Massively Parallel and Distributed Visualization of Neuronal Fibers in Diffusion Tensor MRI Enabled by Logistical Computing and Internetworking

Micah Beck[1], Jian Huang[1], Yong Zheng[1], Jean-Patrick Gelas[1], Terry Moore[1], Nathan Fout[2] and Zhaohua Ding[3]

[1]Department of Computer Science, The University of Tennessee, Knoxville, TN

[2]Department of Computer Science, University of California, Davis, CA

[3]Department of Radiology, Vanderbilt University, Nashville, TN

Abstract

Successful visualization of neuronal connectivity in Diffusion Tensor MRI is a necessary tool for neuroscience research. Here, we describe a method that visualizes neuronal fibers by leveraging Bayes rule. Like many other DT-MRI visualization algorithms, significant computing resources are necessary for our software tool to be useful for routine medical research.

Since most medical research institutions do not have in house large-scale parallel computers, it would be beneficial to leverage the computing resources available in the Grid. Using Logistical Computing and Internetworking (LoCI) tools, we built a prototype distributed visualization system that: (i) is easy to use, (ii) does not require scheduling, reservation or authorization and (iii) has sufficient performance to complete a job in a reasonable amount of time. With a fault-tolerant method of parallel job scheduling, we have consistently obtained close to 80% parallel utilization using up to 60 heterogeneous processors distributed across US and Canada.

Keywords: DT-MRI, fiber tracking, conditional probability and Bayes rule, distributed computing, dynamic job scheduling.

1. Introduction

Understanding the structure and function of the human brain has long been a fundamental goal of neuroscience research. Recent advances in medical imaging, in particular, diffusion tensor MRI (DT-MRI) [4] has emerged as potentially powerful tools for the exploration of brain structure by providing clues regarding neuronal connectivity between functionally related cortical regions. Although still on a coarse resolution, DT-MRI is considered revolutionary in that such neuronal orientation information cannot be detected in other imaging modalities.

Discovery of neuronal patterns within the brain by fiber tracking in DT-MRI datasets has been widely studied in the field by many researchers. The resulting visualization would sometimes contain neuronal fibers amounting to tens of thousands to accurately describe the intricate brain structures. To address problems caused by signal noise and partial volume effects (PVE), sophisticated procedures beyond the conventional streamline reconstruction are necessary. In this paper, we describe a scheme of fiber reconstruction leveraging Bayes rule on conditional probability, for which continuous space probability distribution functions (pdf) must be discretized and evaluated at high angular resolutions. A number of pioneering DT-MRI visualization approaches rely on reducing a diffusion tensor field to a vector field of major eigenvectors. Doing so could cause significant error during fiber tracking in brain areas where the major eigenvector is not unique, that is in areas where the tensor is deprolated (i.e. the ellipsoid represented by the tensor matrix is not prolate, or "cigar-shaped", but oblate, "disc-shaped" or spherical). PVE stems from imaging resolutions insufficient to capture regions where nerve fibers branch or cross. Traditional streamline tracking algorithms could be misled by such areas in a way that results in significant deviations in the constructed neuronal fibers. Our visualization algorithm addresses signal noise and PVE by treating local diffusion tensors as true probability distribution functions (pdf) and dynamically follow the underlying track guided by varying local conditional probability. As a result, like many other alternative DT-MRI visualization methods, much computational resource is required.

However, most medical research institutions do not maintain high performance parallel computers in house. It carries much potential impact to provide large-scale visualization capabilities in a scalable manner that leverages the great amount of free resources that exist in the Grid. In addition, their research also demands a high level of interactivity in three respects: it needs to be (i) easy of use, (ii) available on demand without reservation or scheduling and (iii) it needs to deliver results quickly. These requirements motivate a need for a scalable distributed computing framework. While several large-scale computing alternatives exist, the infrastructure provided by Logistical Computing and Internetworking, enables our system to be easily deployed on the world's academic research networks. Internet. With this deployment and using a dynamic and fault tolerant method of job scheduling, we have consistently obtained close to linear speedups using more than 60 heterogeneous processors on NFU-enabled nodes spread across the North American continent [19], without authorization, reservation or scheduling any of those processors. Medical researchers at Vanderbilt Medical School currently use our system for their daily research.

The remainder of the paper describes our approach and the results we achieved. In section 2, we first describe the background of DT-MRI visualization by fiber tracking

and the Logistical Networking technology we use. In Sections 3, we present the details of our visualization algorithm and the NFU enabled distributed computing infrastructure, respectively. A description of our preliminary results and the relevant issues on scalability are in Sections 4 and 5 respectively. We then conclude in Section 6.

2. Background

2.1 DT-MRI Visualization

Water molecules undergo random motion commonly referred to as diffusion, which can be described by a symmetric rank-2 tensor matrix of size 3×3. A common geometric representation of this matrix is an ellipsoid, where the surface of the ellipsoid marks the probability of diffusion in every direction for a given point in space. The size, shape and orientation of the ellipsoid give a complete description of the tensor. The tensor ellipsoid may take on the shape of a sphere, a disc or spindle, depending on the relative magnitude of the tensor eigenvalues. DT-MRI measures the direction-dependency motion of water molecules [4], producing a set of coefficients which are then used to calculate a diffusion tensor. As the direction-dependency of water diffusion is closely related to structural anisotropy of the media, DT-MRI can be used to probe structural features of living tissue. Although still on a coarse spatial scale, DT-MRI is deemed to be revolutionary in that the structural information it provides is invisible to other imaging modalities.

To track linear structural features (e.g. neuronal fibers) in a tensor field, many methods replace each tensor matrix with the major eigenvector. This reduction converts the tensor field to a vector field in which neuronal fibers are constructed as hyper-streamlines or simply streamlines [5, 14, 21]. To address the unavoidable signal noises, these methods employ filtering or regularization [30], which may unfortunately smooth out local details during the process. Poupon et. al [21] regularized the diffusion field by estimating the true tensor matrix on each voxel. They accomplished this by evaluating the conditional probability in the local neighborhood. A heuristic rule-based streamline tracing is then used to trace fibers in the regularized dataset. To address the issue of PVE, tensorlines were developed based on advection-diffusion of particles [28]. In this method, each particle not only tracks local orientation but also considers orientations in its neighborhood and tracing history. While effective in handling isotropic areas by averaging of optional directions to follow, the tensorline approach requires proper weights to be supplied. An evident drawback is that tracking results could be heavily dependent on the values of the weighting parameters. It is not clear what weight values represent "real" fibers. Recently, also by reducing tensors to major eigenvectors, a method is proposed to first segment regions of the most consistent distribution of major eigenvectors with Hough Transform [29]. In the segmented regions, a fiber can be traced from a user-chosen seed and optimized with a cost function penalizing change of fiber directions. This method addresses some aspects of signal noise but not problems caused by PVE.

