Schur Complement Preconditioners for Distributed General Sparse Linear Systems*

Yousef Saad

University of Minnesota, Department of Computer Science and Engineering, 200 Union Street SE, Minneapolis, MN 55455, USA. saad@cs.umn.edu

Summary. This paper discusses the Schur complement viewpoint when developing parallel preconditioners for general sparse linear systems. Schur complement methods are pervasive in numerical linear algebra where they represent a canonical way of implementing divide-and-conquer principles. The goal of this note is to give a brief overview of recent progress made in using these techniques for solving general, irregularly structured, sparse linear systems. The emphasis is to point out the impact of Domain Decomposition ideas on the design of general purpose sparse system solution methods, as well as to show ideas that are of a purely algebraic nature.

1 Distributed sparse linear systems

The parallel solution of a linear systems of the form

$$Ax = b, \tag{1}$$

where A is an $n \times n$ large sparse matrix, typically begins by subdividing the problem into p parts with the help of a graph partitioner [24, 13, 15, 23, 8, 16]. Generally, this consists of assigning sets of equations along with the corresponding right-hand side values to 'subdomains'. It is common that if equation number i is assigned to a given subdomain then unknown number i is assigned to the same subdomain. Thus, each processor holds a set of equations (rows of the linear system) and vector components associated with these rows.

This distinction is important when taking a purely algebraic viewpoint because for highly unstructured or rectangular (least-squares) systems, this is no longer a viable or possible strategy and one needs to reconsider the standard graph partitioning approach used in Domain Decomposition. The next section is a brief discussion of graph partitioning issues.

*This work was supported by NSF under grants ACI-0305120 and INT-0003274 and by the Minnesota Supercomputer Institute.

2 Graph partitioning

Figure 1 shows two standard ways of partitioning a graph. On the left side is a 'vertex' partitioning which is common in the sparse matrix community. A vertex is a pair equation-unknown (equation number i and unknown number i) and the partitioner subdivides the vertex set into p partitions, i.e., p non-intersecting subsets whose union is equal to the original vertex set. On the right side of Figure 1, is a situation which is a prevalent one in finite element methods. Here it is the set of elements (rectangular in this case) that is partitioned. This can be called an element-based partitioning, or, alternatively, an 'edge-based partitioning', since in this case it also corresponds to assigning edges to subdomains.

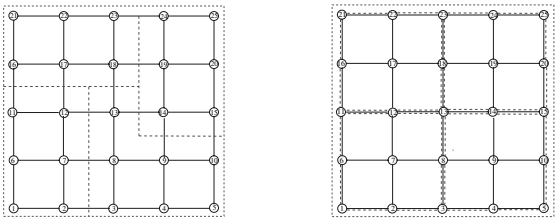

Fig. 1. Two classical ways of partitioning a graph.

The simplest criterion used to partition a graph is to try to minimize communication costs and to ensure at the same time that the work load between processors is well balanced. In this strategy, it is common to model communication costs by counting the number of edge-cuts, i.e., edges that link vertices in different subdomains. Graph partitioners such as Metis [15] and Chaco [13], attempt to partition graphs wit the quality measures just mentioned, in mind. However, a simple look at a general graph will reveal that edge-cuts will not lead to a good model for communication costs. Thus, when k edges connect a single vertex to k non-local vertices we would count k communication instances instead of one.

This observation was exploited in [8] to devise partitioners which lead to reduced communication costs. The authors of [8] used 'Hypergaphs' for this purpose. Hypergraphs are generalizations of graphs in which edges become sets (called *hyperedges* or *nets*) consisting of several vertices, instead of just two. Figure 2 shows a sparse matrix along with its traditional graph representation. Figure 3 shows the hypergraph obtained by defining hyper-edges to be the sets of column entries for each row. A hyperedge is represented by a square. Thus, hyperedge h_6, which corresponds to the 6-th row of the matrix, is the set of the 3 vertices: 1, 6, and 8, as indicated by the links from h_6 (square) to the vertices 1, 6, and 8 (bullets). Similarly $h_7 = \{1, 2, 7, 8\}$.

