

Comparison of Parallel Preconditioners in Anisotropic Diffusion Simulation with Human Brain DT-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

Eric S. Carlson[§]

Department of Chemical Engineering, University of Alabama,
P. O. Box 870203, Tuscaloosa, AL 35487-0203, USA

June 24, 2003

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.

*Technical Report No. 376-03, Department of Computer Science, University of Kentucky, Lexington, KY, 2003.

[†]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.

Key words: anisotropic diffusion simulation, diffusion tensor imaging, differential-algebraic equation, parallel preconditioning techniques

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 [14]. 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].

Diffusion is a three dimensional process which can be modeled by a second order tensor. The molecular mobility in tissues, such as in the brain white matter, may be anisotropic. It is known that anisotropic diffusion in the white matter reveals microscopic properties of the anatomy of the nerve fibers by the fact that water tends to diffuse predominantly along the fibers, because tightly packed myelin membranes, which is widely assumed to be the main barrier for water diffusion, restricts diffusion perpendicular to the axons [3, 22]. The basic principle of the diffusion tensor (magnetic resonance) imaging (DT-MRI) stems from the orientation information provided by the phenomenon of water diffusion anisotropy in the white matter [15]. The diffusion tensor imaging unveils the diffusion behavior of water in tissue on a voxel by voxel basis. For each voxel, the diffusion tensor yields the diffusion coefficient corresponding to any direction in space. Given this information, diffusion anisotropy may become the reason to explain the fact that the diffusion across fiber axes encounters greater obstacle or restriction than along them. The direction of the highest diffusion coefficients is therefore believed to point along a putative fiber bundle traversing the voxel. Thus, the panoramic view of the fastest diffusion direction can be generated to provide a visualization of the white matter pathways and their orientation.

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 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 [13, 19].

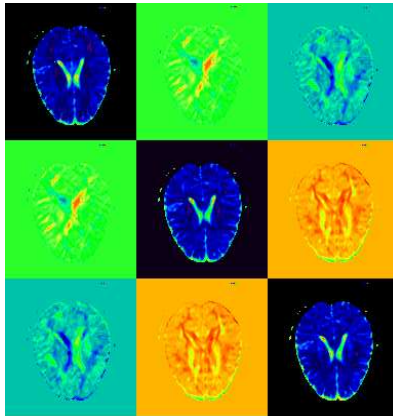


Figure 1: An axial slice of a diffusion tensor volume. It shows the diffusion tensor components, corresponding to the diffusion tensor matrix D .

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.,

$$J = -d\nabla C, \quad (1)$$

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

$$J = -D\nabla C, \quad (2)$$

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 and other typical systems, the tensor is symmetric. In a reference frame $[x', y', z']$ that coincides with the principal or self directions of diffusivity, the offdiagonal terms do not exist and the tensor is reduced only to its diagonal terms. In practice, however, measurements are made in the reference frame

$[x, y, z]$ of the MRI scanner gradients, which usually do not coincide with the diffusion frame of the tissue [2]. Hence, it is important to note that each component of the flux vector J may include contributions from all components of the concentration gradient.

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

$$\frac{\partial C}{\partial t} = \nabla \cdot (D \nabla C), \quad (3)$$

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.

In Cartesian coordinate system, the Equation (3) is expressed as

$$\begin{aligned} \frac{\partial C}{\partial t} = & \frac{\partial}{\partial x} (D_{xx} \frac{\partial C}{\partial x} + D_{xy} \frac{\partial C}{\partial y} + D_{xz} \frac{\partial C}{\partial z}) \\ & + \frac{\partial}{\partial y} (D_{yx} \frac{\partial C}{\partial x} + D_{yy} \frac{\partial C}{\partial y} + D_{yz} \frac{\partial C}{\partial z}) \\ & + \frac{\partial}{\partial z} (D_{zx} \frac{\partial C}{\partial x} + D_{zy} \frac{\partial C}{\partial y} + D_{zz} \frac{\partial C}{\partial z}). \end{aligned} \quad (4)$$

This equation 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 [21]. 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.

