{l+1} z_l = h_l'$ *using GMRES + ILU factors*
5. *Else*
6. *Call VARMS-solve*(A_{l+1}, h_l')
7. *Endif*
8. *Ascend, i.e., compute* $f_l'' = f_l' - L_l^{-1} F_l z_l$
9. *Back-Substitute* $y_l = U_l^{-1} f_l''$

Note that in the analogous WARMS-solve algorithm, line 6 would be simply replaced by the line:

6w. Solve $A_{l+1} z_l = h_l'$ using GMRES preconditioned by VARMS$(A_{l+1}, *)$.

3.2. *ARMS-2*

Inner–outer cycles, such as WARMS, require matrix–vector operations with the iteration matrix A_l at level l. One possible way to do this is simply to store the matrix A_l at each level as was discussed in [22]. The permuted form of the Schur complement at level l is the matrix of level $l+1$, i.e.,

$$P_{l+1} S_l P_{l+1}^{\mathrm{T}} = \begin{pmatrix} B_{l+1} & F_{l+1} \\ E_{l+1} & C_{l+1} \end{pmatrix} \tag{7}$$

Since the blocks $B_{l+1}, E_{l+1}, F_{l+1}, C_{l+1}$ are available from the next level, the product $S_l \times w$ may be obtained with two permutations and matrix–vector products with these blocks. On the last level, this cannot be done and the obvious remedy is to store the last Schur complement matrix in order to perform these products. An alternative obviates the need to store this last Schur complement matrix by computing $S_l \times w$ as

$$S_l \times w = (C_l - E_l B_l^{-1} F_l) w \tag{8}$$

which uses matrices from the current level instead of the next one. The inverse of B_l is applied by using its (incomplete) LU factors.

This approach can be used at any level not just the last one. In other words we can use (8) instead of the procedure based on (7) for multiplying S_l by a vector. This approach, referred to as ARMS-2, is currently used in our codes, in contrast with the earlier implementations in References [22, 7]. The two results will of course be (slightly) different in general. An interesting observation is that if B_l is factored exactly, then an algorithm based on formula (8) produces the action of the Schur complement matrix exactly, regardless of dropping in the Schur complement. This suggests that an algorithm based on formula (8) is more accurate than one based on the use of (7) and this is confirmed by the noticeable improvement observed in the quality of the resulting preconditioning operation.

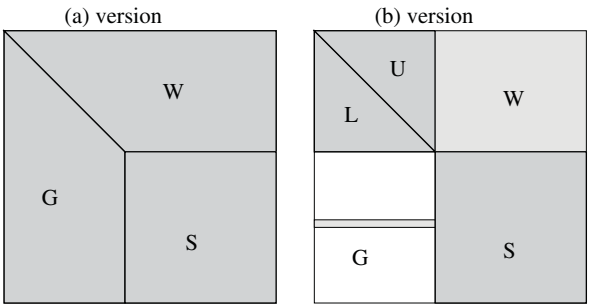

Figure 3. Differences between the old (a) and new (b) versions. In the a version G, W, and S are stored. In the b version L, U and S are stored.

3.3. *A few implementation details*

In References [7, 22] an ILUT-like algorithm was proposed for computing the factorization (4). Let n_B be the dimension of the block B. In words, the method in References [7, 22] performs a variant of the IKJ-version of Gaussian elimination process whereby the index k in the loop runs from 1 to $\min(n_B, i-1)$ (instead of from 1 to $i-1$ as is done in standard Gaussian elimination). As it turns out this results in the factorization (2) and the $(2,2)$ part of the second factor is the desired Schur complement needed for the next level. The new implementation proceeds differently. Referring to (4), we first compute the appropriate factors, $\tilde{L}_l$ and $\tilde{U}_l$, of B_l and an approximation $\tilde{W}_l$, to $\tilde{L}_l^{-1}F_l$. In a second loop an approximation to the Schur complement is computed one row at a time. The l subscripts are dropped to simplify notation.

The differences between the method presented in Reference [22] and the method presented in this paper are displayed schematically in Figure 3. The previous implementation is labeled with an a and the new implementation is labeled with a b. In the a version a row of the upper triangular piece, W, contains entries from both $\tilde{U}$ and $\tilde{L}^{-1}F$. The lower triangular piece G holds $\tilde{L}$ and $E\tilde{U}^{-1}$ explicitly. In the b version dropping is performed separately in the factors $\tilde{L}$, $\tilde{U}$, $\tilde{W}$, $\tilde{G}$ and $\tilde{S}$. The $\tilde{W}$ block is stored temporarily, and then discarded after the Schur complement is computed. Only a single row of $\tilde{G}$ is needed at a time, and it is discarded after the corresponding row of $\tilde{S}$ is computed. Two advantages seen in the b version are lower storage requirements, and the ability to factor $B \approx \tilde{L}\tilde{U}$ accurately without incurring additional costs in $\tilde{G}$ and $\tilde{W}$.

