pARMS: a Parallel Version of the Algebraic Recursive Multilevel Solver *

Z. Li†, Y. Saad‡ and M. Sosonkina‡

September 5, 2001

Abstract

A parallel version of the Algebraic Recursive Multilevel Solver (ARMS) is developed for distributed computing environments. The method adopts the general framework of distributed sparse matrices and relies on solving the resulting distributed Schur complement system. Numerical experiments are presented which compare these approaches on regularly and irregularly structured problems.

1 Introduction

Preconditioning general sparse linear systems remains by far the biggest stumbling block in obtaining good performance for iterative solution methods on high-performance computers. Often, parallel implementations of complex engineering and scientific applications utilize the simplest preconditioning techniques such as the Additive Schwarz procedure. A well-known problem with these preconditioners is that the number of iterations required by the solver to converge may increase substantially as the number of processors grows, and this may drastically reduce the gains made from increased parallelism. In the same vein, standard preconditioners such as incomplete LU factorization (ILU) yield very little parallelism that can be exploited, unless some reordering of the equations is performed leading again to a likely increase of the number of iterations.

In recent years it has become increasingly clear that multilevel approaches are mandatory when dealing with large problems to be solved on massively parallel systems. Another important consideration is the increased complexity of numerical solution software. With this in mind, solvers that attempt to be "general-purpose" have become more important than ever. At the same time these solvers are also facing more challenges as problems are becoming harder to solve.

Two competing, and generally exclusive, classes of iterative methods for solving linear systems are currently being used: multigrid techniques and (preconditioned) Krylov subspace techniques. Multigrid methods can be extremely fast when they work. Their

*This work was supported in part by NSF under grants NSF/ACI-0000443 and NSF/INT-0003274, and in part by the Minnesota Supercomputing Institute

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

‡Department of Computer Science, University of Minnesota - Duluth, 320 Heller Hall, 10 University Drive, Duluth, MN 55812-2496, masha@d.umn.edu.

implementation requires multilevel grids and may also necessitate specific tailoring for particular applications. Algebraic MultiGrid (AMG) methods have been proposed as an alternative but their overall success, which in theory still relies on an underlying PDE problem, has been somewhat limited. 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 convergence rate usually deteriorates as the size of the linear system increases. Although multigrid could be vastly superior for certain problems, users have often sacrificed its speed for the better robustness and generality of preconditioned Krylov solvers. However, as the problems become larger the advantage of multilevel approaches can be overwhelming. The only difficulty is that multilevel methods are somewhat specialized, i.e., their performance is guaranteed to be optimal only for a specific class of problems. The ideal solution would be a method whose cost scales well with the size of the problem and which is as general purpose as the ILU-Krylov combination.

A number of methods were recently developed to try to fill this gap. ILUM [16] and a few related methods [7, 5, 1] showed that this approach is fairly robust and that it scales well with problem size [7, 5], unlike standard ILU preconditioners. The idea was extended to a block version (BILUM) using a sort of domain decomposition strategy [23]. A number of follow-up articles demonstrated the effectiveness of this approach [22, 20, 21]. Our tests indicate that the block approach is generally more efficient and more robust than a standard ILUT-preconditioned GMRES [17] as well as its scalar sibling, ILUM. For hard problems, these attributes often come with the added benefit of reduced memory usage.

Although these preconditioners are also highly parallel, no parallel implementation seems to have been undertaken so far. It is often advocated that the complexity of implementing these methods outweighs any gains made in efficiency over simpler techniques such as the Additive Schwarz procedures. It is hoped that this paper will demonstrate that such is not the case. It is possible to obtain good efficiencies while using only the local data structures used by the Schwarz procedures. A similar point was made in [18] using a Schur complement viewpoint. It was shown that global preconditioners can be designed from techniques which approximately solve the Schur complement system associated with interface variables. In this paper we extend this idea by using the Algebraic Recursive Multilevel Solvers (ARMS) framework.

2 Sequential ARMS – basic notions

The multi-level ILU preconditioners developed in [16, 5, 6, 22, 23] exploit the property that a set of unknowns that are not coupled to each other can be eliminated simultaneously in Gaussian elimination. Such sets are termed "independent sets", see e.g., [14]. In [23], the ILUM factorization described in [16] was generalized by resorting to "group-independent sets". A group-independent set is a set of groups of unknowns such that there is no coupling between unknowns of any two different groups [23]. Unknowns within the same group may be coupled. This is illustrated in Figure 1. In the Algebraic Multigrid literature, certain types of group-independent sets are referred to as "aggregates" [26]. Simple methods for finding standard (point) and group (block) independent sets have been considered in [16, 23] and elsewhere.

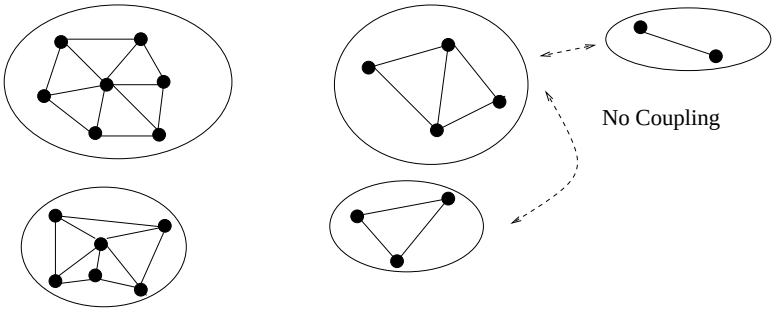

Figure 1: Aggregates, groups or blocks.

In many existing forms of multilevel ILU factorizations [1, 5, 7, 16] the unknowns are reordered, listing the nodes associated with the independent set first, followed by the other unknowns. After this reordering, the original matrix A_l at the l-th level takes the following form

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

where P_l is the independent-set permutation. At the zero-th level ($l = 0$) the matrix A_l is the original coefficient matrix of the linear system under consideration. The above partitioned matrix is then approximately factored as

$$P_l A_l P_l^T \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & A_{l+1} \end{pmatrix} \tag{2}$$

where where I is the identity matrix, L_l and U_l form the LU (or ILU) factors of B_l, and A_{l+1} is an approximation to the Schur complement with respect to C_l,

$$A_{l+1} \approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l). \tag{3}$$

During the factorization process, approximations to the matrices $G_l \equiv E_l U_l^{-1}$ and $W_l \equiv L_l^{-1} F_l$ are computed for obtaining the Schur complement (3) but these two matrices are discarded after A_{l+1} is computed. Typically it is inexpensive to solve linear systems with U_l and L_l since these arise from an ILU-type factorization. Therefore observe that all we need for defining a preconditioning for A_l is to provide a way to solve the reduced system, i.e., the system associated with A_{l+1} obtained by eliminating the unknown associated with the block B_l. It is here that different methods proposed vary.

Solving a linear system with the matrix in (2) requires (1) a forward solve followed by (2) a solve with the coarser level matrix A_{l+1}, 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. We refer to these as a downward and an upward operation, respectively. 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

$$A_{l+1} \times z_l = h'_l \tag{4}$$

from the forward (restriction) operation. The variations that arise are related to the ways in which this coarser level system is solved. In [21] we have implemented several different

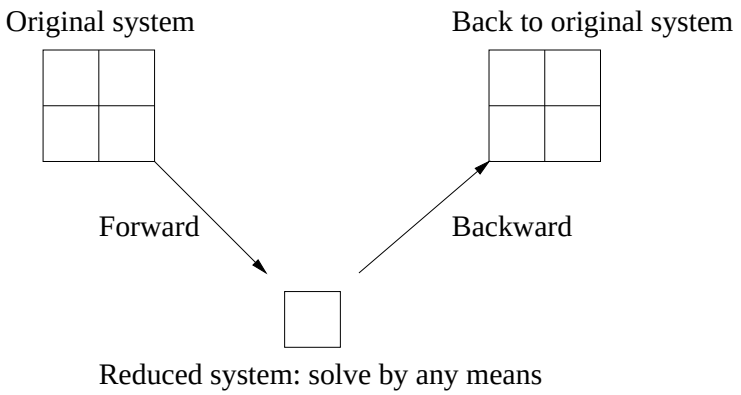

Figure 2: Illustration of a preconditioning operation.

options and tested them. The simplest option available is referred to as VARMS [in analogy with the V-cycle multigrid]. If the current level is not the last, VARMS continues to descend by using the level-structure, then solves the last-level system with GMRES-ILUT, and ascends back to the current level. Another option, not considered in this paper, is referred to as WARMS. At the difference with VARMS, WARMS iterates when solving the intermediate linear systems by using a few steps of GMRES utilizing VARMS as a preconditioner. The last level system is again solved with GMRES-ILUT. For reference, we reproduce here the description of the VARMS recursive solution from [21].

