www.cadanda.com
Computer-Aided Design AND APPLICATIONS

New Multilevel Parallel Preconditioning Strategies of Sparse Matrix for Speeding up CAD System

Tianping Li[1], Kai Wang[2], E Li[3] and Hui Liu[4]

[1]Shandong Normal University, Sdsdltp@163.com
[2]Lawrence Berkeley National Laboratory, Kaiwang@lbl.gov
[3]Shandong Normal University, Ljlchlekybs@126.com
[4]Shandong University of Finance and Economics, liuh_lh@sdufe.edu.cn

ABSTRACT

In the modeling and transformation research of 3D human body organs and tissues based on medical images, according to 3D scene characteristics of huge amount volume in spatial data processing, the parallel computing of large sparse matrices has important significance for performance calculation of speeding up CAD system. In this paper, to avoid the parallel implementations difficulty encountered in the independent set search strategy of CAD system, a new parallel multilevel MSP (Multistep Successive Preconditioning strategies) pre-conditioner is presented. In each level, we use a diagonal value based strategy to permute the matrix into a 2 by 2 block from. During the preconditioning phase, we do forward and backward preconditioning to improve the performance of the pre-conditioner. In our experiments, MMSP shows better algorithmic CPU time and scalability for solving an actual case of CAD matrix with $n = 100^3$ and $nnz = 6940000$, and the better convergence behavior of MMSP confirms the scaling of performance. At the same time, compared with the linear CAD computing systems, our forward and backward preconditioning strategy can solve the different sparse matrices with smaller sparsity ratios and fewer CPU time.

Keywords: 3D spatial data processing, parallel preconditioning strategies, multilevel pre-conditioner

1. INTRODUCTION

With the rapid development of science, engineering and information technology, how to realize the data processing, simulation and modeling of complex 3D human model by real-time computing have been the critical scientific issues in our CAD system. In above applications there are many partial differential equations (PDE), which will be transformed into linear equations by discretization. At the same time, because the deficiency of single processor computer, research on parallel solution of large scale sparse matrix for solving high performance distributed processors and collaborative computing has become very important. So in the CAD system, now people try to deal with sparse linear systems by taking the advantage of the sparse structure in the coefficient matrices to avoid complexity of memory cost and computational cost.

At present, Krylov subspace methods coupled with a suitable preconditioner [9] is one of the most effective methods for solving large sparse matrix problems, as the quality of the preconditioner influences and in most cases dictates the performance of the Krylov subspace solver heavily, the design of parallel preconditioner is the key problem in the study [6,8,13], which focuses on finding a robust preconditioner with rich parallelism both in construction (setup) phase and application (solution) phase. The incomplete LU (ILU) factorizations [4,5,7,10] and Sparse Approximate Inverse (SAI) [1,2,11,14] are two classes of preconditioning techniques which can be used for solving large sparse linear system. Compared with ILU factorizations, SAI preconditioning techniques have the property of possessing high degree of parallelism and are shown to be efficient for certain type of problems.

Then, a multistep successive preconditioning strategy (MSP) was proposed in [12] to compute robust preconditioners based on the sparse approximate techniques. Instead of computing a costly high accuracy sparse approximate inverse preconditioner in one shot, the MSP strategy computes a sequence

Computer-Aided Design & Applications, 12(5), 2015, 572–579, http://dx.doi.org/10.1080/16864360.2015.1014735

Taylor & Francis
Taylor & Francis Group

of low cost sparse matrices to achieve the effect of a high accuracy preconditioner. The resulting preconditioner has a smaller memory cost and is usually more robust and efficient than the standard SAI preconditioners.

There are several construction techniques of construct sparse approximate inverse preconditioners, that can be roughly divided into three classes: (1) based on Frobenius norm minimization, (2) based on ILU factorization computing, (3) factored sparse approximate inverses. Each of them contains a variety of different constructions and has its own merits and drawbacks.

Considering the solution of sparse linear equations:

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

where A is a large nonsingular sparse matrix of dimension n, x is the vector of unknowns, and b is the right-hand side vector.

Eq. (1) may be solved by a direct solver based on a factorization of the sparse matrix A (Gauss eliminations), which is known to be robust. However the Gauss eliminations lack of inherent parallelism, and their $O(n^2)$ complexity of memory cost and $O(n^3)$ complexity of computational cost make them very expensive for solving large problems. So we can choose one of the following equivalent transformations:

$$MAx = Mb \quad \text{or} \quad AMy = b \quad \text{and} \quad x = My \tag{2}$$

where M is a nonsingular matrix of order n that approximation to A^{-1}, M is called a sparse approximate inverse of A[9]. Then MA or AM can be a good approximation to the identity matrix I. In this way, the solution of large scale sparse matrix problems can be computed by the matrix vector simply, this process can be completed efficiently on parallel computers.