As the tensor matrix really represents a probability distribution function (pdf), another class of methods incorporates this idea in tracking. For instance, in [13] tensor matrices are regarded as a pdf which is modulated by Bayes rule on conditional probability. Tracing a fiber from a given seed is equivalent to finding a path through the 3D volume that attains a maximal likelihood, which requires a 3D integration process. Such integration is difficult to compute analytically. A Monte Carlo method is then developed for this purpose. As a more efficient alternative, Poupon et. al [21] approximate the global optimization process with a regularization stage that estimates the true tensor matrix free from noise on each voxel. Their regularization also draws upon conditional probability. After regularization, the conventional numerical integration is then used to trace fibers in the dataset. Besides these known techniques, there are other ways to leverage the idea of conditional probability for neuronal fiber visualization, such as our method in this paper.

2.2 Logistical Computing and Internetworking (LoCI)

The computational demands of the DT-MRI visualization by fiber tracking imposes a delay of up to eight hours in reconstructing 10,000 fibers from a 64x64x18 DT-MRI scan when implemented sequentially on a single Pentium 4 processor of 2.4 GHz. The need for faster response times in clinical settings, together with a desire to offload such computation intensive jobs from personal computing resources suggests the use of a local parallel computing platform, such as a departmental Linux cluster or shared memory processor, or the use of a remote computing center via Grid middleware.

Parallel implementations of DT-MRI over MPI have been tried and very good performance results have been obtained. However, the availability of locally owned solutions is limited by parallel processing resources, which may be in high demand, and so may be unavailable when needed. In fact, many medical institutions do not maintain such computing facility on campus for both technical and policy reasons. At the same time, the usability of some Grid solutions has been found to be limited by the need to integrate the application and the user into a global Grid middleware framework, and to support that middleware framework on the client systems [17]. While such approaches are technically feasible, in our experience they have not provided either the flexibility of use or scalability of deployment that was desired for this application environment. Experience with such systems by author Ding, a radiologist, has shown that analysis and visualization algorithms have parameters that are tricky to

set correctly in general. In addition, there are often several alternative algorithms, such as smoothing, registration and segmentation that may apply well to a particular group of datasets. It would be ideal to be able to spontaneously refine, start or cancel test runs. Therefore, radiologists would desire a computing service that (i) is easy to use, (ii) does not require restrictive scheduling, reservation or authorization and (iii) have sufficient performance.

We built a prototype massively parallel and distributed system to visualize neuronal fibers with these goals in mind. Our system leverages Logistical Computing and Internetworking (LoCI), which takes an approach to the sharing of storage and processing resources that is modeled on the Internet approach to the sharing of network resources. The strategy enables the free sharing of such resources within a community of users, employing a service model that is well adapted to multiplexing resources among competing applications for the sake of scalability, and yet is highly generic, in order to support the greatest flexibility in its use. Like the Internet, the resulting system is not tailored in advance to provide the very highest levels of performance for any single application, and it places on the endpoint the burden of managing and utilizing primitive, generic network services. But, for these same reasons, it is highly scalable and deployable, and these attributes are required in many data intensive medical and scientific applications, such as the kind of diagnostic visualization we are studying here.

2.2.1 Logistical Networking

To achieve the kind of global deployment scalability some high-end applications require for data management, *Logistical Networking* (*LN*) uses a highly generic, best effort storage service, called the Internet Backplane Protocol (IBP), the design of which is shaped by analogy with the design of IP in order to produce a common storage service with similar characteristics. Though it has been implemented as an overlay on TCP/IP (Figure 1), it represents the foundational layer of the ”network storage stack” [6, 20]. Just as IP datagram service is a more abstract service based on link-layer packet delivery, so IBP is a more abstract service based on blocks of data (on disk, memory, tape or other media) that are managed as “byte arrays.” By masking the details of the local disk storage — fixed block size, different failure modes, local addressing schemes — this byte array abstraction allows a uniform IBP model to be applied to storage resources generally. The use of IP networking to access IBP storage resources creates a globally accessible storage service.

As the case of IP shows, however, in order to scale globally the service guarantees that IBP offers must be weakened, i.e. it must present a “best effort” storage service. First and foremost, this means that, by default, IBP storage allocations are time limited. When the lease on an IBP allocation expires, the storage resource can be reused and all data structures associated with it can be deleted. Additionally an IBP allocation can be refused by a storage resource in response to over-allocation, much as routers can drop packets; such “admission decisions” can be based on both size and duration. Forcing time limits puts transience into storage allocation, giving it some of the fluidity of datagram delivery. More importantly, it makes network storage far more sharable, and therefore easier to scale up.

The semantics of IBP storage allocation also assume that an IBP storage resource can be transiently unavailable. Since the user of remote storage resources depends on so many uncontrolled, remote variables, it may be necessary to assume that storage can be permanently lost. In all cases such weak semantics mean that the level of service must be characterized statistically.

IBP storage resources are managed by “storage intermediate nodes”, or “depots,” which are servers on which clients perform remote storage operations. IBP client calls fall into three different groups [20]: **`IBP_allocate`** and **`IBP_manage`** for storage management; **`IBP_store`**, **`IBP_load`**, **`IBP_copy`**, and **`IBP_mcopy`** for data transfer; and **`IBP_status`**, for depot management. The **`IBP_allocate`** function is the most important operation. It is used to allocate a byte array at an IBP depot, specifying the size, duration and other attributes. A chief design feature is the use of capabilities (cryptographically secure passwords) [15]. A successful **`IBP_allocate`** call returns a set of three capabilities for the allocated byte array — one for reading, one for writing, and one for management — that may be passed from client to client, requiring no registration from or notification to the depot.

Basic middleware tools for using this network storage infrastructure have already been developed and are freely available (http://loci.cs.utk.edu). The *Logistical Runtime System* (*LoRS*) consists of a set of tools and associated APIs that allows users to draw on a pool of depots in order to enable the implementation of files and other storage abstractions with a wide range of characteristics, such as large size (through fragmentation), fast access (through caching), and reliability (through replication). LoRS tools also implement some transport layer services such as checksums, encryption, compression, and erasure codes, all of which is implemented at the end-points. The *Logistical Backbone (L-Bone)* is the LN resource discovery service [7]; it maintains information about IBP depots such as hostname, port, storage availability, proximity between depots, etc. Users can also query the L-Bone to determine

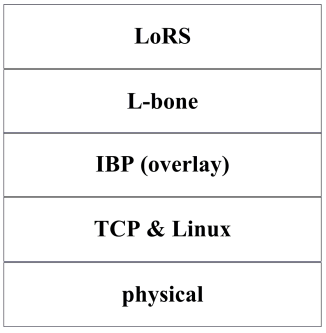


Figure 1. Overlay implementation of the network storage stack.

proximity between depots and the user to improve upload or download performance. This middleware can be used in an open, wide area testbed of IBP depots (currently over 33TB), which today encompasses more than 300 public nodes in 21 countries. This deployment of LN technology provides a rich platform for experimentation with the new approach to scalable network computation described here.

2.2.2 The Network Functional Unit (NFU)

To add processing power to the storage intermediate nodes in a logistical network while retaining deployment scalability, the LoCI approach must supply an abstraction of processing resources local to the depot (i.e. time-sliced operating system execution services) that satisfies the twin goals of providing a generic but sharable computing service, while at the same time leaving that service as exposed as possible to serve the broadest range of purposes of application developers [22]. Following the familiar pattern of other network stacks, all higher layer functions would then be built up on top of these primitive services. We call our new abstraction of the depot's local processing resources the *Network Functional Unit (NFU)* [11], and implement it as an orthogonal extension to the functionality of IBP.

