Proceedings of the 28th IEEE
EMBS Annual International Conference
New York City, USA, Aug 30-Sept 3, 2006

ACCELERATION OF FIBER TRACKING IN DTI TRACTOGRAPHY BY RECONFIGURABLE COMPUTER HARDWARE

Manbir Singh[†], Aditya Kwatra[‡], Chi-Wah Wong[†], Viktor Prasanna[‡]
[†]Depts. of Radiology and Biomedical Engineering, [‡]Dept. of EE-Systems
University of Southern California, Los Angeles, CA 90089

Abstract— **Diffusion Tensor Imaging (DTI) tractography is a computationally intensive procedure. The most time consuming operation is the tracking of fibers from every voxel in the scanned volume. Fiber tracking can be accelerated significantly by use of reconfigurable hardware, such as Field Programmable Gate Arrays (FPGAs), which can track fibers at very high speed by exploiting the flexibility, parallelism and high on-chip bandwidth. Such acceleration has the potential to lead to real-time tractography. In this paper we isolate key kernels within the tracking step and through a simulation study, analyze a specific FPGA architecture comprising deeply pipelined kernel chains running in parallel. Our results suggest that the FPGA based computer architecture could achieve a two orders of magnitude speed-up in the fiber-tracking algorithm over an optimized C-code.**

I. Introduction

Processing DTI tractography data is computationally very expensive. It requires large amount of memory storage, memory bandwidth, computing power, and tens of minutes of data processing to accomplish 3D whole-brain tractography. Instead of relying on seeds within specific regions only, here we assume that tractography would be conducted over the entire brain, with one or multiple seeds per voxel to quantify whole-brain connectivity. An example of 3D tractography is presented in Figure 1, where tracts from a particular brain slice are shown [1]. These tracts provide vital information about brain connectivity and can be used clinically to investigate brain damage or abnormal function. For example, in brain trauma, DTI tractography reveals new and unique information on loss in brain connectivity caused by axonal injury that is difficult to detect by regular MRI or CT scans, and it is critical that such information be available to the clinician as rapidly as possible, preferably in real time so that diagnosis and treatment could begin on the spot. Unfortunately, due to the computational complexity required for DTI 3D tractography, its implementation in real-time is challenging at the present time.

Tractography consists of many sub-tasks out of which fiber tracking is the most time consuming task. A commonly used single-tensor per voxel streamline tractography approach relies on tensor estimation and linear propagation with step size typically around $0.1 - 0.2$mm [2]. The small step size in conjunction with tensor interpolation yields sub-voxel sampling that reduces the probability of duplicate tracts when every voxel is addressed. Propagation termination is based on predetermined thresholds related to the degree of diffusion

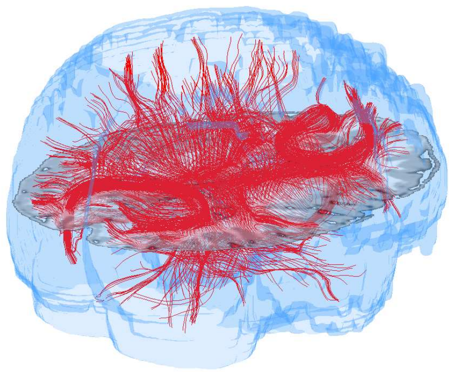
Figure 1: An example of 3D tractography with tracts originated from one slice out of 28 slices. Similar tracts are found for the remaining 27 slices.

anisotropy or deflection angle between steps [2]. The entire fiber tracking procedure can be further divided into three computationally intensive *kernels* – tensor interpolation, tensor diagonalization, and anisotropy calculation [2]. These calculations are performed in each step to determine the new step direction and whether the propagation should continue. These kernels are particularly suited to implementation by FPGAs, which have been an attractive option for computationally intensive applications. Recent research has shown that FPGAs can achieve superior performance compared to general purpose processors [3], [4]. The flexibility offered by reconfigurable FPGAs can be exploited to maximize the achievable parallelism. Very high on-chip bandwidth provided by these devices helps to remove bottle-necks faced by general purpose processors.

