Accelerating DTI Tractography using FPGAs

Aditya Kwatra[1], Viktor Prasanna[1], Manbir Singh[2*]

[1]University of Southern California
Dept. of Electrical Engineering
Los Angeles, CA 90089. USA
{kwatra, prasanna}@usc.edu

[2]University of Southern California
Dept. of Radiology and Biomedical Engineering
Los Angeles, CA 90089. USA
msingh@usc.edu

Abstract

Diffusion Tensor Imaging (DTI) tractography in Magnetic Resonance Imaging (MRI) is a computationally intensive procedure, requiring on the order of tens of minutes to complete tractography of the entire brain. Tractography computations can be accelerated significantly by the use of reconfigurable hardware, such as Field Programmable Gate Arrays (FPGAs). Such acceleration has the potential to lead to real-time tractography, which would greatly facilitate on-site diagnosis and acquisition of additional scans while the patient is still inside the scanner. In this paper we report the development of an FPGA based architecture to accelerate DTI tractography. We identify computationally intensive kernels and design pipelined implementations. Our performance analysis based on the developed architecture gives on the order of 100x speed-up over an optimized implementation in C of tractography on a state-of-the-art processor.

1 Introduction

Many brain imaging operations are computationally intensive as they require large amounts of memory storage, memory bandwidth, and computing power. For example, researchers at UCSD Institute for Neural Computation have designed a new way to parse EEG data and identify the individual signals coming from different areas of the brain [8]. The work enables much more comprehensive view of brain dynamics. However, it was only made possible by exploiting recent advances in mathematics and increases in computing power [9].

Diffusion tensor imaging (DTI) tractography is a relatively new approach to visualize brain connections throughout the volume of the human brain [1]. These tracts provide vital information about brain structure and function. These can be used clinically to detect brain damage or abnormal function. For example, tractography could reveal axonal damage during traumatic brain injury, frequently missed by conventional MRI or CT. An example of 3D tractography is shown in Figure 1 [15]. The state-of-the-art in DTI, however, does not allow real-time whole brain tractography, making it difficult to assess damage during a critical situation such as brain trauma. Real-time tractography would mitigate this limitation, thus accelerating the diagnosis process and selection of appropriate treatment options for the patient.

DTI tractography has been implemented using various techniques. These consist of many tasks, out of which fiber tracking is the most time consuming task. Fiber tracking can be further divided into three computationally intensive sub-tasks called kernels - tensor interpolation, tensor diagonalisation, and anisotropy calculation. These kernels implemented on reconfigurable hardware, in particular Field Programmable Gate Arrays (FPGA) can speed-up the fiber tracking to the order of few seconds.

Tracking of a fiber in DTI tractography is a sequential process [1, 7]. The direction of next step in trajectory propagation of the fiber depends on the previous step of the fiber tract. The direction of the next step can be in any possible direction in 3D space. The next step of the fiber might not be in the same image slice as the previous step. Thus, fiber tracking results in irregular memory accesses. This makes the problem far more complicated than the various image processing techniques that have been implemented in the past on FPGAs.

FPGAs have been an attractive option for computationally intensive applications. The flexibility provided by the FPGAs is exploited to maximize the

*This work was supported in part by grant NIH NIAP50AG05142.

1-4244-0054-6/06/$20.00 ©2006 IEEE

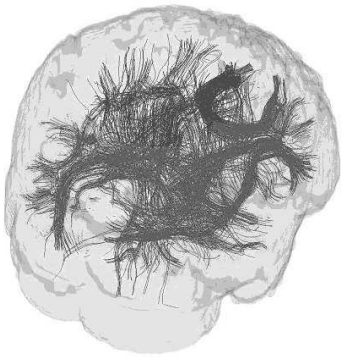

Figure 1. An image of DTI tractography of the brain

parallelism and pipelining of the implementations. Our implementations also exploit the irregular structure of the process of fiber tracking to simultaneously compute on multiple fibers parallelly as well as in a pipelined fashion.

