Parallel Preconditioning in the Analysis of Anisotropic Diffusion Simulation with the Human Brain Diffusion Tensor MRI Data

Ning Kang [*], Jun Zhang [†]
Laboratory for High Performance Scientific Computing and Computer Simulation,
Department of Computer Science,
University of Kentucky,
Lexington, KY 40506-0046, USA
and
Eric S. Carlson [‡]
Department of Chemical Engineering,
University of Alabama,
P. O. Box 870203,
Tuscaloosa, AL 35487-0203, USA

Abstract

We conduct simulations for the 3D unsteady state anisotropic diffusion process in the human brain by discretizing the governing diffusion equation on Cartesian grid and adopting a high performance differential-algebraic equation (DAE) solver, the parallel version of implicit differential-algebraic (IDA) solver, to tackle the resulting large scale system of DAEs. Parallel preconditioning techniques including sparse approximate inverse and banded-block-diagonal preconditioners are used with the GMRES method to accelerate the convergence rate of the iterative solution. We then investigate and compare the efficiency and effectiveness of the two parallel preconditioners. The computational results of the diffusion simulations on a parallel supercomputer show that the sparse approximate inverse preconditioning strategy, which is robust and efficient with good scalability, gives a much better overall performance than the banded-block-diagonal preconditioner.

[*] The research work of this author was supported by the U.S. Department of Energy Office of Science under grant DE-FG02-02ER45961. E-mail: nkang2@csr.uky.edu.

[†] The research work of this author was supported in part by the U.S. National Science Foundation under grants CCR-9988165, CCR-0092532, and ACR-0202934, in part by the U.S. Department of Energy Office of Science under grant DE-FG02-02ER45961, in part by the Kentucky Science and Engineering Foundation under grant KSEF-02-264-RED-002, in part by the Japan Research Organization for Information Science and Technology (RIST), and in part by the University of Kentucky Research Committee. E-mail: jzhang@cs.uky.edu, URL: http://www.cs.uky.edu/∼jzhang.

[‡] The research work of this author was supported by the U.S. Department of Energy Office of Science under grant DE-FG02-02ER45961. E-mail: ecarlson@bama.ua.edu.

1. Introduction

The solution of the general unsteady state anisotropic diffusion equation can be used in the development of improved approaches for the analysis of diffusion tensor magnetic resonance imaging (DT-MRI). DT-MRI is an extension of conventional MRI with the added capability of measuring the random motion of water molecules in all three dimensions, usually referred to as diffusion or "Brownian motion" [3]. DT-MRI renders the information about how water diffuses in tissues containing a large number of fibers, like brain white matter, into intricate three-dimensional representations of the tissues. Thus, it can be exploited to visualize and extract information about the brain white matter and nerve fibers by using fiber traces, which has raised promises for a better understanding of the fiber tract anatomy of the human brain. In combination with functional MRI, it might also be used to study the connectivity between different parts of the brain, which is useful for functional and morphological research on the brain [2].

A number of fiber tracking algorithms have been developed since the appearance of DT-MRI. In [3] a variety of these algorithms are described and reviewed. As the measured quantity in DT-MRI is water diffusion, an intuitive way to understand the diffusion data is to spread a virtual concentration peak of water [8], or to specify a starting point for tractography where a seed is diffused [4]. This

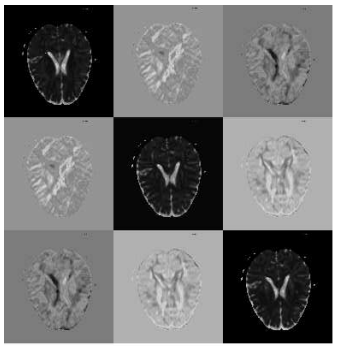

Figure 1. An axial slice of a diffusion tensor volume. It shows the diffusion tensor components, corresponding to the diffusion tensor matrix D.

approach makes use of the full information contained in the diffusion tensor and it is not dependent upon a point to point eigenvalue/eigenvector computation along a trajectory, thus in that sense hopefully is more robust. It is also intuitively related to underlying physio-chemical process [16]. The diffusion process and related transport mechanisms in the brain are discussed in detail in [12].