In this paper, we investigate the use of the MSP strategy to construct a multilevel sparse approximate inverse preconditioner. Because the inherent parallelism provided by the MSP algorithm, we do not need to use an independent set search related strategy to form the multilevel structure. Forward and backward preconditioning strategy is used in the preconditioner application (solution) phase to improve the performance of the resulted multilevel preconditioner. In addition, because the MSP algorithm creates a series of preconditioning matrices, we can create different Schur complement matrices conveniently by taking out different number of these matrices. Therefore, a two Schur complement strategy with Schur complement preconditioning is proposed to decrease the memory cost of the multilevel preconditioner.

2. CONSTRUCTION OF MULTILEVEL PRECONDITIONER BASED ON MSP STRATEGY

In this section, we will discuss the idea of using the MSP strategy in the multilevel structure to construct a multilevel sparse approximate inverse preconditioner, for purpose of combining the strength of the sparse approximate inverse and the multilevel preconditioning to construct a hybrid preconditioner with increased robustness and inherent parallelism.

2.1. Multistep Successive Preconditioning in MSP

As mentioned in section 1, the convergence rate of a Krylov subspace solver applied directly to Eq. (1) may be slow due to the potential ill-conditioning of the matrix A, and Eq. (2) will be easier to solve. For constructing a sparse approximate inverse, the basic idea behind the multistep successive preconditioning strategy is to find a multi-matrix form preconditioner and to achieve high accuracy sparse inverse step by step. In each step we compute a sparse approximate inverse inexpensively then hope to build a high accuracy SAI preconditioner collectively, as the algorithm of MSP strategy [12].

The static sparsity pattern strategy is more attractive to implement on distributed memory parallel computers. A particularly useful and effective strategy is to use the sparsited pattern of the matrix A or A^2, A^3, ... to achieve higher accuracy [3]. The MSP strategy needs the SAI computed at each step cheaply, so in Algorithm of MSP, we choose the sparsited pattern of A_i to be the sparsity pattern of the preconditioning matrix M_i. This algorithm leads to a sequence of matrices M_1, M_2, ... , M_l computed inexpensively according to the sparsity pattern of the matrix A_i, and

$$M_l M_{l-1} \cdots M_1 \approx A^{-1} \tag{3}$$

From the numerical results of [13] we can see that in addition to robustness, the MSP algorithm outperforms standard SAI algorithm both in computational and memory costs.

2.2. Multilevel Preconditioner Algorithm

The multilevel preconditioners may have a structure shown in Fig. 1, which depicts a 4-level multilevel preconditioner. Usually, the construction of a multilevel preconditioner consists of two steps. First, at each level the matrix is permuted into a 2 by 2 block form:

$$A_\alpha \sim P_\alpha A_\alpha P_\alpha^T = \begin{pmatrix} D_\alpha & F_\alpha \\ E_\alpha & C_\alpha \end{pmatrix} \tag{4}$$

where α is the level reference. For simplicity, we denote both the permuted and unpermuted matrices by A_α. Then the matrix can be decomposed into a

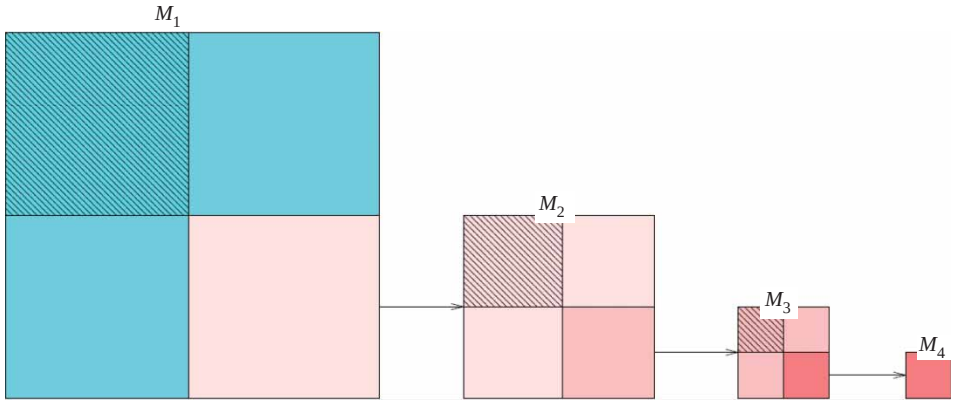

Fig. 1: Structure of 4-level multilevel preconditioning matrices.

