Adapting Algebraic Recursive Multilevel Solvers (ARMS) for solving CFD problems

Yousef Saad[*] Azzeddine Soulaimani [†] Ridha Touihri[†]

Abstract

This paper presents results using preconditioners that are based on a number of variations of the Algebraic Recursive Multilevel Solver (ARMS). ARMS is a recursive block ILU factorization based on a multilevel approach. Variations presented in this paper include approaches which incorporate inner iterations, and methods based on standard reordering techniques. Numerical tests are presented for three-dimensional incompressible, compressible and magneto-hydrodynamic (MHD) problems.

Key words: GMRES, FGMRES, ARMS, ILUT, incompressible, compressible and MHD flows.

1 Introduction

Selecting a suitable solver for handling the linear systems in realistic applications can be a difficult task. On one side of the spectrum of possible choices lies a large collection of special-purpose preconditioners which are tailored to the physical problem at hand. Though these techniques may be vastly superior to contending methods that attempt to be general-purpose, their main weakness is that they are unlikely to be useful for solving other problems with different characteristics. On the other extreme of the spectrum, lies the well-developed class of sparse direct solvers which are the most general and reliable techniques available. It is clear that from the point of view of generality and reliability, direct methods, are the most desirable. However, these methods are prohibitive for realistic 3-dimensional problems. A middle-ground solution is to develop methods that attempt to imitate the generality and robustness of sparse direct solvers. In this regard, ILU-type preconditioners constitute the most popular choice. In this paper we further explore a class of methods in this category, which lies in between multilevel techniques and ILU-type preconditioners.

Multilevel methods can be extremely efficient for certain classes of problems. In this category, multigrid techniques can be viewed as "specialized" since their success is narrowly limited and contingent upon tuning of various parameters and operator choices. The Algebraic MultiGrid (AMG) methods, introduced in the seventies to remedy these limitations, offer an alternative whose success has been mixed since it still depends on the underlying PDE problem. In recent years, a number of preconditioners were introduced in the literature which attempt to combine attributes of the standard

[*]Department of Computer Science and Engineering, University of Minnesota,4-192EE/CSci Building, 200 Union Street S.E., Minneapolis, MN 55455

[†]Département de Génie Mécanique, École de technologie supérieure, 1100 Notre-Dame Ouest, Montréal (Québec), H3C 1K3, Canada email: asoulaimani@mec.etsmtl.ca

incomplete LU factorizations and the algebraic multigrid method. These methods possess features of multilevel methods as well as some features of ILU factorizations. ILUM [26] is one such approach and related work by Botta and co-workers [4, 5], and [30, 31], indicates that this type of approach can be fairly robust and scales well with problem size, in contrast with standard ILU preconditioners. The idea was extended to a block version (BILUM) using dense blocks [30] and then this was further extended into BILUTM [31] which treats the diagonal blocks as sparse. Later BILUTM gave rise to the Algebraic Recursive Multilevel Solver (ARMS) [29], which offers more capabilities and a truly recursive implementation in the C programming language. On parallel platforms ARMS provides a general framework for multilevel parallel ILU preconditioning which can accommodate both fine-grain and coarse-grain parallelism [21].

This paper presents a number of variants of ARMS and illustrates their performance on realistic problems from Computational Fluid Dynamics. Two types of variations are explored. First, various inner-outer combinations are considered, which consist of using Krylov accelerations at some of the levels. Second, we introduce techniques which exploit existing reordering strategies, such as Nested Dissection, in the framework of ARMS. One observation we make in this paper is that the separation of the nodes used to define levels can be quite general. One possibility, for example, is to use a simple criterion of diagonal dominance. The performance of these variations is illustrated in the section on numerical experiments.

2 Background on preconditioned Krylov subspace methods

A preconditioned Krylov subspace method for solving the linear system

$$Ax = b \tag{2.1}$$

consists of an accelerator and a preconditioner. In what follows we call M the preconditioning matrix, so that, for example, the left-preconditioned system

$$M^{-1}Ax = M^{-1}b \tag{2.2}$$

or its right-preconditioned version,

$$AM^{-1}y = b \qquad x = M^{-1}y \tag{2.3}$$

is solved instead of the original system (2.1). The above systems are solved via an "accelerator", a term used to include a number of methods of the Krylov subspace class. We only briefly describe one such method here, namely the right-preconditioned GMRES algorithm [27, 28]. Other techniques and details can be found in [27], for example.

Algorithm **2.1** *Right preconditioned GMRES(m)*

```
1.  Compute $r_0 = b - Ax_0$, $\beta := ||r_0||_2$, $v_1 := r_0/\beta$.
2.  Define the $(m+1) \times m$ matrix $\overline{H}_m = \{h_{i,j}\}_{1 \le i \le m+1, 1 \le j \le m}$. Set $\overline{H}_m = 0$.
3.  For $j = 1, 2, ..., m$ Do:
4.      Compute $z := M^{-1}v_j$ and $w := Az$
5.      For $i = 1, ..., j$ Do:
6.          $h_{ij} := (w, v_i)$
7.          $w := w - h_{ij}v_i$
8.      EndDo
```

```
9.          $h_{j+1,j} = ||w||_2$.
10.         If $h_{j+1,j} = 0$ set $m := j$ and goto 12.
11.         $v_{j+1} := w/h_{j+1,j}$
12.    EndDo
13.    Compute $y_m$ the minimizer of $||\beta e_1 - \overline{H}_m y||_2$ and
14.    Compute $x_m = x_0 + M^{-1} V_m y_m$, where $V_m \equiv [v_1, v_2, \cdots, v_m]$
15.    If satisfied Stop, else set $x_0 := x_m$ goto 1.
```

In the above algorithm, m represents the dimension of the Krylov subspace, i.e., the number of directions used before restarts.

A variation of the above algorithm is available which is motivated by the desire to use a preconditioner M which is itself an iterative solver. This is particularly appealing for domain decomposition methods in parallel environments where a subdomain solve can be itself iterative. In this case, the preconditioning operation in Line of Algorithm 2.1 does not correspond to a matrix solve with the same matrix M. It can be viewed as a solve with a matrix M_j, which varies with the step number j. In this case, the algorithm as it is described above will fail. However, a variation, referred to as FGMRES (Flexible GMRES), has been developed to handle this situation. This variation consists of simply changing Lines 4 and 14 into the following two lines, leaving the rest of the algorithm unchanged:

```
4a.     Compute $z_j := M_j^{-1} v_j$ and $w := A z_j$
14a.    Compute $x_m = x_0 + Z_m y_m$, where $Z_m \equiv [z_1, z_2, \cdots, z_m]$
```

Note that FGMRES requires to save an extra set of vectors, the z_i's, so the storage required by the accelerator is nearly doubled.

A common way to define a preconditioner M is through an Incomplete LU factorization obtained from an approximate Gaussian elimination process. When Gaussian elimination is applied to a sparse matrix A, a large number of nonzero elements may appear in locations originally occupied by zero elements. These 'fill-ins' often have small values and therefore they can be dropped to obtain a sparse factorization, referred to as an Incomplete LU factorization. The simplest of these procedures, ILU(0) is obtained by performing the standard LU factorization of A and dropping all fill-in elements that are generated during the process. In other words, the L and U factors have the same pattern as the lower and upper triangular parts of A (respectively).

More accurate factorizations denoted by ILU(k) and IC (k) (symmetric case) have been defined which drop fill-ins according to their 'levels'. Level-1 fill-ins for example are generated from level-zero fill-ins (at most). So, for example, ILU(1) consists of keeping all fill-ins that have level zero or one and dropping any fill-in whose level is higher.

Another class of preconditioners is based on dropping fill-ins according to their numerical values. One of these methods is ILUT (ILU with Threshold). This procedure uses basically a form of Gaussian elimination which generates the rows of L and U one by one. Small values are dropped during the elimination, using a parameter τ. A second parameter, p, is then used to keep the largest p entries in each of the rows of L and U. This method is denoted by $ILUT(\tau, p)$.

3 Multilevel ILU preconditioners and ARMS

Multi-level ILU preconditioners exploit the property that a set of unknowns that are not coupled to each other can be eliminated simultaneously in a Gaussian elimination type process. These *independent sets* have been extensively used in devising both direct and iterative solution methods for sparse linear systems, see, for example, [16, 23, 20, 8, 26]. Independent set orderings transform the original linear system into the form

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{3.4}$$

in which the B block is diagonal. Since B is diagonal, it is easy to eliminate the u variable to obtain a system with only the y variable. The coefficient matrix for this 'reduced system' is the Schur complement $S = C - EB^{-1}F$ which is still sparse. In ILUM[26], this idea was used recursively. Dropping is applied to S to limit fill-in and an independent set is found to put the resulting reduced system into the above form. This is repeated for a few levels until the system is small enough, or a maximum number of levels is reached. Then the system is solved by some other, standard, means such as an ILUT-GMRES combination.

The concept of independent set has been generalized to *blocks* or *groups* [30, 31]. A group-independent set is a collection of subsets (blocks) of unknowns such that there is no coupling between unknowns of any two different groups (blocks) [30]. Unknowns within the same group (block) may be coupled. This is illustrated in Figure 1. If the unknowns are permuted such that those associated

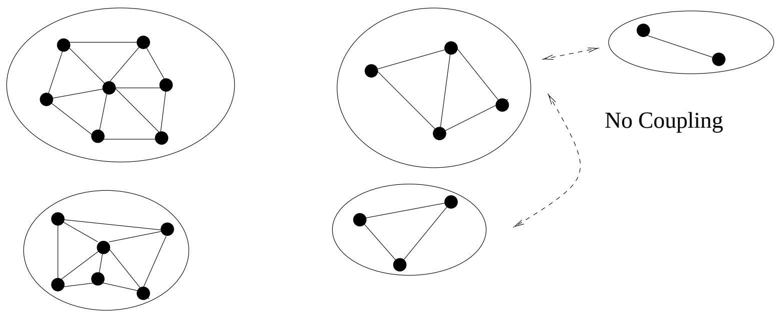


Figure 1: Independent groups or blocks.

with the group-independent set are listed first, followed by the other unknowns, the original coefficient system will take the form (3.4) where now the matrix is not diagonal but block diagonal. An illustration is shown in Figure 2 which shows two block independent reorderings of the same matrix.