Anisotropic systems exhibit a preferential flow direction while isotropic systems have no preference. According to Fick's first law, the flux, J, has magnitude proportional to the concentration gradient, ∇C, and is directed opposite to ∇C, i.e., $J = -d\nabla C$, where the proportionality constant d is the diffusion coefficient. In the presence of anisotropy, the flow field does not follow the concentration gradient directly, for the material properties also affect diffusion. Therefore, the diffusion tensor, D, is introduced to fully describe the molecular mobility along each direction and the correlation between these directions. Thus, the flux is given as $J = -D\nabla C$ and the diffusion tensor is

$$D = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{yx} & D_{yy} & D_{yz} \\ D_{zx} & D_{zy} & D_{zz} \end{pmatrix},$$

where the subscripts xx, xy, xz, etc., denote the values of the individual coefficients in the matrix that can be seen as the influence from directions in the input (being the concentration) on the various directions in the output (being the flux). Unlike the standard MRI data which has one value at each voxel, the DT-MRI data has 9 values at each voxel. Figure 1 shows an axial slice of a diffusion tensor volume data from a human brain. For the brain system on which we are focusing, the tensor is symmetric.

Essentially, we are seeking to solve an unsteady state diffusion equation in an anisotropic medium based on the measured diffusion tensor D. The anisotropic diffusion process, due to conservation of mass, is governed by

$$\frac{\partial C}{\partial t} = \nabla \cdot (D\nabla C), \tag{1}$$

where t is the independent time variable. This equation says that over the time, the rate of change in concentration is proportional to the divergence of the flux.

Equation (1) could be very difficult to solve under the circumstance of the human brain for a few reasons. First, since the brain structure is heterogeneous where anisotropy requires full tensor representation, the second order cross derivatives must be calculated. Second, the diffusion tensor changes drastically between adjacent small regions in the brain tissues. Thus, fine gridding must be used to avoid a crude approximation to the true geometry of interesting structures and this leads to large systems of equations. The third challenge we have to face is that time plays a crucial role in the real environment, such as clinical diagnosis, surgical planning, and neurosurgery [17]. In the sense to be practical, the solution must meet the real-time constraints and achieves good reliability and robustness as well.

Simulations of anisotropic diffusion in a human brain have been studied in [11], in which several standard preconditioning techniques based on incomplete LU (Lower-Upper) factorizations of the coefficient matrix are compared in a sequential environment. However, the ILU schemes are not suitable for parallel environments. In order to perform diffusion simulations over the whole brain with sufficient accuracy and acceptable computational time and memory cost, a parallel implementation of the solution procedure is considered in this paper. The coefficient matrix is distributed to different processors with the scheme of row-wise block striping. We then exploit general purpose modules from the ACTS Toolkit [1], which includes high performance differential-algebraic-equation (DAE) system solvers, as the primary integration tools. For the large scale sparse linear system arising from each integration step, the Krylov subspace method, preconditioned GMRES, is used and a number of highly efficient and robust parallel preconditioners are applied as well to achieve speedy solutions with good accuracy.

The remainder of the paper is organized as follows. Section 2 concisely describes the DAE integration solver as well as its parallel implementation scheme. The parallel preconditioning techniques are discussed in Section 3. In Section 4, we conduct a number of numerical experiments and compare the performance of these preconditioners. The final concluding remarks are given in Section 5.

2. Differential-algebraic-equation solver

Since the tensor data set used in our current simulation is measured and processed on a Cartesian mesh, we discretize

the 3D diffusion Equation (1) on Cartesian grid using finite difference approximation. The central difference in space and backward differentiation formula in time are applied to approximate the spatial derivative and time derivative terms in Equation (1), respectively.

On the boundaries of the heterogeneous system, we assume that it is insulated, i.e., $(D\nabla C)\cdot \boldsymbol{n} = 0$, which corresponds to the Neumann condition. This condition means that the normal part of the gradient of the concentration on the boundary is zero. No material diffuses outside of the boundary. A 3D Gaussian function is selected to be the initial distribution profile of the water concentration in the brain.