2-level structure by a block LU factorization:

$$\begin{pmatrix} D_\alpha & F_\alpha \\ E_\alpha & C_\alpha \end{pmatrix} = \begin{pmatrix} I_\alpha & 0 \\ E_\alpha D_\alpha^{-1} & I_\alpha \end{pmatrix} \begin{pmatrix} D_\alpha & F_\alpha \\ 0 & A_{\alpha+1} \end{pmatrix} \tag{5}$$

where I_α is the generic identity matrix at level. $A_{\alpha+1} = C_\alpha - E_\alpha D_\alpha^{-1} F_\alpha$ is called the Schur complement, which forms the reduced system. The whole process, permuting matrix and performing block LU factorization can be repeated with respect to $A_{\alpha+1}$ recursively, therefore a multilevel structure is produced. The recursion is stopped when the last reduced system $\tilde{A}_\tau$ is small enough so that an efficient solver can be constructed for it.

The preconditioner application process consists of a level by level forward elimination, the coarsest level solution, and a level by level backward substitution. First suppose the right hand side vector b and the solution vector x are partitioned according to the permutation in Eq. (4). At each level we would have:

$$x_\alpha = \begin{pmatrix} x_{\alpha,1} \\ x_{\alpha,2} \end{pmatrix}, \quad b_\alpha = \begin{pmatrix} b_{\alpha,1} \\ b_{\alpha,2} \end{pmatrix} \tag{6}$$

The forward elimination is performed by solving a temporary vector y_α, i.e., for $\alpha = 0, 1, \cdots, \tau - 1$ by solving:

$$\begin{pmatrix} I_\alpha & 0 \\ E_\alpha D_\alpha^{-1} & I_\alpha \end{pmatrix} \begin{pmatrix} y_{\alpha,1} \\ y_{\alpha,2} \end{pmatrix} = \begin{pmatrix} b_{\alpha,1} \\ b_{\alpha,2} \end{pmatrix},$$

$$\text{with} \quad \begin{cases} y_{\alpha,1} = b_{\alpha,1} \\ y_{\alpha,2} = b_{\alpha,2} - E_\alpha D_\alpha^{-1} y_{\alpha,1} \end{cases} \tag{7}$$

The last reduced system may be solved to a certain accuracy by a Krylov subspace iteration to get an approximate solution x_τ. After that, a backward substitution is performed to obtain the solution by solving:

$$\begin{pmatrix} D_\alpha & F_\alpha \\ 0 & A_{\alpha+1} \end{pmatrix} \begin{pmatrix} x_{\alpha,1} \\ x_{\alpha,2} \end{pmatrix} = \begin{pmatrix} y_{\alpha,1} \\ y_{\alpha,2} \end{pmatrix},$$

$$\text{with} \quad \begin{cases} y_{\alpha,1} = b_{\alpha,1} \\ y_{\alpha,2} = b_{\alpha,2} - E_\alpha D_\alpha^{-1} y_{\alpha,1} \end{cases} \tag{8}$$

where $x_{\alpha,2}$ is actually the coarser level solution.

In our algorithm matrix M_α that we compute is a good approximation to D_α^{-1}, so formula (5) will equivalent to:

$$\begin{pmatrix} D_\alpha & F_\alpha \\ E_\alpha & C_\alpha \end{pmatrix} = \begin{pmatrix} I_\alpha & 0 \\ E_\alpha M_\alpha & I_\alpha \end{pmatrix} \begin{pmatrix} D_\alpha & F_\alpha \\ 0 & A_{\alpha+1} \end{pmatrix} \tag{9}$$

The approximate Schur complement matrix is computed as $A_{\alpha+1} = C_\alpha - E_\alpha M_\alpha F_\alpha$. Continue doing this for $A_{\alpha+1}$ at next level, a multilevel preconditioner based on the SAI technique will be produced. Correspondingly, the forward and backward substitutions in the preconditioner application phase change to:

$$\begin{cases} y_{\alpha,1} = b_{\alpha,1} \\ y_{\alpha,2} = b_{\alpha,2} - E_\alpha M_\alpha y_{\alpha,1} \end{cases} \text{and} \quad \begin{cases} x_{\alpha,2} = A_{\alpha+1}^{-1} y_{\alpha,2} \\ x_{\alpha,1} = M_\alpha (y_{\alpha,1} - F_\alpha x_{\alpha,2}) \end{cases} \tag{10}$$