We propose a deeply pipelined architecture for the computation intensive kernels of fiber tracking in DTI tractography. We have implemented these kernels in VHDL using Xilinx ISE 7.1 [17] and simulated them in modelSim 6.0a [10]. We propose to achieve on the order of 100x speed-up over the state-of-the-art software implementation using C on a uni-processor for DTI tractography. Further we propose an overall architecture to accelerate the process of fiber tracking. The acceleration is achieved due to: 1. Deeply pipelined architecture of the kernels linked together to form the kernel chain; 2. Multiple such kernel chains operating in parallel. Based on our preliminary implementations we provide performance analysis using IEEE 64-bit compliant floating point cores developed by our research group [3].

This paper is organized as follows. Section 2 discusses the background of using FPGAs for speeding up DTI Tractography. In Section 3 we discuss the architecture details of our deeply pipelined implementations of the kernel. In Section 4 we provide the high level architecture. Section 5 shows the performance analysis using a C based state-of-the-art software implementation on a uni-processor. We conclude in Section 6.

2 Background

2.1 Diffusion Tensor Imaging Tractography

Tractography based on MRI Diffusion Tensor Imaging (DTI) is a rapidly growing field to map axonal connections through white matter in the human brain. Either alone, or in conjunction with functional MRI, the axonal connections provide important new information to decipher brain structure and function.

The directional dependence of water-molecular diffusion (anisotropic diffusion) forms the basis of mapping axonal tracts in white matter. In general, diffusion is much faster along the fiber than orthogonal to the fiber. Thus by measuring diffusion along different directions in 3D space, it becomes possible to detect and infer the direction of axonal tracts. These axonal tracts can propagate in different directions in 3D space. Each axonal tract of the brain is tracked independent of other axonal tracts. These axonal tracts or fibers are mapped together to provide the required diffusion tensor 3D imaging of the brain. The irregularity of the process of fiber tracking and independence of fiber tracts lead us to consider the use of FPGAs to speed-up DTI tractography.

2.1.1 DTI Processing

Diffusion is a tensor and can be mapped by modifying a gradient echo pulse sequence to become sensitive to diffusion along a particular gradient direction by applying a pair of diffusion pulses (bipolar gradients) along that direction [4]. The MRI signal acquired with a diffusion sensitive pulse sequence will attenuate in proportion to diffusion along the direction of the diffusion gradients. Diffusion is conveniently modeled as an ellipsoid or rank 2 tensor matrix D (3×3 components). From symmetry considerations, this implies that a minimum of six non-collinear gradient directions would be required to estimate the diffusion tensor matrix per voxel [1]. After estimating D per voxel, tractography is commonly accomplished by a streamline tractography (SLT) approach where a seed voxel is connected to surrounding voxels in piecewise linear steps along the direction indicated by the principal eigenvector of the diffusion tensor matrix. The vector associated with the first eigenvalue of the matrix indicates the direction of fastest diffusion and is presumed parallel to the orientation of a fiber lying in the voxel [1].

The anisotropy per voxel can be expressed in many ways but usually the fractional anisotropy (FA), which is computed from the differences around the three eigen values and normalized to the 0-1 range, is commonly

Algorithm 1. Fiber tracking algorithm

Input: - p images of size $m \times n$
where p is the number of scans from the scanner
- Diffusion Tensor Matrix for each voxel
of the given p images.
Output: Set of fibers tracked from each seed-point
Cartesian cube: The centers of eight adjacent voxels
together form a cartesian cube.
Seed-point: The point where fiber tracking is initiated.
Also the center of the cartesian cube.
Num: Total number of seed points
Fiber Step: Minimum step size taken while tracking a fiber.
The Fiber Step is predefined by the user.
Fiber point: The point where the last tracked fiber
step ends.

Do for all seed-points $i = 1...Num$
{

Fetch the x, y and z coordinates of the seed point
Initiate fiber tracking on the chosen seed-point

Do
{

Tensor Interpolation: Calculate the weighted diffusion tensor matrix D based on the distance of the fiber point from the 8 vertices of the cartesian cube.

Tensor Diagonalisation: Calculate eigen values and eigen vectors using SV Decomposition of the D matrix.

Trajectory Propagation: Choose the direction of the next fiber step along the longest eigen vector Interpolate in +/- directions.