The discretization of the Equation (1) and its boundary conditions on the Cartesian grid generates a large scale system of semi-explicit differential-algebraic equations (DAEs) with the form

$$F(t, f, f') = 0, \tag{2}$$

where f and f' are N-dimensional vectors corresponding to the discretized values of C and $\partial C/\partial t$. In this paper, we consider using a high performance DAE solver included in the ACTS Toolkit, IDA, which stands for Implicit Differential-Algebraic solver. we first briefly overview the algorithms used in IDA for solving DAEs, then take a look at how it is implemented in parallel. See [5, 6, 10, 14] for more details.

The IDA solver uses the backward differentiation formula (BDF) method to approximate the time derivative in (2), implemented in a variable order, variable step form. The application of BDF to the DAE system (2) leads to a nonlinear algebraic system to be solved at each time step, which is

$$G(f_n) \equiv F\Big(t_n, f_n, h_n^{-1}\sum_{i=0}^{k}\alpha_{n,i} f_{n-i}\Big) = 0, \tag{3}$$

where f_n is the calculated approximation to $f(t_n)$ and the step size is $h_n = t_n - t_{n-1}$. $\alpha_{n,i}, i = 0, 1, \cdots, k$, are the coefficients of the BDF method, which are uniquely determined by the order of k and the step size at the previous times. IDA solves the nonlinear system (3) by a modified version of Newton iteration method. This results in a linear system for each Newton correction, given by

$$f_n^{(m+1)} = f_n^{(m)} - cJ^{-1}G\big[f_n^{(m)}\big], \tag{4}$$

where $f_n^{(m)}$ is the mth approximation to f_n, c is a constant chosen to speed up the rate of convergence of the iteration, and J is some approximation to the system Jacobian

$$J = \frac{\partial G}{\partial f} = \frac{\partial F}{\partial f} + \alpha\frac{\partial F}{\partial f'}, \tag{5}$$

where $\alpha = \alpha_{n,0}/h_n$, which changes whenever the step size or the BDF method order changes. During the course of integrating the system, the IDA solver imposes tolerances on the computed local truncation errors at the nth time step by using weighted root-mean-square (wrms) norm.

For the solution of large scale linear system (4), only scaled preconditioned GMRES method, denoted as SPGMR, is available in the parallel version of the IDA solver. When solving the linear system (4), a preconditioner matrix P must be supplied and need be constructed to approximate J, which leads to an inexpensive linear system solution, and then factored and used for as many time steps as possible.

The parallel version of IDA uses a vector module in its package to achieve parallelism and the MPI (Message Passing Interface) library for all interprocessor communication. The vector module contains a set of mathematical operations on N-vectors (N-dimensional vectors), including vector linear combinations, vector norms, scalar products, and so forth. By separating these operations from the rest of the code, all operations in IDA with significant potential for parallel computation are isolated. Because the parallel form of IDA is intended for an SPMD programming model with distributed memory, all N-vectors are identically distributed across processors such that each processor is solving a contiguous subset of the DAE system.

3. Parallel preconditioning techniques

We can rewrite the linear system (4) as

$$Ax = b, \tag{6}$$

where, A is the system Jacobian matrix in (5), $x = f_n^{(m+1)} - f_n^{(m)}$, $b = -c\,G\big[f_n^{(m)}\big]$. The sparse linear system (6) needs to be solved at each Newton iteration by way of the scaled preconditioned GMRES method. As mentioned before, preconditioning of the linear iteration is essential and beneficial for both robustness and efficiency.

The first parallel preconditioning technique under our investigation is a class of sparse approximate inverse (SAI) preconditioners. The SAI preconditioner, as its name implies, is an approximation to A^{-1}, the inverse of matrix A. Both its construction and its application in the iterative solution, which requires nothing but matrix-by-vector products, allow a large degree of parallelism. Another class of preconditioners that we are interested in is the block-diagonal preconditioning, which is also suitable for the parallel architecture. In this paper, close attention is paid to the banded-block-diagonal (BBD) preconditioners.