Thus we prefer an accurate sparse approximate inverse of D_α during the computation of the multilevel preconditioner, besides it should be computed efficiently with respect to the costs of memory and computation. Generally speaking this can be accomplished in at least two ways. One is to find a D_α during the permutation with some special structures or properties so that the sparse inverse for D_α can be computed cheaply and accurately. Another way to find a good approximation for D_α^{-1} cheaply is to improve the performance of the approximation algorithm. Since the MSP strategy is an improvement for the standard SAI algorithm and outperforms the standard SAI algorithm both in robustness and efficiency, we use the MSP algorithm to compute our multilevel sparse approximate inverse preconditioner. At this time the MSP algorithm will create a series of matrices:

$$M_{\alpha l} M_{\alpha l-1} \cdots M_{\alpha 1} \approx D_\alpha^{-1} \tag{11}$$

where l is the number of steps. The Schur complement matrix is:

$$C_\alpha - E_\alpha M_{\alpha l} M_{\alpha l-1} \cdots M_{\alpha 1} F_\alpha \tag{12}$$

Obviously, the setup and solution phases will change to the corresponding forms. Suppose the MSP algorithm creates a series of matrices as in Eq. (11). Obviously, if we use Eq. (12) as the Schur complement matrix directly, $A_{\alpha+1}$ may be a dense matrix according to our previous discussion. Then we construct the second Schur complement by only using the first several steps of Eq. (11), say only $M_{\alpha 1}$, the second Schur complement matrix is in the following form:

$$C_\alpha - E_\alpha M_{\alpha 1} F_\alpha \tag{13}$$

Because $M_{\alpha 1}$ is usually very sparse according to [13], the second Schur complement (13) may be sparse (at least sparser than the first Schur complement (12)) and can be computed cheaply. Then we use the second Schur complement as the reduced system to construct lower level preconditioner instead of the first Schur complement. In the preconditioning phase, we may use the first Schur complement as the original matrix, the lower level preconditioner constructed by the second Schur complement as the preconditioner in a preconditioned GMRES iteration to improve the solution accuracy, which is called Schur complement preconditioning [15]. Note that we do not compute the exact form of the first Schur complement matrix, and it is stored in a multi-matrix form. During the computations of Schur complement preconditioning, we only do a series of matrix vector products and then use one vector to minus another. We can see if each of these matrices is sparse, the whole memory cost will not be too large. The whole algorithm of preconditioning is as follows:

Algorithm 1 Preconditioner application algorithm

```
1: Function precondition-main (nlevel)
2: Begin
3:    If (nlevel = thelastlevel)
4:       Solve the last level
5:    Else
6:       Forward substitution
7:       Precondition-main (nlevel + 1)
8:       GMRES iteration with Schur complement
9:             preconditioning
10:      Backward substitution
11:   End
12: End
```

2.3. Algorithm Implementation

The success of general sparse linear system solvers depends on sophisticated implementations as heavily as on the innovative underlying ideas, so the implementation issues need to be addressed to build efficient software for distributed memory parallel computers. To solve a sparse linear system on a parallel computer, the coefficient matrix is first partitioned by a graph partitioner and is distributed to different processors uniformly. Suppose that initially the matrix is distributed row by row in each processor as shown in Fig. 2, where the coefficient matrix A is distributed in 4 processors. The shaded parts A_m, $m = 1, \ldots, 4$, in the figure form the block diagonal of A. From the view of graph partition, A_m can be called the local matrix of the processor P_m. And the entries of A_m are the local elements of the processor P_m.

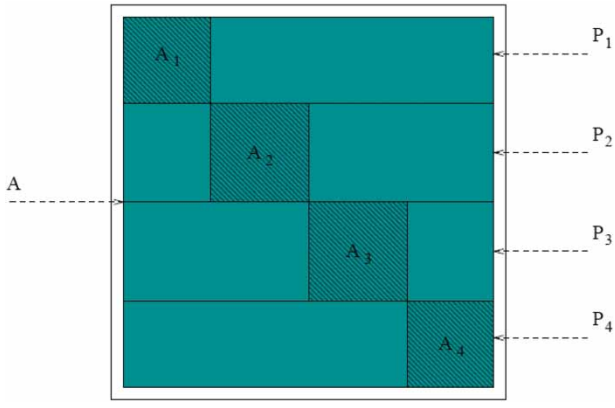

Fig. 2: Distribution of the matrix A in 4 processors.

3. EXPERIMENTAL RESULTS

We implement our parallel multilevel MSP (MMSP) preconditioner, at each level, we use a diagonal value based strategy to permute the matrix into a 2 by 2 block from. A static sparsity pattern based MSP strategy is implemented to compute the approximation of the inverse of D_α. During the preconditioning phase, we do forward and backward preconditioning to improve the performance of the preconditioner. The last level reduced system is solved by a GMRES iteration preconditioned by MSP strategy. In this section, we conduct a few numerical experiments to show the characteristics of MMSP, and we also compare the performance of the MMSP preconditioner with the MSP preconditioner to show the improved robustness and efficiency due to the introduction of multilevel structure. The computations are carried out on a 64 processor (2.4 GHz) subcomplex of an HP Z400 graphics workstation. Unless otherwise indicated explicitly, 4 processors are used in our numerical experiments. For all preconditioning iterations, which include the outer (main) preconditioning iterations, forward and backward preconditioning iterations, Schur complement preconditioning iterations, and the coarsest level solver, we use a flexible

variant of restarted GMRES (FGMRES) in a parallel version.