ARMS exploits block independent sets and implements a multilevel ILU which generalizes ILUM. At the l-th level of ARMS, the coefficient matrix is reordered as in (3.4) where B is now diagonal and then the following block factorization is computed 'approximately' (subscripts corresponding to level numbers are introduced):

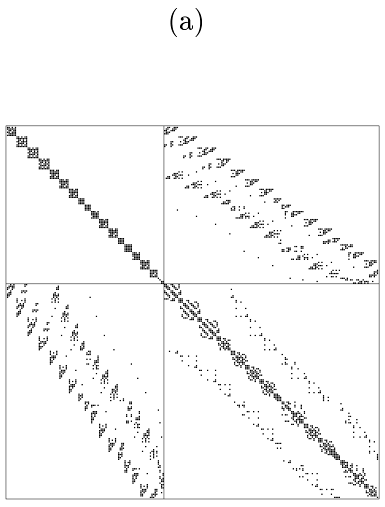
$$\begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & A_{l+1} \end{pmatrix}, \tag{3.5}$$

where

$$B_l \approx L_l U_l \tag{3.6}$$

$$A_{l+1} \approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l) \tag{3.7}$$

Exploting recursivity simplifies the description of the procedure which consists essentially of two steps. First, obtain a group-independent set and reorder the matrix in the form (3.4); second, obtain an ILU factorization $B_l \approx L_l U_l$ for B_l and approximations to the matrices $L_l^{-1} F_l$, $E_l U_l^{-1}$, and A_{l+1}. The process is repeated recursively on the matrix A_{l+1} until a selected number of levels is reached. At



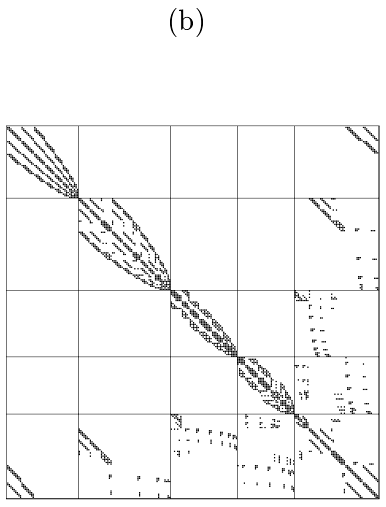


Figure 2: Group-independent set reorderings of a 9-point matrix: (a) Small groups (fine-grain), (b) large groups (coarse-grain).

the last level, a simple ILUT factorization, possibly with pivoting, or an approximate inverse method can be applied.

The A_i's remain sparse but become denser as the number of levels increases, so small elements are dropped in the block factorization to maintain sparsity. Note that the matrices $E_l U_l^{-1}, L_l^{-1} F_l$ or their approximations

$$G_l \approx E_l U_l^{-1} \ , \qquad W_l \approx L_l^{-1} F_l \tag{3.8}$$

which are used to obtain the Schur complement via (3.7) need not be saved. They are computed only for the purpose of obtaining an approximation to A_{l+1}. Once this is accomplished, they are discarded to save storage. Subsequent operations with $L_l^{-1} F_l$ and $E_l U_l^{-1}$ are performed using U_l, L_l and the blocks E_l and F_l. As an example of the many possible variations that can be made, examine the following forward and backward solves associated with the ARMS factorization. In the following, $(f_l^T, g_l^T)^T$ is a right-hand side for the system (3.5), and $(y_l^T, z_l^T)^T$ the solution of the system.

Algorithm **3.1** *RMLS – Recursive Multi-Level Solution*

```
1. Compute $g_l' = g_l - E_l B_l^{-1} f_l$
2. Solve (recursively) the system $A_{l+1} z_l = g_l'$
3. Back-Substitute to get $y_l$, i.e., solve $B_l y_l = [f_l - F_l z_l]$
```

The output vector from the above procedure is the result of one preconditioning operation applied to a given input vector $(f_l^T, g_l^T)^T$. Step 1 is akin to a fine-to-coarse mesh restriction used in multigrid. Step 3 is the reverse operation which can be viewed as a prolongation. Step 2 is where many possible variations can arise, among which the following ones

1. An iterative process is used (recursively) in the forward/backward RMLS sweeps;

2. The preconditioner for this iterative process is defined using the next $(l+2)$ level. This variation leads to procedures reminescent of W-cycles in multigrid.

3. Alternatively, the local B_l block can be used in preconditioning.

The paper [29] describes a scalar version of ARMS. The implementation, written in C, is fully recursive and makes efficient use of storage. The paper also presents a number of comparisons with other existing solution methods, both iterative and direct. ARMS is often superior to ILUT, which is often used as a benchmark for similar comparisons. Note that by a proper choice of parameters (essentielly using a number of levels equal to zero) the code will yield ILUT as one option.

3.1 Block independent set strategies

Several methods are available for obtaining block-independent sets. One of the simplest techniques in this context is based on a form of reverse Cuthill-McKee ordering. A first node (called the root) is selected from which the traversal is done. In the first step of the traversal, all the neighbors of the root are traversed. These nodes constitute the first level in the traversal. Every node that has been visited is marked. At each step a traversal is made from each of the nodes of the previous level. When a certain number of nodes are reached the set of nodes gathered from the traversal will constitute one block. In essence, the procedure starts a standard reverse Cuthill McKee procedure and stops as soon as a given number of nodes have been gatherer in the current set. The nodes thus gathered will form one of the B-blocks. Within each of these B-blocks, the labeling is reversed in a way similar to the reverse Cuthill-Mc Kee ordering.

Algorithm **3.2** *Multiple root reverse Cuthill Mc Kee*

```
1 Set $visited = \emptyset$
2 While $|visited| < n$ do
3   Select a new root $v$; set $set := last\_level := \{v\}$
4   While $|set| < bsize$ :
5       For each $w$ in $last\_level$
6           If $w$ is not marked
8               Add $w$ to $new\_level$
9               Mark $w$
10          EndIf
11      EndFor
12      $set := set \cup new\_level$
13      $last\_level := new\_level$
14  EndWhile
15  Reverse Ordering of set
16  Add all vertices of $last\_level$ to complement set
17  Select a new root from the non marked nodes
18 EndWhile
```

Figure 3 is reminescent of a number of patterns seen with some standard reordering methods. For example Nested Dissection (ND) orderings [16] common in sparse direct solution systems give rise to similar patterns. One can use nested dissection as a basis for generating block-independent sets. Nested dissection is a primary technique used to reduce fill-in in sparse direct solvers. It is easily described by using recursivity and by exploiting 'separators'. The main step of the ND procedure is to separate the graph in three parts, two of which have no coupling between each other. The third set has couplings with vertices from both of the first sets and is referred to as a sepator. The key idea is to separate the graph in this way and then repeat the process recursively in each subgraph.

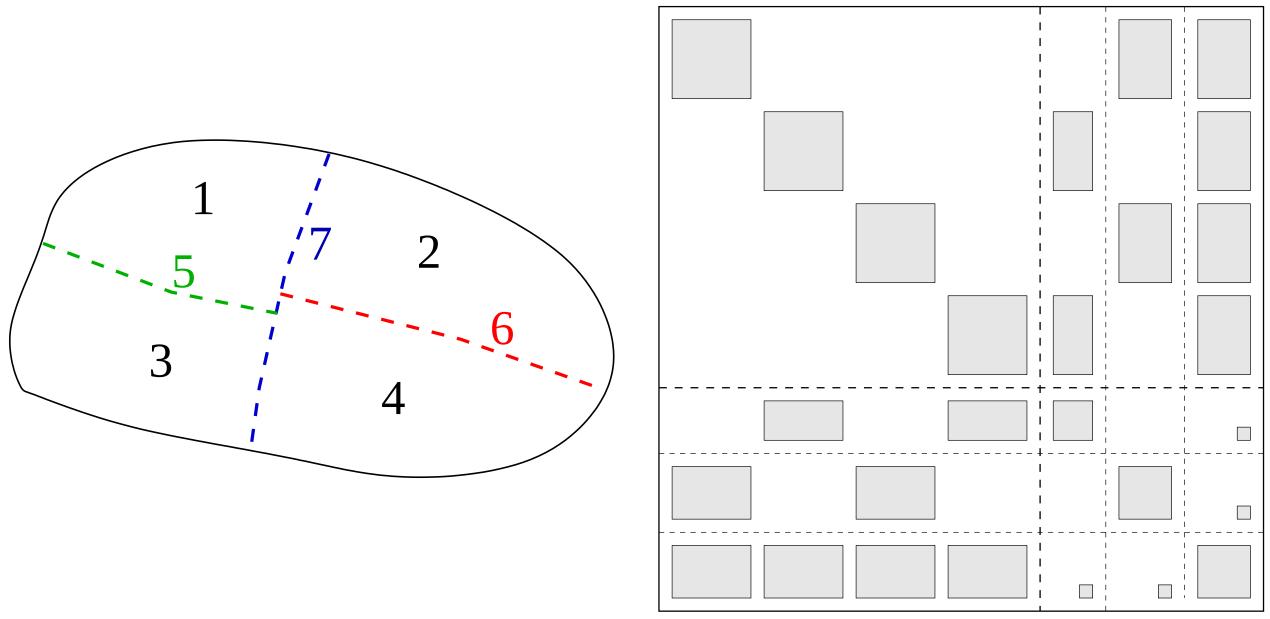


Figure 3: Nested dissection ordering and corresponding reordered matrix

The nodes of the separator are numbered last. An illustration is shown in 3. We implemented this strategy by using the reordering techniques provided in Metis [19]. Results reported in [29] indicate that this techniques does not usually improve upon the simple strategy described earlier. A different way of using Nested Dissection will be discussed in Section 3.3.

3.2 Diagonal Dominance Filtration

In order to improve the robustness of the factorization the numerical values are also considered. This is used to 'filter out' the rows that are the least diagonally dominant to the complement set in any group-independent set strategy. The heuristic is simple yet quite effective for improving robustness for problems that are highly indefinite.

A row is rejected to the complement set whenever a certain measure of its diagonal dominance relative to the other rows is poor. To implement this, 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 $0 \leq \hat{w}(i) \leq 1$ and that when $a_{ii} \neq 0$ the inverse of $\hat{w}(i)$ is $\hat{w}(i)^{-1} = 1 + \sum_{j \neq i} |a_{ij}/a_{ii}|$. These weights are close to one for strongly diagonally dominant rows, and close to zero for very non-diagonal 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,

$$w(i) = \frac{\hat{w}(i)}{\max_{j=1,\ldots,n} \hat{w}(i)}, \quad i = 1, \cdots, n. \tag{3.9}$$

Variations exist in which the weights combine both column and row diagonal dominance ratios.

3.3 ARMS with reordering

The block independent set reordering discussed above can be viewed as a reordering technique (coupled with a filtration mechanism) which allows to obtain a B block that is easy to invert, in the sense that its factorization will introduce little fill-in. In the extreme case of ILUM (block-size of one) no fill-in occurs in the factorization of B. There are many reordering methods used to reduce fill-in prior to a factorization. These are mainly used for direct methods but their usefuleness for iterative methods has been also extensively studied; see for example, [3, 9, 10, 12, 11, 6].

Among the standard reorderings, it is known (at least experimentally) that the best reordering technique for a direct solver is not necessarily the best one for ILU. Among these techniques, the reversed Cuthill Mc Kee reordering is known to perform quite well for ILU.

We can also combine reordering strategies with ILU in a different way. Experiments suggest that it is important to filter out those rows that are non-diagonally dominant and order them last, yielding a system in the form (3.4), where B does not have any particular structure. After this is done, we can apply a fill-reducing ordering to the B matrix. This strategy has been tested in the numerical experiments section. We have tried three different reordering strategies.

ARMS-RCM This uses the reverse Cuthill McKee ordering technique from the SPARSPAK package [7]. Note that the block-independent set algorithm of Section 3.1 can also yield a RCM ordering by taking an infinite block-size. However, it does not search for a good initial node (pseudo-peripheral node) to start the traversal as is done by the SPARSPAK routine.

ARMS-MMD The minimum degree (MD) algorithm is a very popular strategy for minimizing fill-in in Gaussian elimination. At any given step of Gaussian elimination, the node with the smallest degree is selected as the next pivot. The idea is that by doing so, we will minimize an upper bound for the numbe of fill-ins that will be introduced.

The Multiple Minimum Degree algorithm is a variation due to Liu [22, 15] which works with independent sets of pivots at each step. (The degrees of nodes adjacent to any vertex in the independent set, are updated only after all vertices in the set are processed).