Block factors of matrices used in the preconditioning process are stored in a linked list of C structures [23] or 'structs' which have as members other structs for matrices stored in compressed sparse row (CSR) format. For VARMS, the matrices that are needed at each level are the LU factors of the B matrix along with the blocks F, E. In this case the C blocks are not needed. Only the last one (last last Schur complement) is required in order to be able to compute matrix–vector products via (8). Another struct is required to store the ILUT (or any other factorization such as ILUTP) of the last Schur complement along with the associated C block. In WARMS, the C block is required at each level in order to permit matrix–vector products via (8) (Figure 4).

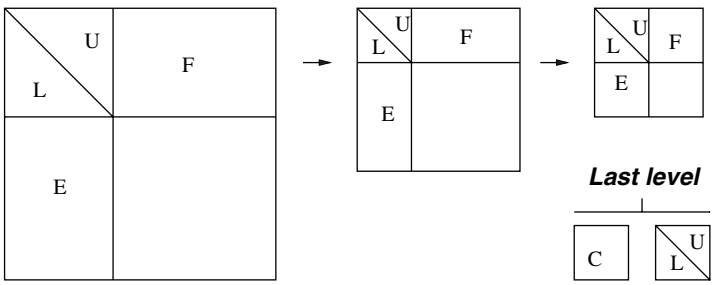

Figure 4. The matrices stored in the VARMS block factorization.

3.4. 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}. This means that there is no dropping in the B-block or the intermediate matrices G_l and W_l used to obtain A_{l+1}. It is also assumed that ARMS-2 is used, meaning that the action of product of the Schur complement by an arbitrary vector is performed via formula (8). 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 solving 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.

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

Therefore, 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 ARMS-2 the solves in lines 4 and 6 of Algorithm 3.2 and its WARMS variant involve an approximation $\tilde{S}_l$ to the Schur complement S_l that is different from the matrix A_{l+1}. We mentioned in the previous section that under the current assumption of exact solves with S_l and no dropping other than in the $(2,2)$ part, the matrix S_l used in (8) is actually the exact Schur complement for the matrix at level l. Thus, under this assumption, a one-level ARMS-2 preconditioner becomes exact.

It is common when analysing block-ILU type preconditioners to make assumptions on the approximation to the Schur complement under consideration [24]. 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

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

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

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

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

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 non-singular an arbitrary eigenvalue λ_i of $S_l A_{l+1}^{-1} = S_l(S_l + R_{22})^{-1}$ is non-zero 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

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

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

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

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 than the tolerance, relative to the norm of the original row. The assumption we make is based on this (though not equivalent to it):

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

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

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

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

Proof
From (13) it follows that

$$\|R_{22}^{\mathrm{T}}\|_1 \leqslant \tau \|S_l^{-\mathrm{T}}\|_1$$

Hence,

$$\|R_{22}^{\mathrm{T}}x\|_1 \leqslant \|R_{22}^{\mathrm{T}}\|_1\|x\|_1 \leqslant \tau\|S_l^{-\mathrm{T}}\|_1\|x\|_1 \tag{15}$$

On the other hand,

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

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

The result of (11) can now be exploited with $\gamma = \tau\kappa_1(S_l^{\mathrm{T}})$. Indeed we mentioned earlier that (11) is also valid for γ defined from (12) instead of (10). This result is pessimistic in that γ is only guaranteed to be less than 1 for values of τ that are less than the condition number of S_l.

4. ARMS REORDERING

The goal of the block-independent set reordering is simply to obtain a matrix of the form (1) where B has a block-diagonal structure. A few strategies for finding block independent sets have been discussed in References [7, 6]. However, many other existing methods can be adapted for this purpose. In fact, all methods that are based on some form of domain decomposition—whether from a graph point of view (ND) or physical point of view (mesh partitioning) can be exploited to provide one level of the ARMS reordering.