ALGORITHM **2.1** *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* $y_l = U_l^{-1} f''_l$

3 Preconditioning of distributed sparse linear systems

The framework we adopt for solving large sparse linear systems on parallel platforms is that of a distributed sparse linear system. This viewpoint generalizes Domain Decomposition methods to irregularly structured sparse linear systems. A typical distributed system arises, for example, from a finite element discretization of a partial differential equation on a certain domain. To solve such systems on a distributed memory computer, it is common to partition the finite element mesh by a graph partitioner and assign a cluster of elements which represent a physical subdomain to each processor. Each processor then assembles only the local equations associated with the elements assigned to it. The general

assumption is that each processor holds a set of equations (rows of the linear system) and a vector of the variables associated with these rows.

Figure 3 shows a 'physical domain' viewpoint of a sparse linear system. As is often done, we will distinguish between three types of unknowns: (1) Interior variables are those that are coupled only with local variables by the equations; (2) Interdomain interface variables are those coupled with non-local (external) variables as well as local variables; and (3) External interface variables are those variables in other processors which are coupled with local variables. The local equations can be represented as shown in Figure 4. Note that these equations are not contiguous in the original system. The matrix represented in the figure can be viewed as a reordered version of the equations associated with a local numbering of the equations/unknowns pairs.

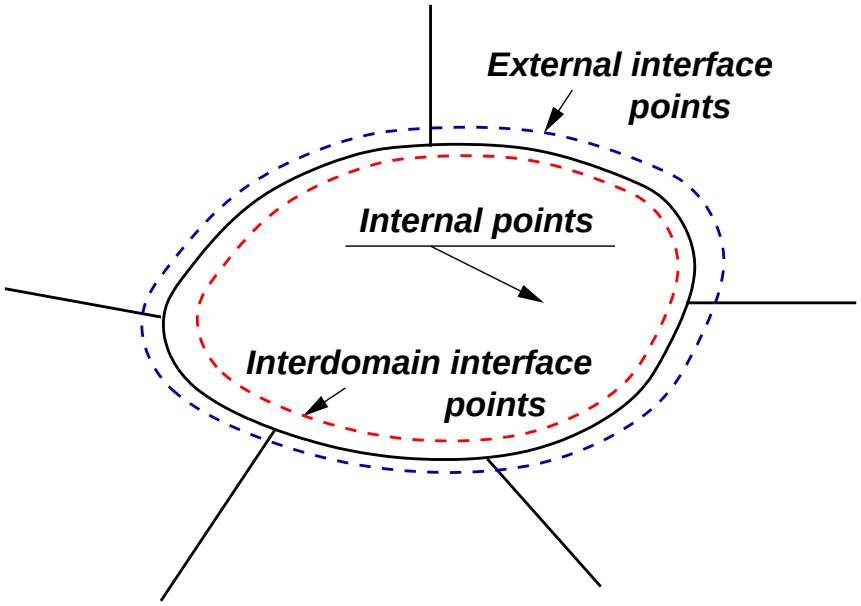

Figure 3: A local view of a distributed sparse matrix.

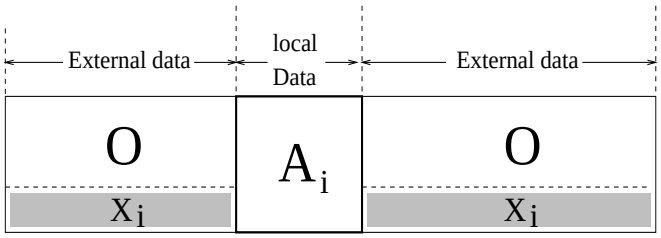

Figure 4: A partitioned sparse matrix.

As can be seen in Figure 4, the rows of the matrix assigned to a certain processor have been split into two parts: a *local* matrix A_i which acts on the local variables and an *interface* matrix X_i which acts on remote variables. These remote variables must be first received from other processor(s) before the matrix-vector product can be completed

in these processors. Most data structures for distributed sparse matrices list the interface nodes separately, usually by ordering them after the interior nodes. This 'local ordering' of the data presents several advantages, including more efficient interprocessor communication, and reduced local indirect addressing during matrix-vector products. It should be noted that the use of block sparse row format can also yield a substantial reduction in indirect addressing. The zero blocks in the figure shown are due to the fact that local internal nodes are not coupled with external nodes.

Thus, each local vector of unknowns x_i, $(i = 1, \ldots, p)$ is split in two parts: the subvector u_i of internal nodes followed by the subvector y_i of interdomain interface variables. The right-hand side b_i is conformally split in the subvectors f_i and g_i,

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix} \; ; \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix} \; . \tag{5}$$

The local matrix A_i residing in processor i as defined above is block-partitioned according to this splitting, leading to

$$A_i = \left(\begin{array}{c|c} B_i & E_i \\ \hline F_i & C_i \end{array} \right) \; . \tag{6}$$

With this, the local equations can be written as follows:

$$\begin{pmatrix} B_i & E_i \\ F_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix} . \tag{7}$$

The term $E_{ij}y_j$ is the contribution to the local equation from the neighboring subdomain number j and N_i is the set of subdomains that are neighbors to subdomain i. The sum of these contributions, seen on the left side of (7), is the result of multiplying a certain matrix by the external interface variables. It is clear that the result of this product will affect only the interdomain interface variables as is indicated by the zero in the upper part of the second term in the left-hand side of (7). For practical implementations, the subvectors of external interface variables are grouped into one vector called $y_{i,ext}$ and the notation

$$\sum_{j \in N_i} E_{ij} y_j \equiv X_i y_{i,ext}$$

will be used to denote the contributions from external variables to the local system (7). In effect, this represents a local ordering of external variables to write these contributions in a compact matrix form. With this notation, the left-hand side of (7) becomes

$$w_i = A_i x_i + X_{i,ext} y_{i,ext}. \tag{8}$$

Note that w_i is also the local part of the matrix-vector product Ax in which x is a vector which has the local vector components x_i.

3.1 Additive Schwarz preconditioning

Preconditioners for distributed sparse linear systems are best designed from the local structure described above. Additive and (variants of) multiplicative Schwarz procedures

are the simplest preconditioners available. Additive Schwarz procedures, update the local solution by the vector obtained from solving the linear system formed by the local matrix and the local residual. The exchange of data is done through the computation of the residual. In simple terms, the Additive Schwarz preconditioners can be stated as follows:

ALGORITHM **3.1** *Additive Schwarz*

1. *Update local residual* $r_i = (b - Ax)_i$
2. *Solve* $A_i \delta_i = r_i$
3. *Update local solution* $x_i = x_i + \delta_i$

This loop is executed on each processor simultaneously. Exchange of information takes place in Line 1, when the (global) residual is updated. Note that the residual is "updated" in that only the y-part of the right-hand side is changed. The local systems $A_i \delta_i = r_i$ can be solved in three ways: (1) By a (sparse) direct solver, (2) by using a standard preconditioned Krylov solver, or (3) by performing a backward-forward solution associated with an accurate ILU (e.g., ILUT) preconditioner. Experiments show that option (3) or option (2) with only a very small number of inner steps (e.g., 5) is quite effective.

3.2 Schur complement techniques

Schur complement techniques refer to methods which iterate on the interdomain unknowns only, implicitly using internal unknowns as intermediate variables. These techniques are at the basis of what will be described in the next sections. Schur complement systems are derived by eliminating the variable u_i from the system (7). Extracting from the first equation $u_i = B_i^{-1}(f_i - F_i y_i)$ yields, upon substitution in the second equation,

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i', \tag{9}$$

where S_i is the "local" Schur complement

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{10}$$

The equations (9) for all subdomains i ($i = 1, \ldots, p$) constitute a global system of equations involving only the interface unknown vectors y_i. This global reduced system has a natural block structure related to the interface points in each subdomain:

$$\begin{pmatrix} S_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2 & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \ldots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}. \tag{11}$$

The diagonal blocks in this system, the matrices S_i, are dense in general. The off-diagonal blocks E_{ij}, which are identical with those involved in the system (7), are sparse.

