Using the Parallel Algebraic Recursive Multilevel Solver in Modern Physical Applications[⋆]

X. Cai[a], Y. Saad[b], M. Sosonkina[c,*]

[a]*Simula Research Laboratory and University of Oslo, P.O. Box 1080, Blindern, N-0316 Oslo, Norway*

[b]*Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455, USA*

[c] *Department of Computer Science, University of Minnesota Duluth, 320 Heller Hall, 1114 Kirby Dr., Duluth, MN 55812, USA*

Abstract

The recently developed *Parallel Algebraic Recursive Multilevel Solver* (pARMS) is the subject of this paper. We investigate its behavior in solving large-scale sparse linear systems. In particular, we study the effect of a few parameters and different algorithms on the overall performance by conducting numerical experiments that stem from a number of realistic applications including magneto-hydrodynamics, nonlinear acoustic field simulation, and tire design.

Key words: Parallel algebraic multilevel preconditioning; Distributed sparse linear systems; Nonlinear acoustic field simulation; Magnetohydrodynamic flows; Tire design

[⋆] 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.
[*] Corresponding author.
Email addresses: `xingca@simula.no` (X. Cai), `saad@cs.umn.edu` (Y. Saad), `masha@d.umn.edu` (M. Sosonkina).
URLs: `http://www.ifi.uio.no/~xingca` (X. Cai), `http://www.cs.umn.edu/~saad` (Y. Saad), `http://www.d.umn.edu/~masha` (M. Sosonkina).

1 Introduction

Viable solutions of modern computational problems that arise in science and engineering should efficiently utilize the combined power of multi-processor computer architectures and concomitant algorithms. For many large-scale applications, solving large sparse linear systems is the most intensive computational task. The important criteria for a suitable solver include numerical efficiency, robustness, and good parallel performance. However, many existing parallel solvers are either designed for a particular problem class, such as symmetric positive definite linear systems, or are very application- and data format-specific. There is a limited selection of "general-purpose" distributed-memory iterative solution implementations. Among the better known packages that contain such implementations are PETSc [1] and *hypre* [6]. While the former focuses mainly on domain decomposition preconditioning techniques, the latter has a wide range of preconditioners including various distributed incomplete LU factorizations and an algebraic multigrid method. In this paper, we consider the Parallel Algebraic Recursive Multilevel Solver (pARMS) [14], which is a suite of distributed-memory iterative accelerator and preconditioners targeting the solution of general sparse linear systems. It adopts a general framework of distributed sparse matrices and relies on the solution of the resulting distributed Schur complement systems. We will discuss some issues related to the performance of pARMS on parallel computers for a few sparse linear systems that arise in realistic applications. Section 2 gives an introduction to the framework of distributed sparse linear systems, whereas Section 3 explains the ARMS/pARMS methods and discusses issues that arise especially in the pARMS method. Section 4 contains numerical experiments arising from three realistic applications. Finally, Section 5 presents some concluding remarks.

2 Distributed Sparse Linear Systems

The framework of distributed linear systems [18,21] provides an algebraic representation of the irregularly structured sparse linear systems arising in the Domain Decomposition methods [25]. A typical distributed system arises, e.g., 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 and assign a cluster of elements representing a physical sub-domain to one processor. Each processor then assembles only the local equations restricted to its assigned cluster of elements. In the case where the linear system is given algebraically, a graph containing vertices that correspond to the rows of the linear system can be partitioned. For both cases, the general assumption is that each processor holds a set of equations (rows of the global linear system) and the associated unknown variables.

Figure 1 shows a 'physical domain' viewpoint of a distributed sparse linear system, for which a vertex-based partitioning has been used without overlapping the unknowns. As is often done, we will distinguish between three types of variables: (1) Interior variables are those that are coupled only with local variables by the equations; (2) Inter-domain interface variables are those coupled with non-local (external) variables as well as local variables; and (3) External interface variables are those variables that belong to neighboring processors and are coupled with the above types of local variables. The local equations can be represented as shown in Figure 2. Note that these equations need not be contiguous in the original system. The matrix represented in Figure 2 can be viewed as a reordered version of the equations associated with a local numbering of the equations/unknowns pairs.