Sparse approximate inverse preconditioners. The SAI preconditioning technique discussed here is based on the idea of the least squares (Frobenius norm) minimization [9],

using *a priori* sparsity patterns [7]. We seek to approximate the inverse of a matrix A (usually sparse) by a sparse matrix P, such that $AP \approx I$ in some sense, where I is the identity matrix. In the case of the right SAI preconditioning being used here, we strive to minimize $\|AP - I\|$ in the Frobenius norm, i.e., $\|AP - I\|_F$, to make $AP \approx I$, in order to achieve fast convergence with a reasonable cost. Since the Frobenius norm of a square matrix is defined by $\|A\|_F = \sqrt{\sum_{i,j=1}^{N} |a_{ij}|^2}$, we have

$$\|AP - I\|_F^2 = \sum_{j=1}^{N} \|(AP - I)e_j\|_2^2 = \sum_{j=1}^{N} \|Ap_j - e_j\|_2^2, \tag{7}$$

where p_j and e_j are the jth column of the matrix P and that of the identity matrix I, respectively. The solution of the minimization function (7) can be decoupled into N independent least squares problems

$$\min_{p_j} \|Ap_j - e_j\|_2, \quad j = 1, 2, ..., N. \tag{8}$$

It is apparent that inherent parallelism lies in the solution of (8) in that each column p_j of P can be computed independently of one another. Thus the approximate inverse P of A can be constructed by solving (8) in parallel.

Under the consideration of the algorithm complexity and the computational and memory cost, it is desirable that P is a sparse matrix and has a good sparsity pattern, i.e., a good distribution of nonzeros in the matrix. There are quite a few heuristic strategies proposed to specify a good sparsity pattern for the matrix P, in both *a priori* way and adaptive way [7, 9]. Here, we are interested in the method of building a prescribed sparsity pattern so that the SAI can be yielded immediately by minimizing (7). For PDE problems, as in our case, the sparsified patterns of the original matrix A can be employed *a priori* as effective SAI patterns. Here "sparsified" means that certain small entries of A are dropped before its sparsity pattern is extracted. Once we get a sparsity pattern for the sparse approximate inverse P, it can be computed by solving the minimization problem (8). Detailed discussions about *a priori* sparsity patterns and approaches of deriving the SAI matrix can be found in [7, 9].

Block-diagonal preconditioners. In general, this class of preconditioners is based on the block Jacobi method where a preconditioner can be derived by a partitioning of the variables. The basic idea is to isolate the preconditioning so that it is local to each processor. In fact, on parallel computers it is natural to let the partitioning coincide with the division of the variables over the processors. In the following paragraphs, we briefly describe the construction of the banded-block-diagonal (BBD) preconditioners [10], which is closely related to the DAE system (2).

First, the spatial domain of the computational PDE problem is subdivided into M non-overlapping subdomains. Each of these subdomains is then assigned to one of the M processors to be used to solve the PDE system (in our case, the DAE system) in parallel. Corresponding to the domain decomposition and distribution of the system over the processors, there is a decomposition of the solution vectors f and f' of the DAE system (2) into M disjoint blocks f_m and f'_m. Also, the function $F(t, f, f')$ is decomposed into blocks F_m correspondingly, and the block F_m depends not only on (f_m, f'_m) but also on the solution vector components residing in neighboring blocks. Thus we have $F(t, f, f') = [F_1, F_2, ..., F_M]^T$ and each of the blocks F_m is uncoupled from the others. The preconditioner associated with this decomposition is

$$P = \mathrm{diag}[P_1, P_2, ..., P_M], \tag{9}$$

where P_m is the difference quotient of

$$J_m = \frac{\partial F_m}{\partial f_m} + \alpha \frac{\partial F_m}{\partial f'_m}. \tag{10}$$

The preconditioner matrix is taken to be banded with upper and lower half-bandwidths defined as the number of nonzero diagonals above and below the main diagonal, respectively.

The solution of the complete preconditioned linear system $Px = b$ is equivalent to solving each of the equations

$$P_m x_m = b_m, \tag{11}$$

which can be done by a banded LU factorization of P_m followed by a banded backsolve. Obviously, both procedures can be performed completely in parallel for $m = 1, ..., M$.

