SIAM J. SCI. COMPUT.
Vol. 24, No. 4, pp. 1141–1156

MSP: A CLASS OF PARALLEL MULTISTEP SUCCESSIVE SPARSE APPROXIMATE INVERSE PRECONDITIONING STRATEGIES*

KAI WANG† AND JUN ZHANG†

Abstract. We develop a class of parallel multistep successive preconditioning strategies to enhance efficiency and robustness of standard sparse approximate inverse preconditioning techniques. The key idea is to compute a series of simple sparse matrices to approximate the inverse of the original matrix. Studies are conducted to show the advantages of such an approach in terms of both improving preconditioning accuracy and reducing computational cost, compared to the standard sparse approximate inverse preconditioners. Numerical experiments using one prototype implementation to solve a few sparse matrices on a distributed memory parallel computer are reported.

Key words. sparse matrices, parallel preconditioning, sparse approximate inverse

AMS subject classifications. 65F10, 65F50, 65N55, 65Y05

PII. S1064827502400832

1. Introduction.

The need for solving very large sparse linear systems arising from many important applications has been pushing the development of sparse linear system solvers for parallel computers. Direct solvers, based on a factorization of the sparse matrices, are extremely robust, but their memory and floating point operation requirements grow faster than a linear function of the order of the matrix, because original zeros fill in during the factorization. While preconditioned Krylov subspace methods are considered to be some of the most suitable candidates for solving large sparse linear systems and the parallelization of many popular Krylov subspace solvers no longer poses a big problem, the quest for robust parallel preconditioners is still challenging [29].

Simple parallel preconditioners such as Jacobi or block Jacobi methods, although they are easy to implement, have the inherent weakness of being not robust for difficult problems. Their lack of robustness inhibits them from being used in industrial standard software packages. Other parallel preconditioners based on multicoloring strategy may have restricted applicability as the parallelism extracted from this strategy is limited. Domain decomposition based methods have been exploited extensively in parallel linear system solvers and preconditioners [4, 10, 24, 34]. Important progress has been made recently concerning the parallelization of incomplete LU (Cholesky) factorization preconditioners [23, 27, 28]. Furthermore, there are two additional classes of more advanced parallelizable preconditioners that seem to be more robust than the simple preconditioners. One is based on multilevel block incomplete LU (ILU) factorization, which is built on successive block independent set ordering and block ILU factorization. For detailed discussions of several sequential and parallel multilevel ILU preconditioning techniques, we refer readers to [1, 2, 30, 31, 32, 33].

*Received by the editors January 10, 2002; accepted for publication (in revised form) July 17, 2002; published electronically January 31, 2003. This research was supported in part by the U.S. National Science Foundation under grants CCR-9902022, CCR-9988165, CCR-0092532, and ACI-0202934, in part by the Japanese Research Organization for Information Science & Technology, and in part by the University of Kentucky Research Committee.

http://www.siam.org/journals/sisc/24-4/40083.html

†Laboratory for High Performance Scientific Computing and Computer Simulation, Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046 (kwang0@csr.uky.edu, www.csr.uky.edu/∼kwang0; jzhang@cs.uky.edu, www.cs.uky.edu/∼jzhang).

In this paper, we will be concerned with another class of parallelizable preconditioning techniques based on computing a sparse approximate inverse of the original matrix [7, 14, 21, 38]. These preconditioners have the property of possessing a high degree of parallelism in the solution process and are shown to be efficient for certain type of problems. However, for large classes of general sparse matrices, straightforward implementations of many sparse approximate inverse techniques lead to inefficient preconditioners on distributed memory parallel computers. Realistic parallel implementation to compute these preconditioners is also a big problem, due to the nature of interprocessor communications associated with the dynamic sparsity pattern search.

Some recently proposed parallel implementation of sparse approximate inverse preconditioner construction may require the sparsity pattern to be specified a priori [12]. For certain matrices, this may lead to poor sparse approximate inverse preconditioners, or to very high computational cost if some higher level static sparsity pattern is specified. The balance between preconditioner construction cost and preconditioner accuracy presents a challenge to practical efficient parallel implementation of most sparse approximate inverse techniques.

We investigate a class of multistep successive sparse approximate inverse preconditioning (MSP) techniques. A sequence of sparse matrices are computed cheaply using an existing parallel sparse approximate inverse technique. The product of these sparse matrices is used to approximate the true inverse of the original matrix. Thus, instead of computing a costly high accuracy sparse approximate inverse preconditioner in one shot, we compute a series of cheap sparse approximate inverse preconditioners to achieve the effect of a high accuracy preconditioner. The sparsity pattern is adjusted when a new sparse approximate inverse matrix is computed.

We develop algorithms to compute MSP efficiently on high performance parallel computers. Specifically we design and test a prototype MSP implementation to show that this class of preconditioning strategies is both robust and scalable with a different number of processors.

This paper is organized as follows. In section 2, we introduce some basic knowledge of sparse approximate inverse preconditioning techniques. We then give a detailed discussion of our motivation, ideas, and computational strategies for multistep successive preconditioning in section 3. In section 4, we give some numerical results to demonstrate the advantages of the new preconditioning strategies. Section 5 contains some brief concluding remarks.

2. Background. Many scientific and engineering models are established on the basis of a few partial differential equations governing certain physical properties and quantities. Computer simulations and modelings of such problems require discrete solution of these partial differential equations. Most sparse matrices are derived from discretizing linear or nonlinear partial differential equations by finite difference, finite element, finite volume, or other methods on structured or unstructured domains. Thus, real life sparse matrices may be very large and may not have regular structures. The efficient preconditioned iterative solution of general sparse matrices is therefore an important step in conducting large scale computer simulation and modeling. One area of active research is to construct robust parallel preconditioners for unstructured sparse matrices, so that the preconditioned linear systems can be solved efficiently by an iterative solver on parallel computers.

2.1. Sparse approximate inverse preconditioning. A sparse approximate inverse is a sparse matrix M which is a good approximation to A^{-1}, the inverse of

a general nonsingular sparse matrix A. The major driving force behind the search for efficient sparse approximate inverse preconditioners is their potential advantages in parallel computing. The idea is that, if such a matrix M can be constructed by some means, the preconditioning process is just a matrix-vector operation and is relatively easy to implement on high performance parallel computers [26], compared to the inherent sequential nature of the triangular solution procedures in the ILU factorization preconditioning techniques.