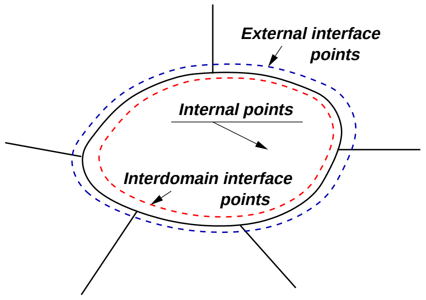

Fig. 1. A local view of a distributed sparse linear system

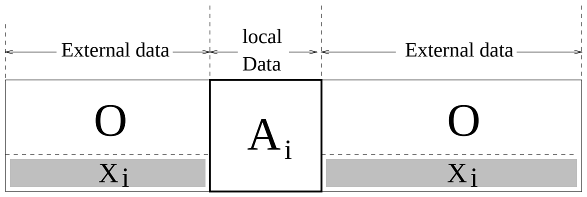

Fig. 2. The corresponding local matrices A_i and X_i

The rows of the matrix assigned to a certain processor have been split into two parts: a *local* matrix A_i which acts only on the local variables and an *interface* matrix X_i which acts only on the external interface variables. These external interface variables must be first received from neighboring processor(s) before a distributed matrix-vector product can be completed. Thus, each local vector of unknowns x_i $(i = 1, \dots, p)$ is also split into two parts: the sub-vector u_i of interior variables followed by the sub-vector y_i of inter-domain interface variables. The right-hand side b_i is conformally split into the sub-vectors f_i and g_i,

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix} \; ; \quad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix} \; . \tag{1}$$

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

$$A_i = \left(\begin{array}{c|c} B_i & F_i \\ \hline E_i & C_i \end{array}\right) . \tag{2}$$

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

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix} . \tag{3}$$

The term $E_{ij}y_j$ is the contribution to the local equations from the neighboring sub-domain number j and N_i is the set of sub-domains that are neighbors to sub-domain i. The sum of these contributions, seen on the left side of (3), is the result of multiplying the interface matrix X_i by the external interface variables. It is clear that the result of this product will affect only the inter-domain interface variables as is indicated by the zero in the upper part of the second term on the left-hand side of (3). For practical implementations, the sub-vectors 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 (3). 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 (3) becomes

$$w_i = A_i x_i + X_i y_{i,ext} . \tag{4}$$

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

2.1 Additive Schwarz Preconditioning

Preconditioners for distributed sparse linear systems are best designed from the local data 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 a 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 **2.1** *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, where 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. In particular, a multi-level ILU type procedure could be used to solve $A_i\delta_i = r_i$ approximately [16,3,4,24,23].

2.2 Schur Complement Techniques

Schur complement techniques refer to methods which iterate on the inter-domain interface unknowns only, implicitly using interior 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 variables u_i from (3). Extracting from the first equation $u_i = B_i^{-1}(f_i - F_i y_i)$ yields, upon substitution in the second equation,

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i', \tag{5}$$

where S_i is the "local" Schur complement

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{6}$$

The equations (5) for all sub-domains i ($i = 1, \ldots, p$) constitute a global system of equations involving only the inter-domain interface unknown vectors y_i. This global reduced system has a natural block structure related to the inter-domain interface points in each sub-domain:

$$\begin{pmatrix} S_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2 & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \ldots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}. \tag{7}$$

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 (3), are sparse.

The system (7) can be written as $Sy = g'$, where y consists of all inter-domain 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 [19] is to exploit methods that *approximately* solve the reduced system (7) to develop preconditioners for the original (global) distributed system. Once the global Schur complement system (5) is (approximately) solved, each processor will compute the u-part of the solution vector (see (1)) by solving the system $B_i u_i = f_i - E_i y_i$ obtained by substitution from (3). In summary, a Schur complement iteration may be expressed by the following algorithm:

Algorithm **2.2** *Schur Complement Iteration*

```
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 $B_i u_i = f_i - E_i y_i$.
```

For convenience, (5) is rewritten as a preconditioned system with the diagonal blocks:

$$y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} \left[g_i - E_i B_i^{-1} f_i \right] . \tag{8}$$

This can be viewed as a block-Jacobi preconditioned version of the Schur complement system (7). This global system can be solved by a GMRES-like accelerator, requiring a solve with S_i at each step.