The system (11) can be written as

$$Sy = g',$$

where y consists of all interface variables $y_1, y_2, \ldots, y_p$ stacked into a long vector. The matrix S is the "global" Schur complement matrix. An idea proposed in [18] is to exploit methods that *approximately solve the reduced system* (11) to develop preconditioners for the original (global) distributed system. The resulting "global" preconditioner is said to be "induced" from the technique used for the Schur complement. Once the global Schur complement system (9) is (approximately) solved, each processor will compute the u-part of the solution vector (see (5)) by solving the system $B_i u_i = f_i - E_i y_i$ obtained by substitution from (7). In summary, a Schur complement iteration may be expressed by the following algorithm:

ALGORITHM **3.2** *Schur Complement Iteration — Template*

1. *Forward: compute local right-hand sides* $g_i' = g_i - E_i B_i^{-1} f_i$
2. *Solve global Schur complement system* $Sy = g'$
3. *Backward: substitute to obtain* u_i, *i.e., solve* $B_i u_i = f_i - E_i y_i$

Here, the similarity with ARMS is worth mentioning. The first step of every Schur complement algorithm considered in this paper is to form the new right-hand side $g_i' = g_i - F_i B_i^{-1} f_i$. This step is identical to the forward operation used for ARMS and corresponds to solving with the L factor in a block LU factorization (see [18]). The second step, which corresponds to the lower block in the diagram of Figure 2, requires that we (approximately) solve the Schur complement system (9) in some way. At this stage, the y variables are approximated. Finally, the third operation is the backward step, which computes the rest of the unknowns by substitution.

We now consider step two, solution of system (9), in more detail. For convenience, (9) is rewritten as a preconditioned system with the diagonal blocks:

$$y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} \left[g_i - E_i B_i^{-1} f_i \right]. \tag{12}$$

This can be viewed as a block-Jacobi preconditioned version of the Schur complement system (11). This global system can be solved by a GMRES-like accelerator, requiring a solve with S_i at each step.

In [18], two techniques were examined for solving linear systems with S_i. The first one is based on the observation that an LU (or ILU) factorization of S_i can be easily extracted from an LU (or ILU) factorization of A_i. Specifically, if A_i has the form (6) and it is factored as $A_i = L_i U_i$, where

$$L_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \quad \text{and} \quad U_i = \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix}.$$

then, $L_{S_i} U_{S_i}$ is equal to the Schur complement S_i associated with the partitioning (6). In other words, *an ILU factorization for the Schur complement is the trace of the global ILU factorization on the unknowns associated with the Schur complement.* For a local Schur complement, the ILU factorization obtained in this manner leads to an approximation $\tilde{S}_i$ of the local Schur complement S_i. The second method examined in [18] was based on using approximate inverse techniques for computing an approximation to the local Schur complement S_i.

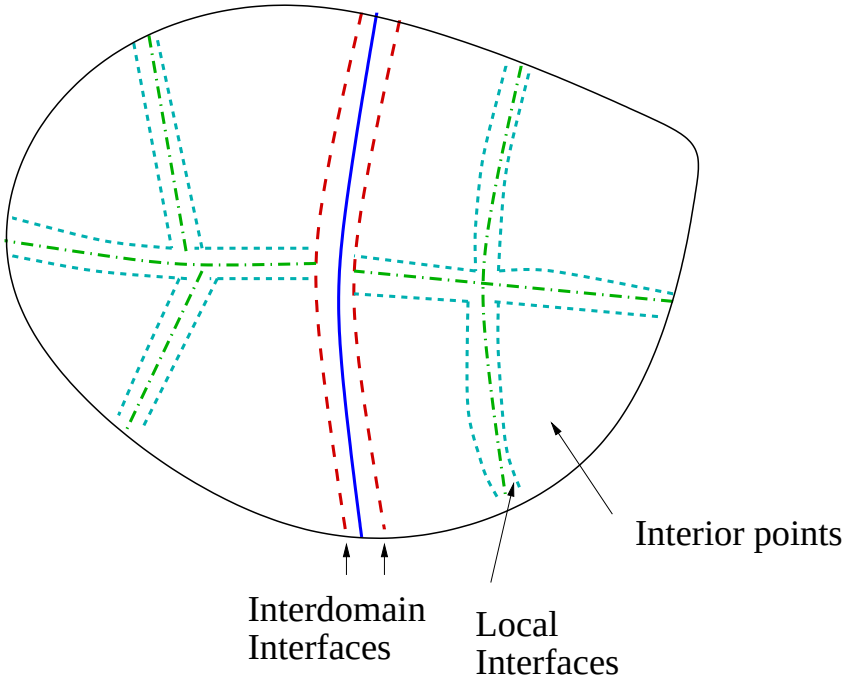

Figure 5: A two-level group-independent set.

4 Parallel implementation of ARMS (pARMS)

In order to extend ARMS to a parallel framework, it is necessary to begin by generalizing the notion of group-independent sets into Distributed Group-Independent sets (DGIS). A DGIS is a set of subsets (groups) from all processors such that the unknowns of different groups (within and across processors) are not coupled. It is clear that there is already one such set available from the partitioning which is the set consisting of the internal nodes, subdomain by subdomain. Distributed group-independent sets are easily obtained by further subdividing these sets of interior nodes. Doing so will yield another category of points: local interface points. An illustration is shown in Figure 5 for a simple case with two subdomains. The thick line in the middle contains no vertices – it only illustrates the separation of the two domains. The partitioning, which is vertex-based, is the same as the one seen in the previous section. Once the system is distributed among processors, using a graph partitioner, an independent set reordering is applied locally. One variation is that the independent set ordering does not disturb, i.e., it does not reorder, the interdomain interface points. Instead, independent sets are found among the interior points only.

With this viewpoint, the local matrix can be represented as in Figure 6, in which I1 represents the set of local interface points, while I2 represents the set of interdomain interface points. In parallel ARMS, the points I1 and I2 are all treated as interface points, in the sense that the Schur complement system considered may include both sets of points. When both types of points are included, we will refer to this system as the *expanded Schur complement*. It is clear that, when dealing with two types of interface points, some differences will arise. For example, I1-points require no communication when solving the linear systems.

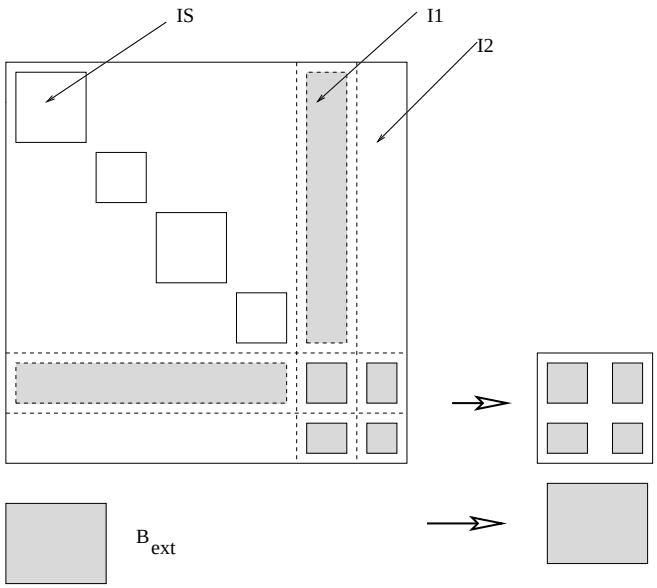

Figure 6: Matrix associated with 2-level partitioning

There are three distinct ways of using ARMS in a parallel environment. The simplest is to use an Additive Schwarz procedure in which the local solver uses the ARMS preconditioner. The second approach relies on a Schur complement-type technique in which, as was just explained, the Schur complement relates to equations associated with (local and inter-processor) interface points. This second approach yields a whole set of algorithms which arise from the different ways the Schur complement system is solved. We can solve the Schur complement system by a block-Jacobi procedure, and this would be similar to the work in [18]. In these two approaches, the ARMS reordering is applied locally only. If the system to solve is very large, the Schur complement system becomes itself quite costly to solve and it may be necessary to extend the ARMS reordering for the interdomain interface variables. This constitutes the third approach which is not considered in this paper. In the rest of this section, we present details of the expanded Schur complement system and describe the preconditionings acting on it.

4.1 Global expanded Schur complement system

In this subsection, the local processor index $i, i = (1, \ldots, p)$, will be omitted whenever it is clear that the related operations are local to processor i, The partitioning of the interior unknowns u into group-independent sets defines their splitting into interior points $\tilde{u}$ and local interface points $\hat{u}$. The local system (7) can be permuted accordingly into a block form in which the points $\tilde{u}$ are labeled first followed by the local interface points $\hat{u}$ and interdomain interface points y (see also Figure 5):

$$\begin{pmatrix} \tilde{B} & \hat{F} & F' \\ \hat{E} & \hat{C} & F'' \\ E' & E'' & C \end{pmatrix} \begin{pmatrix} \tilde{u} \\ \hat{u} \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \sum_{j \in N} E_j y_j \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ \hat{f} \\ g \end{pmatrix}. \tag{13}$$

In system (13), the matrix $\tilde{B}$ is block-diagonal. By eliminating the unknowns $\tilde{u}$ from

(13) in each processor, the *global expanded* Schur complement system, which corresponds to the remaining unknowns $z = (\hat{u}, y)$, is formed.

$$Sz + \begin{pmatrix} 0 \\ \sum_{j \in N} E_j y_j \end{pmatrix} = \begin{pmatrix} \hat{f} - \hat{E}\tilde{B}^{-1}\tilde{f} \\ g - E'\tilde{B}^{-1}\tilde{f} \end{pmatrix} \equiv g', \tag{14}$$

where S is the "local" Schur complement, which corresponds to the local and interdomain interface nodes:

$$S = \begin{pmatrix} \hat{C} & F'' \\ E'' & C \end{pmatrix} - \begin{pmatrix} \hat{E}\tilde{B}^{-1}\bar{F} \\ E'\tilde{B}^{-1}\bar{F} \end{pmatrix}.$$

The structure of this expanded Schur complement system is similar to the one in equation (11) and also has a natural block structure related to the interface points in each subdomain.