In order to achieve its design goals, the abstraction embodied in the NFU must mask enough of the particularities of the local layer processing resources, (e.g. fixed time slice, differing failure modes, local architecture and operating system) to enable lightweight allocations of those resources to be made by any participant in the network. As in the case of IBP storage [10], the strategy for implementing this requirement is to mirror the IP paradigm. Just as IP is a more abstract service based on link-layer datagram delivery, IBP's Network Functional Unit is a more abstract service based on computational fragments (e.g. OS time slices) that are managed as "operations." The independence of NFU operations from the attributes of the particular local layer is established by working through the same features of resource aggregation, fault detection, and global addressing. Table 1 displays the results for the NFU side by side with IBP.

The name "Network Functional Unit" was chosen to fit the pattern established by other components of the LN infrastructure, which expresses an underlying vision of the network as a computing platform with exposed resources that are externally scheduled by endpoints. The archetype here is a more conventional computing network: the system bus of a single computer (historically implemented as a backplane bus), which provides a uniform fabric for storing and moving data. This was the analysis that was invoked in the naming of the fundamental protocol for data transfer and storage the "Internet Backplane Protocol." In extending that analogy to include computation, we looked for that component of a computer that has no part in data transfer or storage, serving only to transform data placed within its reach. The Arithmetic Logic Unit (ALU) seemed a good model, with its input and output latches serving as its only interfaces to the larger system. For this reason, we have named the component of an IBP depot that transforms data stored at that depot the *Network Functional Unit*.

	IBP (Storage)	**NFU (Processing)**
Resource Aggregation	Aggregation of access layer blocks masks the fixed block size	Aggregation of execution layer time slices masks the fixed slice size
Fault Detection	Fault detection with a simple failure model (faulty byte arrays are discarded) masks the variety of different failure modes	Fault detection with a simple failure model (faulty operations terminate with unknown state for write-accessible storage) masks the variety of different failure modes
Global Addressing	A uniform capability name space masks the difference between local layer storage addressing schemes.	A uniform operation namespace, masks the difference between local layer processing resources

Table 1. A comparison between IBP and NFU.

However, since NFU service, like the IBP core storage service, is implemented as an *overlay* on top of TCP/IP, this gives rise to a serious problem. The chronic vulnerability of IP networks and LN to Denial of Service (DoS) attacks, on bandwidth and storage resources respectively, will apply equally to the NFU's computational resources. Another problem is that the classic definition of a time slice processing service is based on execution on a local processor, so it includes strong semantics that are difficult to implement scalably in the wide area network.

Following that line of analysis, and the model of IBP, we address both of these issues by weakening the semantics of compute resource allocation in the NFU. Most importantly, NFU allocations are time limited, and the time limits established by local depot policy makes the compute allocations that occur on them transient. But all the semantics of NFU operations are weaker in ways that model computation accessed over the network. In all cases the weak semantics mean that the level of service must be characterized statistically.

As illustrated in Figure 2, we can identify a logical progression of functionality in intermediate nodes: the router forwards datagrams, exercising control over movement in the spatial dimension by choosing between output buffers. The depot adds control over movement in the temporal dimension by enabling the storage of data in an IBP allocation as it passes through. Finally, the NFU is implemented as a module, added to an IBP storage depot,

that transforms stored data. If we consider spatial direction, time and value to be coordinates of a single space, then the state of any data item is a point in this vector space, and the progression is one of increasing simultaneous control over multiple dimensions.

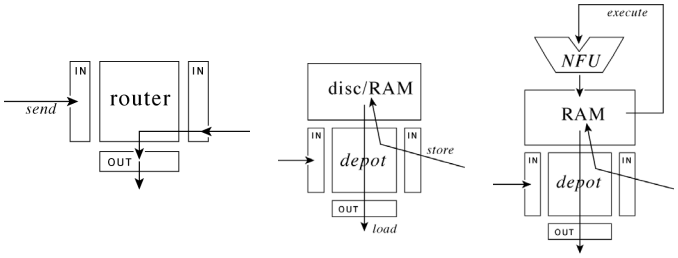


Figure 2: Intermediate nodes to manage bandwidth (IP router), storage (IBP Depot), and computation (NFU-enabled Depot)

Since a depot may model either disk or RAM storage resources, some NFU operations may apply only to data stored in RAM, while others may also apply to data stored on disk. Restricting an operation to data held in RAM forces any necessary movement between disk and RAM to be explicitly directed by the end-point using IBP, just as in data movement between depots. The NFU extends the IBP protocol and API by implementing a single additional operation, `NFU_op`:

```
NFU_op(depot, port, operation,
cap_0,... cap_8)
```

The details of the current NFU API can be found in the reference manual [12]. The asynchronous nature of the pipelined API described in the technical report on our experimental results [18] adds considerably to the complexity of the actual API calls. The `NFU_op` is used to invoke an operation at an IBP depot, specified by the IP address and port it binds to. The operation is specified as an integer argument, whose meaning is set by a global registry of operation numbers. The arguments to an operation consist of a list of capabilities (cryptographically secure names) [15] for storage allocations on the same depot where the operation is being performed. Thus, there is no implicit network communication performed by a given depot in responding to an `NFU_op` call. The capabilities specified in this call can enable reading or writing, and the limitations of each are reflected in the allowed use of the underlying storage as input or output parameters. The number and type of each capability are part of the signature of the operation, specified at the time the operation number is registered. Any violation of this type information (for instance, passing a read capability for an output parameter) may cause a runtime error, but it is not checked by the implementation of `NFU_op` at the client side. Aliasing between capabilities is also not detected, since in some operations it is desirable and an interface that declares when to allow it would add unnecessary complexity.

3. Our System

3.1 Fiber Segment Probabilities

As alluded to in the previous discussion, fiber tracking methods need to address signal noise and partial volume effects (PVE). Both may cause reconstructed fibers to deviate from the underlying physical neuronal fiber. We approach is based on obtaining an accurate estimate of the probability of a given fiber segment connecting two voxels. This probability is certainly a function of the local tensor, but how should it be computed? We could directly use the local tensor, which provides a pdf capable of answering the question, but PVE still cause errors in deprolated regions.

In order to address this problem we developed a framework based on conditional probabilities. The idea behind conditional probabilities is that of updating an estimate of the current probability based on past information. More specifically, conditional probability allows us to better calculate the probability of an event D_i occurring given the fact that another event D_k has occurred. If we further consider the events D_k and D_i as members of an event space U of size n containing many events D, then the conditional probability $P(D_i|D_k)$ for any D_k and D_i in U can be computed using Bayes Rule:

$$P(D_i|D_k) = \frac{P(D_i)P(D_k|D_i)}{\sum_{j=1}^{n} P(D_j)P(D_k|D_j)} \qquad (1)$$

The analogy to computing fiber segment probabilities can be found by letting the event space U be all possible directions of fiber propagation. The event D_i is the fiber following direction i, and thus the term $P(D_i|\ D_k)$ is the probability of a fiber taking the outgoing direction i given that it came from direction k. The term $P(D_i)$ is the unconditional probability of a fiber following direction i and can be taken directly from the pdf given by the local tensor.