4. Numerical experiments

In this section, we present numerical results for the performance of SAI and BBD preconditioners on the simulation of the anisotropic diffusion in the human brain. The resolution of the diffusion tensor MRI data set is $128 \times 128 \times 16$ with each voxel size being $2.5 \times 2.5 \times 7.5\, mm^3$ defined on the Cartesian mesh. Natural ordering is used toward the points in the Cartesian grid. The numerical tests are conducted on a 32-processor (HP PA-RISC 8700 processors running at 750 MHz) subcomplex of an HP superdome supercomputer at the University of Kentucky. Each processor has two gigabytes local memory. During the experiments, four processors are used for numerical simulations unless otherwise indicated explicitly. The total integration time is 10^4 seconds. The sparse linear systems are solved by GMRES(20) with restarts no more than 10.

The construction of SAI preconditioners with *a priori* sparsity pattern in our numerical tests involves two dropping tolerances τ_1 and τ_2. The first one, τ_1, is used to sparsify

τ_1	τ_2	$spar$	$iter$	$setup$	$solve$	$total$
0.5	0.0	0.055	168	13.27	44.46	75.78
0.1		0.108	99	17.45	30.31	63.36
0.01		0.155	88	23.50	31.96	67.63
0.005		0.185	85	27.20	33.68	70.86
10^{-3}		0.268	82	37.91	39.06	81.29
10^{-4}		0.304	76	42.89	40.37	85.23
0.0		1.000	74	59.32	40.21	104.44
0.5	10^{-4}	0.055	168	15.13	44.28	77.56
0.05		0.110	99	19.76	27.67	65.62
0.01		0.143	89	24.55	25.77	68.62
10^{-3}		0.144	87	26.35	25.40	69.92
0.0		0.144	87	28.78	25.23	72.20
0.0	0.5	0.055	168	20.95	43.54	82.64
	0.005	0.055	148	21.15	39.31	78.77
	0.0005	0.110	97	24.94	27.36	70.43
	10^{-4}	0.144	87	28.78	25.23	72.20
0.005	0.1	0.055	168	16.73	44.35	79.33
	0.005	0.055	148	16.65	39.25	74.18
	10^{-3}	0.105	99	20.31	27.71	66.13
	10^{-4}	0.144	87	25.26	25.42	69.12
10^{-3}	0.1	0.055	168	18.11	45.25	81.56
	0.005	0.055	148	18.22	39.14	75.50
	10^{-3}	0.105	99	21.68	27.73	67.63
	10^{-4}	0.144	87	26.35	25.40	69.92

Table 1. Performance data of SAI preconditioners with varying τ_1 and τ_2.

w_1	w_2	$spar$	$iter$	$setup$	$solve$	$total$
3	3	0.384	206	8.32	54.10	80.53
5			140	11.16	37.15	66.40
6			201	12.48	52.47	83.08
7			133	14.03	36.07	68.29
9			141	16.16	37.41	71.72
4	4	0.494	199	10.17	56.34	84.73
5			181	11.56	50.98	80.70
6			208	12.86	58.33	89.27
7			134	14.36	38.84	71.40
9			141	16.79	40.28	75.16
5	5	0.604	188	11.97	56.72	86.76
6			208	13.23	62.65	94.31
7			134	14.71	41.57	74.35
9			177	17.39	53.72	89.27
6	6	0.713	208	13.74	65.81	97.52
7			135	15.14	43.90	77.00
9			181	18.03	58.01	94.08
7	7	0.823	170	15.76	58.24	91.99
8			133	17.08	46.88	81.90
9			181	18.29	61.66	98.02

Table 2. Performance data of BBD preconditioners with varying w_1 and w_2.

the coefficient matrix A in (6) to extract its sparsity pattern. Once an SAI matrix P is computed according to the sparsity pattern of A, we drop small entries of P with respect to τ_2. The resulting matrix P becomes the preconditioner for Equation (6). When constructing BBD preconditioners, we also have to choose two parameters for use in its computing procedure. The first one is w_1, used in the difference quotient approximation, such that P_m in Equation (11) is computed as a matrix with bandwidth $2 \times w_1 + 1$. Then, the second parameter, w_2, is applied to P_m such that it only retains bandwidth $2 \times w_2 + 1$. Here, the upper and lower half-bandwidths are treated to be the same since our linear system is structured with equal half-bandwidths.