3 Parallel Algebraic Recursive Multilevel Solver

Multi-level Schur complement techniques available in pARMS [14] are based on techniques which exploit block independent sets, such as those described in [22]. The idea is to create another level of partitioning of each sub-domain. An illustration is shown in Figure 3, which distinguishes one more type of interface variables: *local interface variables*, where we refer to local interface points as interface points between the sub-sub-domains. Their couplings are all local to the processor and so these points do not require interprocessor communication. These sub-sub-domains are not obtained by a standard partitioner but rather by the *block independent set* reordering strategy utilized by ARMS [22].

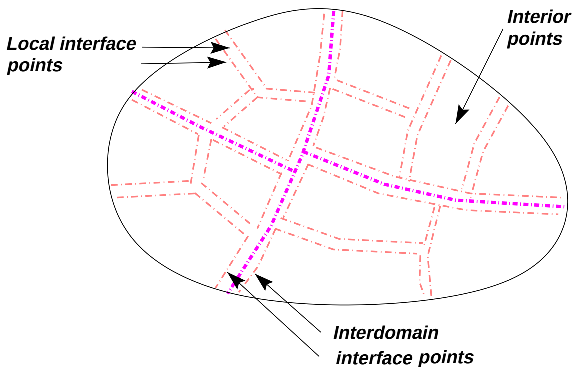

Fig. 3. A two-level partitioning of a domain

3.1 ARMS and pARMS

In order to explain the multilevel techniques used in pARMS, it is necessary to discuss the sequential multilevel ARMS technique. In the sequential ARMS, the matrix coefficient of the at the l-th level is reordered in the from

$$P_l A_l P_l^T = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} , \tag{9}$$

where P_l is a "block-independent-set permutation", which can be obtained in a number of ways. At the zero-th level ($l = 0$), the matrix A_l is the original coefficient matrix of the linear system under consideration. The above permuted matrix is then approximately factored as

$$P_l A_l P_l^T \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & A_{l+1} \end{pmatrix} , \tag{10}$$

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,

$$A_{l+1} \approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l). \tag{11}$$

During the factorization process, approximations to the matrices for obtaining the Schur complement (11) are computed.

To define a recursive multilevel strategy, all we need is a strategy for solving the reduced system obtained by eliminating the unknown associated with the block B_l. This leads to a certain system with the matrix A_{l+1}. Now recursivity is invoked and this system is partitioned again in the form (9) in which l is replaced by $l+1$. At the last level, the reduced system is solved using GMRES preconditioned with ILUT [17]. In the parallel version of ARMS, the same overall strategy is used except that now the global block-independent sets are across domains.

Consider a one-level pARMS for simplicity. In the first level reduction, the matrix A_1 that is produced, will act on all the interface variables, whether local or inter-domain. Thus, a one-level pARMS would solve for these variables and then obtain the interior variables in each processor without communication. We denote by*expanded Schur complement* the system involving the matrix A_1 that acts on inter-domain and local interface unknowns. For a more detailed description of pARMS see [14].

3.2 Diagonal Shifting in pARMS

Extremely ill-conditioned linear systems are difficult to solve by iterative methods. A possible source of difficulty is due to the ineffective preconditioning of such systems. The preconditioner may become unstable (i.e., has large norm of its inverse). To stabilize the preconditioner, a common technique is to shift the matrix A by a scalar and use this shifted matrix $A + \alpha I$ during preconditioning, see, e.g., [15]. Because the matrix is shifted, its preconditioner might be a rather accurate approximation of $A + \alpha I$. It is also more likely to be stable. However, for large shift values, the preconditioner might not represent accurately the original matrix A. So the choice of the shift value is important and leads to a trade-off between accuracy and stability of the preconditioner. We have considered this trade-off in [10,20]. In [7], a strong correlation between stability of the preconditioner and the size of $\mathcal{E} = \log(\|(LU)^{-1}\|_{\text{inf}})$ is shown and is suggested as a practical means of evaluating the quality of a preconditioner. We can inexpensively compute $\mathcal{E}_\alpha = \log(\|(LU)^{-1}e\|_1)$, where e is a vector of all ones and LU is an incomplete LU factors of $A + \alpha I$. The estimate $\mathcal{E}_\alpha$ can be used in choosing a shift value: if this estimate is large, then we increase shift value and recompute (adjust) the preconditioner. Note that efficient techniques for updating a preconditioner when a new shift value is provided is beyond the scope of this paper. One such technique has been outlined in [8].