The simplest of these techniques consists of a repeated breadth-first search traversal where the algorithm breaks the traversal whenever enough levels are reached to form a subgraph of the desired size, see References [7, 6]. When the algorithm restarts, it takes the next non-visited node and initiates a BFS traversal from it. Nodes that are visited are marked so they will not be visited again in the next expansion. In order to improve the robustness of the factorization the numerical values are also considered. A row is rejected to the complement set whenever it does not show enough diagonal dominance relative to the other rows. To determine which rows to reject, a vector of weights w is computed as follows. First, some diagonal dominance coefficients are computed as

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

Note that $0 \leqslant \hat{w}(i) \leqslant 1$ and that when $a_{ii} \neq 0$ the inverse of $\hat{w}(i)$ is $\hat{w}(i)^{-1} = 1 + \sum_{j\neq i}|a_{ij}/a_{ii}|$. These weights are close to one for strongly diagonally dominant rows, and close to zero for very non-diagonally dominant rows. For matrices that have all of their rows far from being diagonally dominant a strategy based on using the above weights might reject all the rows. 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, we can use instead

$$w(i) = \frac{\hat{w}(i)}{\max_{j=1,\ldots,n}\hat{w}(i)}, \quad i = 1, \ldots, n \tag{17}$$

Variations also exist whereby the weights combine both column and row diagonal dominance ratios.

A second strategy which we have used is based on the class of ND algorithms [18], already mentioned in Section 2. ND algorithms exploit graph separators. A separator in a graph is a set of vertices which splits the graph into two subgraphs such that there is no coupling between nodes of the two subgraphs. The nodes of this separator are labeled last and the process is repeated recursively on each of the two subgraphs until a desirable block-size is reached. This reordering has been used as a means of reducing fill-in in sparse Gaussian elimination, see Reference [18] for details. The problem reduces to that of finding good separators. In recent years much progress has been made in ND reorderings as the idea of separating the graph into two subgraphs became of prime importance for parallel processing. In particular, a nested dissection technique based on the Metis partitioner is available as part of the Metis package [25]. The leaves of the ND tree represent the blocks of the independent set. In more recent fill-reduction strategies, these blocks are also relabelled using a different reordering technique—typically one that is based on a minimum degree-type ordering. The tests shown in the experiments reorder these blocks with the well-known multiple-minimum degree algorithm of George and Liu [26]. We refer to this combination as ND-ARMS. ND-ARMS also incorporates diagonal dominance selection. The weights (17) are computed for all the rows belonging to the leaves of the ND tree. Those rows among this set that do not satisfy the diagonal dominance criterion, are rejected to the complement set (Figure 5).

The above discussion leads to some comments on the relationships between ARMS and sparse direct solvers based on ND. It is clear that one level of the ND-ARMS, without dropping would result in a nested dissection sparse direct solver. The Schur complement resulting from the first level reduction would be basically dense in most cases. However, ARMS resparsifies it by dropping small elements. The resulting system can now be reordered again using ND and the process repeated until the Schur complement is small enough. Thus, ARMS can be viewed as a sort of recursive nested dissection—which can be used only in the presence of dropping.

5. NUMERICAL TESTS

The experiments have been conducted on a PC with two Intel Pentium III processors with a clock speed of 866 MHz and 900 MB of main memory. In the first test a set of matrices is selected and three methods are compared for their overall performance on these matrices. In the second, we illustrate a number of facts using two matrices from Computational Fluid Dynamics.

5.1. Comparison with ILUT and ILUTP

To give a rough idea of how the performance of ARMS compares with that of ILUT and ILUTP, we selected a number of matrices that are known for being relatively difficult to solve with iterative methods and run all three methods under roughly comparable conditions. The matrices selected here are all available from the matrix-market.¶ The matrix names, along with their sizes, number of non-zero elements, and a short description of their origin is given

¶`http://math.nist.gov/MatrixMarket`. The FIDAP matrices in this experiment are the same as those in Reference [27], except that the two smallest matrices have been omitted.