In the tables containing numerical results, the notations of τ_1, τ_2, w_1, and w_2 are just explained in the previous paragraph; $spar$ means the sparsity ratio, which is derived from the number of nonzero entries in the preconditioner matrix divided by that of the Jacobian matrix; $iter$ shows the total number of scaled preconditioned GMRES iterations during the course of integration; $setup$ is the CPU time in seconds to construct the preconditioners; $solve$ measures the CPU time to solve preconditioned systems by GMRES iterations; $total$ gives the total CPU time in seconds for solving the DAE problem, including the time for doing initialization (almost constant in every case) which is not reported.

Table 1 shows the performance results obtained by using the SAI preconditioner, with two dropping tolerances τ_1 and τ_2. We can see that the SAI preconditioner converges in all the cases. When τ_2 is fixed, with the decrease of τ_1, it takes more time to construct the preconditioner, as shown in the column $setup$. Since more entries are kept in the sparsity pattern with the reduction of τ_1, the convergence performance of the GMRES method gets better with less iterations in the column of $iter$, but with the price of more memory consumed, as specified by the $spar$ values. The CPU time spent in solving the problem, as in entries of $total$, is getting larger when we choose a smaller dropping tolerance τ_1 with a fixed τ_2, except in the case of $\tau_1 = 0.5$, which has a much larger number of iterations than the rest of the cases, resulting in more time in the iteration phase, because of its less accurate sparse approximate inverse. We notice that when τ_2 is set to be 0, with τ_1 getting closer down to 0, the iteration time, $solve$, tends to increase rather than decrease, as seen in the case $\tau_2 = 10^{-4}$. It implies that the threshold value τ_2 plays an important role in the CPU time cost of the iteration procedure.

The second scenario being analyzed here is to see the effects on computational costs of choosing different τ_2, when τ_1 sets fixed. As we mentioned, τ_2 is a dropping threshold, i.e., small entries will be eliminated from the computed SAI matrix with respect to τ_2. From the lower half of Table 1, it is apparent that the SAI preconditioner is getting a better accuracy with the value of τ_2 going down, which leads to increased sparsity ratio, $spar$, and an improved convergence behavior, as revealed by the reduced number of iterations, $iter$, and iteration time, $solve$. However, it has to pay the price for the rising cost to construct a higher quality preconditioner, shown by the $setup$ column with the time running longer, since more entries are kept and computed in the preconditioner. Therefore, it is difficult for $total$ to manifest a monotonic decrease or increase along with τ_2, since we have to deal with the tradeoff between either making the SAI preconditioner as precise as possible to reduce the iteration time but with the price of more construction expense, or spending less time in the construction step but with the risk of needing more computational efforts to do iterations. The last remark we point out is that when the value of τ_2 gets down to 0 with τ_1 keeping unchanged, the iteration time, $solve$, bounces up if τ_2 reaches 0, instead of declining further. The reason is that no element is dropped from the computed SAI matrix which has a lot of entries with small magnitudes. Thus a large portion of the CPU time is wasted on doing computations for these small elements during the iteration step, where little is gained. This results in a longer time for the solution to get converged.

The numerical results of applying the BBD preconditioners are contained in Table 2. The definition of the two parameters w_1 and w_2 tells us that they are both the half-bandwidth of matrices, where w_1 is for matrices used in the difference quotient approximation and w_2 is the half-bandwidth for the actual BBD preconditioner employed during iterations. Thus the sparsity ratio only depends on w_2 and larger w_2 means more entries in the preconditioner matrix. Table 2 gives a clear picture that the construction cost of the BBD preconditioners goes up along with the increase of w_2 as well as w_1, which is reflected in the $setup$ column. However, from the values of the number of iterations and the time for iterations illustrated in columns of $iter$ and $solve$, we could not see a consistently improved convergence behavior here with changes of half-bandwidths. The reason may lie in the construction of banded structure of the preconditioner matrix, where the GMRES method thus the iterative solution is very sensitive to the availability of some specific entries. Therefore, it can be concluded from Table 2 that it will be difficult for us to expect a better convergence rate even if more entries are retained in the matrix for difference quotient approximation and the final preconditioner matrix. This leads to a fact that the convergence behavior is sometimes unpredictable with the selection of w_1 and w_2.