We have investigated a system level architecture to accelerate the process of fiber tracking. The three kernels mentioned above were implemented using VHDL in Xilinx ISE 7.1 [5] and simulated in ModelSim 6.0a [6]. Acceleration is achieved in two stages: 1. Pipelining of the kernel chain; 2. Multiple kernel chains running in parallel. Using the IEEE 64-bit compliant floating point cores developed by our group [7], with the kernels running at 100MHz, we were able to achieve a speed-up on the order of $100\times$ over a current optimized software implementation using C on a uni-processor. Our approach, implementation details and performance comparison for these kernels pertinent to tractography are presented in this paper.

Algorithm 1: **Fiber tracking algorithm**

```
Input: Diffusion Tensor Matrix for each voxel of the given
        volume of size $x \times y \times z$, and set of seed points
Output: Set of fibers tracked from each seed-point.
Cartesian Cube: A cube in Cartesian space joining the
        centers of adjacent voxels
Seed point: A point where the fiber originates, located at
        the center of Cartesian cube
Assumption: Each cartesian cube contains one seed point.
for $i = 1$...Total number of seed points
{
    do /* Loop to traverse fiber steps */
    {
      Tensor Interpolation: Weighted diffusion tensor
      (matrix $D$), based on the distance of the seed-point
      from the 8 vertices of the cartesian cube, is calculated.
      Tensor Diagonalization: Eigen-values and eigen-
      vectors are calculated by Singular Value Decomposit-
      ion of $D$
      Trajectory Propagation: Direction of next step is
      chosen from the eigen-vector corresponding to the
      largest eigen-value. Direction is taken along the
      angle smaller to previous step.
      Anisotropy Calculation: Fractional Anisotropy is
      calculated.
    } while (FA $\geq$ specified value and inter-step angle
    $\leq 45$ degrees)
    /* Fiber tracking continues till the condition on FA is */
    /* violated */
} /* Fibers tracked from each seed point */
```

II. Method

DTI tractography has certain properties that can be exploited to accelerate its implementation on FPGAs. On analysis of the DTI tractography algorithm, we have made certain observations which are listed in Section II-A. We also identified three computation intensive kernels on profiling the C-code for DTI tractography.

A. Fiber Tracking Algorithm

The fiber tracking algorithm is discussed in Algorithm 1. The following observations are noted:

1) Tracking of two fibers are independent of each other.
2) Tracking a fiber is a sequential process. Next step is chosen at the end of processing of the initial step.
3) At step size of 0.2mm, for example, and voxel size of 2mm×2mm×4mm, a track navigates through a voxel with at most 25 steps. Tracking starts from the center of the cartesian cube and proceeds in two directions opposite to each other.
4) The fibers might converge together or diverge out into two branches. Such situation is taken into consideration by starting from multiple seed points within the set of images, which increases the processing time in direct proportion to the number of seeds per voxel.

Table I: Profiling of the C-code for DTI tractography

Kernels	Execution Time (secs)
Tensor Interpolation	18.668
Tensor Diagonalisation	128.002
Anisotropy Calculation	5.934

We profiled the C based software implementation for DTI tractography on an Intel Xeon Processor [8] running at 800 MHz with 4GB RAM. The results of profiling the C-code are shown in Table I. The three time consuming kernels thus identified were found to be: Tensor Interpolation, Tensor Diagonalization, and Anisotropy Calculation. The kernel implementations are discussed in Section II-C and the performance analysis is provided in Section III.

B. Accelerating DTI Tractography

The computationally intensive kernels are identified and cascaded together to form a kernel chain. The deeply pipelined kernel chain accepts input data every clock cycle. Thus acceleration is achieved in two stages:

1) Pipelining: Deep pipelining of the kernel chain may be exploited to compute the fiber tracks for independent seed-points in parallel till the pipeline gets saturated. As each fiber tracking is independent of the other, the computation of kernels for different fibers can be interleaved in the pipeline. Each kernel chain should provide on the order of $30\times$ speed-up.
2) Parallelization: The flexibility of the FPGA devices can be exploited to maximize the parallelization. Many such kernel chains can be implemented in parallel and can be used to track independent fibers. Having 10 such kernel chains running in parallel woudl provide an overall speed-up on the order of $300\times$.