ARMS-MND This technique uses the Nested Dissection ordering [16] as obtained from the package Metis [19]. Nested dissection is another popular reordering strategy for minimizing fill-in in Gaussian elimination. It is often used for symmetric positive definite matrices since it does not accomodate pivoting.

ARMS-AMF The Approximate Minimum Fill (AMF) algorithm has received much attention recently, see, e.g., [25]. Instead of minimizing the degree of each new pivot in Gaussian elimination, this strategy attempts to directly minimize the actual fill-in. This would in theory be expensive but techniques based on quotient graphs allow to obtain computationally inexpensive bounds for the amount of fill-in.

3.4 ARMS Cycles

There are many possible variations of the ARMS preconditioner once the ARMS factorization is available. One of the simplest ways to enhance the robustness of the preconditioner is to perform some form of inner iterations. The current version of ARMS allows three such variations which can be described in terms of variations on the ways in which the Schur complement systems are solved.

Below are three simple options that come to mind and which have been implemented and tested. Many other options exist which have not been 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 [29]. However, since this algorithm turns out to be fairly expensive for most practical situations we only mention it here for reference. The VARMS preconditioning step is shown in the following algorithm.

ALGORITHM **3.3** $x_l :=$ *VARMS-solve*(A_l, b_l) – *Recursive Multi-Level Solution*

```
0.  Let $b_l = \binom{f_l}{g_l}$
1.  Solve $L_l\ f_l' = f_l$
2.  Descend, i.e., compute $b_{l+1} := g_l - E_l U_l^{-1}\ f_l'$
3.  If $l = last\_lev$ then
4.      Solve $A_{l+1} y_l = b_{l+1}$ using GMRES+ILU factors
5.  Else
6.      $y_l =$ VARMS-solve$(A_{l+1}, b_{l+1})$
7.  Endif
8.  Ascend, i.e., compute $f_l'' = f_l' - L_l^{-1} F_l\ y_l$
9.  Back-Substitute $u_l = U_l^{-1} f_l''$
10. Return $x_l := \binom{u_l}{y_l}$
```

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

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

In [29] the question was addressed on how to perform the matrix-vector products with the matrices A_i without having to save these matrices. These operations are required whenever an iterative process is used at the intermediate or last levels. To obviate the need to store Schur complement matrices, we can compute $S_l \times w$ as

$$S_l \times w = (C_l - E_l B_l^{-1} F_l) w \tag{3.10}$$

This version uses matrices from the current level instead of the next one and saves storage, most often at the expense of additional arihtmetic operations. The inverse of B_l is applied by using its (incomplete) LU factors.

4 Model Problems and solution strategy

Several three-dimensional CFD model problems are used for testing the proposed preconditioners. Incompressible MHD flows and compressible gas flows are solved using stabilized finite element formulations. These considered test problems present some challenges for iterative methods. In the following, the governing equations and the finite element formulations used are briefly presented.

Let Ω be a bounded domain of $\Re^{nd}$ $(nd = 3)$ with boundary $\Gamma = \partial\Omega$. The outward unit vector normal to Γ is denoted by $\boldsymbol{n}$. Throughout this paper, we consider a partition of the domain Ω into elements Ω^e where piecewise continuous approximations for the independent variables are employed. We will use the standard notation: a subscript h to a function denotes a finite element approximation.

4.1 Magnetohydrodynamic flows

Given a flow field, the conservative magnetohydrodynamic system of equations is obtained from the Maxwell equations, and is written as:

$$\begin{aligned} \frac{\partial \mathbf{B}}{\partial t} - \nabla \times (\mathbf{u} \times \mathbf{B}) - \frac{1}{Re_m} \nabla \times (\nabla \times \mathbf{B}) + \nabla q &= 0 \qquad (4.1) \\ \nabla \cdot \mathbf{B} &= 0 \qquad (4.2) \end{aligned}$$

where Re_m, $\mathbf{B}$, $\mathbf{u}$ and q are respectively the magnetic Reynolds number, magnetic induction field, velocity field and the scalar Lagrange multiplier for the magnetic free divergence constraint ([1] and [2]). For coupled magnetohydrodynamics, this system is solved along with the incompressible Navier-Stokes equations where the body force is $\mathbf{f} = \frac{1}{\mu}(\nabla \times \mathbf{B}) \times \mathbf{B}$. This represents the Lorentz (Laplace) force due to the interaction between the current density $\mathbf{j} = \frac{1}{\mu}(\nabla \times \mathbf{B})$ and the magnetic field, where μ is the magnetic permeability. Since in real applications the magnetic-Reynolds number is small, the discrete variational formulation used for the magnetic problem is a simple stabilized Galerkin formulation [1] and is stated as: find $(\mathbf{B_h},\, q_h)$ such that for all weighting functions $(\mathbf{B_h^*},\, q_h^*)$ one has,

$$\begin{aligned} \int_\Omega \mathbf{B_h^*} \cdot \frac{\partial \mathbf{B_h}}{\partial t}\, d\Omega \;&+\; \frac{1}{Re_m} \int_\Omega \nabla \mathbf{B_h^*} \cdot \nabla \mathbf{B_h}\, d\Omega - \int_\Omega \mathbf{B_h^*} \cdot \nabla \times (\mathbf{u} \times \mathbf{B_h})\, d\Omega \\ &+\; \int_\Omega \mathbf{B_h^*} \cdot \nabla q_h\, d\Omega + (\tau_2 - \frac{1}{Re_m}) \int_\Omega (\nabla \cdot \mathbf{B_h})\, (\nabla \cdot \mathbf{B_h^*})\, d\Omega \\ &-\; \frac{1}{Re_m} \int_\Gamma ((\mathbf{n} \cdot \nabla \mathbf{B_h}) \cdot \mathbf{B_h^*} - (\mathbf{n} \cdot \mathbf{B_h^*})\, (\nabla \cdot \mathbf{B_h}))\, d\Gamma \\ &+\; \int_\Omega q_h^*\, (\nabla \cdot \mathbf{B_h})\, d\Omega + \sum_{\Omega_e \in \Gamma_h} \epsilon_2 \int_{\Omega_e} \nabla q_h \cdot \nabla q_h^*\, d\Omega \;=\; 0. \qquad (4.3) \end{aligned}$$

The parameter ϵ_2 is a function of the mesh size h and local magnetic Reynolds number ($Rem_h = ||\mathbf{u}||hRe_m$) given by $\epsilon_2 = h^2 Re_m/\sqrt{Rem_h^2 + 4}$. The parameter τ_2 is a penalty factor given by $\tau_2 = ||\mathbf{u}||h/2$. The case of fully-coupled MHD problems is described in [2] where a coupling algortithm between magnetic and fluid fields is developed.

4.2 Compressible gas flows

The Navier-STokes equations are solved using a finite element method [33] in terms of the conservative varialbles $\boldsymbol{V} = (\rho, \boldsymbol{U}, E)^t$, i.e. the density, the vector momentum and the specific total energy. In the case of turbulent flows we consider the one equation high Reynolds numbers Spalart-Allmaras turbulence model [34], the vector of unknown variables is then $\boldsymbol{V} = (\rho, \boldsymbol{U}, E, \nu_t)^t$ where ν_t is the turbulent kinematic viscosity. The governing system of equations is written in a compact form as

$$\boldsymbol{V}_{,t} + \boldsymbol{F}^{adv}_{i,i}(\boldsymbol{V}) = \boldsymbol{F}^{diff}_{i,i}(\boldsymbol{V}) + \boldsymbol{\mathcal{F}} \qquad (4.4)$$

where $\boldsymbol{F}^{adv}_{i}$ and $\boldsymbol{F}^{diff}_{i}$ are respectively the convective and diffusive fluxes in the ith-space direction, and $\boldsymbol{\mathcal{F}}$ is the source vector. Lower commas denote partial differentiation and repeated indices indicate summation. The diffusive fluxes can be written

$$\boldsymbol{F}_i^{diff} = \boldsymbol{K}_{ij}\boldsymbol{V}_{,j}$$

while the convective fluxes can be represented by diagonalizable Jacobian matrices $\boldsymbol{A}_i = \boldsymbol{F}^{adv}{}_{i,V}$. Recall that any linear combination of these matrices has real eigenvalues and a complete set of eigenvectors.

It is well known that the standard Galerkin finite element formulation often leads to numerical instabilities for convective dominated flows. In the Galerkin-Least-Squares method (or the generalized Streamline Upwind Petrov Galerkin method) ([18], [17]) the Galerkin variational formulation is modified to include an integral form depending on the local residual $\boldsymbol{\mathcal{R}}(\boldsymbol{V})$ of equation (4.4), i.e. $\boldsymbol{\mathcal{R}}(\boldsymbol{V}) = \boldsymbol{V}_{,t} + \boldsymbol{F}_{i,i}^{adv}(\boldsymbol{V}) - \boldsymbol{F}_{i,i}^{diff}(V) - \boldsymbol{\mathcal{F}}$, which is identical to zero for the exact solution. The SUPG formulation reads : find $\boldsymbol{V}$ such that for all weighting functions $\boldsymbol{W}$,

$$\sum_e \int_{\Omega^e} [\boldsymbol{W}\cdot(\boldsymbol{V}_{,t} + \boldsymbol{F}_{i,i}^{adv} - \boldsymbol{\mathcal{F}}) + \boldsymbol{W}_{,i}\boldsymbol{F}_i^{diff}]\, d\Omega \; - \int_\Gamma \boldsymbol{W}.\boldsymbol{F}_i^{diff}\, n_i\, d\Gamma \tag{4.5}$$

$$+\sum_e \int_{\Omega^e} (\boldsymbol{A}_i^t \boldsymbol{W}_{,i})\cdot\boldsymbol{\tau}\cdot\boldsymbol{\mathcal{R}}(\boldsymbol{V})\, d\Omega = 0.$$

In this formulation, the matrix $\boldsymbol{\tau}$ is referred to as *the matrix of time scales.* The SUPG formulation is built as a combination of the standard Galerkin integral form and a perturbation-like integral form depending on the local residual vector [17]. The objective is to reinforce the stability *inside the elements.* The matrix of time scales used here is given by $\boldsymbol{\tau} = (\sum|\boldsymbol{B}_i|)^{-1}$, for $i = 1, nd$ where $\boldsymbol{B}_i = \frac{\partial Re_{mi}}{\partial x_j}\boldsymbol{A}_j$ and $\frac{\partial Re_{mi}}{\partial x_j}$ are the components of the element Jacobian matrix [33]. The SUPG formulation aims at adding an amount of artificial viscosity in the characteristic directions. Since these formulations lead to a high-order scheme, high frequency oscillations in the vicinity of shocks or stagnation points can occur. A shock-capturing viscosity depending on the discrete residual $\boldsymbol{\mathcal{R}}(\boldsymbol{V})$ as proposed in [33] is employed. More dissipation is then added in high gradient zones to avoid any undesirable local oscillations.

