# Parallel Iterative Methods in Modern Physical Applications *

X. Cai[1], Y. Saad[2], and M. Sosonkina[3]

[1] Simula Research Laboratory and University of Oslo, P.O. Box 1080, Blindern, N-0316 Oslo, Norway
xingca@ifi.uio.no
[2] University of Minnesota, Minneapolis, MN 55455, USA
saad@cs.umn.edu
[3] University of Minnesota, Duluth, MN 55812, USA
masha@d.umn.edu

**Abstract.** Solving large sparse linear systems is a computationally-intensive component of many important large-scale applications. We present a few experiments stemming from a number of realistic applications including magneto-hydrodynamics structural mechanics, and ultrasound modeling, which have become possible due to the advances in parallel iterative solution techniques. Among such techniques is a recently developed Parallel Algebraic Recursive Multilevel Solver (`pARMS` ). This is a distributed-memory iterative method that adopts the general framework of distributed sparse matrices and relies on solving the resulting distributed Schur complement systems. We discuss some issues related to parallel performance for various linear systems which arise in realistic applications. In particular, we consider the effect of different parameters and algorithms on the overall performance.

## 1 Distributed Sparse Linear Systems

The viewpoint of a distributed linear system generalizes the Domain Decomposition methods to irregularly structured sparse linear systems. 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 by a graph partitioner and assign a cluster of elements representing a physical sub-domain to a processor. The general assumption is that each processor holds a set of equations (rows of the global linear system) and the associated unknown variables. 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 the external interface variables. These external interface variables must be first received from neighboring processor(s) before a distributed

* 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.

matrix-vector product can be completed. Thus, each local vector of unknowns $x_i$ $(i = 1, \ldots, p)$ 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 $g_i$. The local matrix $A_i$ residing in processor $i$ is block-partitioned according to this splitting. So the local equations can be written as follows:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{1}$$

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 (1), 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 (1). 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 (1). 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 (1) becomes

$$w_i = A_i x_i + X_{i,ext} y_{i,ext}. \tag{2}$$

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 $x_i$.

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. The local systems 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.

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 (1). Extracting from the first equation $u_i = B_i^{-1}(f_i - F_i y_i)$ yields, upon substitution in the second equation,

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i', \tag{3}$$

where $S_i$ is the "local" Schur complement $S_i = C_i - E_i B_i^{-1} F_i$. The equations (3) 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:

$$\begin{pmatrix} S_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2 & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \ldots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g'_1 \\ g'_2 \\ \vdots \\ g'_p \end{pmatrix}. \tag{4}$$

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 (1), are sparse.

The system (4) 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 [13] is to exploit methods that *approximately solve the reduced system* (4) to develop preconditioners for the original (global) distributed system. Once the global Schur complement system (3) is (approximately) solved, each processor will compute the $u$-part of the solution vector by solving the system $B_i u_i = f_i - E_i y_i$ obtained by substitution from (1).

For convenience, (3) is rewritten as a preconditioned system with the diagonal blocks:

$$y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} \left[ g_i - E_i B_i^{-1} f_i \right]. \tag{5}$$

This can be viewed as a block-Jacobi preconditioned version of the Schur complement system (4). This global system can be solved by a GMRES-like accelerator, requiring a solve with $S_i$ at each step.

## 2 Parallel Implementation of ARMS (pARMS)

Multi-level Schur complement techniques available in pARMS [9] are based on techniques which exploit block independent sets, such as those described in [15]. The idea is to create another level of partitioning of each sub-domain. An illustration is shown in Figure 1, 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 communication. These sub-sub-domains are not obtained by a standard partitioner but rather by the *block independent set* reordering strategy utilized by ARMS [15].

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

$$P_l A_l P_l^T = \begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix}, \tag{6}$$

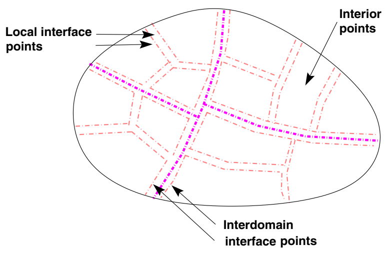

**Fig. 1.** A two-level partitioning of a domain

where $P_l$ is a "block-independent-set permutation", which can be obtained in a number of ways. At the 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

$$P_l A_l P_l^T \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \times \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & A_{l+1} \end{pmatrix}, \tag{7}$$

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 $C_l$,

$$A_{l+1} \approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l). \tag{8}$$

During the factorization process, approximations to the matrices for obtaining the Schur complement (8) are computed. The system with $A_{l+1}$ is partitioned again in the form (6) in which $l$ is replaced by $l + 1$. At the last level, the reduced system is solved using GMRES preconditioned with ILUT [12]. 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 [9].