C. Implementation Details

We designed and implemented the pipelined kernels using VHDL in Xlinx ISE 7.1 [5] and simulated them in ModelSim 6.0 [6]. The architecture was designed to exploit as much parallelism and pipelining as possible to maximize the speed-up. The number of function calls to these kernels are of the order of 10^7, as found on profiling the C-code.

1) Tensor Interpolation: Tensor interpolation is used to calculate the six components of the weighted diffusion tensor symmetric matrix at every step in fiber tracking. The weights are based on the distance of the step point from vertices of the cartesian cube. The basic computation involved in this kernel is as follows:

$$D_{xy} = \Sigma_v \;\; D^v_{xy} \times (X_v - X_0) \times (Y_v - Y_0) \times (Z_v - Z_0)$$

where
X_v, Y_v, Z_v are the x, y, z coordinates of the vertex v
X_0, Y_0, Z_0 are the x, y, z coordinates of the seed point
Σ_v - summation over all 8 vertices

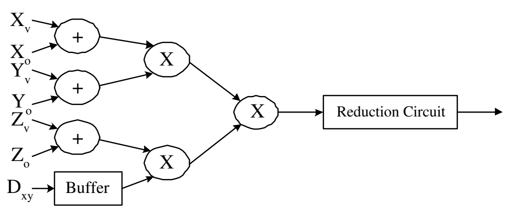

Figure 2: Tensor Interpolation Kernel

The architecture of the implemented kernel is shown in Figure 2. The reduction circuit [9] is used as the fully pipelined high throughput accumulator. The kernel was implemented using the fixed point cores provided by Xilinx [5] and after synthesis and simulation was found to run at maximum frequency of 186.2 MHz.

2) Tensor Diagonalization: The symmetric tensor matrix is a 3×3 matrix, which is diagonalized to calculate the eigen-values and their corresponding eigen-vectors. Steps in fiber tracking proceed along the direction of the longest eigen-vector. The eigen-values and their corresponding eigen-vectors are calculated using the Jacobi based Singular Value Decomposition (SVD) [10] of a symmetric tensor matrix. The following computation is involved for the construction of the orthogonal matrices required for SVD:

$$\theta_{pq} = \frac{D_{qq}-D_{pp}}{2D_{pq}},\ t = \frac{sqn(\theta)}{\theta+\sqrt{\theta^2+1}},\ c = \frac{1}{\sqrt{t^2+1}},\ s = tc,$$
$$D_{new} = Q \times D_{old} \times Q^T$$

where given p=0, q=1

$$Q_{01} = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The orthogonal matrix is multiplied with the diffusion tensor matrix as above to complete one iteration of the SVD. The algorithm keeps iterating for all values of $p = 0$ to 2 and $q = p+1$ to 2. These iterations continue till the off-diagonal elements of the diffusion tensor matrix D, tend to 0. The SVD of a matrix is known to converge in O(n^2) iterations. Thus the maximum iterations for SVD of a 3×3 takes at most 9 iterations. The implementation details of our kernel are shown in Figure 3. The kernel architecture consists of the pipelined implementation of one iteration. All consecutive iterations are interleaved to reuse the same hardware. The kernel was implemented using the fixed point cores provided by Xilinx [5] and after synthesis and simulation was found to run at maximum frequency of 111.3 MHz.

3) Anisotropy Calculation: The anisotropies are calculated to verify the termination of the fibers. The process of tracking a fiber terminates if the fractional anisotropy at a particular step falls below a specified value. The input to this kernel are the eigen-values λ_1, λ_2, and λ_3 obtained from the tensor diagonlization kernel. The kernel involves the following calculations:

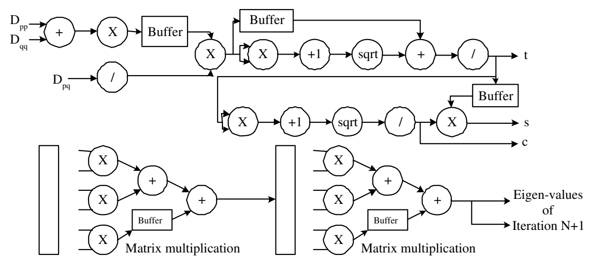

Figure 3: Tensor Diagonalization Kernel

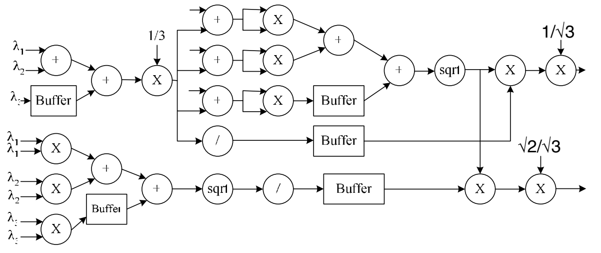

Figure 4: Anisotropy Calculation Kernel

$$\text{FA} = \frac{\sqrt{3\times[(\lambda_1-\overline{\lambda})^2+(\lambda_1-\overline{\lambda})^2+(\lambda_1-\overline{\lambda})^2}}{\sqrt{2\times(\lambda_1^2\times\lambda_2^2\times\lambda_3^2)}}$$

where $\overline{\lambda} = \frac{\lambda_1+\lambda_2+\lambda_3}{3}$

The implementation details of the kernel are shown in Figure 4. The kernel was implemented using the fixed point cores provided by Xilinx [5] and after synthesis and simulation was found to run at maximum frequency of 162.8 MHz.

III. Performance Analysis

Preliminary implementations of the kernels to form a kernel chain were done using the fixed point IP cores available in Xilinx [5]. Based on the maximum frequency for the kernel implementations, we have made a conservative estimate of using 100 MHz as the frequency of the kernels together as a kernel chain. In this section we provide the performance analysis of the kernels when using the IEEE 64 bit floating point cores. The speed-up achieved is mainly due to the deep pipeline and independence among different fiber tracts. Thus input data-set from different fiber tracts can be provided continuously to the pipeline. We have compared our implementation performance with the profiled C-code. The profiling results are shown in Table I. As obtained

Table II: Floating point core details

Floating-point cores	Pipeline stages	Maximum Frequency (MHz)
Adder	14	170
Multiplier	11	170
Divider	58	140
Square Root	55	169

from profiling, each kernel is executed on the order of 10^7 times. Thus the initial combined latency of the kernel chain is neglected when compared to the throughput of the pipeline. Even if the entire latency of the pipeline were about 5000 clock cycles, initial time taken to fill up the pipeline at 100 MHz clock frequency would be just 0.5 μseconds. We discuss each kernel below and analyze the performance improvement.

A. *Deeply Pipelined Kernel Chain*

1) Tensor Interpolation: This kernel uses a fully pipelined tree structure and a reduction circuit. Set of 7, 64-bit floating point inputs are provided every clock cycle and the output is available after every 8 cycles. This delay of 8 cycles is due to the accumulation of the elements of the diffusion tensor matrix from eight vertices of the cartesian cube. The throughput of providing an output after every 8 cycles results in the time taken to process 10^7 such operations to be equal to 0.8 seconds. The same operation on profiling the C-code takes about 16 seconds.

2) Tensor Diagonalization: This kernel receives the input as the six entries of the symmetric diffusion tensor matrix from the tensor interpolation kernel. Tensor interpolation kernel takes 48 cycles to produce 6 elements of the symmetric diffusion tensor matrix. This kernel provides a very high throughput. It is capable of producing an output every clock cycle, but due to the input being delayed by 48 cycles, each output is produced after every 48 cycles. Thus at this rate, both the tensor interpolation kernel and the tensor diagonalization kernel together will be able to process the input data in 4.8 seconds as compared to 2 minutes and 27 seconds of the profiled C-code.

3) Anisotropy Calculations: This fully pipelined kernel is capable of providing an output every clock cycle. The output is delayed by 48 cycles from tensor diagonalization kernel, thus this latency is reflected in the output of the third kernel as well. The three kernels together as a kernel chain produce an output every 48 cycles. Thus total time taken to compute 10^7 such computations of the three kernels together at 100 MHz is about 4.8 seconds as compared to 2 minutes and 32 seconds on profiling the C-code.