In the pARMS implementation, we have adapted a shifting technique for a distributed representation of linear system. Specifically, we perform the shifting and norm $\mathcal{E}_\alpha$ calculation in each processor independently. Thus, each processor i can have a different shift value depending on the magnitude of its $\mathcal{E}^i_\alpha$. Such an implementation is motivated by the observation that shifting is especially important for diagonally non-dominant rows, which can be distinguished among other rows by a local procedure. In each processor, the choice of shift value is described by the following pseudo-code:

Algorithm **3.1** *Matrix shifting*

```
1.  Select initial shift $\alpha \geq 0$: $B = A + \alpha I$.
2.  Compute parallel preconditioner $M$ for $B$.
3.  Calculate local $\mathcal{E}_\alpha$.
4.  If $\mathcal{E}_\alpha$ is large,
5.      Increase shift $\alpha$;
6.      Adjust preconditioner.
```

Note that in Line 6 of Algorithm 3.1, depending on the type of preconditioner, the adjustment operation may be either local or global. For example, Additive Schwarz type preconditioners may perform adjustments independently per processor, whereas all the processors may need to participate in the adjustment of a Schur complement preconditioner. In addition, Lines 3 – 6 may need to be repeated several times.

4 Numerical Experiments

In this section we describe a few realistic applications, which give rise to large irregularly structured linear systems that are challenging to solve by iterative methods. Such applications as ultrasound simulation, magnetohydrodynamics, and tire design are considered. The linear systems arising in ultrasound simulation were generated using Diffpack, which is an object-oriented environment for scientific computing, see [9,13]. The magnetohydrodynamics application has been generously provided by Azzeddine Soulaimani and Ridha Touihri from the "Ecole de Technologie Superieure, Université du Québec", and the linear systems arising in tire design have been supplied by John T. Melson of Michelin Americas Research and Development Corporation. All the numerical experiments have been performed on the IBM SP system at the Minnesota Supercomputing Institute. Each computing node (of type Nighthawk) of the IBM SP system has four 222 MHz Power3 processors sharing 4GB of memory and all the nodes are connected through a high performance switch. We have tested several preconditioning techniques available in our pARMS code when solving the linear systems iteratively. As the criterion for convergence, we have always required that the residual is reduced by a factor of 10^6.

Before describing the applications, let us first introduce some notation for the sake of convenience.

`add_ilut` denotes an additive Schwarz procedure 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` is similar to `add_ilut` but ILU(k) is used as a preconditioner instead of ILUT.

`add_arms` is similar to `add_ilut` but ARMS is used as a preconditioner for local systems.

`sch_gilu0` denotes a method that is based on approximately solving the expanded Schur complement system with a global ILU(0)-preconditioned GMRES. The ILU(0) preconditioning requires a global order (referred to as a schedule in [11]) to traverse the equations. A global multicoloring of the domains is used for this purpose as is often done with global ILU(0).

`no_its` is the suffix added to above methods whenever no local (inner) iterations are used.

`sh` is the suffix added to above methods if they use shifted original matrix for the preconditioner construction.

4.1 Simulation of 3D Nonlinear Acoustic Fields

The propagation of 3D ultrasonic waves in a nonlinear medium can be modeled by the following system of nonlinear PDEs:

$$\nabla^2\varphi - \frac{1}{c^2}\frac{\partial^2\varphi}{\partial t^2} + \frac{1}{c^2}\frac{\partial}{\partial t}\left[(\nabla\varphi)^2 + \frac{B/A}{2c^2}\left(\frac{\partial\varphi}{\partial t}\right)^2 + b\nabla^2\varphi\right] = 0, \tag{12}$$

$$p - p_0 = \rho_0\frac{\partial\varphi}{\partial t}, \tag{13}$$

where the primary unknowns are the velocity potential φ and pressure p. For the involved parameters, c is the speed of sound, ρ_0 is the density, p_0 is the initial pressure, b is the absorption parameter, and B/A is the nonlinearity parameter. The above mathematical model is to be supplemented with suitable initial and boundary conditions.

The numerical scheme consists of using finite elements in the spatial discretization and finite differences for the temporal derivatives. At each time level, the discretization of (12) gives rise to a system of nonlinear algebraic equations involving φ from three consecutive time levels. We apply Newton-Raphson iterations for the nonlinear system. We refer to [5] and the references therein for more information on the mathematical model and the numerical solution method. As a particular numerical test case, we use a 3D domain: $(x, y, z) \in [-0.004, 0.004] \times [-0.004, 0.004] \times [0, 0.008]$. On the face of $z = 0$, there is a circular transducer with radius $r = 0.002$, i.e., the pressure p is given within the circle. On the rest of the boundary we use a non-reflective boundary condition.