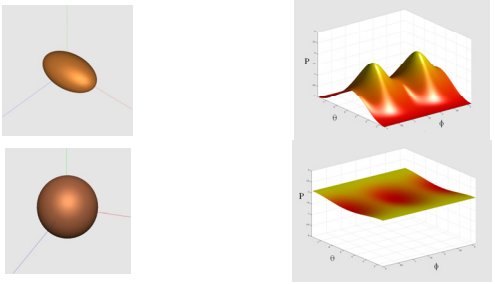


Figure 3. Two different tensors represented as ellipsoids (left) and pdfs (right).

A convenient visualization of this probability profile can be obtained by plotting probability as a function of the two spherical angles θ and ϕ. This creates a surface representing the probability as a function of direction, as shown in Figure 3.

The term $P(D_k|D_i)$ is the probability of a fiber entering from direction k, given that it leaves in direction i. Intuitively this term is related to the concept of bending energy; that is, the energy needed to bend the fiber from k to i. The selection of $P(D_k|D_i)$ can therefore be made based on fiber modeling. If we assume fibers to be somewhat stiff, then $P(D_k|D_i)$ will have a maximum probability along the incoming direction (i.e. no bending) and decrease as the angle widens. The profile of this fall-off will determine the stiffness of the fibers. In practice we choose a smooth profile like a Gaussian or elevated cosine, which allows us to use a single parameter to vary the profile. The cosine-shaped profile is favored due to the simplicity of its computation (a single dot product).

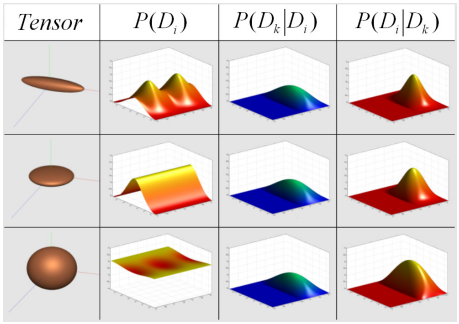


Figure 4. Application of Bayes Rule to different types of tensors using a cosine profile for $P(D_k|D_i)$. The input direction is the z-axis (red line in leftmost column).

Once the selection of the profile of $P(D_k|D_i)$ has been made then the conditional probabilities $P(D_i|D_k)$ can be computed for each outgoing direction D_i given an incoming direction Dk. Figure 4 shows the resulting pdfs for the three types of tensors (prolate, oblate, and spherical) using a cosine profile for $P(D_k|D_i)$. Notice the ambiguity in the case of oblate and spherical tensors is resolved.

With probability analysis, neuronal fiber reconstruction is computed in the following steps. First, a set of starting seeds are chosen by human experts, for instance in the corpus callosum where consistently oriented neuronal fibers are known to exist and PVE effects are low. The initial direction to follow is the major eigenvector direction. Using a given step size, such as 0.1 in voxel size, the next sample point on each fiber is determined. Second, on the new sample point, the local tensor direction is interpolated. From our experiments, high quality interpolation kernels are necessary. We use a 3D Gaussian kernel of 2.0 radii, again measured in voxel sizes along the three coordinate axes. The resulting tensor is then discretized into a 2D absolution probability table (APT) indexed by the two spherical angles θ and ϕ. The resolution that we use is 40 by 40. Third, a second 40 by 40 conditional probability table (CPT) is created to model bending energy using the profile of elevated cosine. In our implementation, we compute this conditional probability table by computing the dot product of the incoming direction of fiber and the direction that each entry in the 40 by 40 table represents according to the corresponding combination of θ and ϕ. Forth, a element-by-element product of APT and CPT is computed, from which we search for the maximum resulting probability. The corresponding direction is the next direction to follow. This procedure is repeated until either (i) the fiber abruptly bends more than 90 degrees or (ii) the new sample point is outside white matter. We discover the boundary of white matter using an anisotropy index proposed by [26].

$$A_\sigma = \frac{1}{\sqrt{6}\overline{D}} \times \sqrt{\sum_{i=x,y,z} \left(D_{ii} - \overline{D}\right)^2 + 2\left(D_{xy}{}^2 + D_{xz}{}^2 + D_{yz}{}^2\right)} \quad (1)$$

where $\overline{D} = \left(D_{xx} + D_{yy} + D_{zz}\right)/3$. A_σ is based on the variance of the average diffusivity in all directions. , as in: $A_\sigma = \sigma(T)/(\sqrt{2}\overline{D})$. $\sigma(T)$ is the standard deviation of the diffusion coefficients measured by the MRI and then encoded as the tensor matrix, T, which symmetric:

$$\begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{xy} & D_{yy} & D_{yz} \\ D_{xz} & D_{yz} & D_{zz} \end{pmatrix}.$$

A 0.25 value of A_σ was found to best describe the surface of the white matter (WM) via repeated experiments. We then construct the WM volume by simulating particles diffusing from corpus callosum within WM and stopping on boundaries marked by 0.25 A_σ. In result, a volumetric mask of WM can be obtained. In Figure 5, the left is a photo of a real WM from a human subject. We were able to obtain a volumetric WM volume with similar visual appearance (Fig. 5 right) as the real photo.

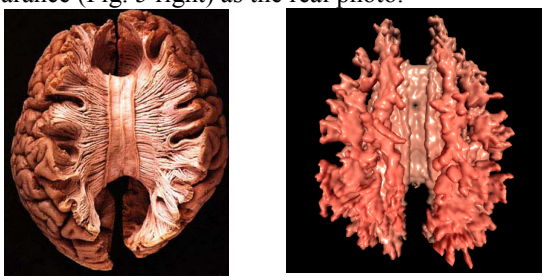


Figure 5. Picture taken of actual WM in the brain (left). Reconstructed WM using A_σ (right).

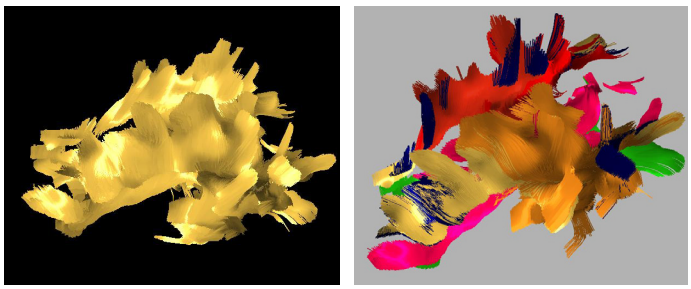


Figure 6. (left) 50,000 reconstructed nerve fibers and 200 bundles among these 50,000 fibers (shown with a different color per bundle)

Using our method of Bayesian fiber reconstruction, very smooth nerve fibers can be obtained from DT-MRI datasets. In Figure 6, we show one sample set of results. From a 64x64x18 DT-MRI dataset, we have tracked 50,000 neuronal fibers (Figure 6 left) and also bundled them according to geometric similarity (Figure 6 right).