B. *Parallel Implementation of Multiple Kernel Chains*

Multiple kernel chains operating on different fiber tracts in parallel can be implemented at the system level. There is no shared computation or memory access restrictions as the tracking of fibers are independent to each other. This will provide a further speed-up on the order of the number of kernel chains implemented in parallel. The maximum memory-FPGA bandwidth allowed is about 8 GBytes/sec, as provided by the existing FPGA modules, e.g. BenDATA-WS FPGA module provided by Nallatech [11]. The input required by the tensor interpolation kernel is one diffusion tensor element every clock cycle. The other inputs can be locally available, if the size of the cartesian cube is known.

Each kernel chain requires one 64-bit floating point element every clock cycle, thus requiring a bandwidth of 0.8 GBytes/sec. At this rate we can support 10 kernel chains in parallel. Bounded by the available area, even if we are able to implement 5 kernel chains, we would be able to complete the computation of the three kernels in 0.96 seconds, a speed-up of 155×.

Table III: Performance Comparison with C-code

Kernels	Time Taken (secs)	
	C-code	FPGA
Tensor Interpolation	18.668	0.8
Tensor Interpolation + Diagonalization	146.67	4.8
Tensor Interpolation + Diagonalization + Anisotropy Calculation	152.604	4.8

IV. Conclusion

The pipelined kernel chain was able to complete the required computation in 4.8 seconds as compared to 2 minutes and 33 seconds obtained by profiling the C-code, thus providing a speed-up of 31×. The results have been summarized in Table III. Further speed-up is also possible when multiple such kernel chains are implemented in parallel. Implementation of 10 such kernel chains, for example, would give a further speed-up of 10×, thus completing the computation of the three kernels in about 0.48 seconds. Reconfigurable computing would be particularly useful when multiple seeds are used. With 10 seeds per voxel, for example, the fiber tracking step could be completed in less than 5 seconds, whereas an optimized C-code would require about 25 minutes. Thus reconfigurable computing has the potential to enable almost real-time tractography in the near future.

References

[1] Singh M, Hwang D, Sungkarat W, and Veera K, "Evaluation of MRI DTI-tractography by tract-length histogram," *Progress in Biomedical Optics and Imaging: Physiology, Function and Structure from Medical Images*, vol. 5746, no. 1, pp. 138–147, 2005.

[2] S. Mori and P. C. M van Zijl, "Fibre tracking: principles and strategies - a technical review," *NMR Biomed*, vol. 15, pp. 468–480, 2002.

[3] Gokul Govindu, Seonil Choi, Viktor K. Prasanna, Vikash Daga, Sridhar Gangadharpalli, and V. Sridhar, "A high-performance and energy-efficient architecture for floating-point based LU decomposition on FPGAs," in *Proceedings of the 11th Reconfigurable Architectures Workshop*, April 2004.

[4] Ling Zhuo and Viktor K. Prasanna, "Scalable and modular algorithms for floating-point matrix multiplication on FPGAs," in *Proceedings of the International Parallel and Distributed Processing Symposium*, New Mexico, April 2004.

[5] Xilinx, Inc., ," `http://www.xilinx.com`.

[6] Mentor Graphics ModelSim, ," `http://www.model.com`.

[7] Gokul Govindu, Ronald Scrofano, and Viktor K.Prasanna, "A library of parameterizable floating-point cores forFPGAs and their application to scientific computing," in *Proceedings of the International Conference onEngineering Reconfigurable Systems and Algorithms*, Toomas Plaks, Ed., June 2005.

[8] Intel Corporation, ," `http://www.intel.com`.

[9] Ling Zhuo, Gerald R. Morris, and Viktor K. Prasanna, "Designing scalable FPGA-based reduction circuits using pipelined floating-point cores," in *Proceedings of the 12th Reconfigurable ArchitecturesWorkshop*, Denver, CO, April 2005.

[10] William H. Press, Saul A. Teukolsky, Willian T. Vetterling, and Brian P. Flannery, *Numerical Recipes in C*, Cambridge University Press, second edition, 1999.

[11] Nallatech, ," `http://www.nallatech.com`.