The incomplete LU preconditioners studied in [11] are not suitable for implementations on parallel computers. 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, i.e., the matrix is divided into groups of complete rows and each group is assigned to one processor. 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. The ACTS Toolkit is a set of tools mostly developed at the national laboratories of the U.S. Department of Energy (DOE) and universities to facilitate the development of high performance computing applications in scientific and complex research areas. ACTS stands for Advanced Computational Testing and Simulation. *One aim of this study is to take advantages of these high performance general*

purpose ACTS tools and to evaluate their usefulness in this particular new application. 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 high performance DAE integration solver as well as its parallel implementation scheme. The parallel preconditioning techniques are discussed in detail in Section 3, including the banded block diagonal preconditioner and the sparse approximate inverse preconditioner. 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

Among the initial steps of numerical simulations on anisotropic diffusion process is to determine a suitable and efficient gridding scheme to discretize the governing partial differential equation. Since the tensor data set used in our current simulation is measured and processed on a Cartesian mesh, we discretize the 3D diffusion equation (4) on the 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 (4), respectively.

On the boundaries of the heterogeneous system, we assume that it is insulated, i.e., $(D\nabla C) \cdot \mathbf{n} = 0$, 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 with an equal standard deviation of $\sigma = 0.2$ is selected to be the initial distribution profile of the water concentration in the brain

$$C \Big|_{t=0} = \frac{1}{\sigma^3 \sqrt{8\pi^3}} e^{-[(x-\mu_x)^2 + (y-\mu_y)^2 + (z-\mu_z)^2]/2\sigma^2}, \quad (5)$$

where μ_x , μ_y , and μ_z are the mean in the x , y , and z direction, respectively.

The discretization of the Equation (4) and its boundary conditions on the Cartesian grid generates a large scale system of semi-explicit differential-algebraic equations (DAEs) with the form

$$F(t, f, f') = 0, \quad (6)$$

where f and f' are N -dimensional vectors corresponding to the discretized values of C and $\partial C/\partial t$, and the initial condition given in (5). Detailed explanations of the DAEs theories and its numerical solution methods on initial-value problems are given in [5]. In this paper, we consider using a high performance DAE solver, the IDA solver in the SUNDIALS suite, one of the software packages contained in the ACTS Toolkit. The name IDA stands for Implicit Differential-Algebraic solver, which is a general purpose solver for the initial value problem for systems of DAEs. SUNDIALS stands for SUite of Nonlinear and Differential/ALgebraic equation Solvers. More detailed information about

the SUNDIALS suite and the IDA solver can be found in [10, 17]. In the following, 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] for more details.

The IDA solver uses the backward differentiation formula (BDF) method to approximate the time derivative in (6), implemented in a variable order, variable step form. It means that at every step, IDA selects the order and step size based on the behavior of the solution, in an attempt to get a solution with the minimum number of steps. The default orders of BDF range from 1 to 5. The application of BDF to the DAE system (6) leads to a nonlinear algebraic system to be solved at each time step, which is

$$G(f_n) \equiv F\left(t_n, f_n, h_n^{-1} \sum_{i=0}^k \alpha_{n,i} f_{n-i}\right) = 0, \quad (7)$$

where f_n is the calculated approximation to $f(t_n)$ and the step size is $h_n = t_n - t_{n-1}$. $\alpha_{n,i}$, $i = 0, 1, \dots, k$, 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 (7) by a modified version of Newton iteration method regardless of the integration method options. This results in a linear system for each Newton correction, given by

$$f_n^{(m+1)} = f_n^{(m)} - cJ^{-1}G[f_n^{(m)}], \quad (8)$$

where $f_n^{(m)}$ is the m th 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

$$J = \frac{\partial G}{\partial f} = \frac{\partial F}{\partial f} + \alpha \frac{\partial F}{\partial f'}, \quad (9)$$

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 n th time step by using the weighted root-mean-square (wrms) norm, and requires it to satisfy the inequality

$$\|E_n\|_{wrms} = \left[\frac{1}{N} \sum_{i=1}^N (E_n^i/w^i)^2 \right]^{1/2} < 1, \quad (10)$$

where the superscript i denotes the i th component, and the i th weight is given by

$$w^i = rtol|f^i| + atol^i \quad \text{or} \quad w^i = rtol|f^i| + atol, \quad (11)$$

where $atol$ and $rtol$ stand for absolute and relative error tolerance, respectively. $rtol$ is a scalar while $atol$ may be either an N -dimensional vector or a scalar.