Note that from one viewpoint, this new representation is really that of a bipartite graph, since the nodes represented by a hyperedge (squares) are linked only to vertices of the graph (bullets). Models similar to the one just illustrated, i.e., based on setting h_i to be the set of column entries of row i, are common in hypergraph partitioning as they tend to yield better cost models for communication, see, [8]. Gains in communication will help reduce the overall run time but these gains are typically in the order of 10-30%, and they often represent a small portion of the

overall execution time. One may ask whether or not the gains could be outweighed by the cost of a higher iteration count. In fact, experimental results suggest that hypergraph partitioning yields as good if not better quality partitionings from the point of convergence. More importantly, we believe that the generality and flexibility of hypergraph models has not yet been fully exploited in Domain Decomposition. Though it is difficult to rigorously build a partitioning that will yield an 'optimal' condition number for the preconditioned matrix, heuristic arguments, see, e.g., [27], may help obtain criteria that can help build good models based on weighted hypergraphs.

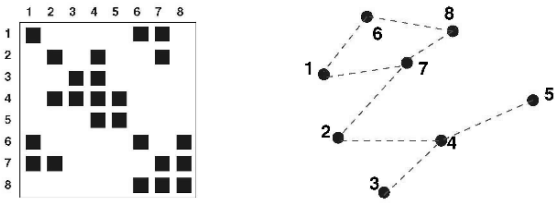

Fig. 2. A small sparse matrix and its classical graph representation.

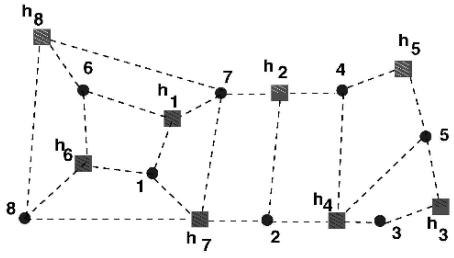

Fig. 3. One possible hypergraph representation of the matrix in Figure 2.

Another potential use of hypergraphs is for solving very irregularly structured problems which do not originate from PDEs. In these situations, the adjacency graph of the matrix may be directed (i.e., pattern of A is nonsymmetric), a situation which is not handled by standard partitioners. A common remedy is to symmetrize the graph before partitioning it, which tends to be wasteful. Domain decomposition ideas can be extended to such problems with the help of hypergraphs [12] or the closely related bipartite models [16].

3 The local system

Once a graph is partitioned, three types of unknowns can be distinguished: (1) Interior unknowns that are coupled only with local equations; (2) Local interface unknowns that are coupled with both non-local (external) and local equations; and

(3) External interface unknowns that belong to other subdomains and are coupled with local equations. Local points in each subdomain are often reordered so that the interface points are listed after the interior points. Thus, each local vector of unknowns x_i is split into two parts: the subvector u_i of internal vector components followed by the subvector y_i of local interface vector components. The right-hand side b_i is conformally split into the subvectors f_i and g_i. When block partitioned according to this splitting, the local system of equations can be written as

$$\underbrace{\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}}_{A_i} \underbrace{\begin{pmatrix} u_i \\ y_i \end{pmatrix}}_{x_i} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \underbrace{\begin{pmatrix} f_i \\ g_i \end{pmatrix}}_{b_i}. \tag{2}$$

Here, N_i is the set of indices for subdomains that are neighbors to the subdomain i. The term $E_{ij}y_j$ is a part of the product which reflects the contribution to the local equation from the neighboring subdomain j. The result of this multiplication affects only local interface equations, which is indicated by zero in the top part of the second term of the left-hand side of (2).

4 Schur complement techniques

Schur complement techniques consist of eliminating interior variables to define methods which focus on solving in some ways the system associated with the interface variables. For example, we can eliminate the variable u_i from (2), which gives $u_i = B_i^{-1}(f_i - F_i y_i)$ and 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

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{4}$$

The equations (3) for all subdomains ($i = 1, \ldots, p$) constitute a linear system involving only the interface unknown vectors y_i. This reduced system has a natural block structure:

$$\underbrace{\begin{pmatrix} S_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2 & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p,2} & \ldots & S_p \end{pmatrix}}_{S} \underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}}_{y} = \underbrace{\begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}}_{g'}. \tag{5}$$

The diagonal blocks in this system, the local Schur complement matrices S_i, are dense in general. The off-diagonal blocks E_{ij}, which are identical with those of the local system (2) are sparse.