4.3 Solution algorithms for nonlinear problems

The solution of partial differential equations of fluid dynamics by any time and space discretization method often leads to solving a non-linear problem in the Euclidian space $\Re^n$:

$$F(x) = 0. \tag{4.6}$$

The preferred strategy for solving (4.6) is Newton's method which generates a sequence of approximations of the form

$$x_{i+1} = x_i + \delta_i \tag{4.7}$$

starting from a certain initial approximation, $x_0 \in \Re^n$. The solution update δ_i at step i is the solution of the linear system

$$J(x_i)\delta_i = -F(x_i). \tag{4.8}$$

where $J(x_i)$ denotes the Jacobian associated with F and evaluated at x_i,

A solution strategy often used to solve many CFD problems consists of two nested loops, the first one over time and the second over Newton's iteration. This can be sketched as follows

Algorithm **4.1** *General solution strategy*

```
1. Do i=1, time − steps
2.     Compute and factor the Jacobian matrix
3.         (or an approximation) when needed
4.     Do k=1,Newton − iterations
5.         Compute the residual $F(x_i)$,
5.         Solve the system (4.8) and
6.         Update the solution by (4.7)
7.         Check for convergence
8.     EndDo
9. EndDo
```

In the exact Newton method, A is the true Jacobian matrix $J(x_i)$. This matrix, however, is not always available or computable analytically. It may also be expensive to recompute at every iteration, as is generally the case in CFD problems. It is often preferable to compute an approximation of the Jacobian once and use it for a prescribed number of time steps and Newton iterations. In our codes, this matrix is used to construct the preconditioner M. The action of the Jacobian on a vector (i.e. the matrix-by-vector product in line 4 of algorithm 2.1 (i.e. Az_j with $z_j = M^{-1}v_j$) is often approximated using a finite difference quotient such as:

$$Az_j \approx [F(x_0 + \epsilon z_j) - F(x_0)]/\epsilon \quad \text{or} \quad Az_j \approx [F(x_0 + \epsilon z_j) - F(x_0 - \epsilon z_j)]/(2\epsilon),$$

where ϵ is an appropriate small number. Simple formulas for computing ϵ are, respectively, $\epsilon = \epsilon_0(||x_0|| + \epsilon_0)$ (with $\epsilon_0 = 10^{-6}$) and $\epsilon = \epsilon_0/||z_j||$ (with $\epsilon_0 = 10^{-2}$).

Two issues of concern at each time step are the convergence of Newton's iterations and the cost of solving the linearized problem (4.8). For steady-state problems, another concern is the global convergence of the method. In practice, two different difficulties may be encountered, the divergence of Newton's iterations and the stagnation of the global loop.

As is well-known, Newton's method will converge only if the initial guess x_0 is close enough to the solution. To enhance the global convergence characteristics of Newton's method, a number of so-called global convergence strategies are often utilized. In the simplest of these, called 'damped' Newton, the update (4.7) is replaced by

$$x_{i+1} = x_i + \lambda_i \delta_i \tag{4.9}$$

in which the damping factor λ_i is less than one and selected so as to achieve sufficient reduction in the form of $\|F(x_i + \lambda_i \delta_i)\|_2$. Trust-region methods are often preferred. Independent of these improvements, the above algorithm may encounter a number of other difficulties related to the time-stepping procedure or the linear solver. Possible cures can be found by using one, or a combination, of the following strategies:

(a) reduce the time step,

(b) select the initial guess carefully,

(c) improve the robustness of the linear system solver by, e.g., increasing the Krylov-space dimension or using a better preconditioner,

(d) use stabilization techniques for discretization,

(e) use multigrid methods or adaptive meshing techniques, etc.

In this paper, a few techniques related to option (c) are proposed and numerically tested. All non-linear CFD problems considered in this paper are solved using the *non-linear version* of FGMRES(m), sometimes referred to as a Jacobian-Free nonlinear FGMRES.

5 Numerical Experiments

The goal of the numerical tests which are reported in this section is to help understand various variations of the ARMS procedure, as well as to show the effect of the choice of some of the parameters. First, we consider both the performance in the solutions of the linear systems extracted from the nonlinear iterative procedure as well for the nonlinear iteration itself. Then, we consider the global convergence of a few nonlinear CFD problems.

5.1 Test on a linear system from compressible flow

In the following tests, we first consider a linear system extracted from the simulation of a laminar compressible flow over a flat plate at Mach= 0.5 and Reynolds = 10000. The geometry is that of a flat profile and the mesh uses mixed tetraedras and hexaedras. The matrix size is $n = 122,321$ and the number of nonzero elements is $nnz = 6,751,053$.

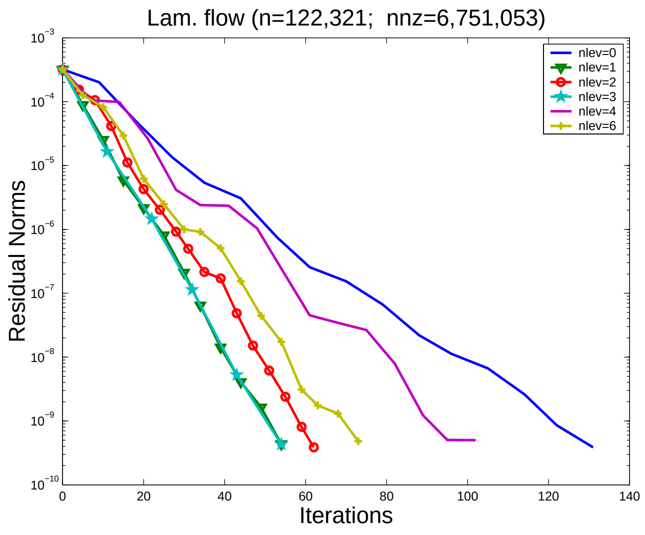
5.1.1 Effect of block-size and number of levels

The parameters `bsize` (block-size) and `nlev` (number of levels) are intimately related. The larger the block size, the smaller is the size of the Schur complement relative to that of the original system. The parameters used for the experiment are listed below

$bsize$	$nlev$	$fill_{inter}$	$fill_{last}$	$fill_{LU}$	$droptol_I$	$droptol_{last}$	tol_{DD}
1000	varies	50	40	40	0.001	0.01	0.1

In the table $bsize$ denotes the block-size selected for the block-independent sets. The actual block-size may differ from the input value of $bsize$ which acts as a threshold – a block is accepted as soon at its size is larger than $bsize$ in the traversal. The subscript I indicates that the parameter is used for the intermediate matrices, while $last$ refers to the last level. A 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 (3.8). The parameter $droptol_{LU}$ is for the ILU factorization of the last Schur complement. The parameter tol_{DD} is the threshold value used for the relative diagonal dominance criterion discussed in Sections 3.1 and 3.2. The number of levels $nlev$ is the maximum number of levels allowed. Similarly, the actual number of levels found may differ from the input value of $nlev$. In this experiment no interlevel iterations are performed. Figure 4 shows the number of iterations and the iteration time required to solve the linear systems when the number of levels varies.

Here, we should mention that when time is considered, then there is no simple rule, or easy way, to determine whether increasing the number of levels will help. This is because, the number of levels is tied to other choices. For example, it is easy to see that if there is substantial dropping at each level, the schur complements that are constructed become less accurate and therefore, more levels will harm convergence. In this case it is best to use one level. This is more or less the situation with the



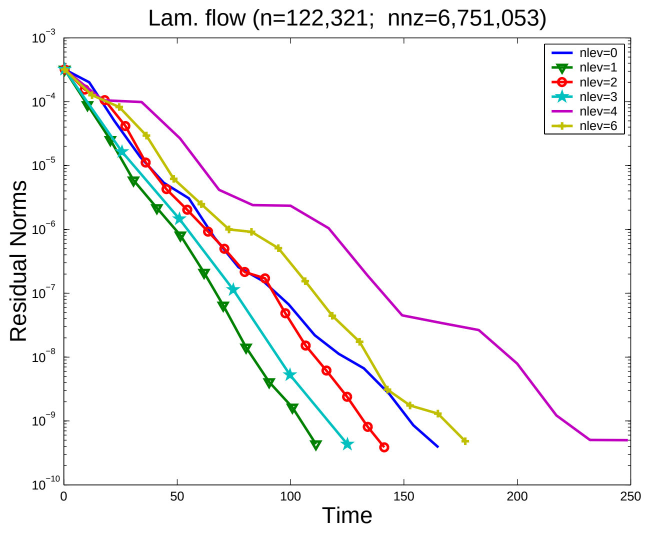


Figure 4: Residual norm vs number of iterations (left) and iteration time (right) for various number of levels

method parameters used in this experiment. It is also worth noting that zero levels (equivalent to ILUT) is often not competitive relative to ARMS. In this particular case, ILUT does still fairly well in terms of overall time. It is better than using 4 or 6 levels for example. In terms of memory used, it is remarkable that in all the tests with ARMS ($nlev >= 1$), the fill factor (i.e., the ratio of the number of nonzero elements in the preconditioning over that of the matrix is about the same (around 1.62). For ILUT this ratio is smaller and close to 1.27.

5.1.2 Effect of inner iterations and iteration cycles

There are many variations of the ARMS preconditioner once the multilevel ARMS factorization has been constructed. One of the simplest ways to enhance the robustness of the preconditioner is to perform some form of inner iterations. The current version of arms allows three such variations which were described in Section 3.4.

In the next test, we experiment with the WARMS variant (in this variant inner iterations are performed only at the last level and the top level). The number of inner iterations allowed at these levels is varied. Tests have been made with $nlev = 1$ and $nlev = 2$. The other parameters are as follows

$bsize$	$nlev$	$fill_{inter}$	$fill_{last}$	$fill_{LU}$	$droptol_I$	$droptol_{last}$	tol_{DD}
100	6	50	40	40	0. 01	0.01	0.1

As can be seen a very loose dropping criterion is selected, so that the resulting factorization is not accurate. The performance of the ILUT factorization using similar parameters is shown for comparison. For the ILUT preconditioner, the fill factor, i.e., the factor of the number of nonzero elements used by the LU factorization over the number of nonzero elements in A, is a small 0.67. For those tests using $nlev = 1$, the fill factor is around 0.79. Thus, the storage requirement in this case is quite modest. The ILUT preconditioner uses slightly less storage, but a slightly more accurate version yielded similar results (stagnation).

It is expected that as the number of inner iterations increases, for the first level and the last, the number of global fgmres iterations should decrease. This will tend to enhance robustness of the global scheme. However, because of dropping, the last level Schur complement matrix is not likely to be too accurate, and as a result, one expects that solving the last Schur complement system accurately is not cost-effective. This is verified. Another observation, is that using only one level can be very effective

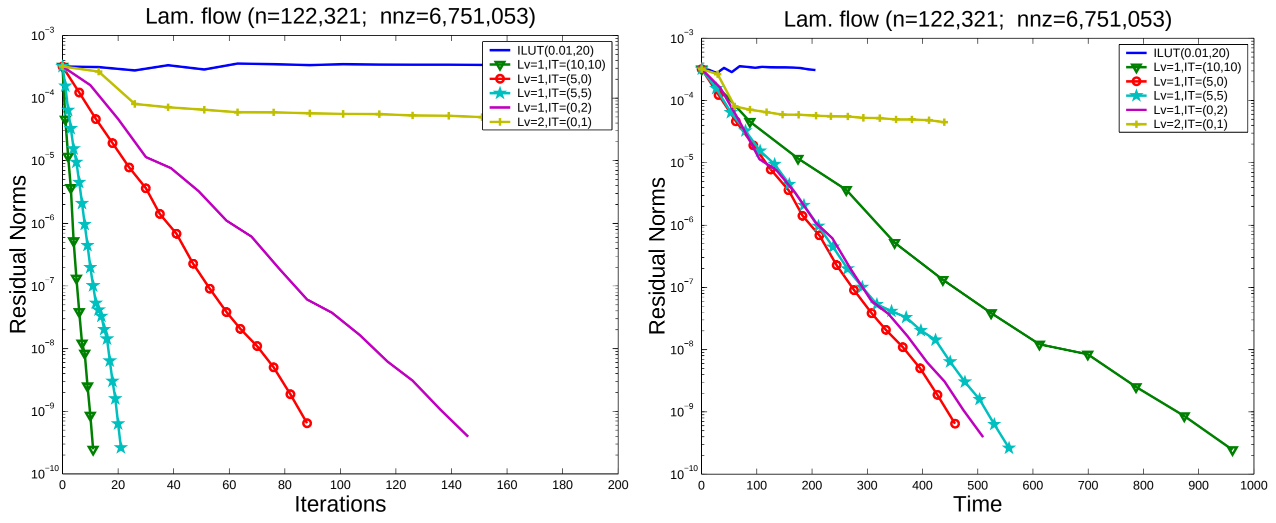


Figure 5: Residual norm vs number of iterations (left) and iteration time (right) for various inner-outer iteration combinations. The first integer in `IT` indicates the number of GMRES iterations at the top level, while the second is the number if GMRES iterations at the last level

in reducing the overall execution time. It is rather remarkable that convergence can still be reached with a preconditioner that is so inaccurate.