3.1. Convection-diffusion Problem Test

A 3D convection-diffusion problem (defined on a unit cube):

$$u_{xx} + u_{yy} + u_{zz} + 1000(p(x,y,z)u_x + q(x,y,z)u_y + r(x,y,z)u_z) = 0 \quad (14)$$

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

$$p(x,y,z) = x(x-1)(1-2y)(1-2z)$$
$$q(x,y,z) = y(y-1)(1-2z)(1-2x)$$
$$r(x,y,z) = z(z-1)(1-2x)(1-2y)$$

The Reynolds number for this problem is 1000. Eq. (14) is discretized by using the standard 7-point central difference scheme and the 19-point fourth order compact difference scheme. The resulting matrices are referred to as the 7-point and 19-point matrices respectively.

We also use MMSP to solve the sparse matrices listed in Tab. 1. The BARTHT1A matrix is from 2D, high Reynolds numbers airfoil problem with turbulence modeling. It belongs to the SPARSKIT collection. The FIDAP matrices were extracted from the test problems provided in the FIDAP package. They arise from coupled finite element discretization of Navier-Stokes equations modeling incompressible fluid flows. The UTM matrices are real unsymmetric matrices arising from nuclear fusion plasma simulations in a tokamak reactor. They are from TOKAMAK set in SPARSKIT collection. The UTM matrices and the other matrices are available from the well-known Harwell-Boeing sparse matrix collection.

3.2. Properties of MMSP

We will present a few numerical results to demonstrate some CPU cost properties of MMSP. The main computational costs in MMSP are matrix-matrix product and matrix-vector product operations. As it is well known [13], these operations can be performed in parallel efficiently on most distributed memory parallel architectures.

We use the 3D convection-diffusion problem to test the implementation efficiency of our MMSP preconditioner. The results in Fig. 3 are from solving an actual case of CAD matrix with $n = 100^3$ and $nnz = 6940000$ using different number of processors. Because the memory limitation of our parallel computers, we can only test the problems starting from 4 processors. To be convenient, we set the speedup in 4 processors case to be 4 (Fig. 3 a). In particular, we point out that it is not the same as the MSP algorithm, in which the number of the MSP iterations is not influenced by the number of processors when the whole problem size fixed. The number of iterations for MMSP does not remain constant. That is because the permutation of the matrix at each level depends on the ordering of the unknowns. Different number of processors have different ordering of the unknowns even for the same problem. Fortunately,

Matrices	n	nnz	Description
BARTHT1A	14075	481125	14075 481125 Navier Stokes flow at high Reynolds number
FIDAP012	3973	80151	3973 80151 Flow in lid-driven wedge
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
FIDAP040	7740	456226	3D die-swell (square die $Re = 1$, $Ca = \infty$)
FIDAPM03	2532	50380	Flow past a cylinder in free stream ($Re = 40$)
FIDAPM08	3876	103076	Developing flow, vertical channel (angle = 0, $Ra = 1000$)
FIDAPM09	4683	95053	Jet impingment cooling
FIDAPM11	22294	623554	3D steady flow, heat exchanger
FIDAPM13	3549	71975	Axisymmetric poppet value
FIDAPM33	2353	23765	Radiation heat transfer in a square cavity
UTM1700A	1700	21313	Nuclear fusion plasma simulations
UTM1700B	1700	21509	Nuclear fusion plasma simulations
UTM3060	3060	42211	Nuclear fusion plasma simulations
WIGTO966	3864	238253	Euler equation model

Tab. 1: Information about some sparse matrices used in the experiments (n is the order of a matrix, nnz is the number of nonzero entries).