If can solve the global Schur complement system (5) then the solution to the global system (1) would be trivially obtained by substituting the y_i's into the first part of (2). A key idea in domain decomposition methods is to develop preconditioners for the *global system* (1) by exploiting methods that *approximately solve the Schur complement system* (5).

Preconditioners implemented in the pARMS library [18] rely on this general approach. The system (5) is preconditioned in a number of ways, the simplest of which is to use a Block-Jacobi preconditioner exploiting the block structure of (5). The S_i's are not explicitly computed. Assuming the notation (2), and considering the LU factorization of A_i, we note that

$$\text{if} \quad A_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix} \quad \text{then} \quad L_{S_i} U_{S_i} = S_i \ .$$

This yields the LU (or ILU) factorization of S_i as a by-product of the LU (resp. ILU) factorization of A_i. Setting up the preconditioner is a local process which only requires the LU (resp. ILU) factorization of A_i.

Other Schur complement preconditioners available in pARMS include methods which solve the system (5) approximately by a parallel (multicolor) version of the ILU(0) preconditioner, and a multicolor block Gauss-Seidel iteration (instead of block Jacobi). In general these work better than the simple block Jacobi technique discussed above. For details see [18].

5 Use of independent sets

Independent set orderings permute a matrix into the form

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \tag{6}$$

where B is diagonal. The unknowns associated with the B block form an independent set (IS), which is said to be maximal if it cannot be augmented by other nodes to form a bigger independent set. Finding a maximal independent set can be done inexpensively by heuristic algorithms [9, 17, 25].

The main observation here is that the Schur complement $S = C - EB^{-1}F$ associated with the above partitioning of the matrix is again a sparse matrix *in general* since B is diagonal. Therefore, one can think of applying the reduction recursively as is illustrated in Figure 4. When the reduced system becomes small

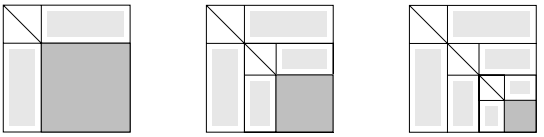

Fig. 4. Three stages of the recursive ILUM process

enough then it can be solved by any method. This is the idea used in ILUM [25], and in a number of related papers [7, 6, 30].

The notion of independent sets can easily be extended to 'group independent sets', in which the matrix B is allowed to be block-diagonal instead of just diagonal. In other words, we need to find "groups" or "aggregates" of vertices which are not coupled to each other, in the sense that no node from one group is coupled with

a node of another group. Coupling within any group is allowed but not between different groups.

Define the matrix at the zeroth-th level to be $A_0 \equiv A$. The Algebraic Recursive Multilevel Solver algorithm (ARMS), see [28], is based on an approximate block factorization of the form

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

Here, $L_l U_l$ is an Incomplete LU factorization of B_l, i.e., $B_l \approx L_l U_l$ and A_{l+1} approximates the Schur complement, so, $A_{l+1} \approx C_l - (E_l U_l^{-1})(L_l^{-1} F_l)$. The matrix A_{l+1} is the coefficient matrix for the linear system at the next level. It remains sparse because of the ordering selected (group independent sets) and due to the dropping of smaller terms. The L-solves associated with the above block factorization amount to a form of restriction in the PDE context, while the U-solve is similar to a prolongation. Note that the algorithm is fully recursive. At the last level (selected in advance, or by exhaustion) a simple ILU factorization is used instead of the one above.

