Sparse Approximate Inverses in Preconditioning Distributed Linear Systems

Maria Sosonkina[†]

Abstract

Using a direct approximation of sparse matrix inverse in preconditioning is viewed as a good alternative to the preconditioning techniques that require a matrix factorization. A sparse approximate inverse is easy to compute and apply, and it is suitable for parallel implementations. For distributed linear systems of varying difficulty, approximate block LU preconditioning using sparse approximate inverse techniques and an incomplete LU factorization used in Block-Jacobi preconditioning are compared.

1 Introduction

Computational challenges of the contemporary world make iterative methods significant "team players" in a sophisticated problem solving process, the overall success of which largely depends on sparse linear system solutions. *Iterative solution techniques* are prominent due to their moderate computational and storage demands: they do not require complete matrix factorization and preserve the sparsity of matrices, which is crucial for large scale scientific applications. Such applications require powerful parallel computing platforms. Hence, it is not surprising that iterative methods are prime candidates for parallel environments, especially since their computations are inherently parallel, which is not the case for direct solution methods. However, using iterative methods in lieu of direct methods poses the question impertinent to solving a linear system with a direct method: how fast can a solution of the desired accuracy can be obtained? The answer often depends on the algebraic properties of the matrix, such as the condition number and the matrix spectrum. For ill conditioned matrices, iterative methods are notorious for their slow convergence and lack of robustness. To make iterative methods more reliable, the technique called *preconditioning* is used, which is essentially a technique for substituting another linear system for the original one, such that the solutions of both linear systems are the same and that an iterative method is likely to solve the new system more easily. Let M be any nonsingular matrix. The solution to the system $Ax = b$ can be calculated by solving the system

$$Bx = (M^{-1}A)x = M^{-1}b = g.$$

[†] Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061-0106, email: masha@cs.vt.edu.

The use of such a matrix is known as (left) preconditioning. Since the goal of using preconditioning is to decrease the computational effort needed to solve the original system, M should be inexpensive to construct and apply and should approximate A in some way. Then $M^{-1}A$ would be close to the identity matrix and iterative methods would likely converge more rapidly when applied to the preconditioned rather than the original system. In practice, B is never explicitly formed.

In distributed environments, the role of preconditioning in accelerating convergence of iterative methods has an important consequence of reducing the parallel overhead, and thus decreasing the total parallel time of computation. (The parallel overhead is the time spent by a parallel method in doing nonessential work, communicating among processors, or in idling of processors.) This role is fulfilled if the construction or application of preconditioning has a small parallel overhead itself. The construction of a parallel preconditioner may follow two general paths: extracting parallelism from efficient sequential techniques and targeting a preconditioner specifically at parallel iterative methods

Each of these two approaches has its advantages and disadvantages. In the first one, the preconditioners promoting a good convergence of a sequential iterative method often have a low degree of parallelism, thus their distributed versions appear to be expensive. The second approach usually yields highly parallel preconditioning, which, however, does not boost the convergence of an iterative method as much as the preconditioning considered in the first approach.

Section 2 gives brief descriptions of preconditioners targeted at parallel environments. In Sections 4 and 5, a variation of a block preconditioner using approximate inverses [4] is extended to general distributed matrices. Section 6 contains numerical results and conclusions.

2 Description of parallel preconditioning.

Among all the preconditioning techniques two main classes can be distinguished: those that require matrix factorization and those that do not. One way to construct the preconditioning matrix M is by means of *incomplete LU factorization* (ILU factorization) of the original matrix A , such that

$$A = LU + E \quad \text{and} \quad M = LU,$$

where L and U are lower triangular and unit upper triangular matrices, respectively, and E is the error of the factorization. The number and pattern of nonzero entries in L and U distinguish different types of ILU factorization of A. For example, an obvious choice for the nonzero structure of the L and

U factors is to take it identical to the sparsity pattern of A. The corresponding incomplete factorization of A is termed ILU factorization with zero fill-in (ILU(0)). However, ILU(0) preconditioning may lead to a crude approximation of the original matrix A. In this case, some other ILU factorization is more appropriate; that is, a factorization with L and U having more nonzeros than in the original matrix, such as ILUT preconditioning [9], which has a dual threshold strategy for dropping fill-in elements.