If (The selected fiber point in different cartesian cube)
Compute the x, y, and z coordinates of the vertices of the new cartesian cube.
Fetch the elements of the diffusion tensor matrix for the eight vertices of the new cartesian cube.

Compute x, y and z coordinates of the new fiber point.

Anisotropy Calculation: Calculate Fractional Anisotropy (FA) at each fiber step.

} **while**($FA \geq specified\ value$)
Fiber tracking from the chosen seed point continues till the FA falls below a specified value.

} Fiber tracking continues on the other seed points.

used as a stopping criterion to end tracts [11, 1].

The problem of fibers crossing within a voxel could mislead fiber tracking [13, 16]. An alternative approach [5] that relies on a statistical model to detect tracts has been proposed to address the problem. In order to map the connectivity of the entire brain, random curves are initiated on a bootstrap of the white matter. From certain randomly selected seed points curves are grown in both directions and elongation stops when maximum steps or the border of the mask is reached. The result is a statistical estimate of the entire brain connectivity, modeled by approximately 100,000 curves. Each of these curves are tracked independently of each other. The propagation direction calculation at each step has much more computations. For one DT-MRI dataset, it took tens of minutes to finish the fiber tracking.

2.1.2 Fiber Tracking Algorithm

Diffusion tensor imaging involves various tasks like filtering, diffusion tensor calculation, fiber tracking, image mapping, 3D visualization, etc. Out of these tasks, fiber tracking is the most computationally intensive task and requires the maximum amount of time. Different DTI tractography techniques use different fiber tracking algorithms. The process of fiber tracking can be further sub-divided into four sub-tasks: 1. Tensor Interpolation; 2. Tensor Diagonalisation; 3. Trajectory Propagation; 4. Anisotropy Calculation. An example of an algorithm for fiber tracking is given as Algorithm 1.

2.2 FPGAs for DTI Tractography

DTI tractography is a tightly coupled computation and is irregular in its memory access. The tracts can be oriented in many directions. The stored data must be accessed without access conflicts to permit effective parallelization. Many brain imaging techniques are computationally intensive, as they require large amounts of memory storage, memory bandwidth, and computing power. To address the need for high performance computing in brain imaging various resources like super-computers, cluster and grid computers, etc. have been utilized [2].

However, those technologies have several limitations: 1. they are expensive, in terms of the infrastructure investment. 2. A highspeed network is required for users to access the computing resources. 3. Maintaining such a computing environment is a tremendous task for computer scientists and engineers. 4. Since the computing power provided by those technologies is centralized or distributed over a limited number of sites,

its usage is restricted. For example, real-time imaging can only be performed where the scanner is close to the available computing power. On the other hand the maximum gate density FPGA costs a few hundred dollars [17] and can easily be used next to the scanner.

The computation intensive task of fiber tracking involves tracking of different fibers which are independent of each other. The calculations involved for the sub-tasks at every fiber step of an axonal tract is independent from any other fiber step at any other axonal tract. This property of diffusion tensor imaging tractography is exploited to develop deeply pipelined kernel architecture implementations on the FPGAs to provide the necessary speed-up to enable real time DTI tractography.

FPGAs provide the flexibility and programmability to optimize the hardware to solve a specific problem. Parallelism and pipelining are the two techniques which provide the most efficient architecture designs on the FPGA . The flexibility of the FPGA devices can be exploited to have multiple kernel chains running in parallel, thus providing further speed-up. The on-chip Block RAM and Distributed RAM available on the FPGA's are used to continuously feed the kernel chains. Reusability of data can be exploited, while different steps of the same fiber tract are being computed. Thus further reducing the FPGA to off-chip memory accesses.

3 Our Architecture

3.1 Observations based on the Algorithm

The following observations were made based on the process of fiber tracking given in Algorithm 1:

1. Tracking of two fibers is independent of each other.

2. Tracking a fiber is a sequential process. Next fiber point is chosen at the end of processing of each fiber step.

3. Fiber tracking in each cartesian cube consists of about 10 fiber steps. This value is as used in the profiled C-code for fiber tracking. As in Figure 2, shows the fiber steps taken from the center solid circle representing the seed-point.