We consider solving the linear system during the first Newton-Raphson iteration at the first time level. The linear system has $185,193$ unknowns and the sparse matrix contains $11,390,625$ nonzero entries. Figure 4 presents the iteration numbers (left) and solution times (right) needed for solving this linear system. Two preconditioning techniques, `sch_gilu0_no_its` and `add_arms_no_its` have been tested on various numbers of processors.

It is observed that `sch_gilu0_no_its` preconditioning consistently leads to a faster convergence than `add_arms_no_its`. Both methods, however, show almost no increase in iterations when the number of processors is increased. The timing results are slightly better for `sch_gilu0_no_its` preconditioner except for the 16-processor case.

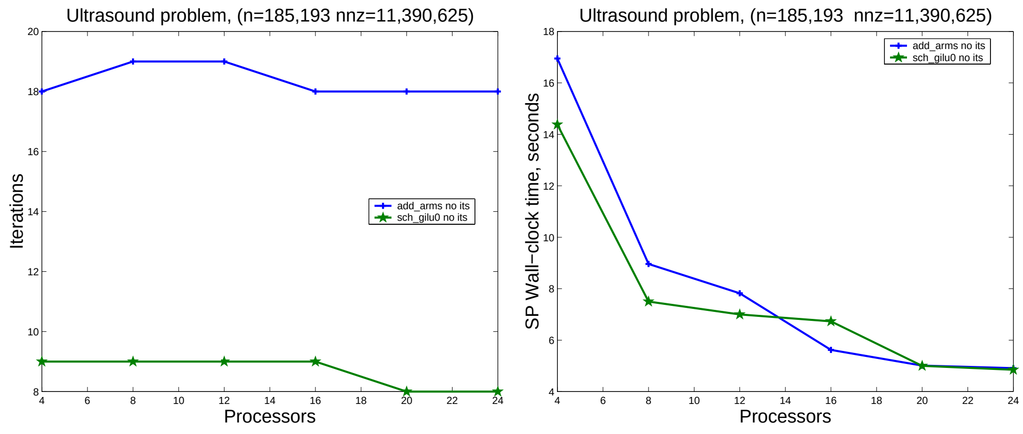

Fig. 4. Iterations (left) and timing results (right) for the (fixed-size) ultrasound problem

4.2 Simulation of Magnetohydrodynamic Flow

In [14], we have described the solution of a rather hard problem which arises from simulating magnetohydrodynamic (MHD) flows. The mathematical model describing the flow consists of the Maxwell equations coupled with the incompressible Navier-Stokes equations. Here, we provide only a brief outline of a sample problem along with its solution and study the solution process when shifting techniques are used. The conservative magnetohydrodynamic system is modeled by the Maxwell equations, written as:

$$\frac{\partial \mathbf{B}}{\partial t} - \nabla \times (\mathbf{u} \times \mathbf{B}) + \eta \nabla \times (\nabla \times \mathbf{B}) + \nabla q = 0 \ , \tag{14}$$

$$\nabla \cdot \mathbf{B} = 0 \ , \tag{15}$$

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

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f} \ , \tag{16}$$

$$\nabla \cdot \mathbf{u} = 0 \ , \tag{17}$$

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 (14-17) 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, we refer to [27].

Here, we only consider solving the linear systems arising from the Maxwell equations. In order to do this, a pre-set periodic induction field $\mathbf{u}$ is used in Maxwell's equation (14). 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 sparse matrix of the resulting linear system (denoted by MHD1) has $n = 485,597$ unknowns and $24,233,141$ nonzero entries. The function q in (14) corresponds to Lagrange multipliers, which arise from imposing the magnetic-free divergence constraint. Its gradient should be zero at steady-state. 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. For the details on the values of the input parameters see [14].