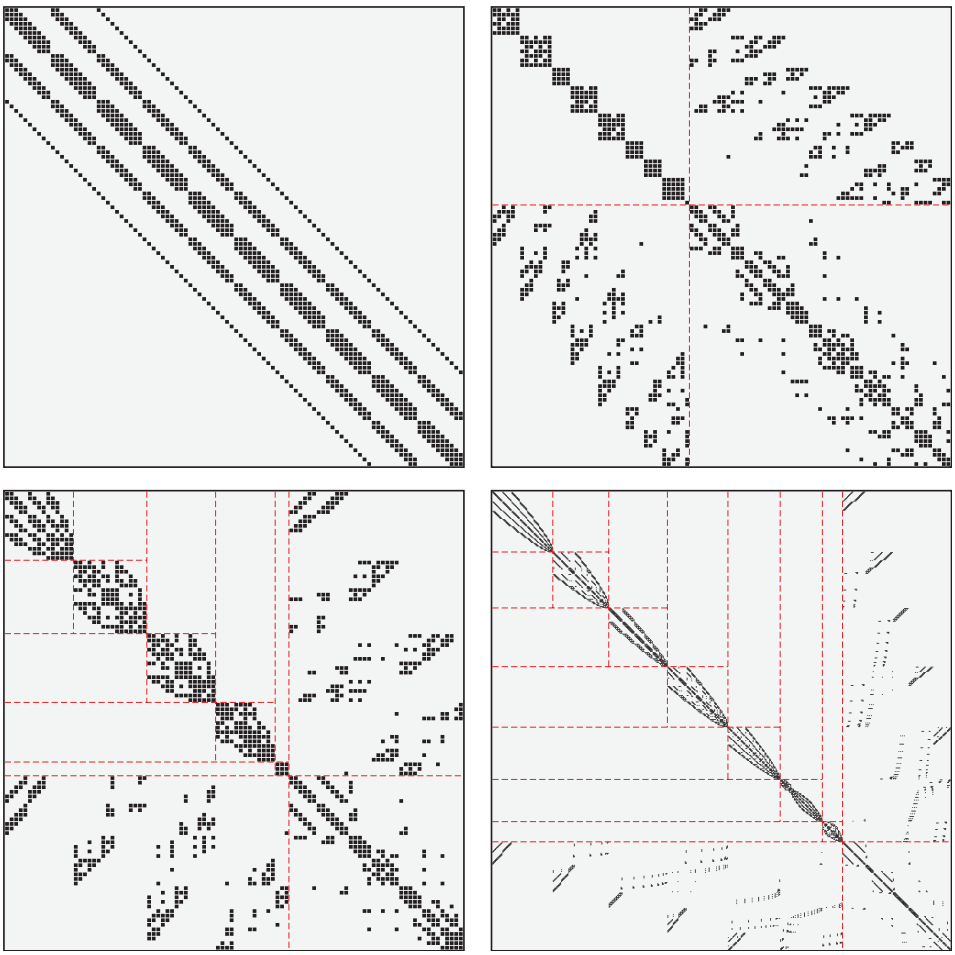

Figure 5. Independent set orderings for 9-point matrices. Top left: original matrix for 10×10 grid. Top right: result using a min block size of 2. Bottom left: result using a min block size of 12. Bottom right: result using a min block size of 50 for a matrix resulting from a 25×20 mesh.

Table I. Parameters for the test in Table III.

bsize	nlev	fill_{I}	$\text{fill}_{\text{last}}$	$\text{fill}_{\text{ILUT}}$	$\text{droptol}_{\text{I}}$	$\text{droptol}_{\text{last}}$	tol_{DD}
500	5	60	50	50	0.0001	0.001	0.2

in Table II. The parameters used for the experiment are listed in Table I. Here, bsize denotes the block-size selected for the block-independent sets and nlev is the maximum number of levels allowed. Recall that the actual block-size itself may differ from the input value of bsize which acts as a threshold: traversal from a point is done by level sets and as soon as the size is above (or equal) bsize, the traversal stops. Similarly, the actual number of levels found may

Table II. A collection of 20 test matrices.

Matrix	*n*	*nnz*	Description
FIDAP04	1601	31850	Hamel flow
FIDAP06	1651	49063	Die swell
FIDAP12	3973	79078	Stokes flow
FIDAP14	3251	65875	Isothermal seepage
FIDAP20	2203	67830	Surface disturbance attenuation
FIDAP23	1409	42761	Fountain flow
FIDAP24	2283	47901	Forward roll coating
FIDAP26	2163	74465	Driven thermal convection
FIDAP28	2603	77031	Two merging liquids
FIDAP31	3909	91223	Dilute species deposition
FIDAP36	3079	53099	Chemical vapor deposition
FIDAP40	7740	456189	3D Die swell
RAEFSKY1	3242	294276	Incompressible flow in pressure driven pipe
RAEFSKY2	3242	294276	Incompressible flow in pressure driven pipe
RAEFSKY3	21200	1488768	Fluid structure interaction turbulence
RAEFSKY4	19779	1328611	Buckling problem for container model
RAEFSKY5	6316	168658	Landing hydrofoil airplane fse model
RAEFSKY6	3402	137845	Slosh tank model
UTM3060	3060	42211	Tokamak plasma reactor model
UTM5940	5940	83842	Tokamak plasma reactor model

differ from the input value of nlev. The subscript 'I' indicates that the parameter is used when generating the intermediate matrices, while 'last' refers to the last level. The drop tolerance indexed with I is for the ILU factorization of each B-matrix at each level as well as for the construction of the temporary matrices $\tilde{G}$, $\tilde{W}$, see, e.g. Equations (3). The fill parameters correspond to the maximum number of non-zeros allowed in each row when constructing the ILU factors for the B matrices at each level as well as for the temporary matrices $\tilde{G}$, $\tilde{W}$. In addition, $\text{fill}_{\text{ILUT}}$ represents the max fill-in allowed in the ILUT factorization of the last Schur complement. Finally, the parameter tol_{DD} is the threshold value used for the relative diagonal dominance criterion discussed in Section 4.