5.1.3 Effect of diagonal dominance filtration

It was mentioned earlier that diagonal dominance filtration is a simple yet very effective technique to improve robustness for highly indefinite matrices. There are several ways in which diagonal dominance filtration can be performed.

In the next experiment we show how a good choice of *tolind* can help improve performance. It has been our experience that it always pays to take a small positive value for *tolind*, i.e., to reject a (relatively) small percentage of rows with poor diagonal dominance, to the complement set.

We consider again the same matrix issued from the laminar flow example seen in the previous subsection. The parameter *tolind* is varied from 0.0 to 0.15 with a step of 0.05. Beyond $tolind = 0.15$, ARMS has difficulties converging or shows an unstable preconditioner. The other parameters used for the test are shown in the following table:

$bsize$	$nlev$	$fill_{inter}$	$fill_{last}$	$fill_{LU}$	$droptol_I$	$droptol_{last}$	tol_{DD}
100	2	50	40	40	0.001	0.01	varies

Results are shown in Figure 6.

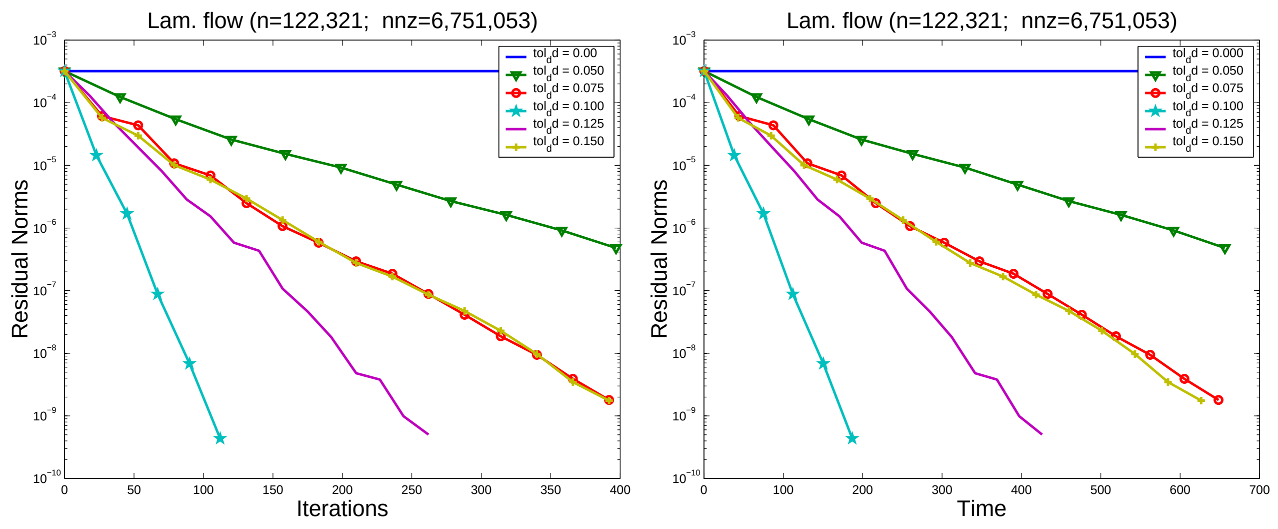


Figure 6: Residual norm vs number of iterations (left) and iteration time (right) for various values of *tolind*

5.2 Tests with reorderings

In section 3.3 we discussed combining standard reorderings with a multilevel strategy. If [ord] is a given ill-reducing ordering (e.g., AMF) then in the methods ARMS-[ord] the least diagonally dominant rows are ordered last and the ordering [ord] is applied to the resulting B block.

In the following tests we compare the strategies outlined in 3.3 with the standard ARMS which uses the block independent set ordering described in Section 3.1. Two test matrices were used. The first one is from an application which simulates Magneto-Hydrodynamics flows.

In the tests that follow, we solve problems which arise from the MHD. A pre-set periodic induction field $\mathbf{u}$ is used in the MHD equations. The physical region is the three-dimensional unit cube $[-1, 1]^3$ and the discretization uses a Galerkin-Least-Squares discretization. The magnetic diffusivity coefficient is $\eta = 1$. The matrix is of size $n = 470,596$ and has $nnz = 23,784,208$ nonzero entries.

We compared the following five methods: ARMS-RCM, ARMS-MMD, ARMS-MND, ARMS-AMF, ARMS-BIS, where BIS refers to the original Block-independent set reordering of 3.1. The parameters used for the experiment are listed below. Note that the block size is used only by ARMS-BIS.

$bsize$	$nlev$	$fill_{inter}$	$fill_{last}$	$fill_{LU}$	$droptol_I$	$droptol_{last}$	tol_{DD}
1000	3	50	30	30	0.0001	0.001	0.3

The results are shown in Table 1. Very similar results were obtained with a set of parameters leading to a slighly less accurate factorization. While the number of iterations to achieve convergence does not vary too much, it is interesting to observe that the memory requirement to achieve this performance shows a pronounced difference. The best overall performers in this case are the MMD and AMF techniques.

We also compared the 5 methods on a set of matrices arising from the simulation of a driven cavity problem. The region is 2-dimensional and the discretization uses bi-quadratic functions for velocities and linear (discontinuous) functions for pressures. Using 40 cells in each direction yields a matrix of size $n = 17,922$ and $nnz = 567,467$ nonzero elements. Though small, the linear system which result from this discretization are not too easy to solve. The system was generated for a Reynolds number

Method	Fill-ratio	Iterations	fact time	its time
ARMS-RCM	2.14	58	4.62e+03	3.21 e+02
ARMS-MMD	1.32	68	1.21e+03	2.91 e+02
ARMS-MND	1.87	68	1.30e+03	3.82 e+02
ARMS-AMF	1.32	66	1.18e+03	2.83 e+02
ARMS-BIS	1.92	69	7.10e+02	3.95 e+02

Table 1: Results with 4 different variations of ARMS based on different reorderings, for the MHD matrix

Method	Fill-ratio	Iterations	fact time	its time
ARMS-RCM	2.37	†	3.29e+01	†
ARMS-MMD	2.18	68	1.00e+00	1.30 e+01
ARMS-MND	2.45	76	1.13e+01	1.94 e+01
ARMS-AMF	2.22	74	1.35e+01	1.46 e+02
ARMS-BIS	1.96	†	8.61e+00	†

Table 2: Results with 4 different variations of ARMS based on different reorderings, for the CAVA0200 matrix

of $R_e = 200$ at the second Newton step. The parameters used for this experiment are the same as those in the previous experiment and the results are shown in Table 2.

A second test was conducted to explore the situation when a more accurate factorization is used. The set of parameters used will emphasize dropping based on drop tolerance rather than on keeping a maximum number of elements per row. This is achieved by taking $fil_I = fil_{last} = 100$. The number of levels and block-size are as before ($bsize = 1,000, nlev = 3$). We used two sets of choices for the $droptol$ parameters, namely $droptol_I = 0.001, droptol_{last} = 0.01$ and then $droptol_I = 0.0001, droptol_{last} = 0.001$. Results are shown in Table 3. It is interesting to notice that overall there is a fairly consistent pattern with the 4 reordering techniques just tested. The RCM and RCM-based ordering BIS, do not perform as well as the more sophisticated reorderings MMD and AMF. The performances of Multiple Minimum degree and Approximate Minimum Fill are very close to each other. It is also interesting to observe that a moderately accurate factorization does pretty well. Finally, we note that the dropping strategy which is based more on drop tolerance rather than on maximum fill per row does better overall than one which uses both (compare with Table 2). This is often observed with ILUT-type techniques.

5.3 Application to the simulation of Ponomarenko MHD dynamo instability

The Ponomarenko dynamo instability [24] is generated by a solid-body helical motion $\mathbf{u} = (u_r, u_\theta, u_z)$ of an incompressible fluid in a cylinder ($r \leq R_1$), surrounded by a cylindrical medium ($R_1 \leq r \leq R_2$). Both materials are electroconducting and characterized by their electroconductivity (resp. σ_1 and σ_2) and by their magnetic permeability (resp. μ_1 and μ_2). The velocity field is defined as $\mathbf{u} = (0, \omega r, U_z)$ with respect to cylindrical coordinates (r, θ and z). The pich of the solid-body helical motion is defined as $\chi = \frac{U_z}{u_\theta(r=R_1)} = \frac{U_z}{R1\omega}$. Here, ω is the angular velocity. For the kinetic MHD problem, the flow is supposed steady so ω and U_z are constant. The magnetic filed $\mathbf{B}$, is the solution of the MHD equations (4.1-4.2). There exists a critical magnetic Reynolds number Rem at which the magnetic

Method	Fill-ratio	Iterations	fact time	its time
With $droptol_I = 0.001, droptol_{last} = 0.01$				
ARMS-RCM	2.60	29	1.44e+01	4.93
ARMS-MMD	2.78	30	1.01e+01	5.19
ARMS-MND	2.77	29	1.11e+01	5.08
ARMS-AMF	2.70	29	1.30e+01	4.93
ARMS-BIS	3.06	38	5.58e+00	5.29
With $droptol_I = 0.0001, droptol_{last} = 0.001$				
ARMS-RCM	4.10	26	5.01e+01	5.97
ARMS-MMD	3.80	28	2.18e+01	6.08
ARMS-MND	3.97	28	2.30e+01	6.36
ARMS-AMF	3.88	27	3.24e+01	5.98
ARMS-BIS	4.05	29	1.15e+01	4.47

Table 3: Results with 4 different variations of ARMS based on different reorderings, for the CAV0200 matrix

field becomes unsteady. This critical value can be captured using a direct numerical solution, i.e. solving MHD equations in a time-accurate fashion and for a long period of time. Since this problem is linear, the precondtioner is built only at the first time step. The performance of the precondtioner is measured in terms of the memory and CPU time consumed. The results shown in this section correspond to the physical parameters $\chi = 1.3$ and $\sigma_1 = \sigma_2$. The initial magnetic field is given by : $\mathbf{B(t=0)} = (P(r)cos(\theta), -(P(r) + rP'(r))sin(\theta), P(r)cos(\theta))$, where $P(r) = (r - R_2)^2$. No-slip boundary conditions $\mathbf{B} = \mathbf{0}$ are imposed on the lateral surfaces ($r = R_2$) and the periodicity is imposed between the two faces $z = 0$ and $z = H$ with $H = 4\chi\pi$. The mesh used contains 33297 nodes and 158720 tetraedras, see Figure 7.