For the solution of the linear system (8), only Krylov (iterative) method is available in the parallel version of the IDA solver. Actually the only iterative method included in the IDA solver so far is the scaled preconditioned GMRES method, denoted as SPGMR.

When solving the linear system (8), 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. Because any nontrivial DAE needs a preconditioner, the Newton iteration test and hence the code reliability is not justified without a reasonable preconditioner. The IDA solver only allows left preconditioning. Also, it is known that the iteration matrix for any nontrivial DAE becomes more and more ill-conditioned as the step size is increased [5]. Therefore, the preconditioner may need to be rescaled. So, scaling is included explicitly in the SPGMR algorithm, using a diagonal scaling matrix whose diagonal elements are the weights w^i of (11).

The parallel version of IDA uses a revised version of the vector module NVECTOR in its package to achieve parallelism and the MPI (Message Passing Interface) library for all interprocessor communication. The NVECTOR module is the key to make possible the shift from the serial computing environment to the parallel computing environment. It contains a set of mathematical operations on N -vectors (N -dimensional vectors), including vector linear combinations, scaling, 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, which allows parallel computation to be neatly implemented in these codes. Because the parallel form of IDA is intended for an SPMD (Single Program Multiple Data) 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. For any given vector operation, each processor performs the operation on its contiguous elements of the input vectors, followed by a global reduction operation where needed. In this way, vector calculations can be done simultaneously with each processor working on its own segment of the vector.

3 Parallel Preconditioning Techniques

We can rewrite the linear system (8) as

$$J[f_n^{(m+1)} - f_n^{(m)}] = -cG[f_n^{(m)}], \quad (12)$$

which needs to be solved at each Newton iteration. To simplify notation, (12) can be abstracted as

$$Ax = b, \quad (13)$$

where A is the $N \times N$ system Jacobian matrix in (9), $x = f_n^{(m+1)} - f_n^{(m)}$ and $b = -cG[f_n^{(m)}]$ are both N -dimensional vectors. The sparse linear system (13) is solved by using the scaled preconditioned GMRES method, which is included in the IDA solver. As mentioned in the previous section, preconditioning of the linear iteration is essential and beneficial for both robustness and efficiency.

The basic idea of preconditioning is as follows. Let P be a matrix which approximates the coefficient matrix A in some way. Preconditioning in an iterative method for solving the linear system (13) means applying the method instead to the equivalent system $P^{-1}Ax =$

$P^{-1}b$. We expect that the preconditioned system would be easier to solve than the original problem, i.e., the matrix $P^{-1}A$ is better conditioned than A or $P^{-1}A$ has a more favorable eigenvalue distribution than A does. In order for preconditioning to be efficient, P should be in some sense close to A and its construction should be inexpensive. In the parallel computing environment, it is ideal that both the preconditioner construction phase and the preconditioned solution phase possess a high degree of parallelism.

The first parallel preconditioning technique under our investigation is a class of sparse approximate inverse preconditioners. The sparse approximate inverse preconditioner, as its name implies, is an approximation to A^{-1} , the inverse of a 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 and can be implemented in parallel without difficulty. Detailed discussion on this class of preconditioners is given below. 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 preconditioners.

Sparse approximate inverse preconditioners. The sparse approximate inverse 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. The right approximate inverse is used here, which is notationally more clear and more convenient for us to illustrate the preconditioning approach. Though it is also possible to approximate A^{-1} from the left, in a sense that $PA \approx I$, and this case is called the left preconditioner.