In addition to the above parameters, we should add that the tolerance used for the stopping criterion is $\varepsilon = 1.e-08$, meaning that the (outer) iteration is stopped as soon as the residual norm is reduced by 8 orders of magnitude. This does not mean that the resulting solution is accurate, as some of the matrices are very ill-conditioned. We allow a maximum of 300 iterations. No inner iteration is used, i.e. the last reduced system is solved by a simple backward–forward sweep with the ILUTP preconditioner. In addition, the diagonal scaling option was turned on. This means that the rows of each generated Schur complement are scaled by their 2-norms, then the columns are scaled in a similar way. We use similar parameters and options for ILUT and ILUTP, namely the drop-tolerance parameter for the ILUT/ILUTP factorization is $\text{tol} = 0.001$ and we allow a maximum of 50 fill-ins in the L and U factors. Diagonal scaling was also applied before the ILU factorization is computed. The first measures of performance shown are related to the setup of the preconditioner. We first show a 'fill-factor' (labelled 'Fill' in the tables) which is the ratio of the number of words required to hold the preconditioner over the number of words required to hold the original matrix. Often, users of iterative methods are especially concerned by this last ratio, which ideally should

Table III. Comparison of three methods on a set of 20 matrices.

	V-ARMS				ILUTP				ILUT			
Matrix	Fill	T_{set}	It	T_{it}	Fill	T_{set}	It	T_{it}	Fill	T_{set}	It	T_{it}
FIDAP004	2.95	1.21	13	0.29	4.53	0.54	10	0.19	4.42	0.42	9	0.16
FIDAP006	2.4	1.49	13	0.39	2.69	0.39	35	0.71	2.65	0.32	25	0.49
FIDAP012	3.09	4.42	13	0.71	3.37	0.86	*	*	0.8	0.18	!	!
FIDAP014	2.74	1.78	*	*	3.29	0.84	!	!	1.32	0.3	!	!
FIDAP020	2.05	1.16	7	0.22	2.51	0.6	33	0.92	2.21	0.43	!	!
FIDAP023	2.4	1.41	14	0.28	2.79	0.4	12	0.2	2.66	0.29	!	!
FIDAP024	2.77	1.35	11	0.3	3.62	0.73	20	0.51	1.71	0.14	!	!
FIDAP026	1.58	1.94	*	*	1.85	0.71	116	3.46	1.74	0.58	!	!
FIDAP028	2.46	2.17	14	0.61	2.82	0.67	63	2.04	2.71	0.58	13	0.41
FIDAP031	1.75	2.13	15	0.8	2.71	0.95	15	0.69	2.32	0.63	11	0.46
FIDAP036	2.62	1.59	9	0.31	3.09	0.41	*	*	2.64	0.3	*	*
FIDAP040	2.09	31	40	8.98	1.68	6.37	24	3.14	1.68	5.6	24	3.13
RAEFSKY1	1.37	6.11	26	2.55	1.05	2.75	18	1.13	0.99	2.34	18	1.13
RAEFSKY2	1.49	11.4	30	3.21	1.06	3.8	21	1.41	1.03	3.36	21	1.36
RAEFSKY3	1.31	29.5	8	4.12	0.87	11.3	15	4.81	0.83	9.76	16	5.17
RAEFSKY4	1.82	69.6	31	19.1	1.4	22.6	*	*	1.39	20.5	*	*
RAEFSKY5	1.01	0.42	2	0.13	0.92	0.23	2	0.09	0.81	0.2	2	0.09
RAEFSKY6	0.92	0.44	2	0.09	0.7	0.2	3	0.11	0.6	0.17	3	0.08
UTM3060	3.61	1.81	58	2.15	5.28	0.82	*	*	4.89	0.59	21	0.62
UTM5940	3.23	3.27	67	4.7	5.35	2.2	*	*	5.01	1.71	*	*

not exceed 2 or 3. For harder problems, one should expect this number to be bigger. Then we show the time it takes to compute the factorization for each of the three preconditioners. Results are shown in Table III.

A star indicates non-convergence in 300 steps. An exclamation point indicates a serious arithmetic failure, such as the occurrence of a NAN. Note that the actual number of levels found may differ from the maximum number of levels allowed. Though these are not shown on the table, for reasons of space, the ending level number was 2 in all cases but FIDAP13 (nlev = 1), FIDAP14, FIDAP40, UTM5940 (nlev = 3) RAEFSKY4 (nlev = 4), and RAEFSKY3 (nlev = 5).

Notice from the table that going from left (ARMS) to right (ILUT) the number of failures increases substantially. Also note that the ARMS factorization is generally more costly to compute. The performance of the set-up phase can be improved. However, it is also important to point out that the memory used is often less than with either ILUT or ILUTP.