4.2 Additive Schwarz for the Schur complement

This consists of applying the Schur complement iteration (Algorithm 3.2) using the Schur complement system derived from pARMS. The system (14), the expanded version of (9), can be rewritten similarly to (12), as a preconditioned system with the diagonal blocks

$$z + S^{-1} \begin{pmatrix} 0 \\ \sum_{j \in N} E_j y_j \end{pmatrix} = S^{-1} g', \tag{15}$$

This is simply a block-Jacobi preconditioned version of the global expanded Schur complement system (14). In [18], incomplete LU factorizations and approximate inverses were used to approximate the local S^{-1}. Here, we apply a local ARMS procedure to obtain similar approximations. In particular, when S is the last level k of the local ARMS recursion, i.e., $S = A_k$, a factorization can be defined as follows. Let A_{k-1} be block factored as in (6) and followed by an (incomplete) LU factorization of the last matrix $A_k = L_k U_k$. Taken separately, this part of the ARMS factorization can be used for the preconditioning of the Schur complement system. For LU factorizations, this result has been established in [17]. In the case of the ARMS preconditioner, it can be easily extended when local ARMS recursively factors S. For a local Schur complement, an ARMS factorization obtained in this manner leads to an approximation $\tilde{S}$ of the local Schur complement S. Instead of the exact Schur complement system (14), or equivalently (15), the following approximate (local) Schur complement system, derived from (15), can be considered in each processor:

$$z + \tilde{S}^{-1} \begin{pmatrix} 0 \\ \sum_{j \in N} E_j y_j \end{pmatrix} = \tilde{S}^{-1} g'. \tag{16}$$

An algorithm is presented in [18] that applies, in each processor, the global approximate Schur LU preconditioner to a block vector $(f_i, g_i)^T$ to obtain the solution $(u_i, y_i)^T$. The algorithm uses a small number of GMRES iterations without restarting to approximate the solution of the local part of the Schur complement system (16).

4.3 Global incomplete LU preconditioning

In the previous subsection, the Schur complement system is solved by a Krylov subspace accelerator with block-Jacobi as a preconditioner. More accurate solutions to the

Schur complement system may lead to a better convergence of the outer iteration. The distributed ILU(0) preconditioner may be a good alternative to a block-Jacobi preconditioner or an SSOR preconditioner described [19]. We may use this approach as well to solve the Schur complement system.

The Schur complement system considered here is again the expanded system which contains local and interdomain interface points (see Figure 5). Interior points can be eliminated first, simultaneously in each processor. Once this is done, the interdomain interface points can be eliminated *in a certain order*. This is no different from standard distributed ILU(0) [17], except that it is applied to the distributed Schur system instead of the original distributed system. The global ordering used is based on a multicoloring of the domains, as this yields good parallelism [17].

As was already mentioned the distributed ILU(0) preconditioner is applied to the *local interface* and *interdomain interface* points. This technique is referred to as Schur Global ILU(0) preconditioner, to distinguish it from the distributed ILU(0) applied to the original distributed system. In what follows *colors[i]* is the color of processor number i, *mycolor* is the color of the current processor, *ncolor* is the number of colors.

ALGORITHM **4.1** *Global ILU(0) factorization*

1. *Perform ARMS elimination on the rows associated with local group-independent sets.*
2. *Perform ILU(0) on the local interface rows.*
3. */* Perform ILU(0) on the interdomain interface rows: */*
4. *for*$(i = 0; \quad i < ncolor; \quad i++)$ {
5. *if* $(mycolor == i)$ {
6. *Perform the ILU(0) factorization for the interdomain interface rows with pivots from the local interface rows completed in step 2;*
7. *Send the completed interdomain interface rows to adjacent processors* j*, with* $colors[j] > mycolor$.
   }
8. *else if* $(mycolor > i)$ {
9. *Receive the factored rows from the adjacent processors,* j *with* $colors[j] == i$*;*
10. *Perform the ILU(0) factorization with pivots received from the external processors in step 7.*
    }
    }

In the above algorithm, steps 1 and 2 can be performed simultaneously in each processor.

The preconditioned Krylov subspace algorithm requires a forward and backward sweep at each step. The distributed forward/backward solution based on this factorization is sketched next.

ALGORITHM **4.2** *Distributed forward and backward sweep*

1. */* Forward solve: */*
2. *Perform the forward solve for the local interface nodes.*

3. for($i = 0; \quad i < ncolor; \quad i++$) {
4. if ($mycolor == i$) {
5. *Perform the forward solve for the interdomain interface nodes;*
6. *Send the updated values of interdomain nodes to the adjacent processors j, with $colors[j] > i$.*
}
7. else if ($mycolor > i$) {
8. *Receive the updated values from the adjacent processors, j, with $colors[j] == i$.*
}
}
9. /* Backward solve: */
10. for($i = ncolor - 1; \quad i >= 0; \quad i--$) {
11. if ($mycolor == i$) {
12. *Perform the backward solve for the interdomain interface nodes;*
13. *Send the updated values of interdomain nodes to the adjacent processors, j, with $colors[j] < i$.*
}
14. else if ($mycolor < i$) {
15. *Receive the updated values from the adjacent processors, j, with $colors[j] == i$.*
}
}
16. *Perform the backward solve for the local interface nodes.*

The above algorithm only shows the forward/backward sweep in the last Schur complement. As in the ILU(0) factorization, the local interface nodes do not depend on the nodes from the external processors, and can be computed in parallel in steps 2 and 16. In the forward solve, the solution of the local interface nodes is followed by an exchange of data and the solution on the interdomain interface. The backward solve works in reverse in that the interdomain interface nodes) are first computed, then they are sent to adjacent processors, and the local interface update follows.

5 Numerical Experiments

In this section, we present a comparison of the preconditioners available in the pARMS framework. Scalability experiments, in which the problem size and number of processors grow, are first reported for two- and three-dimensional PDE problems on regular grids. A performance comparison with a direct solver is included in these tests. We then report on a few tests with fixed-size irregularly structured problems arising in magnetohydrodynamics.

5.1 Naming conventions for the methods tested

For the sake of convenience, let us introduce some notation for the preconditioning options considered in pARMS. In general, the method names are of the form `add_X Y` or

`sch_X Y`, where the prefixes `add` and `sch` indicate whether the preconditioner has local operations only, i.e., it is of Additive Schwarz type, or whether it requires external data communications, i.e., it is of Schur complement type, respectively. The letter `X` may stand for either an incomplete LU factorization, such as ILUT or ILU(k) [17], or for ARMS. The global preconditioners of the `sch` type may be also used with a global ILU(0) preconditioner `gilu0` applied to the expanded Schur complement system. This preconditioning option `sch_gilu0` resembles the one in [18], but it acts on the expanded Schur complement system (13) instead of (9), and it uses an ILU(0) factorization instead of a block Jacobi preconditioner. A preconditioner name may be followed by `Y`, which stands for `no its` and/or `ovlp`, meaning that the preconditioner is applied without inner (i.e., intra-domain) iterations and/or has overlapping, respectively. As is well-known [3, 4, 9, 10, 2] overlapping improves the efficiency and scalability of Additive Schwarz preconditioners. Many ways to overlap nodes have been considered in parallel environments. In particular, Restrictive Additive Schwarz [8] showed good performance. In the experiments, we consider a simple overlapping technique, in which the contributions of the overlapped nodes are averaged upon their exchange among neighboring domains. This type of overlapping has been used by many authors, see, for example, [24, 13, 15]. A few additional details are provided below for each of the preconditioners considered.

`add_ilut.` Additive Schwarz procedure, with or without overlapping, in which ILUT is used as a preconditioner for solving the local systems. These systems can be solved with a given number of GMRES inner iterations or by just applying the preconditioner.

`add_iluk.` Similar to `add_ilut` but uses ILU(k) as a preconditioner instead of ILUT.