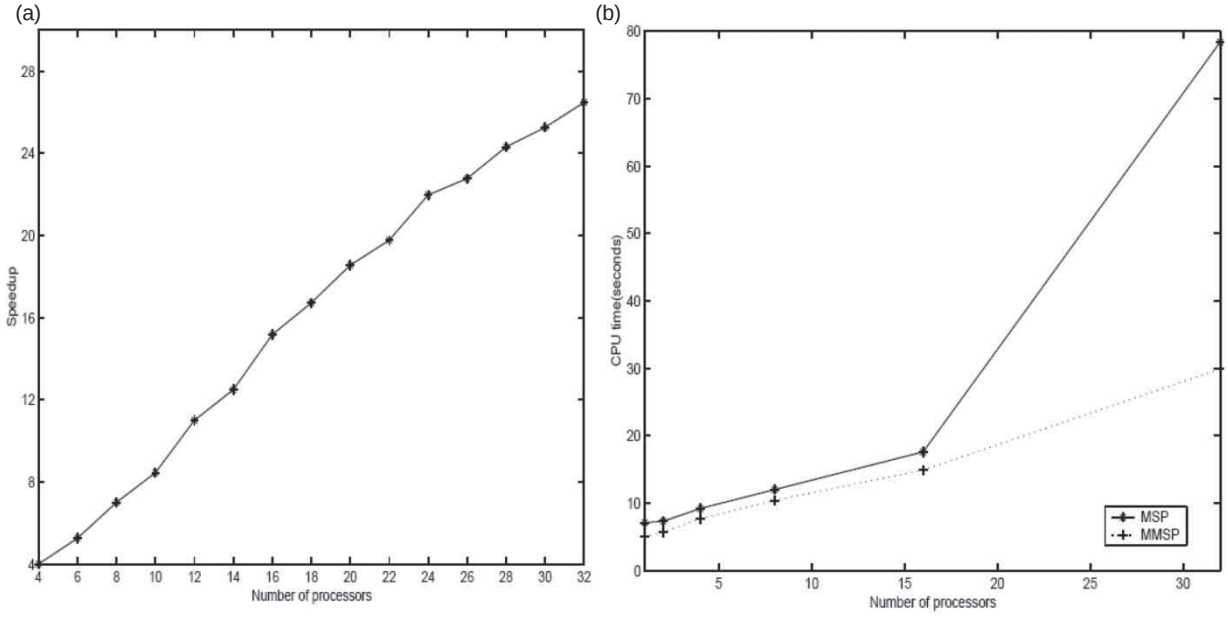

Fig. 3: Experiments of MMSP when solving a 7 point matrix with $n = 100^3$, $nnz = 6940000$, (a): the speedup of MMSP as a function of the number of processors. (b): the total CPU time versus the number of processors of MSP and MMSP.

the number of iterations does not degrade too much with the increase of the number of processors.

In Fig. 3b, the algorithmic scalability of MMSP is tested by solving a series of 19-point matrices. We try to keep the number of unknowns in each processor approximately 253. Therefore when we change the number of processors, the whole problem size increases at the same time. To be comparable, we can see that MMSP shows better algorithmic scalability and efficiency than the MSP algorithm.

We already know that a larger number of steps in MSP will create a preconditioner which converges fast [13]. For the FIDAPM09 matrix, MMSP cannot make it converge using only 1 and 2 steps, in 3 steps case, the preconditioned system converges in 248 iterations. For the FIDAPM33 matrix, MMSP can solve the matrix using 2 and 3 steps, and fails in the 1 step case. Just as we expected, using a larger number of steps leads to a better convergence results.

Then Fig. 4 depicts the relationships of the convergence behavior and the CPU time with respect to different number of forward and backward preconditioning iterations for solving the UTM1700B matrix. Here, the forward and backward preconditioning iteration uses FGMRES(50) algorithm to do certain number of iterations to reduce the 2-norm of the relative residual. The number of iterations is an input parameter. From Fig. 4 we can see that when we increase the number of forward and backward preconditioning iterations from 0 to 5, the number of iterations for solving the preconditioned systems decreases from more than 2000 to around 200 repaidly. And the total CPU time decreases from more than 20 seconds to around 3 seconds at the same time. However, further increasing the number of forward and backward preconditioning iterations from 5 to 60 does not bring very big difference for the convergence of the system, the number of iterations only decreases to around 100, which cannot compensate for the CPU time which increases from 3 to 11 seconds. Therefore, we conclude that the forward and backward preconditioning iteration can improve the convergence of the preconditioning system.

But a large number of forward and backward preconditioning iterations will bring a high CPU time cost. Notice that in the experiments in Fig. 4, the best choice is to perform 5 forward and backward iterations. In the following test, we always use 5 forward and backward preconditioning iterations. We note that how to choose this parameter should be problem dependent, and the choice of 5 may not be the best for all problems.