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 the IBM SP system, 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 tested processor numbers as indicated in Figure 5, which presents the timing results (left) and outer iteration numbers (right). This positive effect can be explained by the fact that the Schur complement system is computed with good accuracy. Figure 5 also shows the usage of the shift value $\alpha = 0.1$ in the `sch_gilu0_sh` preconditioner construction. For this problem, shifting does not help convergence and results in larger numbers of outer iterations. Since a good convergence rate is achieved without shifting of the original matrix, the shift value applied in `sch_gilu0_sh` may be too large and the resulting preconditioner may not be a good approximation of the original matrix. The number of nonzeros in `sch_gilu0_sh`, however, is smaller than in `sch_gilu0`. Therefore, the construction of `sch_gilu0_sh` is always cheaper, and `sch_gilu0_sh` appears to be competitive for small processor numbers.

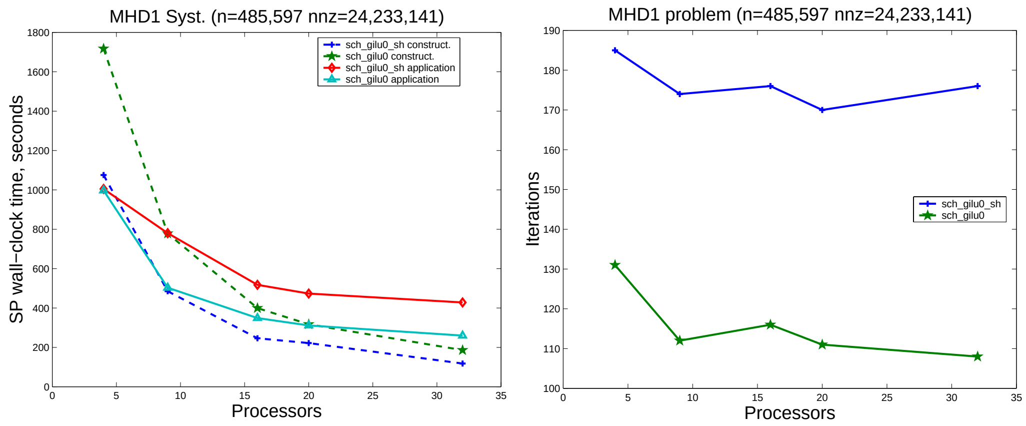

Fig. 5. Solution times (left) and outer iteration numbers (right) for the MHD1 problem with and without diagonal shifting

4.3 Linear Systems Arising in Tire Design

Tire static equilibrium computation is based on a 3D finite element model with distributed loads (see, e.g., [2]). The computation involves minimizing the potential energy $\Pi(\mathbf{u})$ with respect to the displacement field $\mathbf{u} = (u_1, u_2, u_3)$ subject to nonlinear boundary conditions, which change the symmetry of a tire. The equilibrium equations of the model are obtained by setting the variation $\delta\Pi(\mathbf{u})$ to zero, or equivalently

$$\nabla\ \Pi(\mathbf{u}) = 0.$$

The Jacobian matrix of the equilibrium equations is obtained by finite difference approximations. The distributed load is scaled by a (loading) parameter

Table 1
Solution of tire model $\mathcal{M}$ on four processors using different parallel iterative methods. No local iterations are used

Method	α_{final}	$\max_i \mathcal{E}^i_{\alpha=0}$	`Iter`	`Time`
`add_ilu(2)`	0.1	45	475	175.19
`add_ilut`	0.1	116	476	116.55
`sch_gilu0`	0.2	146	566	99.99

λ, and as λ varies the static equilibrium solutions trace out a curve. The difficulty of the finite element problems and concomitant linear systems varies considerably along this equilibrium curve, as well as within the nonlinear iterations to compute a particular point on this curve.

In [26], the problems of varying matrix characteristics are considered. All of the problems pose a challenge for iterative methods since the treatment of stationary solutions of rotation makes the systems extremely ill-conditioned during the nonlinear convergence process. It has been observed that an acceptable convergence was achieved *only* when a rather large shift was applied to the matrix diagonal to stabilize the preconditioner. The size of the shift is very important: while making the preconditioner more stable, large shift values cause the preconditioner to be a poor approximation of the original matrix.

Table 1 shows the results of a few experiments, in which we use pARMS on a sample linear system, medium tire model $\mathcal{M}$, with $49,800$ unknowns and, on average, 84 nonzeros per row in the matrix. In pARMS, a shift α is chosen *automatically*: starting with the zero shift, the preconditioner is reconstructed with a new shift (augmented by 0.1) if the estimate $\mathcal{E}^i_\alpha$ of the preconditioner inverse is large (greater than 7). In Table 1, we list the final value of α, the maximum $\mathcal{E}^i_\alpha$ among all the processors when $\alpha = 0$, the number `Iter` of iterations needed for converge, and the preconditioner application time `Time` spent when running on four processors. Metis [12] has been used for the partitioning of the problem among processors.