`add_arms.` Similar to `add_ilut` but uses ARMS as a preconditioner for local systems.

`sch_ilut.` This method is the same as the one in [18]. It consists of solving the (global) Schur complement system, associated with the interdomain interface variables, using a block-Jacobi preconditioner. The ILUT factorization for the local Schur complement is obtained as a 'trace' of the ILUT factorization on the interface variables. These local ILUT factorizations are then used to precondition the global Schur complement system in a block-Jacobi iteration. For details see [18].

`sch_iluk.` This is the same as `sch_ilut` except that ILU(k) is used instead of ILUT.

`sch_arms.` This is the same as `sch_ilut` except that ARMS is used instead of ILUT.

`sch_sgs.` Symmetric block Gauss-Seidel preconditioning is used in solving the global Schur complement system instead of a block-Jacobi preconditioning as described in [18]. This preconditioning requires a global ordering, which is provided by multicoloring.

`sch_gilu0.` This method differs from the other Schur complement methods in that it is based on approximately solving the expanded Schur complement with a global ILU(0)-preconditioned GMRES. The ILU(0) preconditioning requires a global order (referred to as a schedule in [11]) in which to process the nodes. A global multicoloring of the domains is used for this purpose as is often done with global ILU(0).

5.2 Scaled problem: Two-dimensional case

Consider the elliptic partial differential equation

$$-\Delta u + 100\frac{\partial}{\partial x}\left(e^{xy}u\right) + 100\frac{\partial}{\partial y}\left(e^{-xy}u\right) - 10u = f \tag{17}$$

on a square region with Dirichlet boundary conditions, discretized with a five-point centered finite-difference scheme on a $n_x \times n_y$ grid, excluding boundary points. The mesh is mapped to a virtual $p_x \times p_y$ grid of processors, such that a subrectangle of $r_x = n_x/p_x$ points in the x direction and $r_y = n_y/p_y$ points in the y direction is mapped to a processor. In the following experiments, the mesh-size in each processor is kept constant at $r_x = 100, r_y = 100$. As the number of processors increases, the problem size increases proportionally. For example, on 4 processors the mesh is $(2r_x, 2r_y)$ leading to a problem size of $40,000$ and on 64 processors the mesh is $(8r_x, 8r_y)$ leading to a problem size of $640,000$. The resulting problems become harder as the number of processors grows. In a perfectly scalable situation, the final execution time for solving the problem with size $n = 640,000$ on 64 PEs should be identical with the time for solving the problem of size $n = 40,000$ on 4 PEs. Note that in order to maintain the aspect ratio of the physical domain, we need to consider *square* processor grids of increasing size.

The results have been obtained on an SGI Origin3800. The residual norm reduction of 10^{-6} was achieved by flexible GMRES (FGMRES) [17] with subspace size of 100. The fill-in parameter `lfil` is taken as 60 for all the levels of recursion along with the dropping tolerance of 10^{-4}. The group-independent set size is 20. Timing and iteration results for the fixed local subproblem sizes of 100×100 are presented in Figures 7 and 8 (top and bottom, respectively). As shown in Figure 7, the total solution times for the Additive Schwarz preconditioners with five inner iterations are larger than for the corresponding preconditioners without inner iteration. For Additive Schwarz, the presence of inner iterations does not reduce the number of outer iterations in a significant way. Additive Schwarz with ARMS as a local preconditioner uses less time than Additive Schwarz with ILUT as a local preconditioner although their corresponding iteration numbers are close. The reason is that Additive Schwarz with ARMS requires less memory in both construction and solution. Figures 7 and 8 confirm the advantages of using overlapping in Additive Schwarz.

Figure 8 shows a comparison of two-level preconditioners with various Additive Schwarz procedures. It is noticeable that `sch_gilu0` and `sch_sgs` outperform the other methods for this test case. The performance of `sch_ilut` has also been tested. However, it is not shown in the Figures since this method was not found to be faster than `sch_gilu0` or `sch_sgs` and its advantages over `add_ilut` have been shown in our earlier work [18]. Note a rather interesting spread in the number of iterations for the various methods. It is also remarkable that `sch_sgs` and `sch_gilu0` require an almost identical number of steps to converge, both when inner iterations are performed and when no inner iterations are performed.

5.3 Scaled problem: Three-dimensional case

We solve an analogue of (17), on a three-dimensional cube. The PDE is expressed in an identical way as in (17) but the Laplacean is three-dimensional, i.e., there is no convective

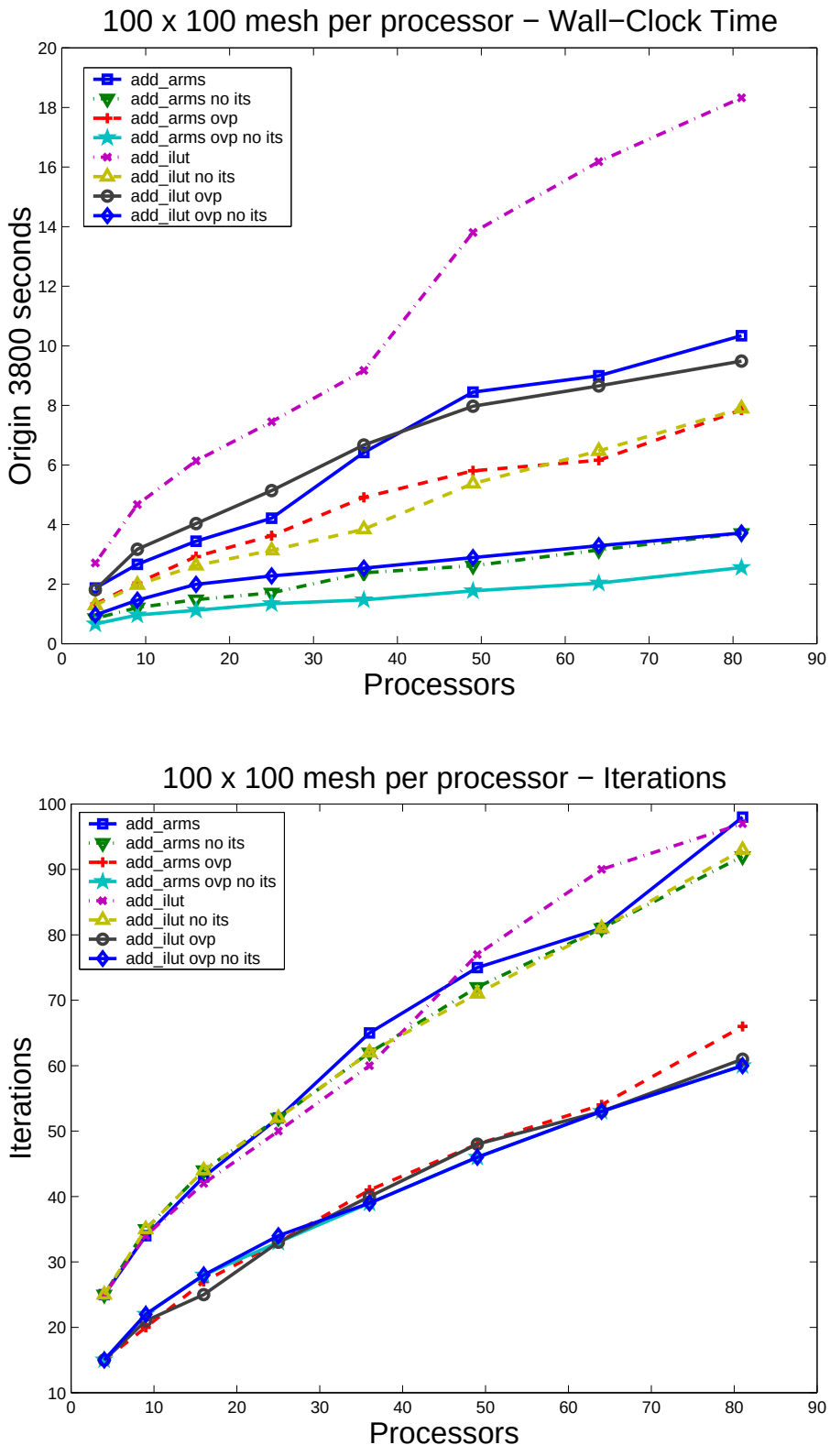

Figure 7: Solution times (top) and iterations (bottom) for a 2D PDE problem with the fixed subproblem size using variations of Additive Schwarz preconditioner.