## 2.1 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., [10]. 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 [6, 14]. In [3], 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 $LU$ are 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 are beyond the scope of this paper. One such technique has been outlined in [4].

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 $\mathcal{E}_{\alpha_i}$. 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 **21** *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.       Choose $\alpha' > \alpha$;
6.       Adjust preconditioner.
```

Note that in Line 6 of Algorithm 21, 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 be repeated several times.

## 3 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. The linear systems arising in ultrasound simulation were generated using Diffpack, which is an object-oriented environment for scientific computing, see [5, 8]. The magnetohydrodynamics application has been provided by A. Soulaimani and R. Touihri from the "Ecole de Technologie Superieure, Université du Québec", and the linear systems arising in tire design have been supplied by J. T. Melson of Michelin Americas Research and Development Corporation. For the sake of convenience, let us introduce some notation.

`add_ilut`. 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`. 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_gilu0`. This method 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 [7]) 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).

The suffixes `no_its` or `sh` are added to the above methods when no local (inner) iterations are used or when the shifted original matrix is used for the preconditioner construction, respectively.

## 3.1 Simulation of 3D Nonlinear Acoustic Fields

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

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 gives rise to a system of nonlinear algebraic equations involving $\varphi$ from three consecutive time levels. We apply Newton-Raphson iterations for the nonlinear system. We refer to [2] 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 $11,390,625$ nonzero entries. Figure 2 presents the iteration numbers (left) and solution times (right) of this linear system on the IBM SP at the Minnesota Supercomputing Institute. Four 222 MHz Power3 processors share 4GB of memory per (Nighthawk) node and are connected by a high performance switch with other nodes. Two preconditioning techniques, `sch_gilu0_no_its` and `add_arms_no_its` have been tested on various processor numbers. Original right hand side and random initial guess have been taken.

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 with increase in processor numbers. The timing results are slightly better for `sch_gilu0_no_its` preconditioner except for the 16-processor case.

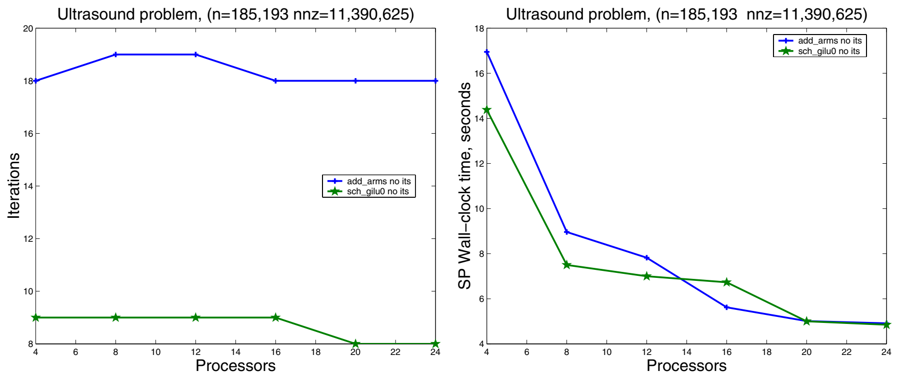

**Fig. 2.** Iterations (left) and timings results (right) for the ultrasound problem

## 3.2 A Problem Issued From Magnetohydrodynamic Flow

In [9], we have described the solution of a rather hard problem which arises from Magnetohydrodynamic (MHD) flows. The flow equations are represented as coupled Maxwell's and the Navier-Stokes equations. Here, we provide only a brief outline of a sample problem along with its solution and note the solution process when shifting techniques are used.

We solve linear systems which arise from the Maxwell equations only. In order to do this, a pre-set periodic induction field is used in Maxwell's equation. 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 linear system (denoted by MHD1) has $n = 485,597$ unknowns and $24,233,141$ nonzero entries. The gradient of the function corresponding to Lagrange multipliers 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 [9].

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 3, 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 3 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.

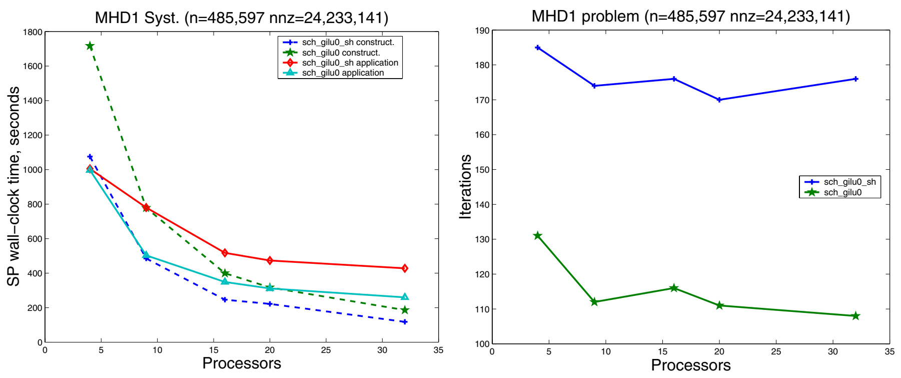