3.3. Comparison of MSP and MMSP

Here we do some comparison between the MSP preconditioner and the MMSP preconditioner. Tab. 2 gives a few experiment results for using MSP and MMSP to solve a few different matrices. For MSP, we adjust the parameter ə and number of steps and try to give the best performance results for solving these matrices. For all results of MMSP we fix the reduce ratio to be 0.67, ə to be 0.05, the number of steps at each level to be 2 and the number of forward and backward iterations to be 5. The number in the parentheses of MSP is the number of steps, and the

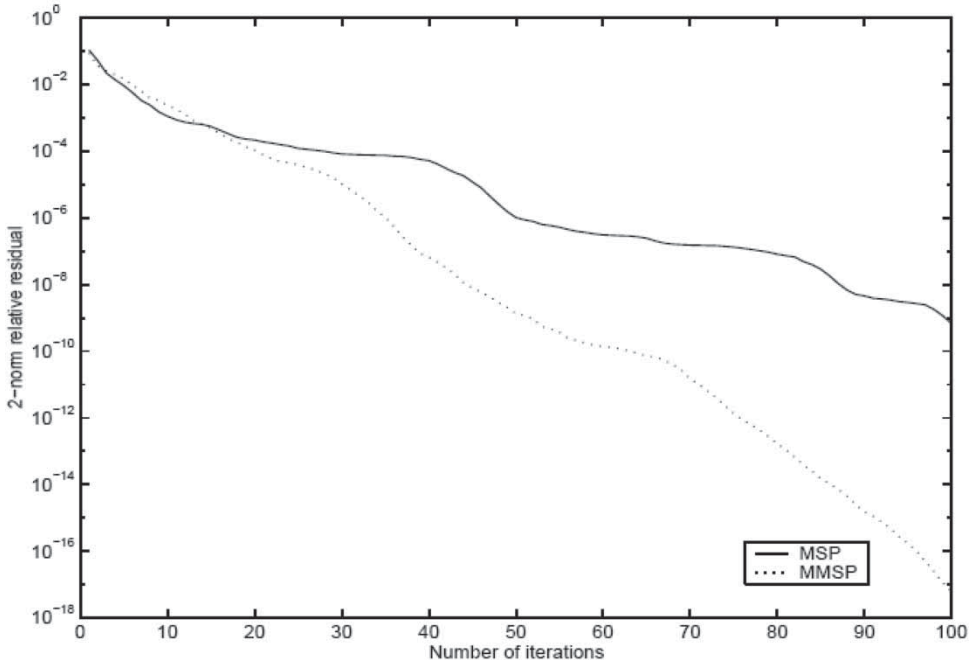

Fig. 4: Convergence behavior of MMSP and MSP for solving the FIDAP028 matrix. (MMSP *step* = 2, = 0.05, *density* = 2.83, *level* = 7) (MSP *step* = 3, ɘ = 0.005, *density* = 5.34).

Matrices	preconditioner	ɘ	density	iter	setup	solve	total
FIDAP024	MSP(3)	0.01	4.87	188	14.4	1.8	16.2
	MMSP(7)	0.05	3.05	39	0.8	0.6	1.4
FIDAPM08	MSP(3)	0.01	3.28	729	48.3	3.4	51.7
	MMSP(8)	0.05	3.02	192	1.1	4.5	5.6
FIDAP012	MSP(-)	-	-	-	-	-	-
	MMSP(8)	0.05	3.38	57	1.1	1.2	2.3
FIDAP040	MSP(-)	-	-	-	-	-	-
	MMSP(8)	0.05	3.46	39	4.3	4.0	8.3
FIDAPM03	MSP(-)	-	-	-	-	-	-
	MMSP(7)	0.05	3.35	62	0.8	1.0	1.8
FIDAPM11	MSP(-)	-	-	-	-	-	-
	MMSP(9)	0.05	6.81	200	16.3	85.1	101.4
FIDAPM13	MSP(-)	-	-	-	-	-	-
	MMSP(8)	0.05	3.58	86	1.1	1.7	2.8
UTM1700A	MSP(-)	-	-	-	-	-	-
	MMSP(7)	0.05	3.45	145	0.7	1.9	2.6
UTM3060	MSP(-)	-	-	-	-	-	-
	MMSP(8)	0.05	4.02	474	1.0	7.9	8.9

Tab. 2: Comparison of MSP and MMSP for solving a few matrices.

number in the parentheses of MMSP is the number of constructed levels.

In Tab. 2 nine matrices are tested by MSP and MMSP respectively, two of them can be solved by the MSP preconditioner, however the MMSP preconditioner can solve them with smaller sparsity ratios and only 10 percent of the CPU time. MSP fails solving remaining seven matrices, which can be solved by MMSP very efficiently. Furthermore, MMSP shows better algorithmic scalability than the MSP algorithm.

4. CONCLUSION

We have developed a multilevel sparse approximate inverse pre-conditioner based on the MSP strategies for solving general sparse matrices in 3D human body organs and tissues research. A prototype implementation is tested to show the robustness and computational efficiency of this class of multilevel pre-conditioners.