The matrix size and number of nonzero elements are $n = 125,052$ and $nnz = 6,354,832$ respectively. Direct numerical simulations were performed using a nondimensional time step of 0.15 and at least 2000 time steps. A critical magnetic Reynolds number was found at $Rem = 17.5$ for our simulation, to be compared with the critical value of $Rem = 17.7$ found by Gailitis [14]. Figure 8 shows the history of total magnetic energy. There is an instant when this energy starts to increase. This means that the magnetic field extracts energy from the flow field. For this test problem, we compared ILUT and ARMS preconditioners using the following parameters

$bsize$	$nlev$	$fill_{inter}$	$fill_{last}$	$fill_{LU}$	$droptol_I$	$droptol_{last}$
3000	5	nnz/n	nnz/n	nnz/n	0.0005	0.0005

In order to obtain a time-accurate solution, the convergence criteria for GMRES iterations is that the residual norm be reduced by a factor of 0.0005. A direct simulation of 2000 time steps takes almost 9 hours of CPU time on a $2Ghz$ PC machine. We found that ARMS preconditioner saves almost 10% of CPU time as compared to ILUT while using almost the same amount of memory. The histories of the residual norm (figure 8) for ARMS and ILUT are undistinguishable. Finally, figure 9 shows isocontours of the magnetic energy. This is well concentrated around radius R_1 in a helical shape.

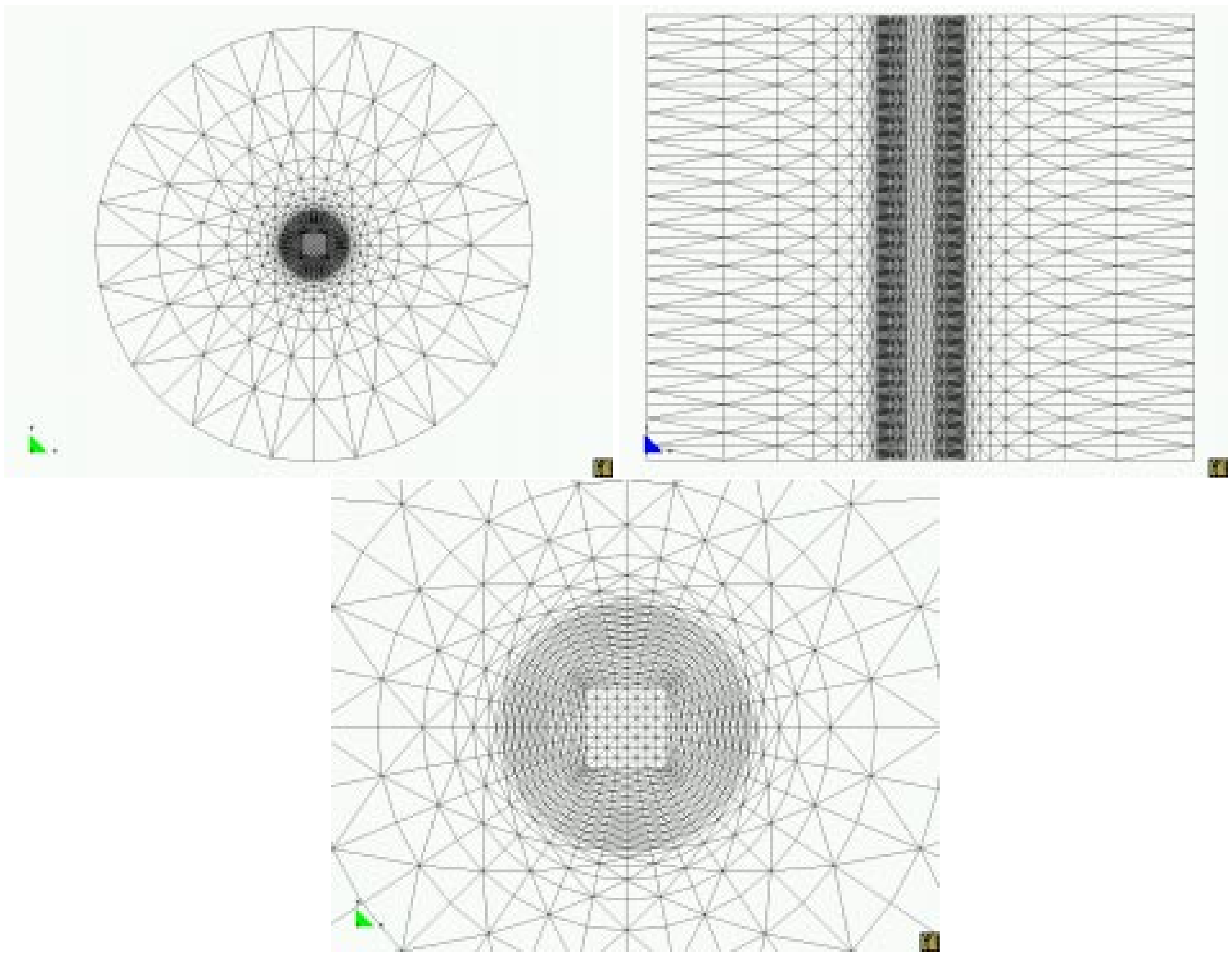


Figure 7: Mesh used for the Ponomarenko MHD test.

5.4 Tests on compressible flows

We consider the problem of a flow over a flat plate using a relatively fine mesh in the neighborhood of the plate. It is well known that some convergence difficulties can be encountred when looking for a steady solution of a low mach number (Mach ≤ 0.2), especially when using the conservative variables as the dependent unknowns. In the case of slightly compressible flows, one can use other sets of variables [13] employing the pressure as the dependent variable instead of the density. Another procedure uses what is called 'a preconditioning' of the Navier-Stokes equations where the time dependent term $\boldsymbol{V}_{,t}$ is multiplied by a preconditioning matrix P aimed at damping the spurious sound waves. In this work we prefer to use the same finite element method for all flow regimes and we rely on the algebraic preconditioners and the stabilization techniques to speed up the iterative procedure.

The mesh uses mixed tetraedras and hexaedras, the total number of elements is $150,064$. The first nodes are located at a distance of 510^{-4} from the plate (figure 10).

The matrix size is $n = 174,701$ and the number of nonzero elements is $nnz = 9,630,337$. We first consider a flow at a Mach number of 0.2 and at a Reynodls number of 10^4. The parameters used for this experiment are listed below

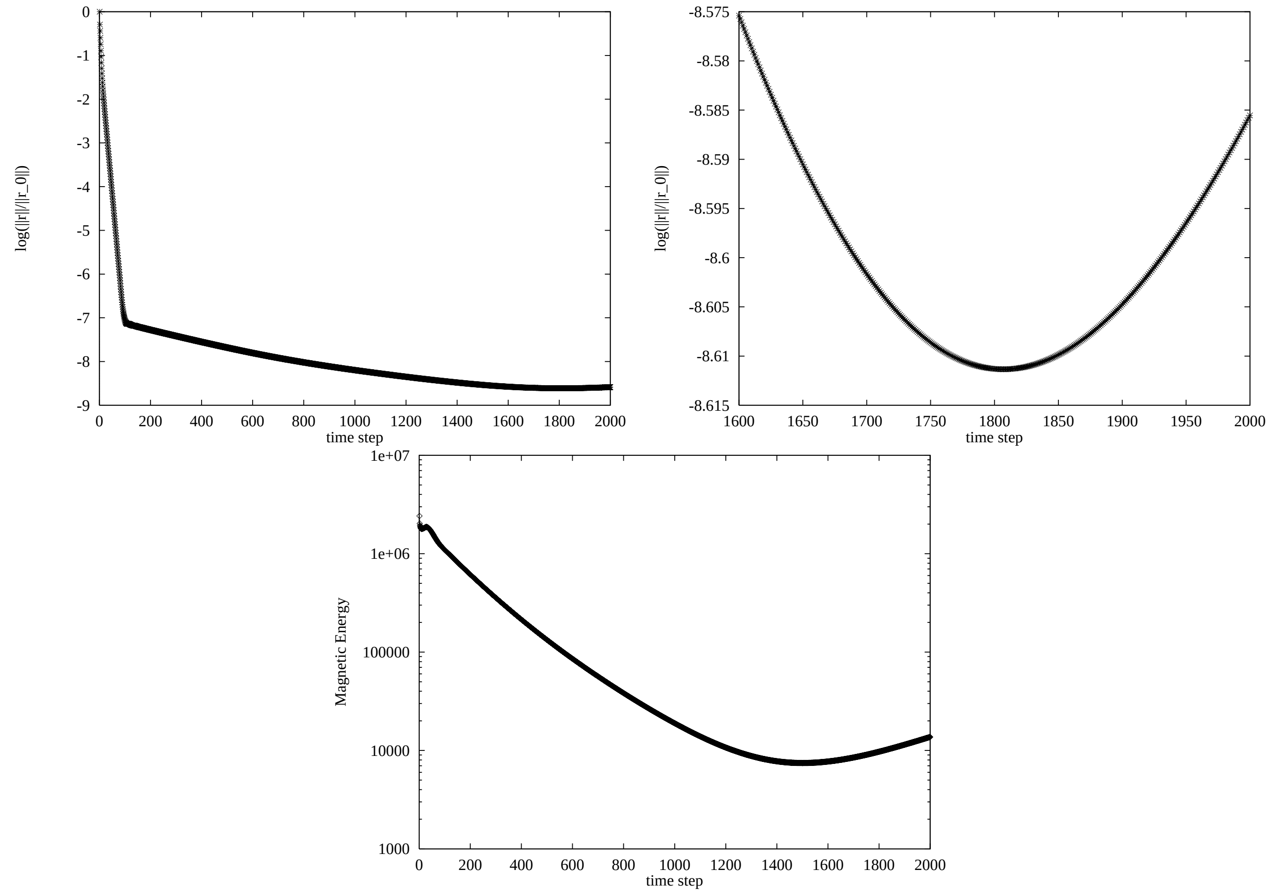


Figure 8: Ponomarenko MHD test, history of the residual norm with a zoom around the instability (top) and magnetic energy (bottom).

$bsize$	$fill_{inter}$	$fill_{last}$	$fill_{LU}$	$droptol_I$	$droptol_{last}$	
500	nnz/n or 2 nnz/n	nnz/n or 2 nnz/n	nnz/n or 2 nnz/n	0.0001	0.001	

As can be seen from this table, the fill-in parameters of the preconditioner have been related to the sizes of matrix nnz and n using some heuristic simple formulas. For the case of $nlev = 0$, the preconditioner corresponds to ILUT. Regarding the ARMS preconditionner, several tests showed that a good value of $nlev$ is 5. A smaller value generates a more stable but much more expensive preconditioner. For this problem, $bsize$ is 500. A larger value makes the preconditioner more expensive to build. It was observed also that the fill-in parameters have a big impact on the quality of the preconditioner and on the memory used. The solution procedure uses the time-stepping approach. After every 5 time steps the preconditioner is rebuilt and the time step is increased by an amount of 0.1 (i.e. we used a variable nondimensional time step Δt according to $\Delta t = \Delta t + 0.1$). Figure 11 shows typical convergence histories. It is clearly shown that the ILUT preconditioner with a fill-in nnz/n tends to a stagnation while for the same fill-in the ARMS based preconditioner behaves pretty well. Howerver, doubling the fill-in parameter makes ILUT preconditioner behave almost identically to ARMS preconditioner. This remark generally holds for many tests performed so far on various kind of CFD problems. ILUT can compete with ARMS but requires more fill-in.