5.2. *Tests with the BARTH matrices*

These matrices originate from a 2D high Reynolds number airfoil problem and have been supplied by Tim Barth of NASA Ames.[‖] 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

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

Table IV. Parameters for first test with BARTH1 matrix.

bsize	nlev	fill_{I}	$\text{fill}_{\text{last}}$	$\text{fill}_{\text{ILUT}}$	$\text{droptol}_{\text{I}}$	$\text{droptol}_{\text{last}}$	tol_{DD}
500	5	100	100	100	0.0001	0.001	0.05

Table V. Performance measures for the BARTH1 matrix.

Method	Fill	lev	iter	Setup sec	Iters. sec
BFS	3.21	5	21	2.12e + 01	6.34e + 00
NDARMS	3.51	2	56	3.58e + 01	2.21e + 01
ILUT	5.04	0	200*	3.39e + 01	6.29e + 01

Table VI. Parameters for first test with BARTH2 matrix.

bsize	nlev	fill_{I}	$\text{fill}_{\text{last}}$	$\text{fill}_{\text{ILUT}}$	$\text{droptol}_{\text{I}}$	$\text{droptol}_{\text{last}}$	tol_{DD}
500	5	200	200	200	0.0001	0.0001	0.05

481,125 non-zeros and the BARTHT2A matrix which uses a higher order discretization has has 1,311,725 non-zeros. These matrices are discussed in Reference [28]. Originally, the ILUT factors of the first order discretization matrix BARTHT1A were used to precondition the second order matrix BARTHT2A, see Reference [28] for details. In this paper we simply solve linear systems formed from both matrices independently.

We compare two ARMS preconditioners with no inner iteration, with the ILUT preconditioner. Various levels of fill are used. In our first experiment, we used the parameter shown in Table IV. Three methods are compared in this example. First, is VARMS with the simple BFS strategy for building independent sets. Then, we run the ND based ARMS. This algorithm uses the same strategy for rejecting the rows that are the least diagonally dominant. Three measures of performance are shown. The most important ones are the execution times and the memory used. We show the times required to set-up the preconditioners, then the time required for GMRES(40) to converge as well as the corresponding number of GMRES iterations. The convergence tolerance was 1.e-09, meaning that iteration is stopped as soon as the residual norm is reduced by nine orders of magnitude. The results are shown in Table V. A star indicates non-convergence in 200 steps. The column lev shows the actual number of levels found in the factorization.

A similar test was done with the matrix BARTH2. Since this matrix is slightly more ill-conditioned, we modified the parameters in order to obtain a more accurate factorization (the parameters used in Table VI). In addition, diagonal scaling was turned on. The rows of each generated Schur complement are scaled by their 2-norms, then the columns are scaled in a similar way. Note that the limit of 200 for fill-in per row is used as an upper limit and is not indicative of the number of non-zero elements kept, which is also controlled by the droptol parameter. The results are shown in Table VII. It has been our general observation that, as is shown in these two tests, ND does not provide a better reordering than the simpler multiple-BFS strategy.

Table VII. Performance measures for the BARTH2 matrix.

Method	Fill	lev	iter	Setup sec	Iters. sec
BFS–ARMS	2.25	4	21	1.06e + 02	1.47e + 01
ND–ARMS	2.32	3	47	8.80e + 01	3.07e + 01
ILUT	3.82	0	200*	3.05e + 02	1.43e + 02

Table VIII. Parameters for the inner–outer iteration test with BARTH1 matrix.

bsize	nlev	fill_{I}	$\text{fill}_{\text{last}}$	$\text{fill}_{\text{ILUT}}$	$\text{droptol}_{\text{I}}$	$\text{droptol}_{\text{last}}$	tol_{DD}
1000	1	40	20	20	0.0001	0.01	0.05

An interesting experiment consists of exploring the effect of the parameter tol_{DD} on the overall performance. This was done for the standard BFS variant. Without going into the details, we note the general observation that the performance is often poorer for $\text{tol}_{\text{DD}} = 0$ and then improves—sometimes substantially.

To give a rough idea of a comparison with a direct solver, the SuperLU code [29] with approximate minimum degree (APM) required 6.203 M of memory words to factor the BARTH1A matrix, resulting in a fill-factor of 12.89. For the BARTH2A matrix, 13.07 M words are needed resulting in a fill-factor of about 10. The factorization times on the same machine are 9.92 and 44.2 s, respectively. It is worth noting that sparse direct solution codes usually exploit dense computations (similar rows are treated as one unit leading to block computations) better than iterative codes, so the raw speeds are usually much better. It has been our experience that for 2D problems such as the one tested here, the gains in execution time, if any, are often minor. Gains in memory can be more convincing, and perhaps overwhelming for large 3D problems.