To summarize our neuronal fiber visualization algorithm, we provide a rather detailed pseudo code in Figure 7. All operations we use are generic and can be implemented by using conventional numeric libraries.

```
fiber_reconstruction (vector3 current_position,
                      vector3 in_direction,
                      float   stepsize)
{
  matrix  APT, CPT; // 40 by 40
  matrix  tensor;   //  3 by  3
  matrix3 dtmri;    // 3D volume of DT-MRI
  matrix3 WM;       // 3D volume of WM mask

  //40 by 40 matrix with each element being a.
  //vector. This stores the direction
  //corresponding to each combination of
  //spherical angles used in APT and CPT
  matrix  directions;

  vector3 new_direction;
  vector3 new_position;
  int     maximum_id;

  tensor = interpolate(dtmri, current_position);

  //check whether in white matter
  if !lookup(WM, current_position) return;

  APT = discretize(tensor);
  CPT = batch_dotproduct(directions,
                         input_direction);

  APT = elementwise_ matrix_multiply(APT, CPT);
  maximum_id = maximum(APT);
  new_direction = convert(maximum_id);

  //check whether making a sharp turn
  if (dotproduct(in_direction,new_direction)<= 0)
       return;

  //4th order runga cutta numeric integration
  new_position = runga_cutta4(current_position,
                 new_direction, stepsize);

  store(new_position);
  fiber_reconstruction(new_position,
                       new_direction,
                       stepsize);
}
```

Figure 7: The pseudo code of Bayesian fiber reconstruction.

When the NFU library contains numeric libraries like LAPACK and LINPACK [2], we can implement fiber reconstruction in NFU with little efforts. All that is needed is to implement NFU operations, such as interpolate, discretize, anisotropy, dotproduct, batch_dotproduct, elementwise matrix multiply, etc., with the numeric libraries.

Currently due to lack of time, we haven't completed the full NFU operation library to implement fiber reconstruction at such low level granularity, although we expect to finish this work in a few weeks. Now, we created a coarse level NFU operation called 'fiber_reconstruction' directly from our production visualization code. We do not expect the obtained test results in performances to vary between the coarse and fine level NFU operations.

3.2 Parallelization and Job Scheduling

In our system, we put seeds in the DT-MRI volume in areas specified by users. In a typical run, a user could position seeds in an area corresponding to the center of corpus callosum, by moving a 3D bounding box in 3D within the white matter surface that we have constructed.

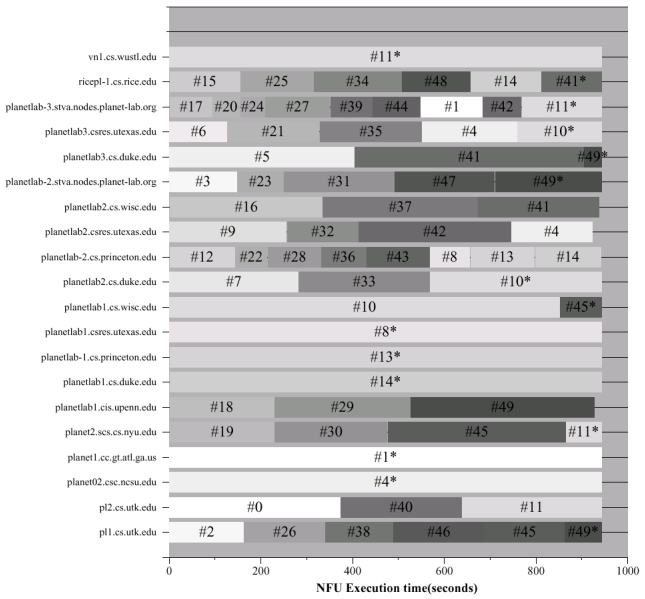
Our system then automatically generates a user controllable number of seeds, each corresponds to a neuronal fiber. The entire set of seeds is then partitioned into a linear list of job assignments containing equal number of seeds. After a one-time data deployment stage during which the DT-MRI dataset is replicated on distributed processors, we start to assign jobs to processors from the job list for parallel processing.

In neuronal fiber reconstruction, little inter-fiber dependency exists. Therefore, parallelizing the algorithm would seemingly be straightforward. However, due to the arbitrary length of each individual fiber, the workload to compute neuronal fibers varies between partitions. It is difficult to predict the exact workload beforehand and dynamic load balancing is indispensable. In addition, our job-scheduling scheme must be fairly robust for a rather open environment of distributed computing. Since our targeted computation infrastructure is completely distributed and has no mechanisms for reservation or scheduling, we can make few assumptions in regard to reliability or quality of service. A processor may be down or busy. In addition, processors as well as the corresponding network connections from the client will be of differing speeds. Hence, in general we would like to dynamically discover fast processors and assign as many jobs to them as possible and at the same time avoid being stalled by slow or faulty processors.

We devised three mechanisms for heterogeneous parallel processing: (i) a priority queue of measured speed of processors, (ii) a priority queue of unfinished jobs and (iii) a buffered job assignment scheme.

The priority queue for processors is indexed by the number of jobs that each processor has completed. This measurement roughly reflects the performance of each processor. The more jobs a processor has finished, the higher priority this processor has. The second priority queue, which maintains unfinished jobs, is indexed by the number of processors currently working on each job. This priority helps to rank unfinished jobs in "likelihood to quickly finish". The fewer processors are competing to finish an unfinished job, the higher the priority. Initially, all

processors are assigned equally the same priority, '0'. During parallel computing, after a job is completed and returned to the client, the corresponding processor's priority is incremented. At first, the priority of each job is '0", signifying that no processors are working on it.



50 partitions with 20 processors

Figure 8. A typical case of job assignment on 20 processors using 50 partitions. The domain name of each processor used is listed to the left. The horizontal axis depicts the duration of the run in seconds. Each number in the graph is an ID of a job partition. All numbers with "*" behind it represent jobs that has never been finished.

When parallel computing first starts, the scheduler randomly assigns one job to each processor, incrementing the corresponding priorities in both priority queues. When a processor completes its job and returns the results, the completed job is removed from job queue. Then, there are three possible scenarios: (1) there are still unassigned jobs, (2) there are still unfinished jobs and (3) all jobs are finished. Case 3 signifies the completion of the entire process. In Case 1, one would directly assign any one of the unassigned jobs to the empty processor, since there are no differences among those jobs. However, Case 2 would require some special consideration. In theory, we would like to always use our fastest processor for the unfinished job that is least likely to finish quickly. If we always immediately assign a new job to each processor that has just become empty, it would be hard to implement this concept. We opt to use a buffered job assignment scheme. Specifically, after the initial round of job assignments, we process any further job assignments every three seconds. That is, we wait three seconds for processors to finish jobs. In each three-second span, a few processors may get done. Among them, we assign the highest priority unfinished job to the processor that has become empty during the past three seconds. We then assign the unfinished job with the second highest priority to the second highest priority empty processor. In this manner, we inherently address fault tolerance in the same framework.

For the sake of resource control, we limit the maximum number of processors that can work on the same job simultaneously. In our test, this number is set to 4.

We illustrate a typical result of job scheduling in Figure 8. Here we show the domain name of each machine in the Planet Lab [1, 19] that we used via LoCI tools. The shaded areas in the graph shows the duration of each numbered job being computed on various processors. The time as measured in seconds is recorded on the horizontal axis. All job IDs' with an "*" behind are jobs that were never finished on each respective processor during the run. In this test run on 20 processors with 50 partitions, we can very easily notice highly varying performances in the distributed wide area system. We chose this relatively small run for ease of illustration.