Preconditioning with approximate inverses [2], [3], and [6] favors parallel implementations, which is evident from the following description. Consider the following minimization problem in the Frobenius norm to find a right approximate inverse preconditioner M:

$$\min_{M \in \mathcal{S}} \|I - AM\|_F^2,$$

where $\mathcal{S}$ is the set of $n \times n$ sparse matrices. It follows that this minimization problem can be decoupled into n minimization problems of the form

$$\min_{m_j} \|e_j - Am_j\|_2^2, \quad j = 1, 2, ..., n,$$

where e_j and m_j are the jth columns of the identity matrix and a matrix $M \in \mathcal{S}$, respectively. Note that each if n columns can be computed independently. Thus, a parallel sparse approximate inverse algorithm arises from computing a column on a processor. Different strategies for selecting a nonzero structure of the approximate inverse are proposed in [6] and [3]. In the former work, the initial sparsity pattern is taken to be diagonal with further fill-in allowed depending on the improvement in the minimization. The latter suggests to control the sparsity of the approximate inverse by dropping certain nonzero entries in the solution or search directions of a suitable iterative method (e.g., GMRES [11]) used to solve the system $Am_j = e_j$ such that $\min_{m_j} \|e_j - Am_j\|_2^2, \quad j = 1, 2, ..., n$.

There exist several parallel implementations of an approximate inverse preconditioning [1] and [5], which are shown to be efficient preconditioners for a variety of general sparse matrices. However, the case when this preconditioning becomes expensive are not rare, which may happen when the entries of the actual inverse are of the same magnitude. To reduce the cost of preconditioning with approximate inverses, this technique may be used on matrix blocks. A success of such usage was reported in [4], where approximate inverses were exploited in conjunction with other preconditioning techniques to construct block preconditioners.

Block preconditioning is particularly suitable for parallel processing, since it promotes an efficient utilization of processors by providing a good load balancing. In sparse linear algebra computations, the matrix is often partitioned by blocks of rows among processors. An identification of preconditioner blocks with blocks of matrix rows assigned to a processor leads to a highly parallel preconditioning (cf., for example, domain decomposition preconditioning such as block-Jacobi and block-Gauss-Seidel). In the next two sections, an approach to constructing a block preconditioning using approximate inverses taken in [4], is extended to distributed general matrices.

4 Distributed matrix representation

Let matrix rows are partitioned according to some partition of vector components and the matrix be structurally symmetric (corresponding to an undirected graph). Then a distributed matrix-vector multiplication (the only operation with a distributed matrix required by an iterative method) can be efficiently implemented, as shown in [10], by introducing a preprocessing stage performed concurrently on each processor. The purpose of this stage is twofold. Firstly, preprocessing creates the lists of the local vector components to be communicated repeatedly at each matrix-vector multiplication. Secondly, preprocessing reorders vector components and nonzeros of a local submatrix to facilitate overlapping of communication and computation while performing global matrix-vector multiplication. As a result of the preprocessing stage, local vector components are decoupled into *internal* vector components that do not participate in communication, and *local interface* vector components that are requested by neighboring processors, such that internal components are ordered before local interface components. The indices of *external interface* vect or components that are received from neighboring processors are also identified. The reordered local submatrix A_d (Figure 1) is represented by two matrices: the square matrix G_d of dimensions $n_d \times n_d$ corresponding to the number of vector components in the processor d, and the rectangular $d_1 \times d_2$ matrix X_d, where d_1 is the number of local interface components and d_2 is the number of external interface components. To perform a matrix-vector multiplication, the submatrix G_d is first multiplied by the local vector components $x_d(1 : n_d)$, then the X_d part of A_d is multiplied by the received external interface components $x'_d(1 : d_2)$ and the results of these two multiplications are added.