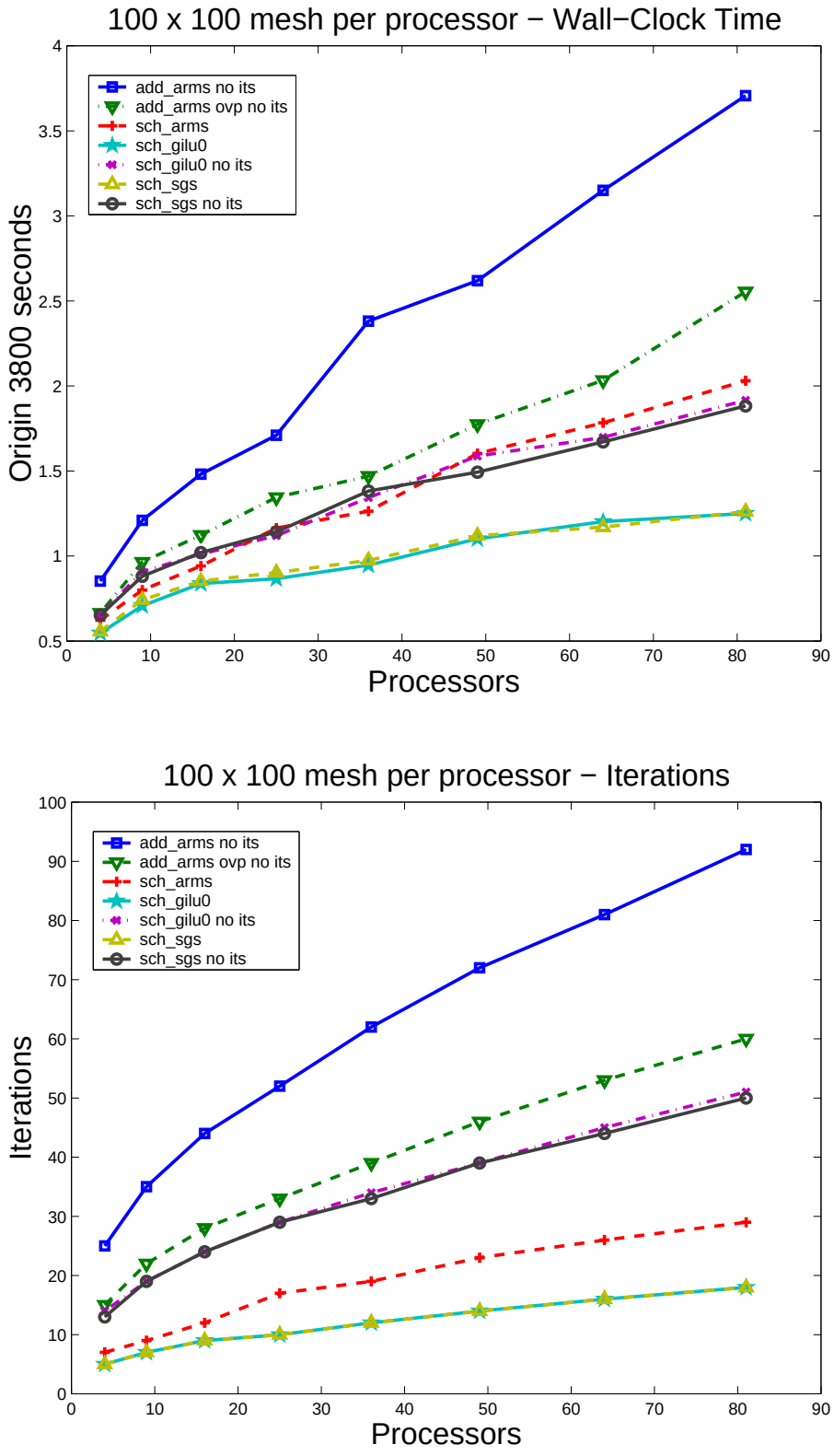

Figure 8: Solution times (top) and iterations (bottom) for a 2D PDE problem with the fixed subproblem size using four different preconditioners.

Table 1: Comparison of preconditioners for 3D PDE problem

	NP = 8		NP = 27		NP = 64	
Preconditioner	`iter`	`time`	`iter`	`time`	`iter`	`time`
`add_arms no its`	24	15.64	32	19.31	41	25.72
`add_arms no its ovp`	13	13.58	20	18.05	26	22.37
`sch_arms`	10	15.10	14	19.35	20	30.31
`sch_gilu0`	8	13.9	11	18.26	15	31.35
`sch_gilu0 no its`	15	12.98	22	16.72	29	20.93
`sch_sgs`	8	15.68	11	20.73	14	28.83
`sch_sgs no its`	13	13.72	19	17.12	24	22.00

term along the z direction. The PDE is set on a cube with Dirichlet boundary conditions, and it is discretized with a seven-point centered finite-difference scheme on a $n_x \times n_y \times n_z$ grid, excluding boundary points.

The experiments have been performed on 8, 27, and 64 processors of an IBM SP with the same parameters as in two-dimensional case. In Table 1, the columns `iter` and `time` contain iterations numbers and total times, respectively. It can be seen that, for this problem type, `sch_gilu0 no its` outperforms other methods mainly because it requires less computational work.

5.4 Comparison of pARMS and PSPASES

We now compare pARMS with a parallel sparse direct solver. Among the parallel solvers available to us, we selected to use PSPASES (Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems) [12]. Because PSPASES is a static code in the sense that it does a symbolic factorization prior to the numerical factorization, it is extremely efficient. On the other hand the code is restricted to solving Symmetric Positive Definite matrices. Another restriction is that the number of processors should be a power of 2. For the tests, we use simply the Poisson equation

$$-\Delta u = f \tag{18}$$

on two-dimensional rectangular region with Dirichlet boundary conditions. The problem is discretized as in the previous examples, with centered finite differences. In the first experiment, we solve a two-dimensional Poisson equation and select the mesh size on each processor to be 100×100. PSPASES has been used with the default parameters supplied with the package distribution. In Table 2, TTIME, CTIME, FTIME, and STIME stand for total time, preconditioner construction time, factorization time, and solution time respectively. All times are in seconds on the SGI Origin 3800. As shown in Table 2, for two-dimensional problem and small numbers of processors (fewer than 64), PSPASES is better than pARMS. On 64 processors, pARMS is slightly faster than PSPASES. In the next test, we solve the Poisson equation on a three-dimensional cube with Dirichlet boundary conditions. The problem size is $30 \times 30 \times 30$ per processor. The timing results are presented in Table 3, in which the columns have the same meaning as in Table 2 and an

Table 2: Comparison of pARMS and PSPASES for a 2D problem

	sch_gilu0			PSPASES		
NP	TTIME	CTIME	STIME	TTIME	FTIME	STIME
4	0.78	0.47	0.29	0.67	0.63	0.045
8	0.99	0.56	0.41	0.75	0.70	0.05
16	1.20	0.55	0.64	1.14	1.07	0.07
32	1.74	0.69	1.04	1.77	1.68	0.09
64	2.97	0.79	2.17	3.19	3.03	0.16

Table 3: Comparison of pARMS and PSPASES on a 3D problem

	sch_gilu0			PSPASES		
NP	TTIME	CTIME	STIME	TTIME	FTIME	STIME
8	10.70	5.44	5.24	86.17	85.50	0.67
64	27.78	7.92	19.85	*	*	*

asterisk indicates that the direct solver failed due to insufficient memory. For this problem, sch_gilu0 is much faster than PSPASES, as Table 3 shows. This comparison gives only a rough idea on the relative performance of pARMS and PSPASES since pARMS is not yet an optimized code. In addition, the pARMS iterative solution of the system does not take advantage of the problem symmetry. Other direct solution codes we have tried were either too slow or not ready for a distributed parallel environment.

5.5 A problem issued from magnetohydrodynamic flow

The linear system solved in this section is a fairly hard problem which arises from Magnetohydrodynamic (MHD) flows. The flow equations are represented as coupled Maxwell's and the Navier-Stokes equations. The conservative magnetohydrodynamic system is modeled by the Maxwell equations, written as:

$$\frac{\partial \mathbf{B}}{\partial t} - \nabla \times (\mathbf{u} \times \mathbf{B}) + \eta \nabla \times (\nabla \times \mathbf{B}) + \nabla \mathbf{q} = 0 \tag{19}$$

$$\nabla \cdot \mathbf{B} = 0\,, \tag{20}$$

where η, $\mathbf{B}$, $\mathbf{u}$ and q are, respectively, the magnetic diffusivity coefficient, magnetic induction field, velocity field, and the scalar Lagrange multiplier for the magnetic-free divergence constraint. In fully coupled magnetohydrodynamics, this system is solved along with the incompressible Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \tag{21}$$

$$\nabla \cdot \mathbf{u} = 0\,, \tag{22}$$

where p, ν and $\mathbf{f}$ are, respectively, pressure, kinematic viscosity, and body force. The coupling between the two systems is through the body force $\mathbf{f} = \frac{1}{\mu}(\nabla \times \mathbf{B}) \times \mathbf{B}$, which

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.