4. Results

In this section, we present the results of our prototype system for massively parallel and distributed visualization of DT-MRI datasets. All testing was performed using NFU-enabled IBP depots in PlanetLab [1, 19]. Up to 80 processors were used. The hardware configurations of these processors vary. The hardware configuration ranges from 1.3GHz to 2.66 GHz Pentium 4 processors, with 300MB to 2GB RAM. For details of PlanetLab hardware configuration, please refer to [1]. The DT-MRI testing dataset is of a clinical resolution of 64x64x18 of 2.6 MB in storage. The white matter data that is constructed requires roughly 2.4 MB storage. In total, each processor needs 5MB of data to proceed. In all tests, we reconstruct a total of 9368 fibers from a typical area of interest specified by user in the center of corpus callosum.

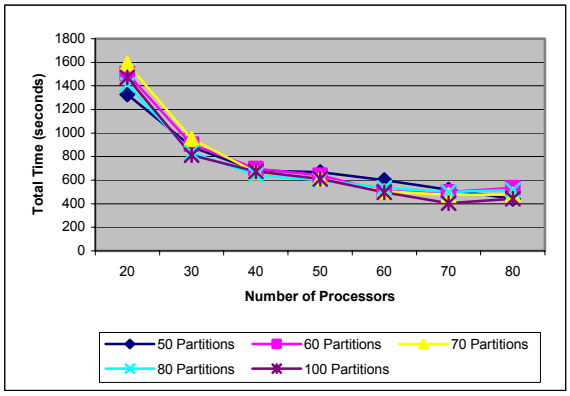


Figure 9. The total running time for jobs using 50 to 100 partitions run on 20 to 80 processors. Each data point in this graph represents an average of 5 test runs.

In Figure 9, we plot the wall clock time for test runs with combinations using 50 to 100 job partitions and using 10 to 80 processors. With each particular combination, 5 tests were run and only the average running time is shown. Due to the large dynamic range of measured time, the

difference among curves needs to be closely examined. The overall speedup effects are very obvious no matter how many partitions are used. In general, we need a sufficient number of partitions (usually as many as the number of processors being used) to obtain good load balancing, but increasing the number of partitions much beyond that would not further lead to significant improvements in performance. For instance, we would expect to use 200 partitions, had there been more than 150 processors for us to use for good load balancing.

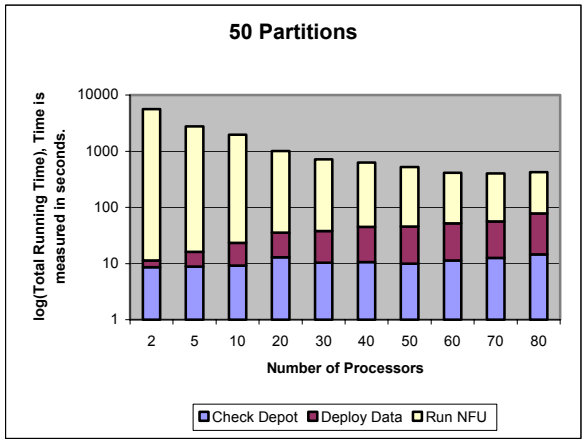


Figure 10. The total running time (shown in logarithm scale) for a job to complete with 50 partitions using 2 to 80 processors, respectively. In the total running time, three operations are included: "run NFU", "Deploy Data" and "Check Depot".

We further analyze the overheads in NFU calls in Figure 10, which illustrates the total running time (in logarithm scale to show differences in orders of magnitudes) for a job to complete with 50 partitions using 2 to 80 processors, respectively. In the total running time, three operations are included: "run NFU", "Deploy Data" and "Check Depot". The latter two operations are both executed only at the beginning of the parallel run, while "run NFU" is the fiber reconstruction function that is repeatedly invoked. When more processors are used than the total number of partitions, in a way we use several processors to compute a common partition and compete for speed. From Figure 10 we can see that "Check Depot" runs in a small constant time and "Deploy Data" requires time that increases with the number of processors. Our current implementation of "Deploy Data" uses a straightforward unicast approach. The effects on our overall results are minimal, however, because the "run NFU" operations consume at least an order of magnitude more time than either of the other two services.

To better analyze the full overhead caused by using load-balancing issues we show results in Figure 11 and 12. We measure a total processor time as the total running time of the parallel run times the number of processors used. Figure 11 shows a roughly linear increase in total processor time (P) versus the total number of processors (n). That is we can model P with the following formula: $P = a \times n + T$, where T corresponds to the total work of the run independent of the number of processors used and a is a measure of parallel processor inefficiency. A linear regression analysis estimates T to be 16,817 seconds and a as 180.5 seconds. Most of such overhead is due to inefficiency caused by load imbalance. To this end, a system with perfect linear speedup and zero inefficiency would produce a horizontal line in Figure 11.

However, this is not the most interesting way to examine the result in Figure 11. $P = a \times n + T$ can be rewritten as $D = P/n = a + T/n$, where D is simply the total running time or duration. After a certain point, such as 60 processors, the speedup or parallel utilization one obtains in parallel becomes less significant as shown in Figure 12.

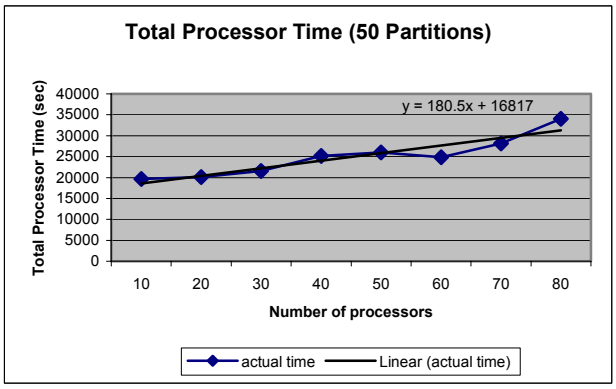


Figure 11. (Curve) The total processor time vs. the number of processors used, from 10 to 80. Here, 50 partitions are used. The vertical axis is the result of multiplying the total running time for the run and the number of processors used. (Straight line) The result of a linear regression analysis on the curve obtained.

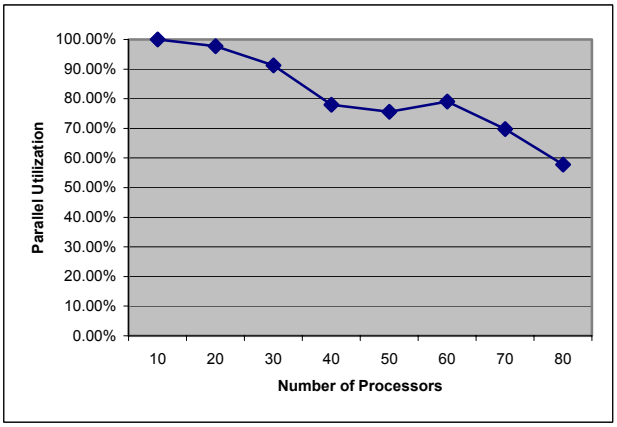


Figure 12. Parallel Utilization measured up to 80 processors using 50 partitions in this particular run.