4. The fibers might converge together or diverge out into two branches. Such a situation is taken into consideration by starting from multiple seed points within the set of images. Fiber tracts originating from these multiple seed points are independent of each other.

Table 1. Profile of the C-code for DTI tractography on a uni-processor

Profiling Results	
Total number of operations : 10^7 *Average number of fiber steps per cartesian cube* : 10	
Kernels	*Execution Time(secs)*
Tensor Interpolation	18.7
Tensor Diagonalisation	128.0
Anisotropy Calculation	5.9

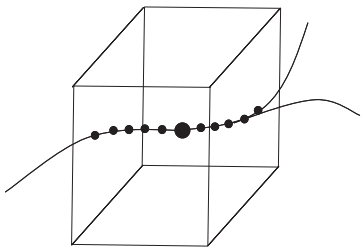

Figure 2. Fiber tracking in a cartesian cube

We profiled the C based software implementation of the DTI tractography on an Intel Xeon Processor [6] running at 800 MHz and having 4GB RAM. The results of profiling the C-code are as shown in Table 1. The three time consuming kernels thus identified are 1. Tensor Interpolation 2. Tensor Diagonalisation 3. Anisotropy Calculation.

3.2 Acceleration of Computationally Intensive Kernels

We develop deeply pipelined architectures of the computationally intensive kernels. We have exploited the property of the fiber tracts being independent of each other and the flexibility of FPGA devices to support a deeply pipelined and parallel architectures. The data-sets for multiple seed-points are provided to the deeply pipelined kernels to initiate multiple fibers to be tracked in parallel.

We implemented the three computation intensive kernels using 64 bit integer cores provided by Xilinx [17]. The implementations were done using VHDL in Xilinx ISE 7.1 [17] and simulated in modelSim 6.0 [10]. We further provide performance analysis using the 64 bit floating point cores developed by our research group [3]. We use 64 bit cores so as to provide a fair comparison with the available state-of-the-art C-code which uses double precision floating point data.

Tensor Interpolation

Tensor interpolation is used to calculate the six

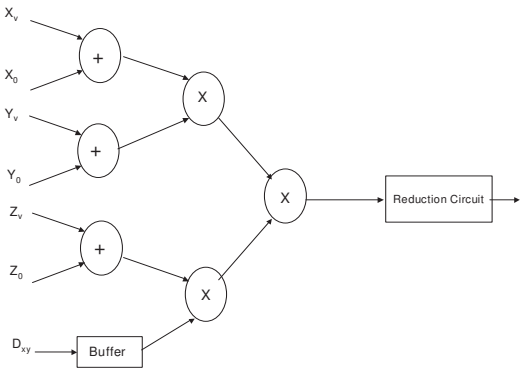

Figure 3. Tensor Interpolation Kernel

components of the 3×3 weighted diffusion tensor symmetric matrix at every step in fiber tracking. The weights are based on the distance of the fiber 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
v represnts the vertices of the cartesian cube
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

The architecture of the implemented kernel is as given in Figure 3. Seven inputs i.e. $X_v, Y_v, Z_v, X_0, Y_0, Z_0,$ *and* D_{xy} are provided to the tree structured pipeline during every clock cycle. The output from the tree structured pipeline is accumulated using a high through-put pipelined reduction circuit [18]. The tensor interpolation kernel pipeline produces an output every 8 cycles. This is due to the accumulation of the weighted diffusion tensor matrix elements from the eight vertices of the cartesian cube. Thus, to output the six elements of the symmetric diffusion tensor matrix: $D_{00}, D_{01}, D_{02}, D_{11}, D_{12}, and\ D_{22}$, the pipelined kernel provides a set of results every 48 cycles. The kernel was implemented using the integer cores provided by Xilinx [17]. After synthesis and simulation, the kernel was found to run at a maximum frequency of 186.2 MHz.