6 Highly indefinite problems: nonsymmetric orderings

Perhaps one of the most significant advances on "general purpose iterative solvers" of the last few years is the realization that permuting a matrix in a nonsymmetric way, before applying a preconditioning, can lead to a robust iterative solution strategy [11, 10, 2]. By permuting A nonsymmetrically we mean a transformation of A of the form PAQ^T, where P and Q are two different permutations. In particular, a significant difference between this situation and the standard one where $P = Q$, is that non-diagonal entries will be moved into the main diagonal. In fact the gist of these methods is to move large entries of the matrix onto the diagonal. This was explored for many years by researchers in sparse direct methods, as a means of avoiding dynamic pivoting in Gaussian elimination [22].

In [10, 11], a (one-sided) permutation P was sought by attempting to maximize the magnitude of the product of the diagonal entries of PA. Here we briefly outline a method which also attempts to place large entries onto the diagonal, by using a more dynamic procedure based on Schur complements. The idea here is to adapt the ARMS algorithm outlined earlier by exploiting nonsymmetric permutations. We will find two permutations P (rows) and Q (columns) to transform A into

$$PAQ^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix} . \tag{8}$$

No particular structure is assumed for the B block. The only requirement on P, Q is that for the resulting matrix in (8), the B block has the 'most diagonally dominant' rows (after nonsym perm) and few nonzero elements (to reduce fill-in). Once the permutations are found and the matrix is permuted as shown above, we can proceed exactly as for ARMS by invoking a multi-level procedure. So, at the l-th level we reorder A into PAQ^T, and then carry out an approximate block factorization identical with that of (7), except that the left-hand side is now PAQ^T instead of PAP^T. The rationale for this approach is that it is critical to have an accurate and

well-conditioned B block, [3, 4, 5]. In the case when B is of dimension 1, one can think of this approach as a form of complete pivoting ILU.

The B block is defined by the *Matching set* $\mathcal{M}$ which is a set of n_M pairs (p_i, q_i) where $n_M \leq n$ with $1 \leq p_i, q_i \leq n$ for $i = 1, \ldots, n_M$ and $p_i \neq p_j$, for $i \neq j \qquad q_i \neq q_j$, for $i \neq j$ The case $n_M = n$ yields the (full) permutation pair (P, Q). A partial matching set can be easily completed into a full pair (P, Q) by a greedy approach.

The algorithm to find permutation consists of 3 phases. First, a *preselection* phase is invoked to filter out poor rows by employing a criterion based on diagonal dominance. The main goal of this preselection phase is only to reduce the cost of the next phase. Second, a *matching* phase scans candidate entries in order given by the preselection algorithm and accepts them into the $\mathcal{M}$ set, or rejects them. Heuristic arguments, mostly based on greedy procedures, are used for this. Finally, the third phase *completes the matching set* to obtain a pair of (full) permutations P, Q, using a greedy procedure.

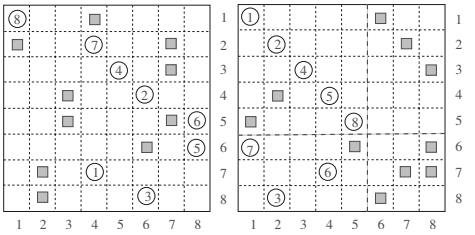

Fig. 5. Illustration of the greedy matching algorithm. Left side: a matrix after the preselection algorithm. Right side: Matrix after Matching permutation.

An illustration of the matching procedure is shown in Figure 5. The left side shows a certain matrix after the preselection procedure. The circled entries are the maximum entries in each row and they are assigned a rank based on the diagonal dominance ratio (the higher the better) and possibly the number of nonzero entries in the row (the fewer the better). The greedy matching algorithm will simply traverse these nodes in the order by which they are ranked, and then determine whether or not to assign the node to $\mathcal{M}$. Thus, entries labeled 1 (a_{74} in original matrix) and 2 ($a_{4,6}$ in original matrix) are accepted. Entry labeled 3 (a_{86}) is not because it is already in the same column as $a_{4,6}$. The algorithm continues in this manner until exhaustion of all nodes. This yields a partial permutation pair which is then completed arbitrarily. The matrix on the right shows the permuted matrix. The B block, separated by longer dash lines, is then eliminated and the process is repeated recursively on the Schur complement, in the same manner as the ARMS procedure. Details can be found in [26], along with a few more elaborate matching procedures.