5.3. Inner iterations at last level

We now consider again the matrix BARTH1A. The goal of this experiment is to illustrate the flexibility of the ARMS framework. We will show how ARMS can be adapted to solve a harder problem using less memory. The parameters used above seem to indicate that it would be hard to solve a system with the matrix BARTH1A with a preconditioner requiring less than 3.2 times the memory size of the original matrix. Our tests confirm that this is more or less the case. However, this is only if no iteration is performed at the intermediate levels or the last level. In the next experiment we will reduce the memory requirement substantially and see if the system can be solved using inner iterations.

In the experiment we use only one level. In order to be close to the situation of Section 3.4, we select a set of parameters that will produce an accurate ILU factorization of B, but a rough Shur complement factorization. Specifically, the table of parameters in Table IV is replaced by the one in Table VIII.

The last reduced system is now solved iteratively with GMRES preconditioned with ILUTP. The GMRES iteration is not restarted and takes two parameters: a maximum number of steps (which is also the dimension of the Krylov subspace) and a tolerance with which to solve the systems. We set the tolerance to $\varepsilon = 0.001$ (iteration is stopped when residual norm is

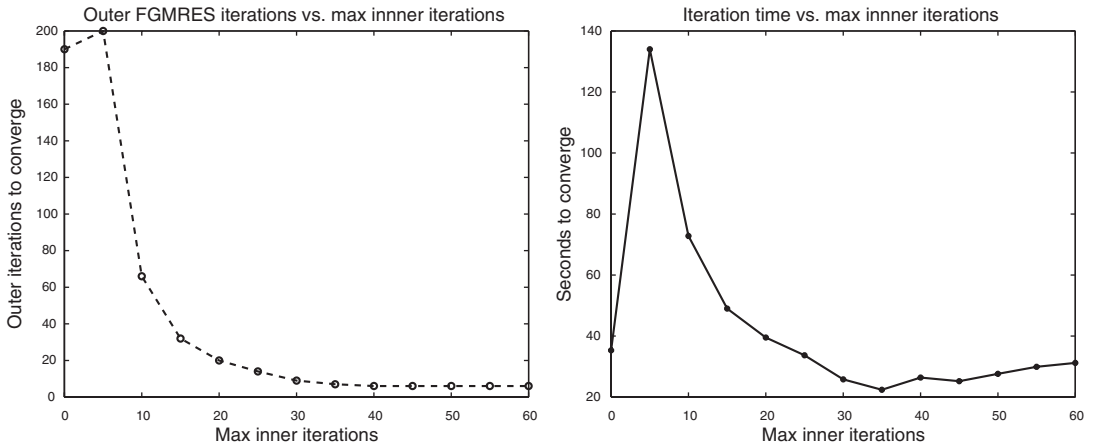

Figure 6. Convergence of inner-outer FGMRES/VARMS as a function of maximum number of inner iterations.

reduced by ε and varied the parameter $\mathrm{im}_{\max}$ from 0 (no iteration) to 60, with an increment of 5. The resulting number of steps and the time required to converge are shown in Figure 6.

The Fill-factor achieved with these parameters is 1.62. At the beginning, i.e., for $\mathrm{im} = 0$ and $\mathrm{im} = 5$, convergence is difficult to achieve. Surprisingly enough, $\mathrm{im} = 0$ works better than $\mathrm{im} = 5$. As im increases, performance increases substantially until a level is reached when the additional inner iterations are no longer cost-effective. If we were to reduce the drop-tolerance for the intermediate levels from 0.0001 to an even lower value we will be able to reduce the number of outer iterations below the limit of seven that has been reached. In words, it is not worth solving the last reduced system with an accuracy that is higher than the level of tolerance with which this Schur complement was computed.

6. CONCLUSION

The method presented here generalizes BILUTM [7] and existing related multilevel ILU strategies [4, 12, 3, 5, 22, 6] by allowing a whole array of additional possibilities within the same framework. In particular, by better exploiting the recursive nature of multilevel block ILU factorizations, it is possible to devise many inner–outer iterations for solving the inter-level or last-level systems. 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. Numerical experiments show that the strategies for selecting the block-independent set are very important. For example, a simple technique for rejecting rows from the *B*-set based on a rather trivial diagonal dominance criterion has been shown to make a big difference in performance. The close relationship discussed in Section b4 between ARMS and the ND-based sparse direct solvers, suggests that there is a potential to build far more robust iterative solvers than what is currently available, by exploiting the best techniques of both worlds.