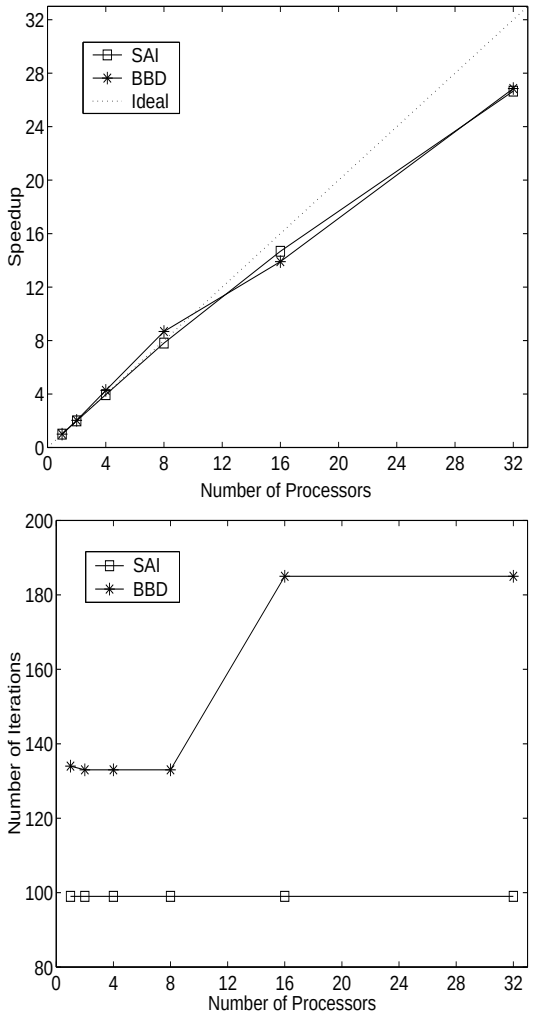

Figure 2. Scalability comparisons of the SAI and BBD preconditioners. Parameters are selected as, for SAI, $\tau_1 = 0.05$, $\tau_2 = 0.001$; for BBD, $w_1 = 8$, $w_2 = 5$.

When making comparisons between data in Table 1 and in Table 2, we found that the SAI preconditioner has a much more predictable behavior than the BBD preconditioner. Further observation reveals that with appropriate choices of parameters, the SAI case has a higher construction cost than the BBD case, while the BBD case delivers a poorer convergence performance than in the case of SAI. Thus the SAI produces a higher quality preconditioner with a better accuracy than BBD, although suffering from more computational efforts. Roughly speaking, though, the total CPU time spent under the SAI preconditioner seems to be less than the BBD preconditioner in our current testing cases.

By comparing the data in the $spar$ columns, we can see that the SAI preconditioners need much less storage space than the BBD preconditioners do.

Further comparison between the SAI and BBD preconditioners is made by comparing its scalability, as shown in Figure 2. The bottom plot gives curves for the number of iterations versus the number of processors. When the number of processors increases from 1 to 32, the number of iterations of the BBD preconditioner changes a lot, displaying an oscillating curve. This phenomenon is due to the fact that, as the number of processors grows larger, the number of independent computational subdomains increases and it becomes more difficult to keep these subdomains from having predominantly local coupling. So the performance of the BBD preconditioner is significantly affected by the number of processors used. On the contrary, the SAI preconditioner presents a much better and desired behavior, where the number of iterations remains constant, fully independent on the number of processors. The top plot of Figure 2 shows speedup curves for both preconditioners. Since we are interested in the robustness and performance of preconditioners and the preconditioned iterative solver, the time value used in calculating the speedup for this plot is only the summation of the preconditioner construction time and iteration time. It can be seen from the plot that both preconditioners have good speedup results, close to linear. We also notice that there exists superlinear speedups for the BBD preconditioner. This can be attributed to the caching effects. When we dispatch the problem onto multiple processors, the subproblems are obviously a fraction of the original problem size. With a smaller problem size, we are most likely to get a higher cache hit rate, and the result, even after considering the communication time, is still better than the time on a single processor with more cache misses.