In the case of the right approximate inverse preconditioning, 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

$$\|AP - I\|_F^2 = \sum_{j=1}^N \|(AP - I)e_j\|_2^2 = \sum_{j=1}^N \|Ap_j - e_j\|_2^2, \quad (14)$$

where p_j and e_j are the j th column of the matrix P and that of the identity matrix I , respectively. The solution of the minimization function (14) can be decoupled into N independent least squares problems

$$\min_{p_j} \|Ap_j - e_j\|_2, \quad j = 1, 2, \dots, N. \quad (15)$$

It is apparent that inherent parallelism lies in the solution of (15) 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 (15) in parallel.

In general, the degrees of freedom of the minimization problem (14) are the nonzero values in P as well as their locations. 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. When P is sparse, each least-squares matrix is small with the number of columns equal to that of nonzeros in its corresponding p_j . 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]. The adaptive schemes, as referred in [7], start the algorithm of computing an approximate inverse with an initial pattern, and this pattern is updated progressively with a new minimization problem being solved each time, until the residual norm is small enough, or a maximum number of nonzeros have been reached. Usually, this strategy is capable of generating high-quality sparse approximate inverse preconditioners, yet it tends to be very expensive. Another option is to build a prescribed sparsity pattern so that the sparse approximate inverse can be yielded immediately by minimizing (14). We are particularly interested in this method in that for PDE problems, as in our case, the sparsified patterns of the original matrix A can be employed *a priori* as effective approximate inverse patterns. It is relatively cheap to compute compared with the adaptive procedures. Here “sparsified” means that certain small entries of A are dropped before its sparsity pattern is extracted. If a higher accuracy is desired, the sparsity patterns of powers of the sparsified matrices A^2, A^3, \dots , may be used instead, but with the price of significantly increased computational cost [20]. Details about *a priori* sparsity patterns can be found in [7]. Thus, in the sense of tradeoffs between accuracy and computational cost, this paper focuses on choosing only the sparsified nonzero positions of A to be *a priori* sparsity patterns.

Once we get a sparsity pattern for the sparse approximate inverse P , it can be computed by solving the minimization problem (15). Let n_2 be the number of certain indices of p_j such that the entries of p_j at these locations are nonzeros in terms of its sparsity pattern. We denote the n_2 nonzero entries of p_j as \tilde{p}_j . Next, let A_j be the submatrix of A such that it contains n_2 columns of A corresponding to \tilde{p}_j . Since A is sparse, we might eliminate all rows which are identically zero in the submatrix A_j , leading to a reduced submatrix with n_1 rows which are not identically zero, denoted as \tilde{A}_j . It follows that solving (15) for p_j is reduced to solving a small least-squares problem for p_j of order $n_1 \times n_2$

$$\min_{\tilde{p}_j} \|\tilde{A}_j \tilde{p}_j - \tilde{e}_j\|_2, \quad j = 1, 2, \dots, N, \quad (16)$$

where \tilde{e}_j is a vector that contains n_1 entries of e_j corresponding to the rows of \tilde{A}_j . We note that the matrix \tilde{A}_j is very small because A and P are sparse matrices. It has full rank if A is nonsingular. One of the methods to solve (16) is to use the QR factorization of \tilde{A}_j [9]

$$\tilde{A}_j = Q_j \begin{pmatrix} R_j \\ 0 \end{pmatrix}, \quad (17)$$

where R_j is a nonsingular upper triangular $n_2 \times n_2$ matrix and Q_j is an $n_1 \times n_1$ orthogonal matrix, i.e., $Q_j^{-1} = Q_j^T$. Let $\tilde{c}_j = Q_j^T \tilde{e}_j$, the solution of (16) is $\tilde{p}_j = R_j^{-1} \tilde{c}_j (1 : n_2)$. We solve \tilde{p}_j for each $j = 1, 2, \dots, N$, independently, and finally get the approximate inverse matrix P , which minimizes $\|AP - I\|_F$ for the given sparsity pattern.

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 preconditioners, which is closely related to the DAE system (6). More details can be found in [10].