Tensor Diagonalisation

The 3×3 symmetric tensor matrix is diagonalised to calculate the eigen values and their corresponding eigen vectors. The next fiber-step in the process of fiber tracking is decided in the direction of the longest eigen vector. The eigen values and their corresponding eigen vectors are calculated by the Jacobi based Singular Value (SV) Decomposition [14] of the symmetric tensor matrix. SV Decomposition of the symmetric matrix always produces a diagonal matrix, whose elements represent their respective eigen values. SV Decomposition involves multiplication of two orthogonal matrices with the tensor diagonal matrix. The elements of the orthogonal matrix of each iteration are calculated to make a pair of the off diagonal elements of the symmetric diffusion tensor matrix tend to zero. The following computation is involved in the Tensor Diagonalisation kernel:

$$\theta_{pq} = \frac{D_{qq} - D_{pp}}{2D_{pq}}$$

$$t = \frac{sgn(\theta)}{\theta + \sqrt{\theta^2 + 1}}$$

$$c = \frac{1}{\sqrt{t^2 + 1}}$$

$$s = tc$$
$$D_{new} = Q \times D_{old} \times Q^T$$

where given *p=0, q=1*, the orthogonal matrix is constructed as follows:

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

The orthogonal matrix Q is constructed using the c and s values. The orthogonal matrix is multiplied to the diffusion tensor matrix as above to complete one iteration of the SV Decomposition. 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 implementation details of the kernel are as shown in Figure 4. The pipelined architecture of this kernel is capable of producing an output every clock cycle.

The designed pipeline of the kernel is equivalent to one iteration of the Jacobi SV Decomposition algorithm. The latency of the pipeline is a multiple of the rate at which the kernel receives the inputs from the Tensor Interpolation kernel, i.e. 48 clock cycles. The Jacobi SV Decomposition of a $n \times n$ matrix converges in n^2 cycles. Thus, the Jacobi SV Decomposition of the tensor diagonal matrix of the order 3×3 will converge in at most 9 iterations. As $48 > 9$ thus, every consecutive iteration can be fed back to the pipeline, without any possibility of data hazards. Thus in the worst case, even if all the data elements take 9 iterations to converge, we will produce an output every 48 clock cycles.

The value of c is also calculated by feeding back the

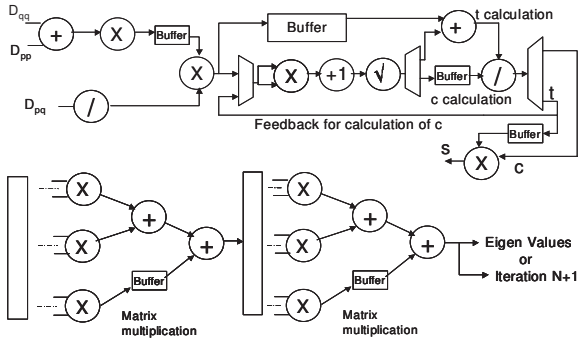

Figure 4. Tensor Diagonalisation Kernel

value of t to the pipeline. The data collisions in the pipeline are avoided, because of the delay of 48 clock cycles between consecutive inputs to the Tensor Diagonalisation kernel. First 9 of these idle clock cycles are used for the 9 iterations of the SV Decomposition. The next 9 idle clock cycles are used for the calculation of the value of c. Once the values of s and c are calculated, we determine the orthogonal matrices, which are multiplied with the tensor matrix.

The kernel was implemented using the integer cores provided by Xilinx [17]. After synthesis and simulation, the kernel was found to run at a maximum frequency of 111.3 MHz.

Anisotropy Calculation

This kernel involves the following calculations:

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

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

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

The implementation details of the kernel are as shown in Figure 5. The architecture of this kernel consists of a deep pipeline of a set of operations being performed in a sequential manner, while exploiting parallelism where ever possible. The architecture is capable of producing an output every clock cycle.

The kernel was implemented using the integer cores provided by Xilinx [17]. After synthesis and simulation, the kernel was found to run at a maximum frequency of 162.8 MHz.

4 Experimental Results

The preliminary implementations of the kernels were done using the 64 bit integer IP cores available by Xilinx [17]. Based on the maximum frequency of the kernel implementations, we have made a conservative estimate of using 100 MHz as the frequency to operate the implemented kernels as a kernel chain. In this section we provide performance analysis based on our experimental results of the implementation of the kernels. For a fair performance analysis with the C-code, we use the IEEE 64 bit floating point cores developed by our research group [3] instead of the 64 bit integer cores used in the kernel implementation.