Parallel processor utilization is a common metric used to measure the efficiency of the parallel algorithm. To compute parallel utilization, the total running time using one processor is often used as the reference, R. The parallel utilization with x processors, whose total running time is P,

is computed as: $R/P/_x$. However, when heterogeneous processors are used, a different reference is necessary. To choose any single processor as the benchmark would be misleading. In our experiments, we use an ensemble of 10 processors as the benchmark and compute parallel utilization using $R/P/_{x/10}$. In this case R is the total running time using 10 processors. The resulting curve is shown in Figure 12. It is obvious that around 80% parallel utilization can be obtained with as many as 60 processors. Any additional processors beyond 60 may still help to obtain better absolute performance via competition; however, the scalability already starts to drop. In fact, this result should be expected since parallel processors won't be kept busy unless there are sufficient job partitions to go around.

5. Issues in Scalable Computing

The NFU can be thought of as a primitive mechanism for invoking remote processing, and in that sense it is similar to distributed computing mechanisms such as Network Enabled Servers used in the Grid Computing environments [3, 16, 17, 25, 27], or EveryWare [23]. However, a primary design goal of the NFU is to define a common service that is as scalable as possible, so that the limits of scalability are determined by the particular functions defined, not by the service framework itself. To that end, the NFU requires an administrative interface to install procedure calls (referred to as an installation interface) different from the interface to invoke a procedure call (referred to as an invocation interface). This is due to the fact that installing code is a heavyweight operation requiring a level of trust between client and depot that imposes scalability constraints. Currently, adding an NFU operation to a depot through the installation interface is a manual administrative procedure that involves installing an operation executable image in a depot directory or re-linking of the depot process with an operation library. By separating the installation and invocation interfaces, we make the invocation interface lightweight and achieve maximum scalability.

Besides separating interfaces, in order to achieve maximum scalability there are three additional aspects of scalability to address. First, an NFU operation must place limits on the size of the allocations that it operates on and the amount of processor resources that it can use. For instance, if an operation is defined to operate on allocations that are larger than 4GB in size, it may not be efficiently implemented on a depot running a 32-bit operating system. Second, an operation cannot run for a very long time. Each call represents a large allocation of processor resources, and so it makes it more difficult for a depot to fairly serve a large community of users, maintaining a minimum level of service to each.

The third aspect of scalability stems from the notion that the functions implemented on a depot must be generic, which is less intuitive outside of the networking community. The widespread deployment of an operation that supports a specific application requires a large community of depot owners to make an administrative decision to deploy that operation on their depot, which they may be loathe to do if it does not serve a sufficiently large segment of the user community that share the depot. Each installation of an additional NFU operation takes some resources on the depot, whether the operation is invoked or not, potentially diminishes the locality of caches and memory management on the network node, and adds to the complexity of depot management if updates to the operation implementation are issued. After all, each operation added to the depot increases the number of trusted parties. Due to design or programming errors or malicious actions, there would be more chances for the depot to misbehave or be compromised. For these reasons, scalability is improved by constructing a minimally necessary set of NFU operations comprised of generic functions that are basic and thus more likely to be reliable. This argues for the NFU operations to be very primitive at the level of ALU and FPU operations, and for the IBP protocol to be used for composing large numbers of such fine-grained operations. The arguments against this approach are those of complexity at the endpoints and performance.

The idea that maximal scalability is achieved by the minimization of trust between client and depot brings up the issue of whether an depot can be used in the implementation of processing as part of a reliable server by endpoints that do not trust it at all. If such a service were highly scalable but could not be used to create a reliable service, then it might be useless. In that case, we might need to accept some requirements of trust, and lower our expectations of scalability.

5.1 Generality Mapping Fiber Reconstruction of NFU

The NFU operation that we used in our experiments is called `fiber_reconstruction`, and it is quite specific to the DT-MRI application. According to our analysis, this specificity will have a negative impact on the scalability of its deployment in a public network, so we are interested in expressing it in terms of more primitive, generic operations.

If the NFU library contains numeric libraries like LAPACK [2] and LINPACK, we can implement fiber reconstruction by invoking the more generic operations in these libraries, and composing these operations using the IBP protocol, to correspond to the pseudo code given in Figure 7. The operations required include interpolation, discretization, anisotropy, dotproduct, and matrix multiply.

Due to lack of time, we haven't completed the full NFU operation library required to implement fiber reconstruction at such low level granularity, although we expect to finish this work and present performance results in the full version of the paper. Because of the fact that the code is straight line, there are no complex control

dependences that must be resolved by the client, as can be seen by inspection of the pseudo code given in Figure 7. The IBP protocol is also evolving to allow some control dependences to be resolved without requiring the intervention of the client [11].

5.2 Data Distribution

One of the limitations of scalability that can be seen in our experimental results is due to the need to distribute the entire MRI dataset to every node participating in the fiber tracing computation. While the size of the dataset, 6 MB, is not large enough to inhibit its implementation on any depots that currently comprise of the L-Bone, the need to move the data to every depot in order to process it strains the data movement capabilities of the network that connects them.

Part of the problem is simply the implementation strategy. The current implementation uses a simple iterated unicast strategy to distribute the data, which results in the saturation of the networking capacity of the client or its connection to the backbone, and requires time that increases linearly with the number of target depot. A more scalable approach is to use some form of hierarchical distribution or overlay multicast [8], which can operate in time logarithmic in the number of target depots, or even faster if the multicast tree is built to conform to the topology of the network.

We have implemented a hierarchical distribution scheme, although it does not conform to the topology of the network. Early results, illustrated in figure Y, show greatly reduced data distribution times. Due to a lack of time, we have not been able to rerun all of our experiments using this new data distribution mechanism, but will do so for the final version of the paper.

As the number of processors used is scaled beyond 100, it becomes necessary to use some nodes that are not well connected to the international research backbone networks, and for which a transfer of even 6 MB from the University of Tennessee might represent an impediment to contributing to a computation that may last for less than a minute. For this reason, it may be valuable to express the computation as a composition of operations on a partition of an MRI, requiring an additional step for these partitions to be merged. The result would be more scalable in its ability to be used to take advantage of depots that are constrained in bandwidth, processor and even storage resources.

5.3 Security

Distributed computing using untrusted depots is a harder problem that moving data across or storing data on untrusted depots because of the need to process the data while it resides on depots. In the case of data movement and storage, end-to-end data encryption can be used to make data inaccessible to intruders who might tap wires or even to malicious depot operators [24]. In the case of processing, even data that is transferred to the depot in encrypted form is typically decrypted before processing takes place, making the computation vulnerable to being compromised at the depot.

There are several approaches to this problem, which is particularly acute when dealing with medical data. Part of the solution may lie in approaches to obscuring the data by dividing it and distributing it, so that a private dataset cannot be recovered from the public network without monitoring a large amount of traffic or many geographically distributed networks. Another approach is to obscure the data without using strong encryption, for instance by intermixing it with dummy data, or by overlaying it with known, random patterns that have a known effect on the results that can be factored back out after the computation has been completed. It may even be possible to compute on strongly encrypted data in some cases.