It is uncommon to solve the fully coupled problem described by the equations (19-22) along with their coupling via the body forces, because this usually requires an excessive amount of memory. Instead, segregated approaches are often applied which alternatively solve the two coupled problems until a certain convergence criterion is satisfied. For time-dependent problems, these coupling iterations are embedded into the time-stepping procedure. For a few details on this problem, its discretization, and the segregated solution procedure see [25].

In the tests that follow, we solve problems which arise from the Maxwell equations only. In order to do this, a pre-set periodic induction field $\mathbf{u}$ is used in Maxwell's equation (19). 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 $\eta = 1$. Two systems were generated. The first one (denoted by MHD1) has $n = 485,597$ unknowns and $24,233,141$ nonzero elements. As was mentioned above the function q in (19) corresponds to Lagrange multipliers, which arise from imposing the magnetic-free divergence constraint. Its gradient should be zero at steady-state. The second system is obtained by simply imposing Dirichlet boundary conditions for q. This results in a slightly smaller linear system of size $n = 470,596$ equations and $nnz = 23,784,208$ nonzeros. As it turns out, the second system is slightly easier to solve by iterative methods than the first one. Though the actual right-hand side was supplied, we preferred to use an artificially generated one in order to check the accuracy of the process. A random initial guess was taken. Little difference in performance was seen when the actual right-hand and a zero vector initial guess were used instead.

The following input parameters have been chosen to solve this problem: the residual is to be reduced by 10^6, the FGMRES restart value is 60, and at most five inner iterations may be allowed at each preconditioner application to attain 0.01 accuracy of the residual norm reduction. The parameters for ARMS are as follows: the number of levels, the independent-set block size, fill-in factors, and dropping tolerances for the intermediate and the last levels are 5, 5000, 60, 60, 10^{-5}, and 10^{-4}, respectively. Large block size and fill-in factors are prompted by the hardness of the problem. The resulting total number of nonzero memory locations required for the preconditioner was about 2.5 times the number of nonzeros in the original matrix. We observed that all the methods without inner iterations experienced stagnation for the MHD1 problem. Additive Schwarz (`add_arms no its`) with or without overlap does not converge for any number of processors while the Schur global ILU(0) (`sch_gilu0 no its`) stagnates when executed on more than nine processors. On four and nine processors, `sch_gilu0 no its` converges in 188 and 177 iterations, respectively. On an IBM SP, this amounts to 2,223.43 and 1,076.27 seconds, respectively. This is faster than 2,372.44 and 1,240.23 seconds when five inner iterations are applied and the number of outer iterations decreases to 119 and 109 on four and nine processors, respectively. The benefits of iterating on the global Schur complement system are clear since the Schur complement-based preconditioners converge for all the processor numbers tested as indicated in Figure 9, which shows the timing results (top) and outer iteration numbers (bottom). This positive effect can be explained by the fact that the Schur complement system is computed with good accuracy.

Comparing the two preconditioners in Figure 9 (bottom), we observe that `sch_gilu0`

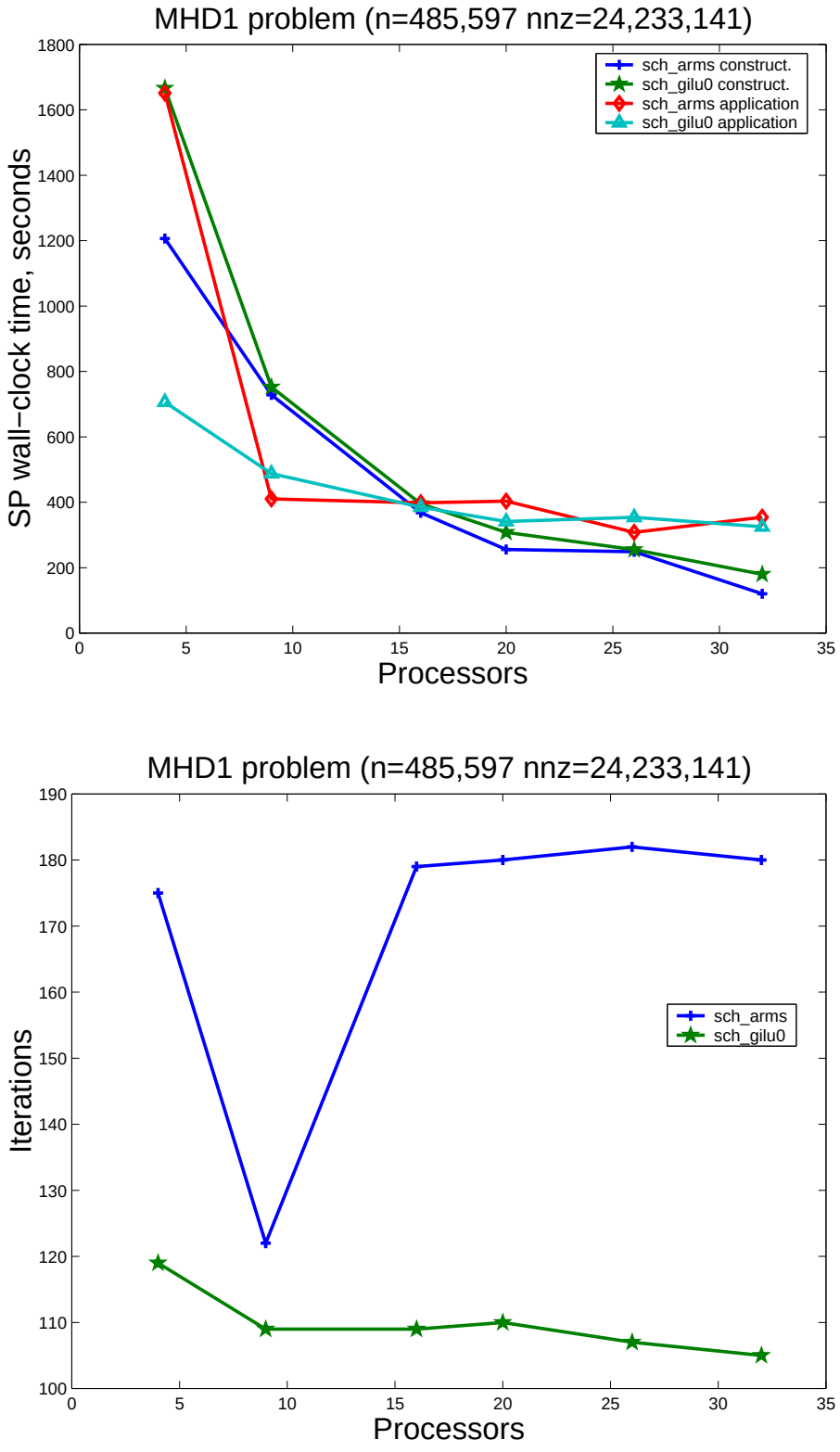

Figure 9: Timing results (top) and outer iterations (bottom) for the (fixed-size) MHD1 problem.

always requires fewer than `sch_arms` outer iterations and shows no iteration increase with the increase in processor numbers. The construction of `sch_arms`, however, is somewhat cheaper (cf. the "star" and "cross" curves in Figure 9, top) because it involves no interprocessor communication. The `sch_gilu0` application time advantage, seen for smaller processor numbers, vanishes as more processors are used (cf. the "rhombus" and "triangle" curves) since the cost of `sch_gilu0` increases compared with that of `sch_arms` as the number of processors increases, whereas the difference in numbers of iterations stays almost constant, for the MHD1 problem.

The MHD2 problem is easier to solve. In fact, this is a good example where iterating on the global Schur Complement is not mandatory. The convergence may be achieved with a simple Additive Schwarz-type preconditioner (`add_arms no its`) without overlapping, which, in this case, is quite competitive with respect to time (Figure 10, top). This is in spite of the rather high iteration counts for `add_arms no its`, relative to the other methods tested (Figure 10, bottom). All the Schur complement techniques produce fewer iterations to convergence. Applying `sch_gilu0` without inner iteration decreases solution times and makes `sch_gilu0 no its` faster than `add_arms not its` up to 32 processors.

In summary, the purpose of presenting these two examples of realistic problems is to show the flexibility of the developed pARMS framework. The difficulty and size of the problem at hand can call for a more powerful preconditioner, which can be readily provided by increasing fill-in and allowing more iterations on the global Schur complement variables. For easier problems (e.g., MHD2), Additive Schwarz methods or inexpensive variations of Schur complement techniques may be adequate and are provided as particular cases within the pARMS package.