There exist several techniques to construct sparse approximate inverse preconditioners. They can be roughly categorized into three classes [8]: sparse approximate inverses based on Frobenius norm minimization [14, 21], sparse approximate inverses computed from an ILU factorization [19], and factored sparse approximate inverses [7, 38, 39]. Each of these classes contains a variety of different constructions and each of them has its own merits and drawbacks. Since our new concept of multistep successive preconditioning strategies can be applied to almost all of these sparse approximate inverse preconditioning techniques, we will only go into details on one particular sparse approximate inverse approach.

The sparse approximate inverse technique that we discuss here is based on the idea of least squares approximation. This is also the one that initially motivated research in sparse approximate inverse preconditioning [5, 6]. Consider a sparse linear system

$$Ax = b, \tag{2.1}$$

where A is a nonsingular general square matrix of order n. The convergence rate of a Krylov subspace solver applied directly to (2.1) may be slow due to the potential ill-conditioning of the matrix A. In order to speed up the convergence rate of the iterative methods, we may transform (2.1) into an equivalent system

$$MAx = Mb, \tag{2.2}$$

where M is a nonsingular matrix of order n. If M is a good approximation to A^{-1} in some sense, then MA can be a good approximation to the identity matrix I. It follows that the equivalent system (2.2) will be easier to solve, compared to solving (2.1), by a Krylov subspace solver.

A particular class of sparse approximate inverse preconditioners is constructed based on the Frobenius norm minimization idea. Since we want M to be a good approximation to A^{-1}, it is ideal if $MA \approx I$. This approach is to approximate A^{-1} from the left, and M is called the left preconditioner. It is also possible to approximate A^{-1} from the right, so that $AM \approx I$, which is termed as the right preconditioner. In the case of the right preconditioning, the equivalent preconditioned system analogous to (2.2) is

$$AMy = b \qquad \text{and} \qquad x = My. \tag{2.3}$$

In fact, the right preconditioning approach is easier for us to illustrate the Frobenius norm minimization idea, which will be described in detail in the following paragraphs.

In order to have $AM \approx I$, we want to minimize the functional

$$f(M) = \min_{M} \|AM - I\| \tag{2.4}$$

for all possible nonsingular square matrices M of order n, with respect to a certain norm. Without any constraint on M, the minimization problem (2.4) has an obvious

solution, i.e., $M = A^{-1}$. This obvious solution is undesirable for at least two reasons. First, the computational cost for solving the unconstrained minimization problem (2.4) is prohibitively high. Second, for most sparse matrices A, their inverses A^{-1} are dense, which will cause memory problems for large scale matrices encountered in many practical applications.

Thus we are interested in a constrained minimization such that M has a certain sparsity pattern (nonzero structure), i.e., only certain entries of M are allowed to be nonzero. Given a set of sparsity patterns (this set is usually unknown a priori) Ω, we minimize the functional

$$f(M) = \min_{M \in \Omega} \|AM - I\|. \tag{2.5}$$

Although any norms can potentially be used in the above definition, a particularly convenient norm is the Frobenius norm which is defined for a matrix $A = (a_{ij})_{n\times n}$ as $\|A\|_F = \sqrt{\sum_{i,j=1}^{n} a_{ij}^2}$ [29]. With the Frobenius norm, the minimization problem (2.5) is decoupled into n independent subproblems and can proceed as (using square for convenience)

$$\|AM - I\|_F^2 = \sum_{k=1}^{n} \|(AM - I)e_k\|_2^2 = \sum_{k=1}^{n} \|Am_k - e_k\|_2^2, \tag{2.6}$$

where m_k and e_k are the kth column of M and that of I, respectively. It follows that the minimization problem (2.5) is equivalent to minimizing the individual functions

$$\|Am_k - e_k\|_2, \qquad k = 1, 2, \ldots, n, \tag{2.7}$$

with certain restrictions placed on the sparsity pattern of m_k. In other words, each column of M can be computed independently. For the moment, we assume that the sparsity pattern of m_k is given a priori, i.e., a few, say n_2, entries of m_k at certain locations are allowed to be nonzero, the rest of the entries are forced to be zero. Denote the n_2 nonzero entries of m_k by $\tilde{m}_k$ and the n_2 columns of A corresponding to $\tilde{m}_k$ by A_k. Since A is sparse, the submatrix A_k has many rows that are identically zero. After removing the zero rows, we have a reduced matrix $\tilde{A}_k$ with n_1 rows. The individual minimization problem (2.7) is reduced to a least squares problem of order $n_1 \times n_2$:

$$\min_{\tilde{m}_k} \|\tilde{A}_k \tilde{m}_k - \tilde{e}_k\|_2, \qquad k = 1, 2, \ldots, n. \tag{2.8}$$

We note that the matrix $\tilde{A}_k$ is usually a very small rectangular matrix. It has full rank if A is nonsingular.

There are a variety of methods available to solve the least squares problem (2.8). One approach, proposed by Grote and Huckle [21], is to solve (2.8) using a QR factorization as

$$\tilde{A}_k = Q_k \begin{pmatrix} R_k \\ 0 \end{pmatrix}, \tag{2.9}$$

where R_k is a nonsingular upper triangular $n_2 \times n_2$ matrix. Q_k is an $n_1 \times n_1$ orthogonal matrix such that $Q_k^{-1} = Q_k^T$. The least squares problem (2.8) is solved by first computing $\tilde{c}_k = Q_k^T \tilde{e}_k$ and then obtaining the solution as $\tilde{m}_k = R_k^{-1} \tilde{c}_k(1 : n_2)$. In

this way, $\tilde{m}_k$ can be computed for each $k = 1, 2, \ldots, n$, independently. This yields an approximate inverse matrix M, which minimizes $\|AM - I\|_F$ for the given sparsity pattern.

The inherent parallelism is obvious in the process of computing $\tilde{m}_k$ independently of each other. It can be implemented straightforwardly on a model parallel random access machine without considering communication cost [26].

2.2. Static and dynamic sparsity patterns. The remaining problem for constructing a sparse approximate inverse preconditioner is to decide how to choose a good sparsity pattern for M. There are a few heuristic strategies; both static and dynamic sparsity pattern approaches have been proposed [14, 16, 25].

The dynamic sparsity pattern strategies can usually compute better sparse approximate inverse preconditioners, given a certain sparsity ratio (density) for M. However, dynamic strategies are usually expensive and more difficult to implement on parallel computers, although the implementation by Barnard, Bernardo, and Simon [3] is certainly attractive. Let us take a look at, for example, the dynamic strategy proposed by Grote and Huckle [21]. Given an initial sparsity pattern, the least squares problem (2.8) is solved and an initial $\tilde{m}_k$ is computed. Another subminimization is then conducted to find which of the zero positions in m_k can be augmented into $\tilde{m}_k$ so that the residual in (2.8) can be reduced most. The difficulty is associated with the matrix $\tilde{A}_k$. As $\tilde{m}_k$ is augmented, the corresponding rows and columns of $\tilde{A}_k$ have to be augmented also. These rows and columns are not necessarily in the local processor and have to be transferred from other processors on a distributed memory architecture. If this sparsity pattern search procedure is to be executed for a few times for each k, the communication cost is likely to present a big problem on a distributed memory computer with many processors on which many $\tilde{m}_k$ are computed and updated simultaneously.

It seems that a static (a priori) sparsity pattern can be more attractive to implement on distributed memory parallel computers. This is, however, not an easy answer. Although communications are still needed to assemble local matrix $\tilde{A}_k$, the biggest problem for static sparsity pattern is that, for general sparse matrices, there is no useful information to determine whether a static pattern is a good one before a sparse approximate inverse is computed and tested. On the other hand, for certain matrices, numerical experiments show that some static sparsity patterns may be good. For example, banded sparsity patterns can be used for banded matrices [6, 22].

A particularly useful and effective strategy is to use the sparsity pattern of the matrix A or A^T. Chow [13] proposes using sparsified patterns of A as the sparsity pattern for M. Here "sparsified" means that certain small entries of A are removed before its sparsity pattern is extracted. For achieving higher accuracy, the sparsity patterns of (sparsified) A^2, $A^3, \ldots$ may be used. Here the matrices $A^2, A^3, \ldots$ are not explicitly computed, only their sparsity patterns are extracted from that of the matrix A with binary operations. Several auxiliary strategies are proposed to make such an approach practically useful. A software package, ParaSails, which implements the static sparsity pattern sparse approximate inverse preconditioning, has been announced to the public [11, 12].

3. Multistep successive preconditioning. It has been noticed that high accuracy sparse approximate inverse preconditioners may be difficult and very expensive to compute using a static sparsity pattern as in ParaSails [12, 28]. Experimental results indicate that, compared to incomplete Cholesky factorization, sparse approximate inverse preconditioning wins only when the factorization is not required to be

TABLE 3.1
Test results using ParaSails from [11].

Sparsity pattern	Sparsity ratio	Iteration	Setup time	Solution time
A	0.25	754	2.0	39.3
A^2	0.47	539	40.0	33.7
A^3	0.80	243	491.2	20.4

very accurate [28]. This is because it is difficult to determine a good static sparsity pattern a priori. Table 3.1 shows some test data using ParaSails with different levels of sparsity patterns from Chow's paper [11]. It is to solve a symmetric positive definite matrix with $n = 12,205$ and about 1.4 million nonzeros. Timings were taken on an IBM SP computer with PowerPC 604e (332MHz) processors. Four processors were used for the computation.

We can see that the use of higher level sparsity patterns, such as those of A^2 and A^3, can lead to better sparse approximate inverse preconditioners. This is indicated by the reduction in the number of preconditioned iterations (column 3). However, the CPU time in seconds needed to construct the preconditioners with higher accuracy (the setup time, column 4) increases substantially. The reduction in the solution time (column 5) does not compensate for the huge increase in setup time. Hence, it is difficult to justify in this case computing higher accuracy (more robust) sparse approximate inverse preconditioners. Unless, as Chow points out [11], we are in a situation that the higher accuracy sparsity pattern will be used in latter computation, the initial high cost of extracting high accuracy sparsity pattern may be amortized.

We can approach the problem of choosing a suitable sparsity pattern in another way. Suppose a (simple and cheap) sparse approximate inverse preconditioner M_1 is computed for the matrix A, using any available sparse approximate inverse construction techniques, e.g., ParaSails with the sparsified pattern of A. If somehow we find that M_1 is not very efficient, we can compute another sparse approximate inverse preconditioner M_2 for the preconditioned linear system

$$M_1 A x = M_1 b. \tag{3.1}$$

Here, for discussion convenience, we return to using left preconditioning. We note that the systems (2.1) and (3.1) are equivalent, if M_1 is nonsingular as assumed. Thus, we compute another (simple and cheap) sparse approximate inverse preconditioner M_2 for the matrix $A_2 = M_1 A$. The combined preconditioner is then $M_2 M_1$ for the matrix A. Here are a few comments to justify our MSP in the form of product matrix $M_2 M_1$.

- If we use the sparsity pattern (sparsified, if applicable) of A for M_1, which is cheap as we see from the experimental data in Table 3.1, the sparsity pattern of $A_2 = M_1 A$ is similar to that of A^2. However, computing a level 2 sparse approximate inverse preconditioner for A using the sparsity pattern of A^2 is different from computing a simple level 1 sparse approximate inverse preconditioner for A_2 using the sparsity pattern of A_2. The latter is much cheaper. This is because the cost of the standard sparse approximate inverse using (2.9) increases with q^3, where q is the average number of nonzero entries per column of $M_i, i = 1, 2$. By writing the sparse approximate inverse matrix M as a product of two (or a few) sparse matrices, the cost of computing and applying the preconditioner M increases only linearly with the number of the sparse matrices used in the preconditioner.[1]

[1] This cost relationship was suggested by one of the anonymous referees.

- The computation of $A_2 = M_1 A$ can be done efficiently on parallel computers. If p is the average number of nonzeros in each row of A, the cost of computing A_2 is approximately equal to p folds of applying M_1 on a dense vector, or less than that of $p/2$ preconditioned iteration steps, assuming that M_1 uses the sparsity pattern of A. Moreover, when we sparsify A_2 using a threshold parameter, the obtained sparsity pattern is more accurate than that of A^2, as it reflects the true pattern of A_2. The pattern of A^2 is computed from that of A using binary operations on the graph of A, without considering the size of the entries of A^2. Thus some useful information may get lost.
- If M_1 is an approximation to A^{-1}, albeit not a very good one, then $A_2 = M_1 A$ tends to be closer to I than A does, or A_2 tends to be more diagonally dominant than A does. Thus, computing a sparse approximate inverse for A_2 is usually easier than computing one for A, given the same conditions. In the numerical results section, we give some experimental results to justify this argument.
- Intuitively the inverse A^{-1} of a sparse matrix A is dense. Thus we expect an accurate approximation M for A^{-1} will generally be a dense matrix. This conflicts with our initial goal which is to find a sparse approximate inverse matrix M. However, using the product of two sparse matrices $M_2 M_1$ to approximate A^{-1}, we expect that $M_2 M_1$ may be capable of holding more information than a single matrix M can. In this viewpoint, using $M_2 M_1$ as a preconditioner is to some extent like using the factored sparse approximate inverse preconditioners [7, 38, 39]. Chow and Saad [15] also pointed out the potential advantages of using a few sparse matrices to approximate the inverse of a general sparse matrix, but no numerical results were reported.

A reader with recursive thinking will have already figured out the next step in the successive sparse approximate inverse procedure. If $M_2 M_1$ is not good enough for preconditioning the matrix A in question, we compute a third sparse approximate inverse matrix M_3 for the product matrix $A_3 = M_2 M_1 A$. This procedure can be continued for a few times to obtain a sequence of sparse matrices $M_1, M_2, \ldots, M_l$ such that $M_l M_{l-1} \cdots M_2 M_1 \approx A^{-1}$. If each M_i is good (but not necessarily very good) in some sense, we may expect that

$$\lim_{l \to \infty} M_l M_{l-1} \cdots M_2 M_1 = A^{-1}.$$

The algorithm for computing an MSP can be written as follows.

ALGORITHM 3.1.

0. *Given the number of steps $l > 0$, the threshold tolerance τ, and the filter parameter ϵ*
1. *Let $A_1 = A$*
2. *For $(i = 1; i < l; i++)$*
3. *Sparsify A_i with respect to τ*
4. *Compute a sparse approximate inverse $M_i \approx A_i^{-1}$*
5. *Drop small entries of M_i with respect to ϵ*
6. *Compute $A_{i+1} = M_i A_i$*
7. *End*
8. *Sparsify A_l with respect to τ*
9. *Compute a sparse approximate inverse $M_l \approx A_l^{-1}$*
10. *Drop small entries of M_l with respect to ϵ*
11. *$\prod_{i=1}^{l} M_i$ is the preconditioner for $Ax = b$*

Because the density of the matrix M_i increases as i increases, at each step we keep them sparse by dropping small size entries of A_i (before the sparse approximate inverse computation) and M_i (after the sparse approximate inverse computation). This is indicated in lines 3, 5, 8, and 10 in the algorithm. Such sparsification procedures are called preprocessing (τ) and postprocessing (ϵ) phases, respectively. We note that when $l = 1$, the algorithm is the same as a standard sparse approximate inverse algorithm.

A number of parallel implementations of some sparse approximate inverse preconditioned iterative solvers have been published. We can use any existing sparse approximate inverse packages, such as ParaSails of Chow [11, 12] and SPAI_1.1 of Barnard, Bernardo, and Simon [3], as the backbones for our MSP preconditioned iterative solvers.

In MSP, the values of the entries of the matrix $A_{i+1} = M_i A_i$, as well as its sparsified graph, will be used to construct the next step preconditioner M_2. It is intuitively correct that our strategies are likely to build good sparsity patterns step by step, if the sizes of the entries are indicators of certain locations of large entries of the sparse approximate inverse matrix. We expect that the MSP preconditioning strategy is *unlikely* to generate worse preconditioners than the original standard sparse approximate inverse preconditioning technique used as its backbone.

4. Experimental results. We conduct a few numerical experiments using a preliminary prototype code with MSP strategies outlined in the previous sections. This particular implementation uses ParaSails of Chow [12] as the backbone to build our MSP with different steps. This MSP code is mostly written in C programming language, with a few LAPACK routines written in Fortran programming language. The interprocessor communications are handled by MPI. The computations are carried out on a 32 processor (750MHz) subcomplex of an HP superdome (supercluster) at the University of Kentucky. Unless otherwise indicated explictly, 4 processors are used in our numerical experiments.

In all tables containing numerical results "*np*" is the number of processors used; "iter" shows how many iterations it takes for the preconditioned GMRES(50) to reduce the residual norm by 8 orders of magnitude. We also set an upper bound of 5000 for the GMRES iteration; a symbol "-" in a table indicates lack of convergence. "Density" stands for the sparsity ratio. In MSP, this is the sum of the number of nonzero entries of all M_i divided by the number of nonzero entries of the original matrix A. "Setup" is the total CPU time in seconds for constructing the preconditioner; "solve" is the total CPU time in seconds for solving the given sparse linear system using the preconditioned GMRES(50); "total" is the sum of "setup" and "solve." "τ" and "ϵ" are the two preprocessing and postprocessing parameters used in ParaSails and in MSP.

4.1. Test problems. In this subsection, we introduce the test problems which will be used in our experiments. The right-hand sides of all linear systems are constructed by assuming that the solution is a vector of all ones. The initial guess is a zero vector.

Convection-diffusion problem. The two-dimensional convection-diffusion problem

$$-u_{xx} - u_{yy} - 10\left(\sin x \cos \pi y \, u_x - \cos \pi x \sin y \, u_y\right) = 0 \tag{4.1}$$

is defined on the unit square. Here the so-called Reynolds number is 10. Dirichlet boundary conditions are assumed, but the artificial right-hand side mentioned pre-

TABLE 4.1
Information about some sparse matrices used in the experiments (n is the order of a matrix; nnz is the number of nonzero entries).

Matrices	n	nnz	Description
FIDAP024	2283	48733	nonsymmetric forward roll coating
FIDAP028	2603	77653	two merging liquids with one external interior interface
FIDAP031	3909	115299	dilute species deposition on a tilted heated plate
FIDAP036	3079	53851	chemical vapor deposition
FIDAP037	3565	67591	flow of plastic in a profile extrusion die
FIDAPM08	3876	103076	developing flow, vertical channel (angle = 0, Ra = 1000)
FIDAPM10	3046	53842	2D flow over multiple heat sources in a channel
PORES2	1224	9613	reservoir modeling
SHERMAN1	1000	3750	oil reservoir modeling, black oil simulation, shale barriers
PSMIGR1	3140	543162	demography, US intercounty migration 1965–1970
RAEFSKY1	3242	294276	flow in pressure driven pipe, time = 05
RAEFSKY2	3242	294276	flow in pressure driven pipe, time = 25

viously is used. The equation is discretized by using the standard 5-point central difference scheme. The resulting matrix is referred to as the 5-point matrix.

A three-dimensional convection-diffusion problem (defined on a unit cube)

$$u_{xx} + u_{yy} + u_{zz} + 1000\,(p(x,y,z)\,u_x + q(x,y,z)\,u_y + r(x,y,z)\,u_z) = 0 \tag{4.2}$$

is used to generate some large sparse matrices to test the implementation scalability of MSP. Here the convection coefficients are chosen as

$$\begin{aligned} p(x,y,z) &= x(x-1)(1-3y)(1-2z), \\ q(x,y,z) &= y(y-1)(1-2z)(1-2x), \\ r(x,y,z) &= z(z-1)(1-2x)(1-2y). \end{aligned}$$

The Reynolds number for this problem is 1000. Equation (4.2) is discretized by using the standard 7-point central difference scheme and the 19-point fourth order compact difference scheme [37]. The resulting matrices are referred to as the 7-point and 19-point matrices, respectively.

Test matrices. We also use MSP to solve the sparse matrices listed in Table 4.1.

The FIDAP matrices[2] were extracted from the test problems provided in the FIDAP package [20]. They arise from coupled finite element discretization of Navier–Stokes equations modeling incompressible fluid flows. The RAEFSKY matrices are from modeling incompressible flow in pressure driven pipe and are available from the University of Florida Sparse Matrix Collection [17].[3] The other matrices are from the well-known Harwell–Boeing sparse matrix collection [18].[4]

4.2. Comparison of preconditioning MA and A. We first compare the difference between preconditioning MA and A using ParaSails. The purpose of the comparison is to show that the matrix MA is usually more attractive than the matrix A to be used to construct a sparse approximate inverse preconditioner, which implies that (2.2) may be easier to solve, compared to solving (2.1).

The test results listed in Table 4.2 are from solving the two-dimensional convection-diffusion problem (4.1). The first column is the number of rows (unknowns) of the

[2]All FIDAP matrices are available online from MatrixMarket of the National Institute of Standards and Technology (http://math.nist.gov/MatrixMarket).

[3]http://www.csis.ufl.edu/∼davis/sparse.

[4]http://math.nist.gov/MatrixMarket.

TABLE 4.2
Comparison of preconditioning A and MA for solving the 5-point matrices.

	$Ax = b$		$MAx = Mb$	
n	Density	Iter	Density	Iter
100^2	2.58	195	2.58	139
200^2	2.59	354	2.59	249
250^2	2.59	443	2.59	354
300^2	2.59	535	2.59	400
350^2	2.59	576	2.59	427
400^2	2.60	681	2.60	536
450^2	2.60	821	2.60	625
500^2	2.60	864	2.60	688

matrices. The results in the second and third columns are to solve $Ax = b$ using a sparse approximate inverse preconditioner with the sparsity pattern of A^2. The results in the fourth and fifth columns are to solve $MAx = Mb$, which can be divided into two steps. First we use the sparsity pattern of A to obtain a sparse matrix $M \approx A^{-1}$, then we solve $MAx = Mb$ using a sparse approximate inverse preconditioner with the sparsity pattern of MA. In the experiments, we set the parameters τ and ϵ in ParaSails to be 0, so that no entries are dropped during the preprocessing and post-processing phases [12]. This implementation makes the sparsity pattern of MA the same as that of A^2.

We can see from Table 4.2 that the number of iterations needed to solve the matrix $MAx = Mb$ is usually 20% less than that to solve the matrix $Ax = b$ directly. This means that, with the same sparsity pattern or preconditioner density, which is indicated in the "density" columns, the preconditioned matrix MA can be solved faster than the matrix A. This motivates us to develop the MSP strategies.

4.3. Properties of MSP. Here we present numerical results to demonstrate some favorable properties of MSP strategies.

Diagonal dominance property. Figure 4.1 depicts the relationship between the number of steps in constructing MSP and the ratio of strongly diagonally dominant rows of A_i in solving a few sparse matrices using MSP. The number in the parentheses after the matrix name in Figure 4.1 is the number of iterations to solve the linear system using MSP with 8 steps.

From Figure 4.1 we can see that as the number of MSP steps increases, the ratio of strongly diagonally dominant rows of A_i increases quickly. In 4 of the test cases, the strongly diagonal dominance ratio finally reaches 1.0, i.e., 100%, after only a few steps. It is well known that diagonally dominant matrices are comparably easy to solve by iterative methods. Each of the linear systems can be solved with the 8-step MSP in no more than 10 iterations.

In the experiments we also find that when the ratio of the strongly diagonally dominant rows approaches 1, the structure of A_i tends to be similar to that of the identity matrix I with many small offdiagonal entries, compared to the magnitudes of the main diagonal entries. It is possible to use a diagonal matrix to approximate the strongly diagonally dominant product matrix. Hence, we need only to compute the main diagonal entries of the last matrix, and its inverse can be computed straightforwardly.

Sparsity ratio and number of iterations. Table 4.3 gives results of using MSP with different numbers of steps to solve the FIDAP031 matrix. We point out that if we use an oversparsified pattern of A to construct a preconditioner for A in the first step,

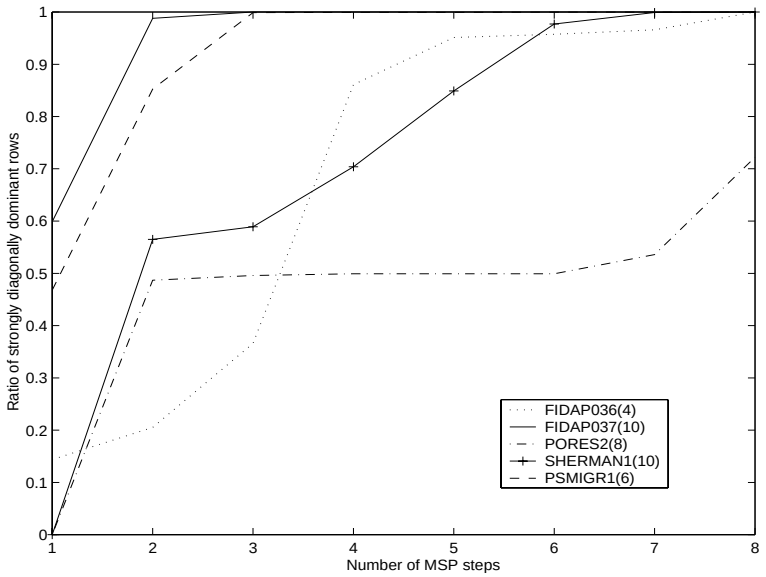

FIG. 4.1. *Relationship between the number of MSP steps and the ratio of strongly diagonally dominant rows.*

the resulting preconditioner may not converge. However, this "poor" preconditioner can be used as M_1 in MSP as the basis to construct M_2, and M_2M_1 may make the preconditioned solver converge. In our situation, we think M_2M_1 may still not be good enough, because it converges in 1439 iterations. We then use M_2M_1 as the basis to construct M_3. In our tests, $M_3M_2M_1$ seems to be a good preconditioner for A, and it converges in 573 iterations. Continuing, we find that using the 6-step MSP, the convergence is achieved in 342 iterations.

Our other experiments also indicate that a larger number of steps leads to better convergence results. But it is not the case that the more steps in MSP the better the constructed preconditioner. This is because in each step we compute a matrix $M_i \approx A_i^{-1}$. The memory cost of M_i will be counted into the total memory cost of the preconditioner, as well as the construction cost. This will obviously increase both the computational cost and the memory cost for MSP with a large number of steps. In Table 4.3 we notice that with 6 steps, the preconditioner converges in 342 iterations, but the total computational cost is 5 times as much as that with 2 steps. The reduction in the solution time does not compensate for the increase in the setup time. So unless convergence is not achieved with a lower number of steps or we want to solve a matrix repeatedly with many right-hand sides, we do not recommend too many steps in real applications, even though it may yield better convergence results. For the case reported in Table 4.3, we think that MSP with 2 or 3 steps is a good compromise between reasonable computational cost and good convergence results.

Table 4.4 lists experimental results using MSP to solve the FIDAPM10 matrix. We set the number of steps to be 2 and 3, then change the values of τ and ϵ in each case to see the relationship between the sparsity ratio and the number of iterations. For convenience, the values of τ and ϵ are always chosen to be equal.

From Table 4.4 we can see that increasing τ and ϵ leads to a smaller sparsity ratio due to the fact that more entries are dropped during the preprocessing and

TABLE 4.3
*Comparison of MSP with different number of steps to solve the FIDAP*031 *matrix* ($\tau = \epsilon = 0.03$).

Steps	Density	Iter	Setup	Solve	Total
1	0.38	-	0.3	-	-
2	1.01	1439	1.7	2.5	4.2
3	1.43	573	4.4	1.8	6.1
4	1.62	392	8.0	1.4	9.4
5	1.72	398	14.2	1.7	15.9
6	1.78	342	19.5	1.8	21.3

TABLE 4.4
Comparison of MSP using different preprocessing and postprocessing parameters (τ *and* ϵ) *to solve the FIDAPM*10 *matrix.*

Steps	$\tau(=\epsilon)$	Density	Iter	Setup	Solve	Total
2	0.001	4.58	426	10.0	0.8	10.8
	0.005	3.37	465	5.2	1.8	7.0
	0.01	2.71	484	4.5	0.7	5.2
	0.05	1.17	730	0.6	0.7	1.3
	0.07	0.85	936	0.5	0.9	1.4
3	0.001	12.70	99	110.6	0.7	111.3
	0.005	7.80	145	34.4	0.8	35.2
	0.01	5.80	169	15.9	0.8	16.7
	0.05	1.67	482	1.4	0.8	2.2
	0.07	1.14	646	0.8	0.9	1.7
	0.09	0.85	889	0.6	1.0	1.5

postprocessing phases. These sparsification strategies may degrade the convergence rate of the preconditioner to some extent and may increase the number of iterations. However, with different steps, a smaller sparsity ratio does not necessarily mean poorer convergence. For instance, in the 2-step case, MSP with a sparsity ratio of 2.71 converges in 484 iterations; in the 3-step case, MSP with a sparsity ratio of 1.67 converges in 482 iterations.

4.4. Comparison of ParaSails and MSP. In Table 4.5, we compare the performance of MSP and ParaSails (PAS). In each test, we choose parameters carefully and try to keep the memory costs (sparsity ratio) of these two preconditioners comparable. The content in the parentheses following "PAS" indicates the a priori pattern used in ParaSails, e.g., PAS(A^2) means that we use the (sparsified) sparsity pattern of A^2. Similarly, for MSP, the number in the parentheses is the number of steps, e.g., MSP(2) means a 2-step MSP.

When constructing a preconditioner, MSP spends less time than ParaSails if the sparsity ratios are comparable. According to our discussions in previous sections, the preconditioner computed from MSP is composed of a number of sparse matrices M_i. The memory cost of each sparse matrix is usually small and each of the sparse approximate inverse matrices can be computed very cheaply. So the total computational cost of these sparse matrices is also small, compared with computing a single sparse matrix with comparable density in the case of ParaSails.