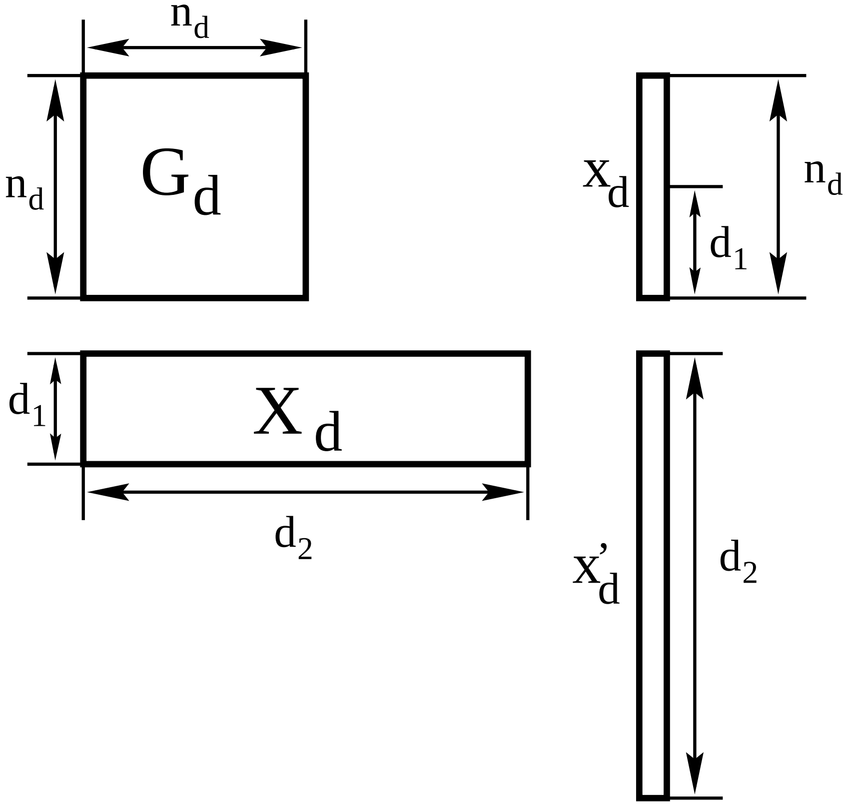

Fig.1. Local matrix A_d and vector components after the preprocessing stage.

5 Distributed approximate block LU preconditioner

Consider the system $Ax = b$ rewritten as

$$\begin{pmatrix} B & E \\ F & C \end{pmatrix} \begin{pmatrix} z \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \text{ where } A = \begin{pmatrix} B & E \\ F & C \end{pmatrix}, \quad x = \begin{pmatrix} z \\ y \end{pmatrix}, \text{ and } b = \begin{pmatrix} f \\ g \end{pmatrix}.$$

A block preconditioner M_A for A can be derived from an approximate block LU factorization of A ($A \approx LU$) as follows:

$$M_A = LU \quad \text{with} \quad L = \begin{pmatrix} B & 0 \\ F & M_S \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} I & B^{-1}E \\ 0 & I \end{pmatrix},$$

where M_S is some preconditioning of the Schur Complement matrix $S = C - FB^{-1}E$. The matrix M_S may be, for example, a sparse approximation to S obtained by one of the approximate inverse techniques mentioned in Section 2, such that a sparse approximate inverse technique is applied to approximate $B^{-1}E$ (as it is done in [4]) with Y. The resulting matrix Y is sparse. Therefore, the preconditioning matrix M_S is also sparse.

In application of the preconditioner M_A to a block vector $(f, g)^t$, at each iteration the following steps should be performed :

1. $z = B^{-1}f$;

2. $y = M_S^{-1}(g - Fz)$;
3. $z = z - B^{-1}Ey$.