Table 2. Floating point core details

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

The speed-up achieved is due to the deeply pipelined design, which exploits the independence of the tracking of individual fiber tracts. Thus, input data-set from different fibers can be continuously provided to the pipeline. The profiled C-code uses 28 images of size 128×128, and each image on an average has 2000 seed points. Thus we have enough data available to us to keep the pipeline busy. However in the tracking of a single fiber the consecutive fiber step is dependent on the outcome of fiber step currently computed. The latency of fetching the input data to maintain the through-put of the pipeline is overlapped with the computation of the anisotropy calculation kernel. Once the next fiber point has been computed and the required data fetched, the data-set is again provided to the pipeline for further fiber tracking.

We have compared our implementation performance with the profiled state-of-the-art C-code. The profiling results are as shown in the Table 1. Each kernel is executed on the order of 10^7 times as seen by the profiling results. Thus the initial combined latency of the kernel chain is neglected when compared to the through-put of the pipeline.

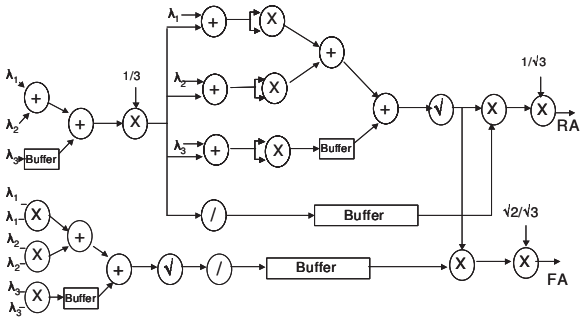

Figure 5. Anisotropy Calculation Kernel

Table 3. Performance Comparison with C-code

Kernels	Execution Time (secs)	
	$C-code$	$FPGA$
Tensor Interpolation	18.7	0.8
Tensor Interpolation+ Tensor Diagonalisation	146.7	4.8
Tensor Interpolation+ Tensor Diagonalisation+ Anisotropy Calculation	152.6	4.8

Tensor Interpolation

This kernel uses a fully pipelined tree structure and a reduction circuit. Set of 7, 64-bit floating point inputs are available every clock cycle. The output is produced every 8 clock cycles. The time taken to process 10^7 such operations to be equal to 0.8 seconds at 100 MHz. The same operation takes about 16 seconds on the C-based software implementation as seen from the profiling results.

Tensor Diagonalisation

One iteration of the SV Decomposition is unrolled to produce a very deep pipelined non-linear kernel implementation. This kernel receives the set of inputs after every 48 clock cycles from the tensor interpolation kernel. Due to the delay between successive inputs, the output is produced after every 48 cycles. Thus at this rate, both the tensor interpolation kernel and tensor diagonalisation kernel together will be able to process the input data in 4.8 seconds at 100 MHz as compared with 2 minutes and 27 seconds of the profiled C-code.

Anisotropy Calculation

The three eigen values $\lambda_1, \lambda_2, and\ \lambda_3$ are provided in parallel to the anisotropy calculation kernel from the tensor diagonalisation kernel. This is a fully pipelined kernel capable of producing an output every clock cycle. The input from the tensor diagonalisation kernel is delayed by 48 clock cycles, thus this latency is reflected in the output of this kernel. Considering the three kernels together as a kernel chain, the output can be produced every 48 cycles. The total time taken to compute for 10^7 such computations with the kernel chain operating at 100 MHz takes about 4.8 seconds as compared with 2 minutes and 32 seconds of profiled the C-code. Thus providing a speed-up of the order of 30x.

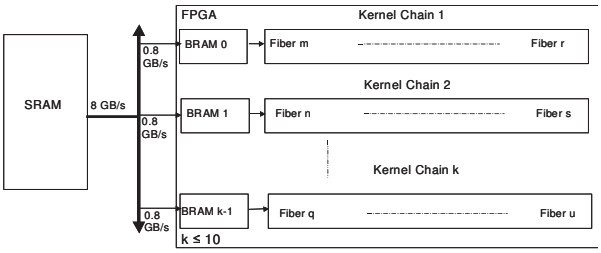

Figure 6. Overall Architecture

5 Overall Architecture

5.1 Parallel Implementation of Multiple Kernel Chains

Multiple kernel chains consisting of the computational intensive kernels can be implemented to run in parallel. This is possible because the computation involved in tracking a fiber is independent of that involved in tracking another fiber. This would provide a further speed-up proportional to the number of kernel chains implemented in parallel. Thus, if we have enough resources to implement 10 such kernel chains, with each chain providing a speed-up of 30x, we can expect an overall speed-up of the order of 300x. We would be able to complete the computation for the data-set of 10^7 data elements in 0.48 seconds as compared with 2 minutes and 32 seconds taken by the profiled C-code. The overall architecture is shown in Figure 6.

5.2 Bandwidth Requirements

Multiple kernel chains operating in parallel will result in increased FPGA to off-chip memory bandwidth requirements. Currently available FPGA modules (e.g. BenDATA-WS FPGA module [12]) provide up to 8 GBytes/sec of maximum memory to FPGA bandwidth.

The inputs required by the tensor interpolation kernel are the *x, y,* and *z* coordinates of the eight vertices; *x, y,* and *z* coordinates of the seed-point; and one of the six elements of the symmetric diffusion tensor matrix every clock cycle, as can be seen in Figure 3. With the known size of the cartesian cube, the values of the *x, y* and *z* coordinates of the vertices and the computed fiber points are locally available. Thus the inputs required from the off-chip memory are the six elements of the diffusion tensor matrix for each vertex of the cartesian cube in operation. These elements can be stored on chip in the BRAM while the fiber tracking continues in the same cartesian cube.

If our pipeline requires one diffusion tensor matrix element every clock cycle to sustain fiber tracking from

multiple seed-points, we would require one 64-bit floating point value every clock cycle. Thus the bandwidth requirement in the worst case would be 0.8 GBytes/sec. However, as the fiber tracking continues with-in the cartesian cube, the on-chip data is reused and no inputs are required from the memory for this computation. Thus during this time the unused bandwidth can be used for speculative pre-fetching of the diffusion tensor matrix elements of the vertices of the cartesian cube surrounding the cartesian cube in operation.

5.3 Overall Performance

Even if we are able to implement 5 kernel chains operating in parallel, we can achieve a further 5x speed-up over the single kernel chain implementation. Thus we should be able to complete the computation in 0.96 seconds as compared with 2 minutes and 32 seconds taken by the C-code on a 800 MHz Intel Xeon processor having 4GB of RAM. Thus providing an overall speed-up of 155x.

6 Conclusion and Future Work

The pipelined kernel chain was able to complete the required computation in 4.8 seconds as compared with the 2 minutes and 32 seconds obtained by profiling the C-code. Thus providing a speed-up of 31x. The results based on a conservative clock speed of 100 MHz have been summarized in Table 3. Further speed-up is also possible when multiple such kernel chains are implemented in parallel. Having 5 such kernel chains in parallel, we will be able complete all computation in 0.96 seconds, a speed up of 155x.

In our analysis we have used 64-bit floating point cores to get a fair comparison with the available state-of-the-art C-code implementation, which uses double precision floating point variables. Fiber tracking can also be achieved using 32-bit floating point numbers with out much concern of loss of precision to the biomedical community. Thus using 32-bit floating point cores in our implementation, will result in faster clock rate and reduced area.

Our present research provides the pipelined kernel implementation for a DTI tractography technique. Our implementations are modular, and can be used for implementation of other DTI tractography techniques, with some changes to the present kernel implementations. Providing a library of kernels would allow the users to speed-up different DTI techniques on the FPGAs. The kernel library would give the users a choice of the technique to be implemented.

References

[1] P. Basser, S. Pajevic, C. Pierpaoli, J. Duda, and A. Aldroubi. In vivo fiber tractography using DT-MRI data. *Magn Reson Med.*, 44(4):625–32, 2000.

[2] BIRN - Biomedical Informatics Research Network. `http://www.nbirn.net`.

[3] G. Govindu, R. Scrofano, and V. K. Prasanna. A library of parameterizable floating-point cores for FPGAs and their application to scientific computing. In T. