Parallel Multistep Successive Sparse Approximate Inverse Preconditioning Strategies of General Sparse Matrices *

Kai Wang†
Laboratory for High Performance Scientific Computing and Computer Simulation,
Department of Computer Science, University of Kentucky,
Lexington, KY 40506–0046, USA

Abstract

We develop new concepts and parallel algorithms of 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. Numerical experiments using one prototype implementation to solve a few general sparse matrices on a distributed memory parallel computer are reported.

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

1 Introduction

Consider a large sparse linear sytem

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

where A is a nonsingular general matrix of order n. A sparse approximate inverse preconditioning technique is first to find a sparse matrix M which is a good approximation to A^{-1}, then to solve a transformed system, in the form of

$$MAx = Mb, \tag{2}$$

by a Krylov subspace accelerator. The major driving force behind the search for efficient sparse approximate inverse preconditioners has been their potential advantages in parallel computing.

There exist several techniques to construct sparse approximate inverse preconditioners. They can be roughly categorized into three classes [3], sparse approximate inverses based on Frobenius norm minimization [7, 10], sparse approximate inverses computed from an ILU factorization [8, 13], and factored sparse approximate inverses [2, 15, 16]. Each of these classes contains a variety of different constructions and each of them has its own merits and drawbacks.

*This research was supported in part by the U.S. National Science Foundation under grants CCR-9902022, CCR-9988165, and CCR-0092532.

†E-mail: kwang0@csr.uky.edu. URL: http://www.csr.uky.edu/~kwang0.

In this paper, we investigate a class of multistep successive sparse approximate inverse preconditioning 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.

This paper is organized as follows. We give detailed discussion on our motivation, ideas, and computational strategies for multistep successive preconditioning in Section 2. In Section 3, we give some numerical results to demonstrate the advantages of the new preconditioning strategies. Section 4 contains some brief concluding remarks.

2 Enhancing Robustness via Successive Preconditioning

A sparse preconditioner may be computed using either dynamic or static sparsity pattern. The use of dynamic sparsity pattern may compute more accurate sparse approximate inverse preconditioners. But parallel implementation of dynamic sparsity pattern search can be quite expensive due to large amounts of data movement. It has also been noticed that high accuracy sparse approximate inverse preconditioners may be difficult and expensive to compute using a static sparsity pattern [6, 12]. Experimental results indicate that, compared to incomplete Cholesky factorization, sparse approximate inverse preconditioning wins only when the factorization is not required to be very accurate [12]. This is because it is very difficult to determine a very good static sparsity pattern *a priori*. Table 1 lists some test data using ParaSails, a software package implementing a sparse approximate inverse preconditioning, with different levels of sparsity patterns from Chow's paper [5]. It is to solve a symmetric positive definite matrix with $n = 12,205$ and about 1.4 million nonzeros.

We can see that use of higher level sparsity patterns, such as those of A^2 and A^3, does 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 setup time. Hence, it is difficult to justify in this case to compute higher accuracy (more robust) sparse approximate inverse preconditioners.

We can approach the problem of choosing a suitable sparsity pattern in another way. Suppose a (*simple and cheap*) sparse approximate inverse preconditioner M_0 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_0 is not very efficient, we can compute another sparse approximate inverse preconditioner M_1 for the preconditioned linear system

$$M_0 A x = M_0 b. \tag{3}$$

We note that the systems (1) and (3) are equivalent, if M_0 is nonsingular as assumed. Thus, we compute another (*simple and cheap*) sparse approximate inverse preconditioner M_1 to the matrix $A_1 = M_0 A$. The combined preconditioner is then $M_1 M_0$ for the matrix A. Here are a few comments to justify our successive sparse approximate inverse preconditioner in the form of product matrix $M_1 M_0$.

- The computation of $A_1 = M_0 A$ can be done efficiently on parallel computers. If p is the

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	491.0	20.4

Table 1: Test results using ParaSails from [5].

average number of nonzeros in each row of A, the cost of computing A_1 is approximately equal to p folds of applying M_0 on a dense vector, or less than that of $p/2$ preconditioned iteration steps, assuming that M_0 uses the sparsity pattern of A. Moreover, when we sparsify A_1 using a threshold parameter, the obtained sparsity pattern is more accurate than that of A^2, as it reflects the true pattern of A_1. 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_0 is an approximation to A, albeit not a very good one, then $A_1 = M_0 A$ tends to be closer to I than A does, or A_1 tends to be more diagonally dominant than A does. Thus, computing a sparse approximate inverse for A_1 is usually easier than computing one for A, given the same conditions. In Section 3.2, we give some test results to justify this argument.
- Intuitively the inverse A^{-1} of a sparse matrix A is dense. Then usually an accurate approximation M for A^{-1} should be a dense matrix. This conflicts with our initial goal which is to find a sparse approximate inverse matrix M. However, if using the product of two sparse matrix $M_1 M_0$ to approximate A^{-1}, we expect that $M_1 M_0$ may be capable of holding more information than a single matrix M is. In this viewpoint, using $M_1 M_0$ as a preconditioner is to some extend like using the factored sparse approximate inverse preconditioners [2, 15, 16].

A reader with recursive thinking will have already figured out the next step in the successive sparse approximate inverse procedure. If $M_1 M_0$ is not good enough for preconditioning the matrix A in question, we compute a third sparse approximate inverse matrix M_2 for the product matrix $A_2 = M_1 M_0 A$. This procedure can be continued for a few times to obtain a sequence of sparse matrices $M_0, M_1, \ldots, M_l$, such that $M_l M_{l-1} \cdots M_1 M_0 \approx A^{-1}$. If each M_i is good (but not necessarily very good) in some sense, we may expect that

$$\lim_{l \to \infty} M_l M_{l-1} \cdots M_1 M_0 = A^{-1}.$$

The algorithm for computing a multistep sparse approximate inverse preconditioner can be written as follows.

ALGORITHM **2.1**

Let $l = stepNum$, $A_0 = A$
For $(i = 1; i <= l; i++)$
    Sparsify A_{i-1}
    Compute a sparse approximate inverse $M_{i-1} \approx A_{i-1}^{-1}$
    Drop small entries of M_{i-1}
    Compute $A_i = M_{i-1} A_{i-1}$
End
Sparsify A_l
Compute a sparse approximate inverse $M_l \approx A_l^{-1}$
Drop small entries of M_l
$\prod_{i=0}^{l} M_i$ is the preconditioner for $Ax = b$

Because the matrix M_i becomes denser and denser as i increases, in each step we keep them sparse by dropping certain small size entries. This is indicated in Lines 3, 5, 8 and 10 in the algorithm. We note that when $stepNum = 0$, the algorithm is the same as a standard sparse approximate inverse algorithm.

There are a few sparse approximate inverse algorithms published by different researchers [2, 7, 16]. There are also a few parallel implementations of some sparse approximate inverse preconditioned iterative solvers [1, 5, 6]. We can use any existing sparse approximate inverse packages, such as ParaSails of Chow and SPAI_1.1 of Barnard et. al. [1], as the backbones for our multistep successive sparse approximate inverse preconditioned iterative solvers.

3 Experimental Results

We conduct a few numerical experiments using a preliminary prototype code with multistep successive sparse approximate inverse techniques outlined in the previous section. This particular implementation uses ParaSails of Chow [6] as the backbone to build our multistep sparse approximate inverse preconditioner. For this reason, we refer to our preconditioner here as *MultiSails* with different steps. The code is mostly written in C programming language, with interprocessor communications being 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. When computing $M_i \approx A_i^{-1}$ in each step, we use the (sparsified) pattern of A_i as the *a priori* sparsity pattern.

In all tables containing numerical results, "n" denotes the dimension of the matrix; "nnz" represents the number of nonzeros in the sparse matrix; "np" is the number of processors used; "iter" shows how many iterations it takes for the preconditioned GMRES(50) to reduce 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. Similarly, "s-ratio" stands for the sparsity ratio. In MultiSails, this is the sum of the number of nonzero entries of each M_i divided by the number of nonzero entries of 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 matrix using the preconditioner; "total" is the sum of "setup" and "solve". "thre" and "filt" are two parameters used by Chow in ParaSails [5]. In MultiSails we also use these two parameters to keep the memory cost small in each step. The content in the parentheses following "PS" indicates the *a priori* pattern used in ParaSails, e.g., PS(A^2) means that we use the sparsity pattern of A^2. Similarly, we use "MS" to denote MultiSails, the number in the followed parentheses is the step number, e.g., MS(2) means a 2 step MultiSails preconditioner.

3.1 Test Problems

In this section, 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

$$-u_{xx} - u_{yy} - 10\,(\sin x \cos \pi y\, u_x - \cos \pi x \sin y\, u_y) = 0, \tag{4}$$

is defined on the unit square. Here the so-called Reynolds number value is 10. Dirichlet boundary

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)
PORES2	1224	9613	reservoir modeling
SHERMAN1	1000	3750	oil reservoir modeling, black oil simulation, shale barriers
PSMIGR1	3140	543162	demography, US inter-county migration 1965-1970
RAEFSKY1	3242	294276	flow in pressure driven pipe, time = 05
RAEFSKY2	3242	294276	flow in pressure driven pipe, time = 25

Table 2: Information about some sparse matrices used in the experiments.

condition is assumed, but the artificial right hand side mentioned previously 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)

$$u_{xx} + u_{yy} + u_{zz} + 1000\,(p(x,y,z)\,u_x + q(x,y,z)\,u_y + r(x,y,z)\,u_z) = 0 \qquad (5)$$

is used to generate some large sparse matrices to test the implementation scalability of MultiSails. 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 value for this problem is 1000. Equation (5) is discretized by using the standard 7-point central difference scheme [14]. The resulting matrices are referred to as the 7-point matrix.

Test matrices. We also use MultiSails to solve a few sparse matrices listed in Table 2.

The FIDAP matrices [1] were extracted from the test problems provided in the FIDAP package [9]. They arise from coupled finite element discretization of Navier-Stokes equations modeling incompressible fluid flows. Some FIDAP matrices may have zero main diagonals and they are difficult to solve by standard incomplete LU factorization preconditioners without high levels of fill-in.

The RAEFSKY1 and RAEFSKY2 matrices are from modeling incompressible flow in pressure driven pipe, and are available from the University of Florida Sparse Matrix Collection. [2] The other matrices are from the well known Harwell-Boeing sparse matrix collection.

3.2 Comparison of solving MA and A

First, we use ParaSails, the software package developed by Chow, to compute a sparse approximate inverse matrix M for the matrix A (using the sparsity pattern of A). We then compute another

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

[2] `http://www.csis.ufl.edu/~davis/sparse`.

	$Ax = b$		$MAx = Mb$	
n	s-ratio	iter	s-ratio	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

Table 3: Comparison of preconditioning MA and A for solving the 5-point matrices.

sparse approximate inverse preconditioner for the product matrix MA. The purpose of this comparison is to show that MA usually is more attractive than A to be used to construct sparse approximate inverse preconditioner, which means Equation (2) may be easier to solve, compared to solving Equation (1).

The test results listed in Table 3 are from solving the two dimensional convection-diffusion problem (4). The first column is the number of rows (unknowns) of the 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 forth 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 sparse approximate inverse preconditioner with the sparsity pattern of MA. In the experiments, we set the parameters "filt" and "thre" in ParaSails to be 0, so that it does not drop anything during the preprocessing and postprocessing phases [6]. This implementation makes the sparsity pattern of MA the same as that of A^2.

We can see from Table 3 that the iteration numbers needed to solve the matrix MA are usually 20% less than that to solve the matrix A directly. That means under the same sparsity pattern or preconditioner density, which is indicated in the "s-ratio" columns, preconditioning matrix MA can get better convergence results than preconditioning matrix A directly. This property motivates us to develop multistep successive sparse approximate inverse techniques.

3.3 Properties of MultiSails

In this subsection, we present results from a few numerical experiments to demonstrate some favorable properties of multistep sparse approximate inverse preconditioners.

Diagonal dominance property. Figure 1 depicts the relationship between the number of steps in constructing the multistep sparse approximate inverse preconditioner and the ratio of strongly diagonally dominant rows of A_i in solving a few sparse matrices using MultiSails. The number after the matrix name in Figure 1 denotes the iteration number to solve the given matrix using the multistep sparse approximate inverse preconditioner with 7 steps.

From Figure 1 we can see that when the step number 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. So after 7 steps, all these matrices can be solved with the multistep sparse approximate inverse preconditioner in no more than 10 iterations.

In the experiments we also find that when the strongly diagonally dominant row ratio ap-

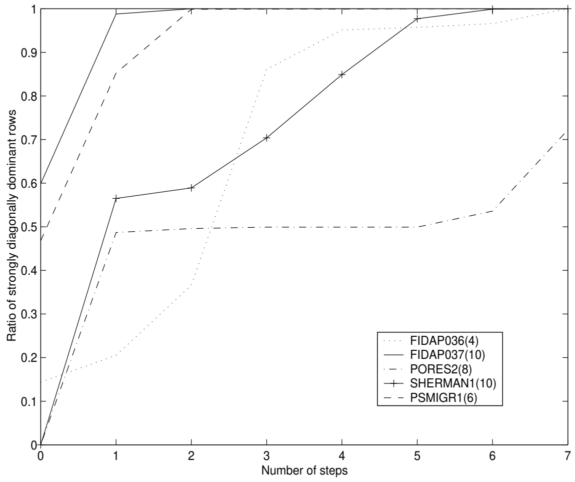

Figure 1: Relationship between the number of steps and the ratio of strongly diagonally dominant rows.

proaches 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. So that we only need to compute the main diagonal entries of the last matrix and its inverse can be computed straightforwardly.

Sparsity ratio and iteration number. Table 4 gives some results from using MultiSails with different steps to solve the FIDAP031 matrix. We note that 0 step here can be regarded as a standard sparse approximate inverse algorithm, i.e., ParaSails in the current case. We point out that if we use an oversparsified pattern of A to construct a preconditioner for A at the 0th step, the resulting preconditioner may not converge. However, this "poor" preconditioner can be used as M_0 in MultiSails as the basis to construct M_1, and M_1M_0 may make the preconditioned solver converge. In our situation, we think M_1M_0 may still not be good enough, because it converges after 1439 iterations. We then use M_1M_0 as the basis to construct M_2. In out tests, $M_2M_1M_0$ seems to be a good preconditioner for A, and it converges after 573 iterations. Continue doing this, we find that the multistep preconditioner converges in 342 iterations after 5 steps.

Our other experiments also indicate that a larger step number leads to better convergence results. But it is not the case that the more steps in MultiSails the better the constructed preconditioner. This is because in each step we compute a matrix $M_i \approx A_i^{-1}$ and the memory cost of M_i will be counted into the whole memory cost of the preconditioner, as well as the construction cost. This obviously will increase both our computational cost and memory cost for MultiSails with a large number of steps. In Table 4 we notice that in the 5th step case, the preconditioner converges in 342 iterations, but the total computational cost is 5 times as much as that in the 1st step case. The reduction in the solution time does not compensate for the increase in the setup time. So unless it does not converge with a lower number of steps or in the case of solving one matrix with many right hand sides, usually we do not recommend too many steps in real applications, even though that may yield better convergence results. In Table 4, we think that 1 or 2 steps is a good compromise between reasonable computational cost and good convergence results.

steps	thre	filt	s-ratio	iter	setup	solve	total
0	0.03	0.03	0.38	-	0.3	-	-
1	0.03	0.03	1.01	1439	1.7	2.5	4.2
2	0.03	0.03	1.43	573	4.4	1.8	6.1
3	0.03	0.03	1.62	392	8.0	1.4	9.4
4	0.03	0.03	1.72	398	14.2	1.7	15.9
5	0.03	0.03	1.78	342	19.5	1.8	21.3

Table 4: Comparison of MultiSails with different number of steps to solve the FIDAP031 matrix.

Matrices	Preconditioner	thre	filt	s-ratio	iter	setup	solve	total
RAEFSKY1	PS(A)	0.01	0.01	0.24	545	5.1	4.3	9.4
	PS(A^2)	0.02	0.02	0.38	148	210.5	4.1	214.6
	MS(1)	0.05	0.05	0.16	207	1.2	1.5	2.8
	MS(2)	0.02	0.02	0.44	50	10.2	0.2	10.5
RAEFSKY2	PS(A)	0.01	0.01	0.38	786	6.0	7.5	13.5
	PS(A^2)	0.02	0.01	1.02	196	263.5	3.3	266.8
	MS(1)	0.05	0.02	0.32	535	2.9	3.3	6.2
	MS(2)	0.02	0.01	0.93	169	31.9	1.1	33.0
FIDAP024	PS(A^2)	0.0	0.0	4.86	-	9.8	-	-
	PS(A^3)	0.01	0.01	6.93	285	52.7	3.3	55.9
	MS(1)	0.001	0.002	4.47	799	12.2	4.4	16.6
	MS(2)	0.01	0.01	4.87	188	14.4	1.8	16.3
FIDAP028	PS(A^2)	0.0	0.0	4.29	789	19.3	6.7	25.9
	PS(A^2)	0.001	0.001	4.16	835	19.5	7.8	27.3
	MS(1)	0.004	0.004	2.97	255	13.4	2.9	16.2
	MS(1)	0.005	0.005	2.75	330	10.7	3.3	14.0
FIDAPM08	PS(A^2)	0.0	0.0	5.21	-	37.8	-	-
	PS(A^3)	0.0	0.0	12.91	-	377.0	-	-
	MS(2)	0.01	0.01	3.28	729	48.3	3.4	51.7
	MS(3)	0.01	0.01	5.12	291	142.2	2.2	144.5

Table 5: Comparison of ParaSails and MultiSails for solving a few sparse matrices.

3.4 Comparison of ParaSails and MultiSails

In Table 5, we give some comparison results between MultiSails and ParaSails for solving a few sparse matrices.

We see that when constructing a preconditioner, MultiSails usually spends less time than ParaSails to reach the same amount of sparsity ratio. According to our discussions in previous sections, the preconditioner computed from MultiSails 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 whole 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 5 show that with the same amount of memory cost (sparsity ratio), MultiSails usually has better convergence performance than ParaSails does. For solving the FIDAPM08 matrix, ParaSails does not converge when using either A^2 or A^3 as its sparsity pattern. However, MultiSails with a 2 step construction converges with a sparsity ratio 3.28.

np	thre	filt	s-ratio	iter	setup	solve	total
4	0.05	0.05	1.74	288	1953.4	232.8	2186.1
8	0.05	0.05	1.74	288	984.1	121.0	1105.0
16	0.05	0.05	1.74	288	501.9	44.4	546.3
24	0.05	0.05	1.74	288	361.7	29.4	391.1
32	0.05	0.05	1.74	288	281.8	24.7	306.5

Table 6: Scalability of MultiSails for solving a 7-point matrix with $n = 100^3$.

3.5 Implementation scalability

According to Algorithm 2.1, the main computational costs in MultiSails are matrix-matrix product and matrix-vector product operations. As it is well known [11], 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 (5) with the 7-point standard central difference scheme [14]. We let the matrix dimension to be 100^3. The nonzero number is 6940000. The matrix is solved by using a 1 step MultiSails. Tables 6 shows the computational results with different numbers of processors. We can see that the MultiSails preconditioner scales very well in this test case. In particular, we point out that the number of iterations remains to be the same in the test, when the number of processors incenses from 4 to 32. This is different from the simple domain decomposition preconditioners whose iteration properties are usually affected by the number of processors (domains) involved [4].

4 Concluding Remarks

We have proposed a class of multistep successive sparse approximate inverse preconditioning strategies for solving general sparse matrices. A prototype implementation named MultiSails is tested to show favorable convergence properties and computational efficiency of this class of new preconditioning strategies. The performance of the MultiSails preconditioner is indeed as good as what we expected. But some detailed work still needs be done in the future work, in order to build a software package that may be used in realistic large scale scientific computation and computer simulations. E.g., "filt" and "thre" are two important parameters in this class of algorithms (both MultiSails and ParaSails). However, in our current implementation, we did not consider different situations in different construction steps and always kept them the same. It is possible that we may be able to choose these parameters adaptively in a more sophisticated implementation.

Finally, we remark that the concepts of multistep successive preconditioning can be applied to other preconditioning techniques. It is also possible to construct multistep successive ILU preconditioners, or to construct multistep hybrid successive preconditioners using several different preconditioning techniques in each steps.

Acknowledgments: The author would like to thank Professor Jun Zhang for his guidance and encouragement during the course of this research work and the U.S. National Science Foundation for supporting this research project.

References

[1] S. T. Barnard, L. M. Bernardo, and H. D. Simon. An MPI implementation of the SPAI preconditioner on the T3E. *Int. J. High Performance Comput. Appl.*, 13:107–128, 1999.

[2] M. Benzi and M. Tuma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994, 1998.

[3] M. Benzi and M. Tuma. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math.*, 30(2-3):305–340, 1999.

[4] 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(5):477–504, 1994.

[5] E. Chow. Parallel implementation and performance characteristics of least squares sparse approximate inverse preconditioners. Technical Report UCRL-JC-138883, Lawrence Livermore National Laboratory, Livermore, CA, 2000.

[6] E. Chow. ParaSails Users' Guide. Technical Report UCRL-JC-137863, Lawrence Livermore National Laboratory, Livermore, CA, 2000.

[7] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.*, 19(3):995–1023, 1998.

[8] A. C. N. van Duin. Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices. *SIAM J. Matrix Anal. Appl.*, 20:987–1006, 1999.

[9] M. Engelman. FIDAP: Examples Manual, Revision 6.0. Technical report, Fluid Dynamics International, Evanston, IL, 1991.

[10] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18:838–853, 1997.

[11] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing*. Benjamin/Cummings Pub. Co., Redwood City, CA, 1994.

[12] P. Raghavan, K. Teranishi, and E. Ng. Towards scalable preconditioning using incomplete Cholesky factorization. In *Proceedings of the 2001 Conference on Preconditioning Techniques for Large Scale Matrix Problems in Industrial Applications*, pages 63–65, Tahoe City, CA, 2001.

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

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

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

[16] J. Zhang. A sparse approximate inverse technique for parallel preconditioning of general sparse matrices. *Appl. Math. Comput.*, 2001. in press.