ACKNOWLEDGEMENTS

This work benefited from the experience and the codes gathered from the earlier paper [7]. Zhongze Li helped fix numerous bugs and provided invaluable assistance with recent versions of the ARMS code. The Minnesota Supercomputing Institute provided the computing resources and an excellent working environment to conduct this research. This work was supported in part by NSF/ACI-0000443, and in part by the Minnesota Supercomputer Institute.

REFERENCES

1. Axelsson O, Larin M. An algebraic multilevel iteration method for finite element matrices. *Journal of Computational and Applied Mathematics* 1997; **89**:135–153.
2. Axelsson O, Vassilevski P. Algebraic multilevel preconditioning methods. II. *SIAM Journal of Numerical Analysis* 1990; **27**(6):1569–1590.
3. Botta EFF Wubs FW. MRILU: it's the preconditioning that counts. *Technical Report W-9703*, Department of Mathematics, University of Groningen, The Netherlands, 1997.
4. Botta EFF, van der Ploeg A, Wubs FW. Nested grids ILU-decomposition (NGILU). *Journal of Computational and Applied Mathematics* 1996; **66**:515–526.
5. Saad Y. ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing* 1996; **17**(4):830–847.
6. Saad Y, Zhang J. BILUM: Block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing* 1999; **20**:2103–2121.
7. Saad Y, Zhang J. BILUTM: A domain-based multi-level block ILUT preconditioner for general sparse matrices. *SIAM Journal on Matrix Analysis and Applications* 2000; 21: to appear in SIMAX.
8. Zhang J. A grid based multilevel incomplete LU factorization preconditioning technique for general sparse matrices. *Applied Mathematics and Computation* 2001; **124**(1):95–115.
9. Zhang J. A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices. *SIAM Journal on Matrix Analysis and Applications* 2000; **22**(3):925–947.
10. Zhang J. A class of multilevel recursive incomplete lu preconditioning techniques. *Korean Journal of Computational and Applied Mathematics*, 2002, to appear.
11. Ruge A, Stüben K. Algebraic multigrid. In *Multigrid Methods*, McCormick S (ed.). *Frontiers in Applied Mathematics*, Vol. 3, Chapter 4. SIAM: Philadelphia, 1987.
12. Botta EFF, van der Ploeg A, Wubs FW. A fast linear-system solver for large unstructured problems on a shared-memory computer. In *Proceedings of the Conference on Algebraic Multilevel Methods with Applications*, Axelsson O, Polman B (eds). 1996; 105–116.
13. Saad Y. *Iterative Methods for Sparse Linear Systems*. PWS publishing: New York, 1996.
14. Leuze R. Independent set orderings for parallel matrix factorizations by Gaussian elimination. *Parallel Computing* 1989; **10**:177–191.
15. Wagner C. *Introduction to Algebraic Multigrid—Course Notes of an Algebraic Multigrid Course at the University of Heidelberg in the Wintersemester*, 1998/99.
16. Adams LM. Iterative algorithms for large sparse linear systems on parallel computers. *Ph.D. Thesis*, Applied Mathematics, University of Virginia, Charlottsville, VA, 1982. Also NASA Contractor Report 166027.
17. Axelsson O, Vassilevski P. Algebraic multilevel preconditioning methods. I. *Numerische Mathematik* 1989; **56**:157–177.
18. George JA, Liu JW. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall: Englewood Cliffs, NJ, 1981.
19. Chang Q, Wong YS, Fu H. On the algebraic multigrid method. *Journal of Computational Physics* 1996; **125**: 279–292.
20. Bank R, Wagner C. Multilevel ILU decomposition. *Numerische Mathematik* 1999, to appear.
21. Wagner C, Kinzelbach W, Wittum G. Schur-complement multigrid, a robust method for groundwater flow and transport problems. *Numerische Mathematik* 1997; **75**:523–545.
22. Saad Y, Suchomel B. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. *Technical Report umsi-99-107*, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1999.
23. Schildt H. *C*: *The Complete Reference*. McGraw Hill, Berkeley, California, 1987.
24. Axelsson O. *Iterative Solution Methods*. Cambridge University Press: New York, 1994.
25. Karypis F, Kumar V. A fast and high-quality multi-level scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing* 1998; **20**:359–392.
26. Joseph W-H Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software* 1985; **11**:141–153.

27. Chow E, Saad Y. Approximate inverse techniques for block-partitioned matrices. *SIAM Journal on Scientific Computing* 1997; **18**:1657–1675.
28. Chapman A, Saad Y, Wigton L. High-order ILU preconditioners for CFD problems. Technical Report umsi-96-14, Minnesota Supercomputer Institute, 1996. *International Journal for Numerical Methods in Fluids* 2000; **33**:767–788.
29. Demmel JW, Eisenstat SC, Gilbert JR, Li XS, Liu JWH. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications* 1999; **20**:720–755.