5.4 Correctness

In addition to problems of security, the use of untrusted depots presents a problem of verifying correctness of the processing performed at those depots [9]. In the case of data movement and storage, redundancy in the form of end-to-end checksums is used to detect errors probabilistically. In the case of computation, redundancy can be introduced in a number of ways, including performing computations multiple times on independent depots or at the client and comparing results. In some cases, the client can verify a computation more efficiently by using the results than by recomputing from the inputs. It is possible to enhance the efficiency of verification in some cases by modifying NFU operations to return more of the details of their computations. Working in a mode where the results of remote computations are untrusted and checked requires a thorough rethinking of the assumptions that pervade much of software engineering concerning the degree to which checks of correctness should be implemented at runtime, which may have application even in cases where calls are performed in local or trusted environments, but errors in logic may exist.

6. Conclusions and Future Work

In this paper we have presented an algorithm to visualize DT-MRI datasets by Bayesian fiber reconstruction. We address effects of signal noise and PVE by studying the conditional probabilities using Bayes rule and treating each tensor matrix as a true pdf. Possible directions of further research on DT-MRI visualization include acceleration algorithms, integrated visualization of even more imaging modalities and animal experiments to verify our results.

We have built a practically useful system for routine radiological experimentation which can make use of available computational resources without authentication or reservation. This system is built on a scalable infrastructure

which adopts a unique architectural approach that promises to scale beyond the boundaries of conventional computational facilities. We have undertaken a research program that addresses many of the hard technical problems posed by this architecture, as described in Section 5, as well as other work not presented here which address the goals of efficiency and high performance in the operation of individual nodes [18].

7. References

1. *http://www.planet-lab.org/.*
2. Anderson, E., et al., *LAPACK Users' Guide*. Third ed. 1999, Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM).
3. Arnold, D., H. Casanova, and J. Dongarra, *Innovations of the NetSolve Grid Computing System.* Concurrency: Practice and Experience, 2002. **14**(13-15): p. 1457-1479.
4. Basser, P.J., J. Mattiello, and D. Le Bihan, *MR diffusion tensor spectroscopy and imaging.* Biophysics Journal, 1994. **66**: p. 259-267.
5. Basser, P.J., et al., *In vivo fiber tractography using DT-MRI Data.* Magn Reson Med, 2000. **44**(625-632).
6. Bassi, A., et al. *The Internet Backplane Protocol: A Study in Resource Sharing*. in *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002)*. 2002 (to appear). Berlin, Germany: IEEE.
7. Bassi, A., et al., *The Logistical Backbone: Scalable Infrastructure for Global Data Grids*, in *Asian Computing Science Conference 2002*. 2002, Springer Verlag: Hanoi, Vietnam.
8. Beck, M., et al. *An Exposed Approach to Reliable Multicast in Heterogeneous Logistical Networks*. in *Workshop on Grids and Advanced Networks (GAN03)*. 2003. Tokyo, Japan.
9. Beck, M., et al. *Scalable, Trustworthy Network Computing Using Untrusted Intermediaries: A Position Paper*. in *DOE/NSF Workshop on New Directions in Cyber-Security in Large-Scale Networks: Deployment Obstacles*. 2003. Lansdowne, Virginia, March, 2003.
10. Beck, M., T. Moore, and J.S. Plank. *An End-to-end Approach to Globally Scalable Network Storage*. in *ACM Sigcomm 2002*. 2002. Pittsburgh, PA: Association of Computing Machinery.
11. Beck, M., T. Moore, and J.S. Plank. *An End-to-End Approach to Globally Scalable Programmable Networking*. in *Future Directions in Network Architecture (FDNA-03), an ACM SIGCOMM 2003 Workshop*. 2003. Karlsruhe, DE: ACM.
12. Beck, M., et al., *Internet Backplane Protocol API 1.4*. 2004 (In preparation), Department of Computer Science, University of Tennessee: Knoxville, TN.
13. Bjornemo, M., et al., *Regularized Stochastic White Matter Tractography Using Diffusion Tensor MRI.* 2002.
14. Delmarcelle, T. and L. Hesselink, *Visualizing second-order tensor fields with hyper streamlines.* IEEE Computer Graphicsand Applications, 1993: p. 25-33.
15. Dennis, J. and E.V. Horn, *Programming semantics for multiprogrammed computations.* Communications of the ACM, 1966. **9**(3): p. 143-155.
16. Epema, D.H.J., et al., *A worldwide flock of condors : Load sharing among workstation clusters.* Journal on Future Generations of Computer Systems, 1996. **12**.
17. Foster, I. and C. Kesselman, eds. *The Grid: Blueprint for a New Computing Infrastructure*. 1999, Morgan Kaufman Publishers. 677.
18. Liu, H., *Experiments in Pipelining the Internet Backplane Protocol*. 2004 (In preparation), Departhment of Computer Science, University of Tennessee: Knoxville, TN.
19. Peterson, L., et al., *A Blueprint for Introducing Disruptive Technology into the Internet*, in *Proceedings of ACM HotNets-I Workshop*. 2002: Princeton, New Jersey, USA.
20. Plank, J.S., et al., *Managing Data Storage in the Network.* IEEE Internet Computing, 2001. **5**(5): p. 50-58.
21. Poupon, C., et al., *Regularization of diffusion-based direction maps for the tracking of brain white matter fascicles.* NeuroImage, 2000. **12**: p. 184-195.
22. Reed, D.P., J.H. Saltzer, and D.D. Clark, *Comment on Active Networking and End-to-End Arguments.* IEEE Network, 1998. **12**(3): p. 69-71.
23. Rich Wolski, et al., *Writing Programs that Run EveryWare on the Computational Grid.* IEEE Transactions on Parallel and Distributed Systems, 2001. **12**(10).
24. Sarmenta, L.F.G., *Sabotage-tolerance mechanisms for volunteer computing systems.* Future Generation Computer Systems, 2002. **18**(4): p. 561-572.
25. Sekiguchi, S., et al. *Ninf : Network based Information Library for Globally High Performance Computing*. in *Proc. of Parallel Object-Oriented Methods and Applications (POOMA)*. 1996. Santa Fe, NM.
26. Shimony, J., et al., *Quantitative Diffusion-Tensor Anisotropy Brain MR Imaging: Normative Human Data and Anatomic Analysis.* Radiology, 1999. **212**: p. 770-784.
27. Siegel, J., *Corba 3 Fundamentals and Programming*. 2nd ed. 2000: John Wiley & Sons. 928.
28. Weinstein, D., G. Kindlmann, and E. Lundberg. *Tensorlines: Advection-Diffusion based Propagation through Diffusion Tensor Fields*. in *Proc. of IEEE Visualization Conference*. 1999. San Francisco, CA.
29. Weng, J.-C., et al. *A Global Approach for Non-invasive Axonal Fiber Tracking on Diffusion Tensor Magnetic Resonance Image*. in *Proc ISMRM Annual Meeting*. 2002. Hawaii.
30. Zhukov, L. and A. Barr. *Oriented Tensor Reconstruction: Tracing Neural Pathways from Diffusion Tensor MRI*. in *Proceedings IEEE Visualization*. 2002. Boston, MA.