As an example, Figure 6 shows an algorithm of this type in action for a highly indefinite and unstructured matrix, BP1000, obtained from the old Harwell-Boeing collection [2]. The matrix pattern is shown in the top left part of the figure. Most of the diagonal entries of the matrix are zero and as a result standard iterative methods will fail. Five levels are required by the procedure with the last block reaching a size

[2] See `http://math.nist.gov/MatrixMarket/`

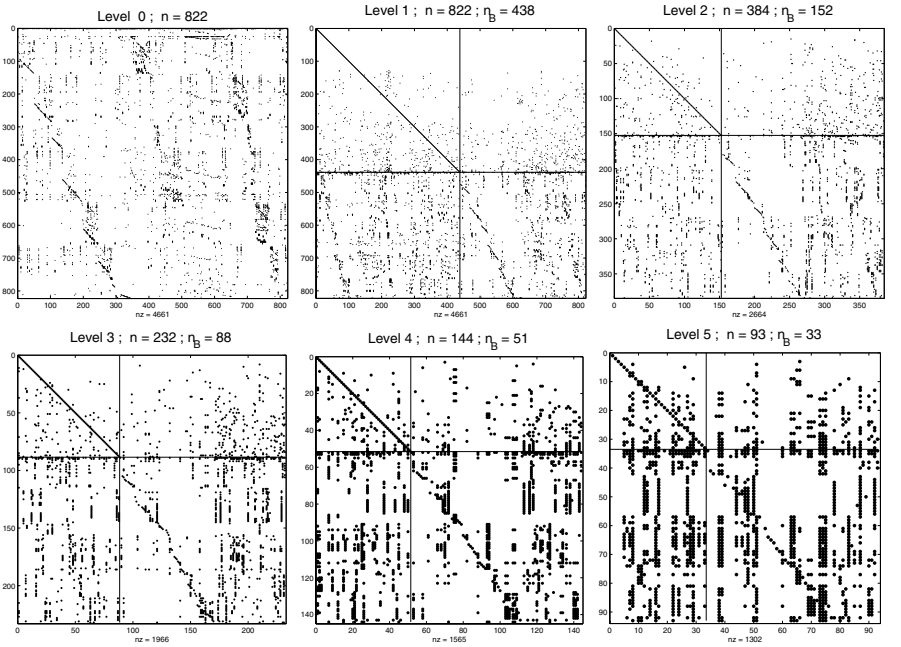

Fig. 6. The Diagonal Dominance PQ-ordering in action for a highly unstructured matrix.

of $n = 60$. With this the resulting preconditioning, GMRES converges in 17 steps. In addition this is achieved with a 'fill-factor' of 2.09, i.e., the ratio of the memory required for the preconditioner over that of the original matrix is 2.09. For additional experiments of more realistic problems see [26].

7 Wirebaskets and hierarchical graph decomposition

It was often observed in the domain decomposition literature that "cross points" play a significant role. This was exploited in [29] in a method known as the wirebasket preconditioner. Recently we have considered a method of the same type from an algebraic viewpoint [14]. This algorithm, called Parallel Hierarchical Interface Decomposition ALgorithm (PHIDAL), descends recursively into interface variables, by exploiting a hierarchy of 'interfaces'. Its main difference with the parallel version of ARMS, is that it uses a *static* ordering instead of a dynamic one. This results in fast preprocessing and, potentially, better parallelism.

To explain the algorithm, consider a graph $\mathcal{G}$ that is partitioned into p subgraphs. However, we now consider an edge-based partitioning, i.e., there are overlapping vertices. The illustration on the left side of Figure 7 shows the graph of a matrix associated with a 5-point FD discretization of a Laplacean on a 2-D domain. One can distinguish three types of nodes: interior, interface, and cross-points. Imagine now that we order the nodes according to this division: we would label all interior

points first, followed by the interface points followed by the cross-points. Of course the points in the same set (in this case whether interior nodes, domain edges) are always labeled together. The result of this reordering would be the matrix shown on the right of Figure 7. We refer to the connected subsets as "connectors". The interiors of the subdomains as well as the domain edges are connectors, as are the cross-points.