Also the data in Table 4.5 shows that with the same amount of memory cost (sparsity ratio), MSP usually has better convergence performance than ParaSails does. For solving the FIDAPM08 matrix, ParaSails does not converge when using the sparsity pattern of either A^2 or A^3. A 3-step MSP converges with a sparsity ratio 3.28, although a 2-step MSP with the same τ and ϵ values does not converge.

TABLE 4.5
Comparison of ParaSails (PAS) and MSP for solving a few sparse matrices.

Matrices	Preconditioner	τ	ϵ	Density	Iter	Setup	Solve	Total
RAEFSKY1	PAS(A)	0.01	0.01	0.24	545	5.1	4.3	9.4
	PAS(A^2)	0.02	0.02	0.38	148	210.5	4.1	214.6
	MSP(2)	0.05	0.05	0.16	207	1.2	1.5	2.8
	MSP(3)	0.02	0.02	0.44	50	10.2	0.2	10.5
RAEFSKY2	PAS(A)	0.01	0.01	0.38	786	6.0	7.5	13.5
	PAS(A^2)	0.02	0.01	1.02	196	263.5	3.3	266.8
	MSP(2)	0.05	0.02	0.32	535	2.9	3.3	6.2
	MSP(3)	0.02	0.01	0.93	169	31.9	1.1	33.0
FIDAP024	PAS(A^2)	0.0	0.0	4.86	-	9.8	-	-
	PAS(A^3)	0.01	0.01	6.93	285	52.7	3.3	55.9
	MSP(2)	0.001	0.002	4.47	799	12.2	4.4	16.6
	MSP(3)	0.01	0.01	4.87	188	14.4	1.8	16.3
FIDAP028	PAS(A^2)	0.0	0.0	4.29	789	19.3	6.7	25.9
	PAS(A^2)	0.001	0.001	4.16	835	19.5	7.8	27.3
	MSP(2)	0.004	0.004	2.97	255	13.4	2.9	16.2
	MSP(2)	0.005	0.005	2.75	330	10.7	3.3	14.0
FIDAPM08	PAS(A^2)	0.0	0.0	5.21	-	37.8	-	-
	PAS(A^3)	0.0	0.0	12.91	-	377.0	-	-
	MSP(3)	0.01	0.01	3.28	729	48.3	3.4	51.7
	MSP(4)	0.01	0.01	5.12	291	142.2	2.2	144.5

TABLE 4.6
Scalability of MSP for solving a 7-point matrix with $n = 100^3$ ($\tau = \epsilon = 0.05$).

np	Density	Iter	Setup	Solve	Total
4	1.74	288	1953.4	232.8	2186.1
8	1.74	288	984.1	121.0	1105.0
16	1.74	288	501.9	44.4	546.3
24	1.74	288	361.7	29.4	391.1
32	1.74	288	281.8	24.7	306.5

4.5. Implementation scalability. According to Algorithm 3.1, the main computational costs in MSP strategies are matrix-matrix product and matrix-vector product operations. As is well known [26], these operations can be performed in parallel efficiently on most distributed memory parallel architectures.

The implementation scalability is tested using a three-dimensional convection-diffusion problem (4.2). For the 7-point matrix we let $n = 100^3$. The number of nonzeros entries is 6940000. The matrix is solved by using a 2-step MSP. For the 19-point matrix we choose $n = 80^3$. The number of nonzeros entries is 9536960. It is solved by using a 3-step MSP. Tables 4.6 and 4.7 show the computational results with different numbers of processors. We can see that MSP scales very well in these two test cases. In particular, we point out that the number of iterations remains constant as the number of processors increases from 4 to 32. This is because all the components of the preconditioned solver (the sparse approximate inverse approach and the GMRES algorithm) are inherently parallel. Its performance does not depend on the ordering of the unknowns and does not degrade with an increasing number of processors. This is different from the simple domain decomposition preconditioners whose convergence performance is usually affected by the number of processors (domains) involved [9, 34].

We remark that for the "solve" time listed in the 5th column of Tables 4.6 and 4.7, the timing results show some superlinear speedup effect for up to 32 processors. This is largely because of the well-known cache effect commonly seen on such large

TABLE 4.7
Scalability of MSP for solving a 19-point matrix with $n = 80^3$ ($\tau = \epsilon = 0.05$).

np	Density	Iter	Setup	Solve	Total
4	0.74	171	1049.9	69.4	1119.3
8	0.74	171	529.1	31.0	560.1
16	0.74	171	269.9	15.0	285.0
24	0.74	171	184.3	12.0	196.3
32	0.74	171	146.6	8.3	154.8

scale parallel computer systems.

5. Concluding remarks. We have proposed a class of multistep successive sparse approximate inverse preconditioning (MSP) strategies for solving general sparse matrices. A prototype implementation is tested to show favorable convergence rate and computational efficiency of this class of new preconditioning strategies.

Our numerical experiments with several sparse matrices show that as the number of MSP steps increases, the matrix A_i we compute in each step tends to become more strongly diagonally dominant. Solving these strongly diagonally dominant matrices is relatively easier, compared to solving the original matrix. Usually, multistep preconditioners with a small number of steps lead to good convergence results, even with a low memory cost. Multistep preconditioners with a large number of steps may be more robust, but they may also lead to both higher computational cost and higher memory cost. It is our experience that the number of steps should be chosen as 2 or 3, provided that the preconditioned solver can converge well.

When compared with ParaSails, MSP is cheaper in its setup phase. Generally speaking, to compute a series of small memory cost (sparse) matrices may be cheaper than to compute a high memory cost (sparse) matrix. However, a series of small memory cost (sparse) matrices together may hold more information about the inverse of the original matrix, it can make the preconditioned solver more robust and converge faster.

The scalability of MSP is also tested in our numerical experiments. The MSP preconditioner seems to scale well for the tested two and three dimensional convection-diffusion problems.

In conclusion, the performance of MSP is indeed as good as we expected. But some detailed work still needs be done in the future study, in order to build a software package that may be used in realistic large scale scientific computation and computer simulations. For example, the preprocessing and postprocessing sparsity parameters are important in this class of algorithms (both MSP and ParaSails). However, in our current implementation, we do not consider different values in different construction steps and always keep both parameters the same. It is possible that we may be able to choose these parameters adaptively in a more sophisticated implementation.