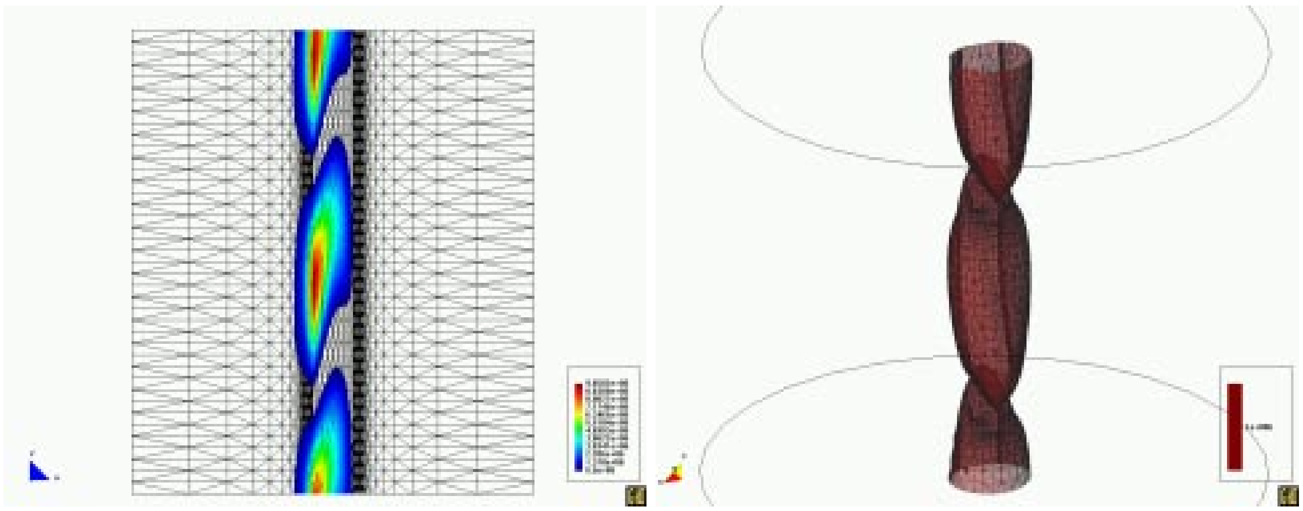


Figure 9: Ponomarenko MHD test. Isocontours of the magnetic energy.

On the other-hand, the fill-in parameters have a direct impact on the memory used. Although the ARMS preconditioner requires a smaller fill-in than ILUT, it uses some small additional memory to keep the multilevel structure. As mensioned earlier, a decision was made to sacrifice some computation to save memory, in keeping this structure. Specifically, the matrices G_l and W_l in (3.8) are discarded once the Schur complement given by (3.7) is computed. The products with the matrices W_l and G_l are performed by using L_l, F_l and E_l, U_l respectively, which are saved. The disadvantage of this approach is that it may entail a nonneglogible additional operation count when the matrices L_l and U_l are large. For the ARMS preconditioner, we found that a fill-in of nnz/n is sufficent but for ILUT we recommand $2nnz/n$. In terms of CPU, ARMS is found to be almost 15 percent more expensive than ILUT for tha same convergence quality. However, this performance is not always obtained. In other tests we found that ARMS can be faster than ILUT depending on how often the preconditoner is rebuilt and on the difficulty of the problem. Indeed, some CPU time saving can be obtained in favor of ARMS if the number of Krylov directions used per time step is found much lower than that required by ILUT. Figure 12 shows the profile of the nondimensional longitudinal velocity at few stations on the plate $(5.0 \leq x \leq 6.0)$ and at station $x = 7$. They are compared to the Blasius profile which is the soltuion for an incompressible flow at zero-pressure gradient.

We performed another test using the same geometry and the same mesh, in the transonic regime, i.e. the Mach number was set to 0.85 and the Reynolds number was $R_e = 10^4$. The parameters used for this test are listed below

$bsize$	$nlev$	$fill_{inter}$	$fill_{last}$	$fill_{LU}$	$droptol_I$	$droptol_{last}$
2000	nlev	2*nz/n	2*nz/n	2*nz/n	0.0001	0.0001

The time marching procedure is used with a variable *local time step* (i.e the element-time step is computed according to $\Delta t = CFL\frac{h^2}{h\lambda+2/Re}$, where h measures the element size, λ the maximum wave speed and CFL the courant number which increases after every 10 time steps $CFL = CFL + 1$). The initial flow field is obtained for a converged solution at a Reynolds number of $R_e = 10^2$. At each time step, only one Newton iteration is allowed, the matrix-free FGMRES algorithm is used with a maximum Krylov subsapce dimension of 40 and the inner iteration is stopped when the euclidian norm of the residual is reduced by a factor of 0.05. We observed that this test case is particularly demanding

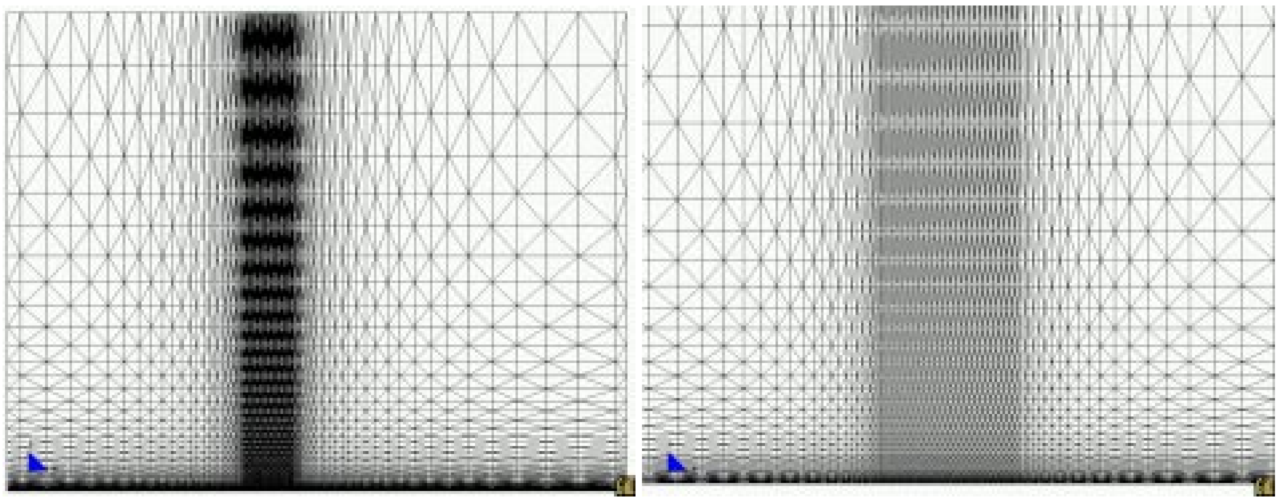


Figure 10: Mesh over a flat plate, close view (right)

on the iterative solver, so the fill-in parameters were set to $2nnz/n$. The global convergence (up to 200 time steps where the residual was reduced to three-order of magnitude) for ILUT and ARMS are almost identical, but FGMRES algorithm takes many more directions to converge with ILUT as compared to when the ARMS preconditioner is used. Figure 13 shows an example of the convergence history for time step number 20. Finally, figure 14 shows the iso-Mach contours for this test case.

Conclusion

We have shown a few preconditioning techniques which are derived from the Algebraic Recursive Multilevel Solver framework. There is virtually no limit in the number of possibilities available in this framework. The main ingredients are a means of selecting nodes to leave for the coarse level, a reordering of the nodes that are kept for the fine level, and various dropping techniques for constructing the next Schur complement. We have seen that performing inner iterations can make up for an inaccurate preconditioner and reestablish convergence in some difficult cases. We have also shown that using standard reordering methods as a means of defining levels can work quite well for the harder and larger matrices.

Acknowledgments

The work of Azzeddine Soulaimani and Rida Touihri has been been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). Yousef Saad acknowledges support from the Army Research Office under contract DA/DAAD19-00-1-0485, and the Minnesota Supercomputing Institute. The authors would like to thank Patrick Amestoy for providing them with a version of his Approximate Minimum Fill code.

References

[1] N. Ben Salah, A. Soulaïmani, W. G. Habashi and M. Fortin. “A Conservative Stabilized Finite Element Method for the magneto-hydrodynamics equations ”, *International Journal for Numerical Methods in*

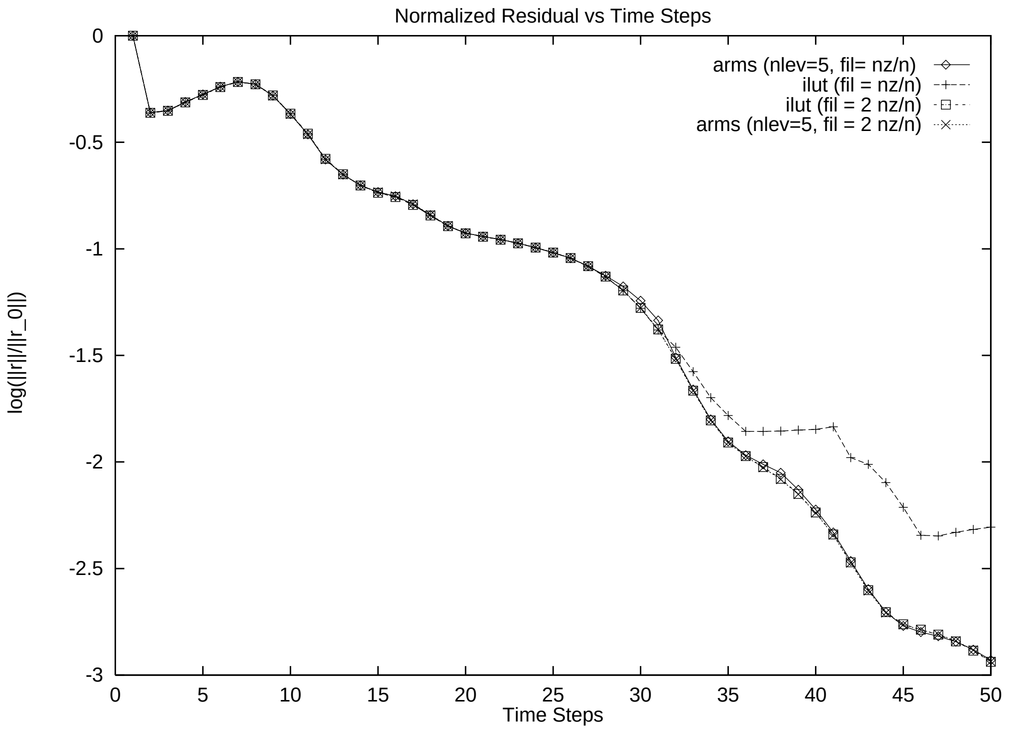


Figure 11: Residual norm vs number of time steps for various number of levels

Fluids, **29**, 535-554, (1999).