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 (6) 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 $F(t, f, f') = [F_1, F_2, \dots, F_M]^T$ and each of the blocks F_m is uncoupled from the others. The preconditioner associated with this decomposition is

$$P = \text{diag}[P_1, P_2, \dots, P_M], \quad (18)$$

where P_m is the difference quotient of

$$J_m = \frac{\partial F_m}{\partial f_m} + \alpha \frac{\partial F_m}{\partial f'_m}. \quad (19)$$

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. However, its upper and lower half-bandwidths need not be the true values of that of J_m , if smaller values provide a more efficient preconditioner. They also do not have to be the same on every processor.

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

$$P_m x_m = b_m, \quad (20)$$

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 $m = 1, \dots, M$. Similar block-diagonal preconditioners could be devised with a different treatment of the blocks P_m . For instance, incomplete LU factorization might be used instead of banded LU factorization.

4 Numerical Experiments

In this section, we present a number of numerical results for the performance of sparse approximate inverse (SAI) and banded-block-diagonal (BBD) preconditioners on the simulation of the anisotropic diffusion in the human brain. The codes are implemented with

the IDA solver package as the primary integration tool. The diffusion tensor MRI data set used in our simulation is provided by Dr. Daniel Gembris at Institut für Medizin im Forschungszentrum Jülich (Institute for Medicine, Jülich Research Center, Jülich, Germany). The resolution of the tensor data set is $128 \times 128 \times 16$ with each voxel size being $2.5 \times 2.5 \times 7.5 \text{ mm}^3$ defined on the Cartesian mesh. The 3D domain is discretized in the Cartesian grid with a 19 point difference stencil such that there are 262,144 unknowns in the resulting linear systems and 4,777,024 nonzeros in the system Jacobian matrix. 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.

The following testing parameters are set fixed for all numerical experiments that we carry out. Four processors are used in our numerical simulations unless otherwise indicated explicitly. The total integration time is 1.0×10^4 seconds. The sparse linear systems are solved by GMRES(20) with the maximum number of restarts being 10. Thus GMRES gets restarted for every 20 iterations to avoid large memory cost. The maximum order in the BDF method is set at 5. In order to have an acceptable convergence rate, the maximum time step size is not allowed to exceed 2.0×10^3 . We select both the relative tolerance and absolute tolerance to be 1.0×10^{-4} . The linear and nonlinear iteration convergence factors are chosen to be 1.0×10^{-5} and 0.01, respectively, in hope of achieving reasonable accuracy without taking too much computing time.

In our numerical tests, the actual construction of SAI preconditioners with *a priori* sparsity pattern involves two dropping tolerances τ_1 and τ_2 . The first dropping tolerance, τ_1 , is used to sparsify the coefficient matrix A in (13), in order to extract the sparsity pattern of A . Once a sparse approximate inverse matrix P is computed according to the sparsity pattern of A , we drop small entries of P with respect to τ_2 , the second tolerance. The resulting matrix P becomes the preconditioner for Equation (13). In order to construct the BBD preconditioners, we also have to choose two parameters for use in its actual computing procedure. The first one needing to be specified is w_1 , used in the difference quotient approximation, such that P_m in Equation (20) 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; *sparsity* 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; *init* counts the CPU time for doing initialization, including reading the tensor data, calculating the coefficient matrix, setting initial profiles, etc.; *setup* is the CPU time in seconds to construct the preconditioners; *solve* is the CPU time in seconds to solve preconditioned systems by GMRES iterations; *total* gives the total CPU time in seconds for solving the DAE problem, i.e., it is the sum of *init*, *setup*, and

solve.

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 *sparsity* 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 $\tau_2 = 0.0001$. It implies that the threshold value τ_2 plays an important role in the CPU time cost of the iteration procedure. More discussion on τ_2 is as follows.