To adapt this algorithm for distributed matrices, the global matrices B, E, F, C, and M_S should be specified in terms of local matrices A_d, G_d, and X_d on each processor (see Section 4 for the descriptions of A_d, G_d, and X_d). Furthermore, the global solution vector $(z, y)^t$ and the right-hand side $(f, g)^t$ need also to be represented via local vector components. For the processor d, Figure 2 displays blocks B_d, E_d, F_d, and C_d of G_d that are parts of B, E, F, and C, respectively. Figure 2 also shows the local vector components, that are parts of $(z, y)^t$ ($(f, g)^t$ is partitioned accordingly).

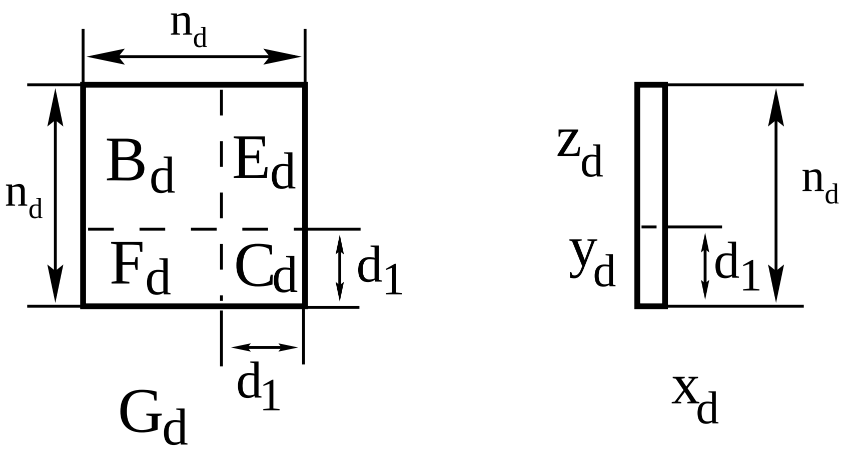

Fig.2. Local submatrix G_d and vector components divided into blocks.

Similarly to [10], the global M_S is constructed from an approximation to the local Schur complement matrix $S_d = C_d - F_d Y_d$, where $Y_d \approx\ B^{-1}E$ and the rectangular matrix X_d on each processor, as shown in Figure 3.

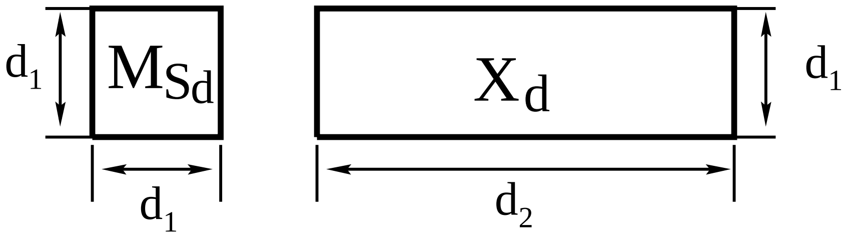

Fig.3. Local part of the preconditioning of the Schur complement.

Once the relation between the blocks of the "global" approximate LU factorization and local matrices is established, a sequence of steps in application of M_A takes the following form (in terms of local matrices): On each processor do in parallel:

(a) $z_d = B_d^{-1} f_d$;
(b) $y_d = M_{S_d}^{-1}(g_d - F_d z_d)$;
(c) $z_d = z_d - B_d^{-1} E_d y$.

Note that steps (a) and (c) do not require any communication among processors, since matrix-vector operations in these steps are performed with the local vector components only. Conversely, in step (b) the solve with M_{S_d} (i.e., $M_{S_d} y_d = g_d - F_d z_d$) global matrix-vector multiplications, since the matrix M_{S_d} incorporated the matrix X_d corresponding to the external interface vector components.