6 Conclusion

The Algebraic Recursive Multilevel Solvers constitute a viable framework for developing a large array of preconditioners for parallel environments. This framework offers many possibilities ranging from simple Additive Schwarz with or without overlapping, to more complex Schur complement techniques. These techniques are developed for the general sparse irregularly structured linear systems. Experiments confirm what is already known from theory, namely that scalability as well as robustness may suffer when simple Additive Schwarz procedures are used. For harder and larger problems, it is essential to use a truly multilevel method, and this again is already known from general theory. It was also shown that inner iterations at the Schur complement level are essential in this case. While direct solvers are still very competitive for 2-D problems, they are likely to be far inferior, or even to fail, for reasonable size 3-D cases, as was shown in Section 5.3.

Not yet explored in this paper are methods which extend the ARMS recursive reduction into the interdomain interface variables. It remains to be seen whether this further step, which is much more complex to implement, is mandated for much larger problems.

Acknowledgments We would like to thank Azzeddine Soulaimani and Ridha Touihri from the "Ecole de Technologie Superieure, Université du Québec", for their generous help in supplying us with the realistic test problems of Section 5.5.

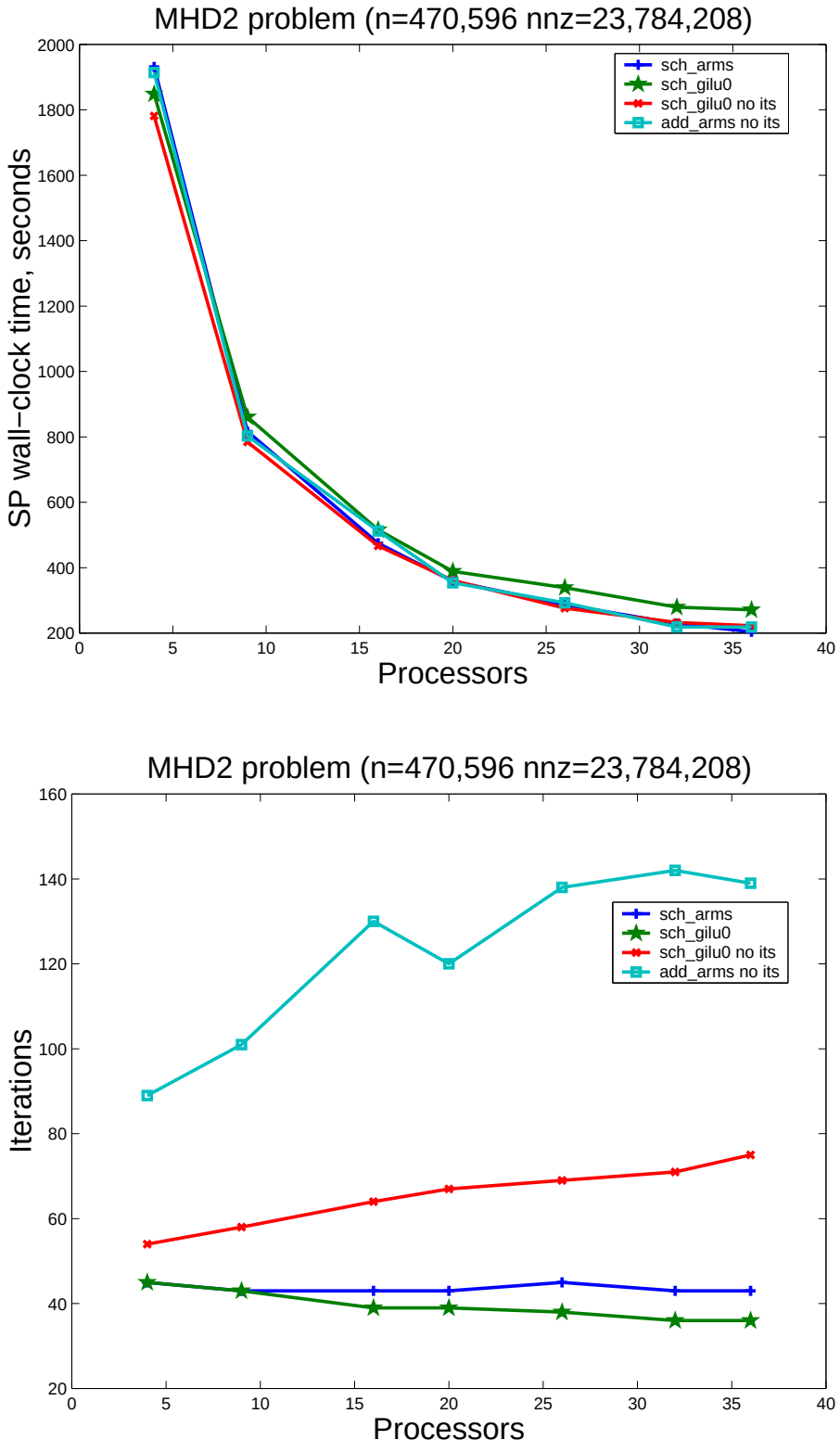

Figure 10: Solution times (top) and outer iterations (bottom) for the (fixed-size) MHD2 problem.

References

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

[2] M. Benzi, A. Frommer, R. Nabben, and D. Szyld. Algebraic theory of overlapping schwarz methods. *Numerische Mathematik*, Apr. 2001.

[3] P. E. Bjørstad. Multiplicative and Additive Schwarz Methods: Convergence in the 2 domain case. In Tony Chan, Roland Glowinski, Jacques Périaux, and O. Widlund, editors, *Domain Decomposition Methods*, Philadelphia, PA, 1989. SIAM.

[4] P. E. Bjørstad and O. B. Widlund. To overlap or not to overlap: A note on a domain decomposition method for elliptic problems. *SIAM Journal on Scientific and Statistical Computing*, 10(5):1053–1061, 1989.

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

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

[7] 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.

[8] X. C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21:792–797, 1999.

[9] Xiao-Chuan Cai, William D. Gropp, and David E. Keyes. A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems. *J. Numer. Lin. Alg. Appl.*, 1:477–504, June 1994.

[10] W. D. Gropp and B. F. Smith. Experiences with domain decomposition in three dimensions: Overlapping Schwarz methods. In *Proceedings of the Sixth International Symposium on Domain Decomposition Methods*, pages 323–333. AMS, 1993.

[11] D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. Technical Report (preprint), Old-Dominion University, Norfolk, VA, 2000.

[12] Mahesh Joshi, George Karypis, Vipin Kumar, Anshul Gupta, and Fred Gustavson. Pspases: Scalable parallel direct solver library for sparse symmetric positive definite linear systems – user's manual (version 1.0.3). Technical report, IBM Watson Res. center, 1999.

[13] S. Kuznetsov, G. C. Lo, and Y. Saad. Parallel solution of general sparse linear systems using PSPARSLIB. In Choi-Hong Lai, Petter Bjorstad, Mark Cross, and Olof B.

Widlund, editors, *Domain Decomposition XI*, pages 455–465, Bergen, Norway, 1999. Domain Decomposition Press.

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

[15] L. Little, Z. Li, H. G. Choi, and Y. Saad. Particle partitioning strategies for the parallel computation of solid-liquid flows. *Computers and Math with Applications*, 2001. accepted for publication, 03-01.

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

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

[18] Y. Saad and M. Sosonkina. Distributed schur complement techniques for general sparse linear systems. *J. Scientific Computing*, 21(4):1337–1356, 1999.

[19] Y. Saad and M. Sosonkina. Enhanced parallel multicolor preconditioning techniques for linear systems. In *9th SIAM Conference on Parallel Processing for Scientific Computing*, Philadelphia, PA, 1999. SIAM.

[20] Y. Saad, M. Sosonkina, and J. Zhang. Domain decomposition and multi-level type techniques for general sparse linear systems. In *Domain Decomposition Methods 10*, Providence, RI, 1998. American Mathematical Society.

[21] Y. Saad and B. Suchomel. 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.

[22] Y. Saad and J. Zhang. BILUTM: A domain-based multi-level block ILUT preconditioner for general sparse matrices. Technical Report umsi-98-118, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1998. appeared in SIMAX, vol. 21, pp. 279-299 (2000).

[23] 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.

[24] A. Soulaimani, A. Rebaine, and Y. Saad. Parallelization of the edge-based stabilized finite element method. In D. Keyes and al., editors, *Parallel Computational Fluid Dynamics, Teraflops, Optimization, and novel formulations*, pages 397–406. North Holland, 2000.

[25] A. Soulaimani, N. B. Salah, and Y. Saad. Enhanced GMRES acceleration techniques for some CFD problems. Technical Report umsi-2000-165, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2000. To appear in Int. Jour. of CFD.

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