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 sparse approximate inverse 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, *sparsity*, 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 sparse approximate inverse 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 banded-block-diagonal (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

τ_1	τ_2	<i>sparsity</i>	<i>iter</i>	<i>init</i>	<i>setup</i>	<i>solve</i>	<i>total</i>
0.5	0.0	0.055	168	18.24	13.27	44.46	75.78
0.1		0.108	99	18.24	17.45	30.31	63.36
0.01		0.155	88	18.38	23.50	31.96	67.63
0.005		0.185	85	18.28	27.20	33.68	70.86
0.001		0.268	82	18.27	37.91	39.06	81.29
0.0001		0.304	76	18.38	42.89	40.37	85.23
0.0		1.000	74	18.23	59.32	40.21	104.44
0.5		0.0001	0.055	168	18.20	15.13	44.28
0.05	0.110		99	18.24	19.76	27.67	65.62
0.01	0.143		89	18.36	24.55	25.77	68.62
0.001	0.144		87	18.23	26.35	25.40	69.92
0.0	0.144		87	18.24	28.78	25.23	72.20
0	0.5	0.055	168	18.22	20.95	43.54	82.64
	0.005	0.055	148	18.37	21.15	39.31	78.77
	0.0005	0.110	97	18.19	24.94	27.36	70.43
	0.0001	0.144	87	18.24	28.78	25.23	72.20
0.005	0.1	0.055	168	18.32	16.73	44.35	79.33
	0.005	0.055	148	18.35	16.65	39.25	74.18
	0.001	0.105	99	18.17	20.31	27.71	66.13
	0.0001	0.144	87	18.51	25.26	25.42	69.12
0.001	0.1	0.055	168	18.28	18.11	45.25	81.56
	0.005	0.055	148	18.22	18.22	39.14	75.50
	0.001	0.105	99	18.29	21.68	27.73	67.63
	0.0001	0.144	87	18.23	26.35	25.40	69.92

Table 1: Performance data of the sparse approximate inverse preconditioners (SAI) with varying dropping tolerances τ_1 and τ_2 .

w_1	w_2	<i>sparsity</i>	<i>iter</i>	<i>init</i>	<i>setup</i>	<i>solve</i>	<i>total</i>
3	3	0.384	206	18.19	8.32	54.10	80.53
5	140		18.18	11.16	37.15	66.40	
6	201		18.20	12.48	52.47	83.08	
7	133		18.27	14.03	36.07	68.29	
9	141		18.18	16.16	37.41	71.72	
4	4	0.494	199	18.31	10.17	56.34	84.73
5	181		18.24	11.56	50.98	80.70	
6	208		18.17	12.86	58.33	89.27	
7	134		18.28	14.36	38.84	71.40	
9	141		18.19	16.79	40.28	75.16	
5	5	0.604	188	18.15	11.97	56.72	86.76
6	208		18.48	13.23	62.65	94.31	
7	134		18.15	14.71	41.57	74.35	
9	177		18.25	17.39	53.72	89.27	
6	6	0.713	208	18.11	13.74	65.81	97.52
7	135		18.11	15.14	43.90	77.00	
9	181		18.20	18.03	58.01	94.08	
7	7	0.823	170	18.17	15.76	58.24	91.99
8	133		18.15	17.08	46.88	81.90	
9	181		18.21	18.29	61.66	98.02	

Table 2: Performance data of the banded-block-diagonal preconditioners (BBD) with varying parameters of w_1 and w_2 .

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 .

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 *sparsity* columns, we can see that the SAI preconditioners need much less storage space than the BBD preconditioners do.