**Fig. 3.** Solution times (left) and outer iterations (right) for the (fixed-size) MHD1 problem with and without diagonal shifting

## 3.3 Linear Systems Arising in Tire Design

Tire static equilibrium computation is based on a 3D finite element model with distributed loads. Computation of static equilibrium involves minimizing the potential energy $\Pi(u)$ with respect to finite element nodal displacements $u^i (i = 1, 2, 3)$ subject to nonlinear boundary conditions, which change the symmetry of a tire. The equilibrium equations of the model are obtained by setting the variation $\delta\Pi(u)$ to zero. equivalently The Jacobian matrix of the equilibrium equations is obtained by finite difference approximations. The distributed load is scaled by a (loading) parameter $\lambda$, and as $\lambda$ 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 [17], 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 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.

In this paper, we show (Table 1) the results of a few experiments with using `pARMS` on an example of a linear system, medium tire model $\mathcal{M}$, in which $n = 49,800$ and the number of nonzeros is approximately $84n$. In `pARMS`, a shift $\alpha$ is chosen *automatically*: starting with the zero shift, the preconditioner is reconstructed with a new shift (augmented by 0.1) if the estimate $\mathcal{E}_\alpha$ of the preconditioner inverse is large (greater than seven). In Table 1, we state the final value of $\alpha$, the maximum $\mathcal{E}_\alpha$ among all the processors when $\alpha = 0$, the number `Iter` of iterations to converge, and the preconditioner application time `Time` spent when running on four processors. Due to the difficulty of this problem,

**Table 1.** Solution of tire model $\mathcal{M}$ on four processors

| Method | $\alpha_{fin}$ | $\max E_{\alpha=0}$ | Iter | Time |
|---|---|---|---|---|
| add_ilu(2) | 0.1 | 46 | 543 | 287.13 |
| add_ilut | 0.1 | 116 | 537 | 211.45 |
| sch_gilu0 | 0.2 | 146 | 575 | 369.00 |

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 shift values.

## 4 Conclusion

In this paper, we have illustrated the performance of the recently developed parallel ARMS (`pARMS`) code on 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 which allows a shift value to be assigned independently in each sub-domain. An automatic procedure for the shift value selection has also been implemented, which stabilizes the distributed preconditioner and often overcomes 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. 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.
2. X. Cai and Å. Ødegård. Parallel simulation of 3D nonlinear acoustic fields on a Linux-cluster. *Proceedings of the* Cluster 2000 *conference.*
3. E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics*, 87:387–414, 1997.
4. E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19:995–1023, 1998.
5. Diffpack World Wide Web home page. *http://www.nobjects.com.*
6. 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.
7. D. Hysom and A. Pothen. A scalable parallel algorithm for incomplete factor preconditioning. Technical Report (preprint), Old-Dominion University, Norfolk, VA, 2000.
8. H. P. Langtangen. *Computational Partial Differential Equations – Numerical Methods and Diffpack Programming.* Springer-Verlag, 1999.
9. 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.
10. T.A Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of computation*, 32:473–497, 1980.
11. Y. Saad. ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing*, 17(4):830–847, 1996.
12. Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS publishing, New York, 1996.
13. Y. Saad and M. Sosonkina. Distributed Schur Complement techniques for general sparse linear systems. *SIAM J. Scientific Computing*, 21(4):1337–1356, 1999.
14. 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.
15. 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.
16. 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).
17. 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.