We remark that the concepts of multistep successive preconditioning may be applied to other preconditioning techniques. It is also possible to construct multistep successive ILU preconditioners [35, 36] or to construct multistep hybrid successive preconditioners using several different preconditioning techniques in each step.

REFERENCES

[1] O. AXELSSON AND P. S. VASSILEVSKI, *Variable-step multilevel preconditioning methods.* I. *Self-adjoint and positive definite elliptic problems*, Numer. Linear Algebra Appl., 1 (1994), pp. 75–101.

[2] R. E. Bank and C. Wagner, *Multilevel ILU decomposition*, Numer. Math., 82 (1999), pp. 543–576.
[3] S. T. Barnard, L. M. Bernardo, and H. D. Simon, *An MPI implementation of the SPAI preconditioner on the T3E*, Int. J. High Perf. Comput. Appl., 13 (1999), pp. 107–128.
[4] T. Barth, T. F. Chan, and W.-P. Tang, *A parallel non-overlapping domain-decomposition algorithm for compressible fluid flow problems on triangulated domains*, in Domain Decomposition Methods, AMS, Providence, RI, 1998, pp. 23–41.
[5] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Util. Math., 22 (1982), pp. 127–140.
[6] M. W. Benson, J. Krettmann, and M. Wright, *Parallel algorithms for the solution of certain large sparse linear systems*, Int. J. Comput. Math., 16 (1984), pp. 245–260.
[7] M. Benzi and M. Tuma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.
[8] M. Benzi and M. Tuma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.
[9] X.-C. Cai, W. D. Gropp, and D. E. Keyes, *A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems*, Numer. Linear Algebra Appl., 1 (1994), pp. 477–504.
[10] T. F. Chan, S. Go, and J. Zou, *Multilevel Domain Decomposition and Multigrid Methods for Unstructured Meshes: Algorithms and Theory*, Technical Report CAM 95-24, Department of Mathematics, UCLA, Los Angeles, CA, 1995.
[11] E. Chow, *Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, Int. J. High Perf. Comput. Appl., 15 (2001), pp. 56–74.
[12] E. Chow, *ParaSails Users' Guide*, Technical Report UCRL-JC-137863, Lawrence Livermore National Laboratory, Livermore, CA, 2000.
[13] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.
[14] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.
[15] E. Chow and Y. Saad, *Approximate Inverse Preconditioners for General Sparse Matrices*, Technical Report UMSI 94/101, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1994.
[16] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank, *Approximate inverse preconditionings for sparse linear systems*, Int. J. Comput. Math., 44 (1992), pp. 91–110.
[17] T. Davis, *University of Florida sparse matrix collection*, NA Digest, 97 (1997).
[18] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.
[19] A. C. N. van Duin, *Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 987–1006.
[20] M. Engelman, *FIDAP: Examples Manual, Revision* 6.0, Technical report, Fluid Dynamics International, Evanston, IL, 1991.
[21] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[22] M. Grote and H. D. Simon, *Parallel preconditioning and approximate inverse on the Connection machines*, in Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, eds., SIAM, Philadelphia, 1993, pp. 519–523.
[23] D. Hysom and A. Pothen, *A scalable parallel algorithm for incomplete factor preconditioning*, SIAM J. Sci. Comput., 22 (2001), pp. 2194–2215.
[24] D. E. Keyes and W. D. Gropp, *A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 166–202.
[25] L. Y. Kolotilina, *Explicit preconditioning of systems of linear algebraic equations with dense matrices*, J. Soviet Math., 43 (1988), pp. 2566–2573.
[26] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, Benjamin/Cummings, Redwood City, CA, 1994.
[27] M. Magolu monga Made and H. A. van der Vorst, *Parallel incomplete factorization with pseudo-overlapped subdomains*, Parallel Comput., 27 (2001), pp. 989–1008.
[28] P. Raghavan, K. Teranishi, and E. Ng, *Towards scalable preconditioning using incomplete Cholesky factorization*, in Proceedings of the 2001 Conference on Preconditioning Tech-

niques for Large Scale Matrix Problems in Industrial Applications, Tahoe City, CA, 2001, pp. 63–65.

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

[30] Y. Saad and M. Sosonkina, *Distributed Schur complement techniques for general sparse linear systems*, SIAM J. Sci. Comput., 21 (1999), pp. 1337–1356.

[31] Y. Saad and J. Zhang, *BILUM: Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., 20 (1999), pp. 2103–2121.

[32] Y. Saad and J. Zhang, *BILUTM: A domain-based multilevel block ILUT preconditioner for general sparse matrices*, SIAM J. Matrix Anal. Appl., 21 (1999), pp. 279–299.

[33] C. Shen and J. Zhang, *Parallel two level block ILU preconditioning techniques for solving general sparse linear systems*, Parallel Comput., 28 (2002), pp. 1451–1475.

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

[35] L. Wang and J. Zhang, *A two step combined stable preconditioning strategy for incomplete LU factorization of CFD matrices*, in Proceedings of the 2002 Copper Mountain Conference on Iterative Methods, Vol. I, T. Manteuffel and S. McCormick, eds., Copper Mountain, CO, 2002.

[36] L. Wang and J. Zhang, *A new stabilization strategy for incomplete LU preconditioning of indefinite matrices*, Appl. Math. Comput., to appear.

[37] J. Zhang, *An explicit fourth-order compact finite difference scheme for three dimensional convection-diffusion equation*, Comm. Numer. Methods Engrg., 14 (1998), pp. 209–218.

[38] J. Zhang, *A parallelizable preconditioner based on a factored sparse approximate inverse technique*, in Proceedings of the 1999 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Industrial Applications, Y. Saad, D. Pierce, and W.-P. Tang, eds., University of Minnesota, Minneapolis, MN, 1999, pp. 193–199.

[39] J. Zhang, *A sparse approximate inverse technique for parallel preconditioning of general sparse matrices*, Appl. Math. Comput., 130 (2002), pp. 63–85.