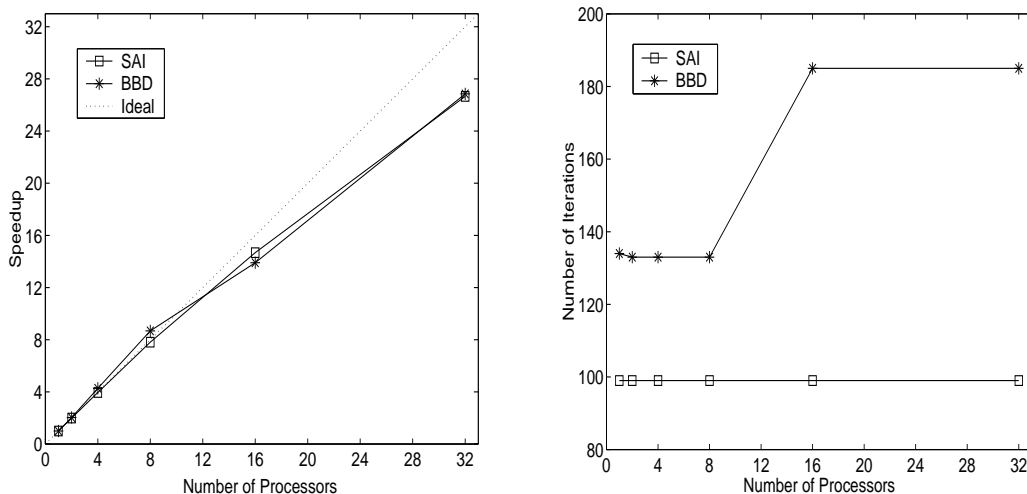


Figure 2: The diagram for scalability comparison of the SAI and BBD preconditioners. Parameters are selected as, for SAI, $\tau_1 = 0.05$, $\tau_2 = 0.001$; for BBD, $w_1 = 8$, $w_2 = 5$. Left: speedup obtained v.s. the number of processors used. Right: the number of iterations v.s. the number of processors used.

Further comparison between the SAI and BBD preconditioners is made by comparing its scalability, as shown in Figure 2. The right 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, completely independent of the change of the number of processors. The left 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, as indicated by entries of *setup* and *solve* in Tables 1 and 2. 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

In the current work, we simulate the anisotropic diffusion process in the human brain, which could be of vital importance for the analysis of DT-MRI. The 3D anisotropic diffusion equation is discretized on a Cartesian grid, which leads to a large scale system of semi-explicit differential-algebraic equations (DAEs). A high performance DAE solver, the parallel version of IDA solver, with scaled preconditioned GMRES iterative method, is employed to solve the resulting large scale system of DAEs. By applying parallel preconditioning techniques for the GMRES method, we conduct numerical experiments to investigate the efficiency and effectiveness of the preconditioners in solving the linear systems arising at each Newton iteration.

Two classes of preconditioners, the SAI and BBD preconditioners, are applied in our testing cases. 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 for our brain diffusion simulation problem. It is the SAI preconditioner whose convergence performance is not affected by the number of processors employed, although both the SAI and BBD preconditioners demonstrate good speedup which is close to linear. The SAI preconditioners take more CPU time to construct, but need less memory space to store, than the BBD preconditioners. The numerical 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 also 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, Technical Report No. 364-03, Department of Computer Science, University of Kentucky, Lexington, KY, 2003.
- [12] C. Nicholson, Diffusion and related transport mechanism in brain tissue, *Reports on Progress in Physics*, 64:815-884, 2001.
- [13] C. Nicholson and E. Syková, Extracellular space structure revealed by diffusion analysis, *Trends in Neurosciences*, 21(5):207-215, 1998.

- [14] National Institutes of Health (NIH), Diffusion Tensor Magnetic Resonance Imaging, *NIH news*,12, 2000.
- [15] C. Poupon, J. F. Mangin, C. A. Clark, V. Frouin, J. Régis, D. Le Bihan, and I. Bloch, Towards inference of human brain connectivity from MR diffusion tensor data, *Medical Image Analysis*, 5:1-15, 2001.
- [16] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA, 1996.
- [17] The SUNDIALS software package web site: <http://acts.nersc.gov/sundials/>.
- [18] 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.
- [19] I. Vorisek and E. Sykova, Evolution of anisotropic diffusion in the developing rat corpus callosum, *J. Neurophysiol.*, 78:912-919, 1997.
- [20] K. Wang and J. Zhang, MSP: A class of parallel multistep successive sparse approximate inverse preconditioning strategies, *SIAM J. Sci. Comput.*, 24:1141-1156, 2003.
- [21] 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.
- [22] C. F. Westin, S. E. Maier, B. Khidir, P. Everett, F. A. Jolesz, and R. Kikinis, Image processing for diffusion tensor magnetic resonance imaging, *Proceedings of Second International Conference on Medical Image Computing and Computer-assisted Interventions (MICCAI)*, 441-452, Cambridge, England, July 1999.