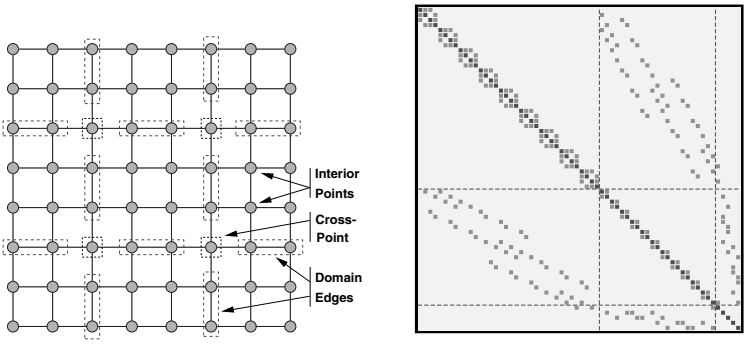

Fig. 7. A small finite difference mesh (left); Pattern of the matrix after the HID ordering.

This ordering is very appealing for parallel processing. If we do not allow any fill-in between the connectors, then the factorization will proceed in parallel at each level. For this example, there are 3 levels: one for the interior points, the second is that of the domain edges, and the 3rd is that of the cross-points. An idea similar to the one discussed here was described in [19, 20] including some analysis [21], though the setting was that of regular meshes. In [14], the above decomposition was extended to general graphs.

An extention of the above definition requires us to *partition the graph into levels of subgraphs with the requirements that the subgraphs at a given level separate those at lower levels.* We will call a connector a connected component in the adjacency graph. A level consists of a collection of connectors with the following requirements: (1) Connectors at any level should separate connectors of previous levels; (2) Connectors of the same level are not coupled (just as in ARMS).

One of the simplest (and clearly not the best) ways to obtain this decomposition is to use the number of domains to which a node belongs. We can label each node u with list $key(u)$ of domains to which it belongs and then define the Level k to be the set of nodes such that $|key(u)| = k + 1$, for $k = 1, 2, \ldots,$. The next task would be to refine the labeling of the connectors to make them independent. The simplest refinement is based on a greedy approach which would relabel a connector by a higher label if it is connected to another connector of the same level. There are many possible refinements, and the reader is referred to [14] for details.

By reordering the nodes hierarchically at the outset, it is possible to create Schur complements that can be made sparse. Once a Schur complement at a given level is constructed it is then possible to create another level. The two important

ingredients of this procedure are: (1) algorithms for building a good levelization (few levels); and (2) good combination of effective dropping strategies and parallel incomplete factorization. Results shown in [14] indicate almost perfect scalability for simple model problems (Poisson's problem on a regular mesh) and good scalability for a much harder problem issued from a Magneto Hydrodynamics problem.

8 Concluding remarks

Schur complement techniques can lead to very successful parallel or sequential iterative procedures for solving general sparse linear systems. One of the most important ingredients that is exploited when taking a purely algebraic viewpoint is to reorder the equations in such a way that the next Schur complement is again sparse. This is exploited in techniques such as MRILU [7, 6] and ILUM [25], MLILU [1] and the closely related ARMS [28], and in PHIDAL [14]. Some of these techniques have their analogue in the classical DD literature, a good example being the PHIDAL preconditioner. Other types of reorderings exploit nonsymmetric permutations in order to first eliminate the easier equations. These techniques do not have obvious analogues in the classical DD literature. Because they represent an important set of tools to bridge the gap between the robustness of iterative methods and that of direct solvers, their extension to parallel computing environments, which is still lacking, is of critical importance.

9 Acknowledgments

Most of the work presented in this paper summarizes collaborative efforts with a number of co-workers. The Algebraic Recursive Multilevel Solvers (ARMS) were developed with Zhongze Li, Masha Sosonkina, and Brian Suchomel. The work on PHIDAL, which is still on-going, is in collaboration with Pascal Henon. The preliminary work on the use of hypergraphs within ARMS is joint work with Masha Sosonkina.