5. Summary and remarks

Two classes of preconditioners, SAI and BBD, are applied in numerical tests to carry out simulations of the anisotropic diffusion process in the human brain. Our test results show that the SAI preconditioners based on *a priori* sparsity pattern provides a more robust and efficient parallel preconditioning technique than the BBD preconditioners. The tests also illustrate that the best performance of the preconditioners can be obtained by choosing optimum values for their corresponding parameters, τ_1 and τ_2 in SAI, and w_1 and w_2 in BBD, which have direct and distinct influences on the quality and the construction expense of preconditioners, the convergence rate of iterative solutions, and the total computational efforts.

Acknowledgments

This study was funded by the U.S. Department of Energy Office of Science under the project "Development of a High Performance Anisotropic Diffusion Equation Solver Using the ACTS Toolkit" (DE-FG02-02ER45961). The first two authors would like to acknowledge DOE's support of their attending the ACTS Workshop, organized by Drs. Tony Drummond and Osni Marques, at the Lawrence Berkeley National Laboratory, in September 2002. We would also like to thank Dr. Daniel Gembris, at Institute for Medicine, Jülich Research Center, Jülich, Germany, for providing the diffusion tensor data set.

References

[1] The DOE ACTS (Advanced CompuTational Software) collection web site: http://acts.nersc.gov/, 2003.

[2] D. Le Bihan, J. F. Mangin, C. Poupon, C. A. Clark, S. Pappata, N. Molko, and H. Chabriat, Diffusion tensor imaging: concepts and application, *J. Magnetic Resonance Imaging*, 13:534-546, 2001.

[3] A. B. M. Bjornemo, *White Matter Fiber Tracking Using Diffusion Tensor MRI*, Master's Thesis, Linkoping University, Sweden, 2002.

[4] P. G. Batchelor, D. L. G. Hill, F. Calamante, and D. Atkinson, Study of connectivity in the brain using the full diffusion tensor from MRI, *Information Processing in Medical Imaging*, 17th International Conference, IPMI'01, June 2001, UC Davies, USA. Published by Springer, *Lecture Notes in Computer Science 2082*, pp. 121-133.

[5] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, SIAM, Philadelphia, PA, 1996.

[6] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, Using Krylov methods in the solution of large-scale differential-algebraic systems, *SIAM J. Sci. Comput.*, 15:1467-1488, 1994.

[7] E. Chow, A priori sparsity patterns for Parallel sparse approximate inverse preconditioners, *SIAM J. Sci. Comput.*, 21:1804-1822, 2000.

[8] D. Gembris, H. Schumacher, and D. Suter, Solving the diffusion equation for fiber tracking in the living human brain, *Proc. of the International Society for Magnetic Resonance Medicine (ISMRM)*, 9:1529, Glasgow, Scotland, April 2001.

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

[10] A. C. Hindmarsh and A. G. Taylor, *User Documentation for IDA, a Differential-Algebraic Equation Solver for Sequential and Parallel Computers*, LLNL Report UCRL-MA-136910, Center for Applied Scientific Computing, LLNL, Livermore, CA, 1999.

[11] N. Kang, J. Zhang, and E. S. Carlson, Performance of ILU preconditioning techniques in simulating anisotropic diffusion in the human brain, *Future Generation Computer Systems* (to appear).

[12] C. Nicholson, Diffusion and related transport mechanism in brain tissue, *Reports on Progress in Physics*, 64:815-884, 2001.

[13] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA, 1996.

[14] The SUNDIALS software package web site: http://acts.nersc.gov/sundials/.

[15] Y. Saad and M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.*, 7:856-869, 1986.

[16] I. Vorisek and E. Sykova, Evolution of anisotropic diffusion in the developing rat corpus callosum, *J. Neurophysiol.*, 78:912-919, 1997.

[17] S. K. Warfield, F. Talos, A. Tei, A. Bharatha, A. Nabavi, M. Ferrant, P. M. Black, F. A. Jolesz, and R. Kikinis, Real-time registration of volumetric brain MRI by biomechanical simulation of deformation during image guided neurosurgery, *Computing and Visualization in Science*, 5:3-11, 2002.