4 Numerical experiments

A flexible variant of the restarted GMRES (FGMRES) [8] has been chosen to solve the original system, because this iterative method permits a change in the preconditioning operation at each step. This situation arises when, for example, an iterative process is used to precondition the input system. In particular, it is possible to use an ILUT-preconditioned GMRES with $lfil$ fill-in elements is used in step (a) to reduce the residual norm by 10^{-3} but no more than for 5 iterations. The same iterative process (with the same parameters) may be used in step (b). Alternatively, $B_d^{-1} E_d$ may be substituted by Y_d in step (c). An approximate solution of $M_{S_d} y_d = g_d - F_d z_d$ is obtained by no more than 20 iterations of ILUT-preconditioned FGMRES. For the original system, flexible GMRES(20) is allowed to perform 300 iterations to achieve a residual norm reduction by 10^{-6}.

In the numerical experiments the following four test problems (Table 1) from the Harwell-Boeing and Davis collections were considered. In Table 1, the heading Scaling denotes the column that shows whether or not the rows and columns of the matrix for a given problem were scaled by 2-norm before the solution process; the column Pattern specifies whether a given problem has a structurally symmetric matrix. Also, in the partitioning of a problem one level of overlapping without averaging was used following [7].

TABLE 1 *Description of test problems.*

Name	n	n_z	Pattern	Scaling	Discipline
bcsstk13	2003	83883	Symm	Yes	Dynamical analysis in structural engineering
fidap020	2203	67429	Unsymm	Yes	Finite element modeling
fidap037	3565	67591	Symm	Yes	Finite element modeling
raefsky1	3242	22316	Unsymm	Yes	Incompressible flow in pressure driven pipe

For convenient reference, the following abbreviations denote preconditioning techniques used in numerical experiments:

BFsy — distributed approximate block LU factorization with local M_{S_d} and $B_d^{-1}E_d$ approximated by S_d and Y_d, respectively;

BFs — distributed approximate block LU factorization with local M_{S_d} approximated by S_d and $B_d^{-1}E_d$ applied using one matrix-vector multiplication followed by solve with B_d;

BJ — approximate block-Jacobi, where ILUT-preconditioned GMRES(20) is used to precondition each submatrix assigned to a processor.

Tables 2 and 3 show iteration numbers required by FGMRES(20) with BFsy, BFs, or BJ till convergence on different numbers of processors. Asterisk indicates nonconvergence. The $lfil$ column specifies the number of fill-in elements permitted in the ILUT factorization. $lfil$ corresponds also to the number of elements in a matrix column created by the approximate inverse technique.

TABLE 2 *Number of the FGMRES(20) iterations for the FIDAP and BCSSTK13 problems.*

Name	Precon	$lfil$	2	4	16	24	36	40	44
fidap020	BFsy	10	18	117	*	5	5	5	6
		20	12	90	*	4	4	5	5
	BFs	10	*	*	*	5	5	5	6
		20	13	*	*	3	3	3	3
	BJ	10	17	*	*	45	47	43	53
		20	12	16	100	44	47	44	53
fidap037	BFsy	10‖20	3	4	4	3	3	3	3
	BFs	10‖20	3	4	3	3	3	3	3
	BJ	10‖20	3	5	7	8	8	8	8
bcsstk13	BFsy	10‖20	5	4	3	3	4	3	3
	BFs	10‖20	5	4	3	3	4	3	3
	BJ	10‖20	6	7	11	12	14	14	15

TABLE 3 *Number of the FGMRES(20) iterations for the RAEFSKY1 problem.*

Name	Precon	$lfil$	4	8	16	24	36	40
raefsky1	BFsy	10	14	13	10	11	8	8
		20	12	11	9	9	8	8
	BFs	10	16	13	10	11	8	8
		20	13	11	9	9	8	8
	BJ	10	85	171	173	273	252	263
		20	82	170	173	271	259	259