Note that the `sch_gilu0` preconditioner is more ill-conditioned initially and thus causes two augmentations of the shift value. For `sch_gilu0`, the larger number of outer iterations (566) may be attributed to the resulting preconditioner being much sparser than the other preconditioners tested. However, this difference in the iteration numbers sustains the timing advantage of a sparser preconditioner despite the communication overhead incurred by `sch_gilu0`.

Due to the difficulty of this problem, which is also unpredictably affected by partitioning, the convergence was not observed consistently on any processor numbers. For example, no convergence has been achieved on eight processors for moderate values of α.

5 Concluding Remarks

In this paper, we have studied the performance of a recently developed pARMS code in several realistic applications. For all the problems considered, it is beneficial to use preconditioners based on Schur complement techniques, enhanced by a local multi-level procedure. In addition, a few inner (local to a sub-domain) preconditioning iterations enhance convergence for a problem arising from a magneto-hydrodynamics application.

We have also proposed an implementation of matrix shifting in the framework of distributed linear systems. It allows a shift value to be assigned independently in each sub-domain. An automatic procedure for the shift value selection has also been implemented to stabilize the distributed preconditioner and overcome stagnation. We would like to underline the flexibility of the pARMS framework, which, with a proper selection of input parameters, allows to choose among many available options for solving real-world problems.

References

[1] S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.0, Argonne National Laboratory, 2001.

[2] K.J. Bathe and E.L. Wilson. *Numerical Methods in Finite Element Analysis.* Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[3] 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.

[4] 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.

[5] X. Cai and Å. Ødegård. Parallel simulation of 3D nonlinear acoustic fields on a Linux-cluster. *Proceedings of the* Cluster 2000 *conference.*

[6] E. Chow, A. Cleary, and R. Falgout. *hypre* User's manual, version 1.6.0. Technical Report UCRL-MA-137155, Lawrence Livermore National Laboratory, Livermore, CA, 1998.

[7] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics*, 87:387–414, 1997.

[8] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19:995–1023, 1998.

[9] Diffpack World Wide Web home page. *http://www.nobjects.com.*

[10] P. Guillaume, Y. Saad, and M. Sosonkina. Rational approximation preconditioners for general sparse linear systems. Technical Report umsi-99-209, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1999.

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

[12] G. Karypis and V. Kumar, *Metis: Unstructured Graph Partitioning and Sparse Matrix Ordering System*, Tech. Rep., Department of Computer Science, University of Minnesota, 1995; also available online at http://www.cs.umn.edu/˜karypis/metis.

[13] H. P. Langtangen. *Computational Partial Differential Equations – Numerical Methods and Diffpack Programming.* Springer-Verlag, 1999.

[14] 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.

[15] T.A Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of computation*, 32:473–497, 1980.

[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 A. Malevsky. PSPARSLIB: A portable library of distributed memory sparse iterative solvers. In V. E. Malyshkin et al., editor, *Proceedings of Parallel Computing Technologies (PaCT-95), 3-rd international conference, St. Petersburg, Russia, Sept. 1995*, 1995.

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

[20] Y. Saad and M. Sosonkina. Enhanced preconditioners for large sparse least squares problems. Technical Report umsi-2001-1, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.

[21] Y. Saad and M. Sosonkina. Solution of distributed sparse linear systems using PSPARSLIB. In B. Kågström et al., editors, *Applied Parallel Computing, PARA'98*, Lecture Notes in Computer Science, No. 1541, pages 503–509, Berlin, 1998. Springer-Verlag.

[22] 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.

[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] 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, 2000.

[25] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, New York, 1996.

[26] M. Sosonkina, J. T. Melson, Y. Saad, and L. T. Watson. Preconditioning strategies for linear systems arising in tire design. *Numer. Linear Alg. with Appl.*, 7:743–757, 2000.

[27] A. Soulaimani, N. B. Salah, and Y. Saad. Enhanced GMRES acceleration techniques for some CFD problems. *Int. Journal of CFD*, 16:1–20, 2002.