[2] N. Ben Salah, A. Soulaïmani and W. G. Habashi. "A Finite Element Method for magneto-hydrodynamics equations ", *Computer Methods in Applied Mechanics and Engineering.*, **190**, 5867-5892, (2001).

[3] M. Benzi, D.B. Szyld, and A. van Duin. Orderings for incomplete factorization preconditioning of non-symmetric problems. *SIAM Journal on Scientific Computing*, 20:1652–1670, 1999.

[4] 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.

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

[6] R. Bridson and W. P. Tang. A structural diagonosis of some IC orderings. *SIAM Journal on Scientific Computing*, 2001.

[7] E. Chu, A. George, J. W.-H. Liu, and E. G.-Y. Ng. Users guide for SPARSPAK–A: Waterloo sparse linear equations package. Technical Report CS-84-36, University of Waterloo, Waterloo, IA, 1984.

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

[9] E. F. D'Azevedo, F. A. Forsyth, and W. P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM Journal on Matrix Analysis and Applications*, 13:944–961, 1992.

[10] E. F. D'Azevedo, F. A. Forsyth, and W. P. Tang. Towards a cost-effective high order ILU preconditioner. *BIT*, 31:442–463, 1992.

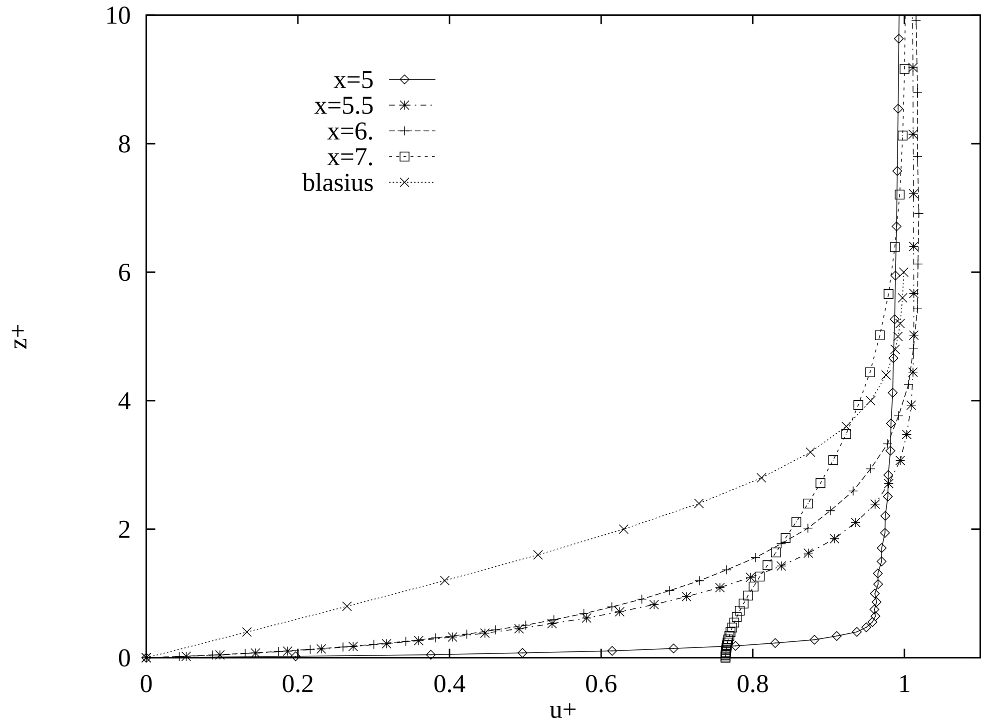


Figure 12: Nondimensional longitudinal velocity profile at different stations

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

[12] L. C. Dutto. The effect of reordering on the preconditioned GMRES algorithm for solving the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Engineering*, 36:457–497, 1993.

[13] N. E. Elkadri E, A. Soulaïmani and C. Deschenes. A finite element formulation of compressible flows using various sets of independent variables. *Computer Methods in Applied Mechanics and Engineering*, 181:161–189, 2000.

[14] A. K. Gailatis. The Helical MHD Dynamo. in *Topological Fluid Mechanics, Proceeding of the IUTAM Symposium*, Cambridge, August 13-18, 1989. Moffatt & Tsinober eds.

[15] A. George and J. W.-H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, March 1989.

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

[17] T.J.R. Hughes, L.P. Franca and Hulbert. "A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective-duffusive equations", *Computer Methods in Applied Mechanics and Engineering,* **73**, 173-189 (1989).

[18] T.J.R. Hughes and Mallet." A new finite element formulation for computational fluid dynamics: III. The generalized streamline operator for multidimensional advective-diffusive systems", *Computer Methods in Applied Mechanics and Engineering*, **58**, 305-328 (1986).

[19] G. Karypis and V. Kumar. A fast and high-quality multi-level scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1998.

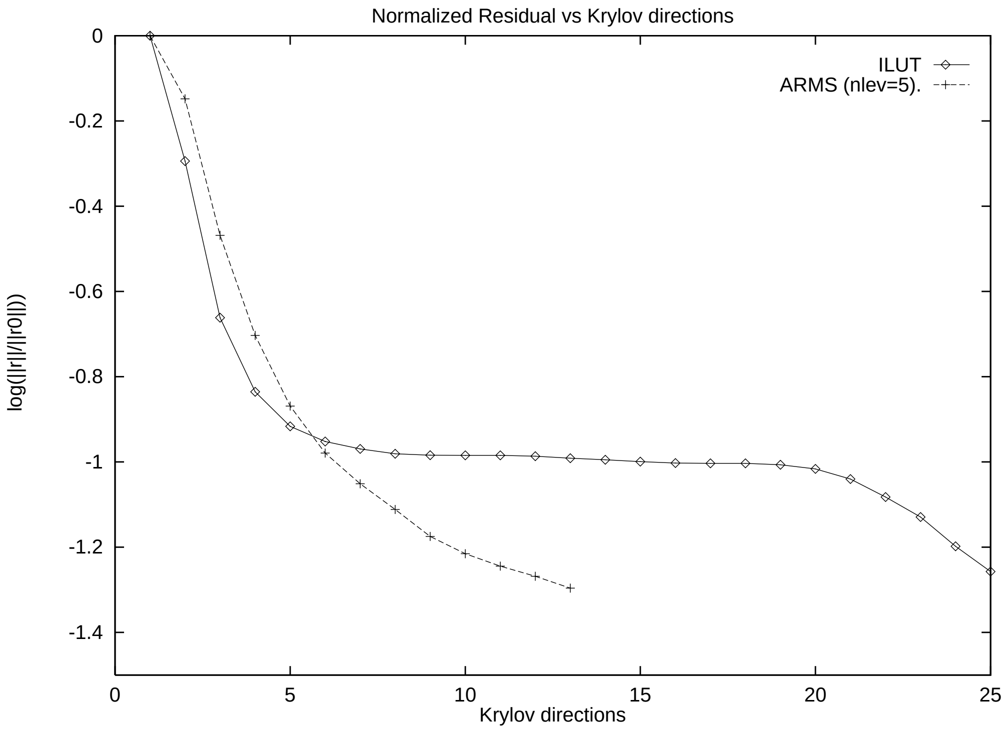


Figure 13: FGMRES convergence for time step number 20 (Free-stream Mach numer =0.85).

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

[21] Z. Li, Y. Saad, and M. Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. Technical Report umsi-2001-100, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.

[22] J. W.-H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11:141–153, 1985.

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

[24] Y. B. Ponomarenko. "On the theory of hydromagnetic dyanmo", *Zh. Prikl. Mekh. Tekhn Fiz (USSR).*, **6**, 47-51, (1973).

[25] E. Rothberg and S. C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM Journal on Matrix Analysis and Applications*, 1998.

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

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

[28] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[29] Y. Saad and B. Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. Technical Report umsi-99-107-REVIS, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001. Revised version of umsi-99-107.

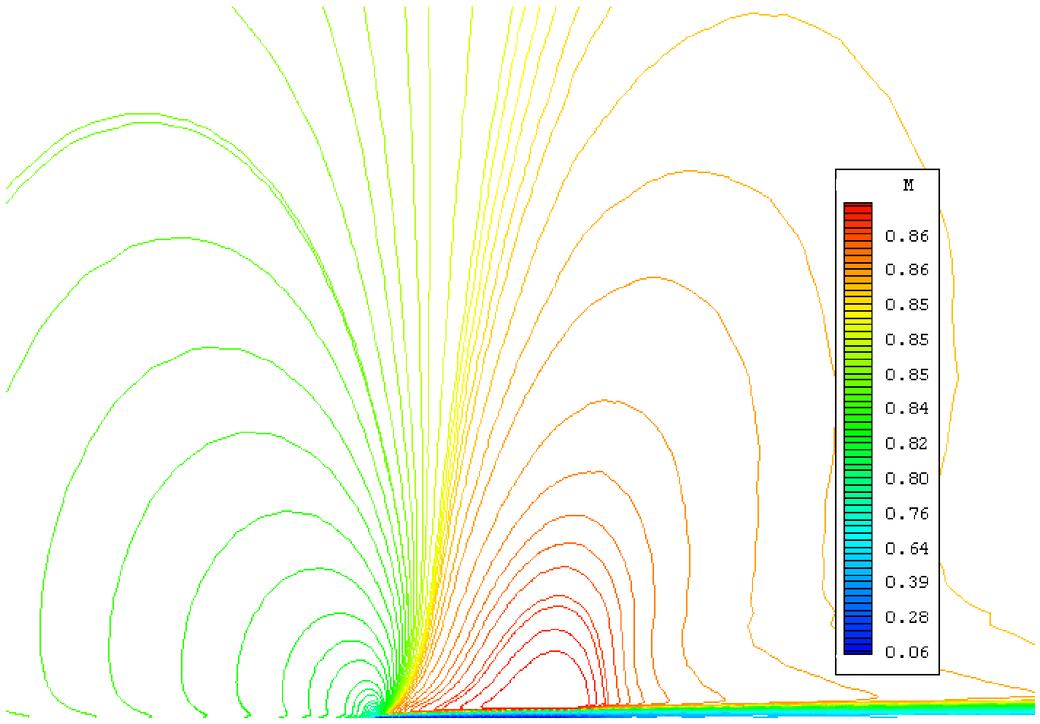


Figure 14: Iso-Mach contours (Free-stream Mach numer =0.85).

[30] Y. Saad and J. Zhang. BILUM: Block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 20:2103–2121, 1999.

[31] Y. Saad and J. Zhang. BILUTM: A domain-based multi-level block ILUT preconditioner for general sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 21:279–299, 2000.

[32] A. Soulaïmani, N. Ben Salah and Y. Saad. "Enhanced GMRES acceleration techniques for some CFD problems.", *International Journal of Computational Fluid Dynamics*, Vol. 16 (1), 1-20, (2002).

[33] A. Soulaïmani and M. Fortin. "Finite Element Solution of Compressible Viscous Flows Using Conservative Variables", *Computer Methods in Applied Mechanics and Engineering*, **118**, 319-350 (1994).

[34] P.R. Spalart and S.R. Allmaras. "A one-equation turbulence model for aerodynamic flows", *La Recherche Aerospatiale*, **1**, 5-21 (1994).