For a given problem, iteration counts for the BFsy and BFs suggest a clear trend of achieving convergence in fewer iterations with increasing number of processors, which means that a high degree of parallelism of these preconditioners does not impede convergence, and may even enhance it significantly (cf. rows 1–4 of Table 2). Furthermore, both BFsy and BFs do not suffer from an information loss as it happens with BJ (because of disregarding the local matrix entries corresponding to the external interface vector components). Note that the effectiveness of BJ degrades with increase of the number of processors for almost every problem in Tables 2 and 3, and all three preconditioners perform similarly on small numbers of processors (e.g., column 1 for the fidap037 problem). Comparison of BFs and BFsy confirms the conclusions of [4] that using Y to approximate $B^{-1}E$ in step 3 is more efficient than applying $B^{-1}E$ directly. For general distributed matrices, this is especially true, since iterative solves with the local parts of B may be very inaccurate (cf., for example, BFsy and BFs on 4 processors in Table 2 for the fidap20 problem). Unstable LU factors in preconditioning of the local parts is the cause of failure of FGMRES(20) with all three preconditionings on 16 processors (Table 2). Increasing the amount of fill-in helps FGMRES(20) with BJ to converge, while no convergence is achieved for FGMRES(20) with BFsy (or BFs).

In the experiments, sparse approximations of Y_d appear to be quite accurate (usually reducing the Frobenius norm to 10^{-2}), which could be attributed to the small dimensions of the matrices used in approximations. This reduction in the Frobenius norm was attained in 10 iterations of the MR method. Smaller numbers of iterations were also tested. Their effect on the overall solution process amounted to one extra iteration on average of FGMRES(20). However, when BFsy was used on 4 processors for fidap20, 260 iterations were required if a sparse approximation of Y_d was constructed with only 5 iterations. Thus, the performance of FGMRES(20) with BFsy on 4 processors (fidap20 problem) had a strong dependency on the quality of Y_d, which is why FGMRES(20) with BJ ($lfil = 20$) outperforms FGMRES(20) with BFsy on 4 processors

6 Conclusions

In this paper approximate inverse techniques have been used to accurately approximate distributed matrix blocks, which arise from partitioning the matrix among processors. The application of these techniques is integrated into distributed preconditioning by an approximate block LU factorization. The obtained preconditioning is designed such that it has parts of its blocks located on each processor, which provides a high degree of parallelism. Also, the resulting preconditioned FGMRES exhibits good convergence with increasing number of processors. A distributed approximate block LU factorization can be adapted to a linear system at hand by choosing appropriate ways to construct and apply parts of its blocks local to a processor.

References

[1] S. T. Barnard and R. L. Clay, *A portable MPI implementation of the SPAI preconditioner in ISIS++*, In proc. Eighth SIAM Conf. on Parallel Processing for Scientific Computing, SIAM, Philadelphia, PA, 1997.

[2] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Math., 22 (1982), pp. 127–140.

[3] E. Chow and Y. Saad, *Approximate inverse preconditioners for general sparse matrices*, Tech. Rep., UMSI 94-101, Univ. Minnesota Supercomputer Institute, Minneapolis, MN 55415, May 1994.

[4] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, Tech. Rep., UMSI 95-13, Univ. Minnesota Supercomputer Institute, Minneapolis, MN 55415, January 1995.

[5] V. Deshpande, M. J. Grote, P. Messmer, and W. Sawyer, *Parallel implementation of a sparse approximate inverse preconditioner*, In Proc. Irregular'96, Santa Barbara, CA, 1996

[6] T. Huckle and M. Grote, *A new approach to parallel preconditioning with sparse approximate inverses*, Tech. Rep., SCCM 94-03, Scientific Computing and Computational Mathematics Program, Stanford Univ., Stanford, CA, 1994.

[7] G.-C. Lo and Y. Saad, *Iterative solution of general sparse linear systems on clusters of workstations*, Tech. Report, UMSI 96/117, Supercomputer Institute, Univ. Minnesota, 1200 S. Washington Ave., Minneapolis, MN 55415, August 1996.

[8] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. Sci. Comput., 14, 461–469, 1993.

[9] Y. Saad, *ILUT: A dual threshold incomplete LU factorization*, Numer. Linear Algebra Appl., 1, 387-402, 1994.

[10] Y. Saad, *Iterative methods for sparse linear systems*, PWS Publishing Company, 1996.

[11] Y. Saad and M. H. Schultz, *GMRES: a generalized minimal residual method for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856 – 869.