From the numerical results presented, we can see that forward and backward preconditioning is an

important strategy for the convergence performance of the multilevel MSP pre-conditioner. A number of forward and backward preconditioning iterations help the convergence of the multilevel MSP pre-conditioner and a carefully chosen number of iterations will make the multilevel MSP pre-conditioner converge fast. Compared with the linear CAD computing systems, our strategy can solve the different sparse matrices with smaller sparsity ratios and fewer CPU time.

In addition, the number of levels also influences the convergence and memory cost of the multilevel MSP pre-conditioner. A large number of levels will produce a good pre-conditioner with a high memory cost. A small number of levels will create a cheap pre-conditioner with low memory cost. The same thing happens when refer to the number of MSP steps used at each level of the multilevel MSP pre-conditioner. Fortunately, because the MSP algorithm creates a series of preconditioning matrices, we can use a two Schur complement strategy to decrease the memory cost, even sometimes the computational cost for the preconditioning phase may increase.

ACKNOWLEDGEMENTS

This work is supposed by National Natural Science Foundation (No. 61173174, 61272245), the Postdoctoral Granted Financial Support of Shandong Province (No. BS2011DX025). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] Benzi, M.; Tuma, M.: A sparse approximate inverse preconditioner for nonsymmetric linear systems, SIAM Journal on Scientific Computing, 19(3), 1998, 968–994. http://dx.doi.org/10.1137/S1064827595294691

[2] Boollhofer, M.; Mehrmann, V.: Algebraic multilevel methods and sparse approximate inverses, SIAM J. Matrix Anal. Appl., 24(1), 2002, 191–218. http://dx.doi.org/10.1137/S0895479899364441

[3] Chow, E.: A priori sparsity patterns for parallel sparse approximate inverse preconditioners, SIAM J. Sci. Comput., 21(5), 2000, 1804–1822. http://dx.doi.org/10.1137/S106482759833913X

[4] Huang, Z.-H.; Shi, P.-L.: Notes on Convergence of an Algebraic Multi-grid Method, Applied Mathematics Letters, 20(3), 2007, 335–340. http://dx.doi.org/10.1016/j.aml.2006.05.002

[5] Huang, Z.-H.; Wang, D.-S.: Chin Parallel Numerical Simulations for Quantized Vortices in Bose-Einstein Condensates, Chinese Physics, 16(1), 2007, 32–37. http://dx.doi.org/10.1088/1009-1963/16/1/005

[6] Liu, Y.-Q.; Yin, K.-X.; Wu, E.-H.: Fast GMRES-GPU Solver for Large Scale Sparse Linear Systems, Journal of Computer-Aided Design & Computer Graphics, 23(4), 2011, 553–560.

[7] Nakajima, K.; Okuda, H.: Parallel iterative solvers with localized ILU preconditioning for unstructured grids on workstation clusters, International Journal of Computational Fluid Dynamics, 12, 1999, 315–322. http://dx.doi.org/10.1080/10618569908940835

[8] Quan, Z.; Xiang, S.-H.: A GMRES based polynomial preconditioning algorithm, Mathematic Numerica Sinica, 28(4), 2006, 365–376.

[9] Saad, Y.: Iterative Methods for Sparse Linter Systems, PWS Publishing, New York, 1996.

[10] Saad, Y.; Zhang, J.: Enhanced multi-level block ILU preconditioning strategies for general sparse linear systems. Journal of Computational and Applied Mathematics, 130, 2001, 99–118. http://dx.doi.org/10.1016/S0377-0427(99)00388-X

[11] Wang, K.; Wang, R.: Nonsingularity Study of SAI Preconditioners for M-matrices, The 2009 International Conference on Scientific Computing, 2009, 89–95.

[12] Wang, K.; Zhang, J.: MSP: a class of parallel multistep successive sparse approximate inverse preconditioning strategies, SIAM J. Sci. Comput., 24(4), 2003, 1141–1156. http://dx.doi.org/10.1137/S1064827502400832

[13] Zhang, J.: Preconditioned Krylov subspace methods for solving non-symmetric matrices from CFD applications, Computer Methods in Applied Mechanics Engineering, 189(3), 2000, 825–840. http://dx.doi.org/10.1016/S0045-7825(99)00345-X

[14] Zhang, J.: A sparse approximate inverse technique for parallel preconditioning of general sparse matrices, Applied Mathematics and Computation, 130(1), 2002, 63–85. http://dx.doi.org/10.1016/S0096-3003(01)00069-8

[15] Zhang, J.: On preconditioning Schur complement and Schur complement preconditioning, Electron. Trans. Numer. Anal., 10, 2000, 115–130.