References

1. R. E. Bank and C. Wagner, *Multilevel ILU decomposition*, Numer. Math., 82 (1999), pp. 543–576.
2. M. Benzi, J. C. Haws, and M. Tuma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 1333–1353.
3. M. Bollöfer, *A robust ILU with pivoting based on monitoring the growth of the inverse factors*, Lin. Alg. Appl., 338 (2001), pp. 201–218.
4. M. Bollöfer and Y. Saad, *ILUPACK - preconditioning software package, release v1.0, may 14, 2004*. Available online at http://www.tuberlin.de/ilupack/.
5. M. Bollöfer and Y. Saad, *Multilevel preconditioners constructed from inverse-based ILUs*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 1627–1650.

6. 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 Proceedings of the Conference on Algebraic Multilevel Methods with Applications, O. Axelsson and B. Polman, eds., 1996, pp. 105–116.
7. E. F. F. Botta and F. W. Wubs, *Matrix renumbering ILU: an effective algebraic multilevel ILU*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 1007–1026.
8. U. V. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel and Distributed Systems, 10 (1999), pp. 673–693.
9. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw Hill, New York, 1990.
10. I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 889–901.
11. ———, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 973–996.
12. B. Hendrickson and T. G. Kolda, *Graph partitioning models for parallel computing*, Parallel Computing, 26 (2000), pp. 1519–1534.
13. B. Hendrickson and R. Leland, *The Chaco User's Guide Version 2*, Sandia National Laboratories, Albuquerque NM, 1995.
14. P. Henon and Y. Saad, *A parallel multilevel ILU factorization based on a hierarchical graph decomposition*, Tech. Rep. UMSI-2004-74, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2004.
15. G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1999), pp. 359–392.
16. T. G. Kolda, *Partitioning sparse rectangular matrices for parallel processing*, Lecture Notes in Computer Science, 1457 (1998), pp. 68–79.
17. M. R. Leuze, *Independent set orderings for parallel matrix factorizations by Gaussian elimination*, Parallel Computing, 10 (1989), pp. 177–191.
18. Z. Li, Y. Saad, and M. Sosonkina, *pARMS: a parallel version of the algebraic recursive multilevel solver*, Numer. Linear Algebra Appl., 10 (2003), pp. 485–509.
19. M. M. monga Made and H. A. van der Vorst, *A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings*, Future Generation Computer Systems, 17 (2001), pp. 925–932.
20. ———, *Parallel incomplete factorizations with pseudo-overlapped subdomains*, Parallel Computing, 27 (2001), pp. 989–1008.
21. ———, *Spectral analysis of parallel incomplete factorizations with implicit pseudo-overlap*, Numer. Linear Algebra Appl., 9 (2002), pp. 45–64.
22. M. Olschowska and A. Neumaier, *A new pivoting strategy for Gaussian elimination*, Lin. Alg. Appl., 240 (1996), pp. 131–151.
23. F. Pellegrini, *SCOTCH 4.0 user's guide*, tech. rep., INRIA Futurs, April 2005. http://www.labri.fr/perso/pelegrin/scotch/.
24. A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning sparse matrices with Eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.
25. Y. Saad, *ILUM: a multi-elimination ILU preconditioner for general sparse matrices*, SIAM J. Sci. Comput., (1996), pp. 830–847.
26. ———, *Multilevel ILU with reorderings for diagonal dominance*, SIAM J. Sci. Comput., 27 (2005), pp. 1032–1057.

27. Y. Saad and M. Sosonkina, *Non-standard parallel solution strategies for distributed sparse linear systems*, in Parallel Computation: 4th international ACPC conference, Salzburg Austria, February 1999, P. Zinterhof, M. Vajtersic, and A. Uhl, eds., vol. 1557 of Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 13–27.
28. Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl., 9 (2002), pp. 359–378.
29. B. F. Smith, *Domain Decomposition Algorithms for the Partial Differential Equations of Linear Elasticity*, PhD thesis, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, September 1990.
30. A. van der Ploeg, E. F. F. Botta, and F. W. Wubs, *Nested grids ILU decomposition (NGILU)*, J. Comp. Appl. Math., 66 (1996), pp. 515–526.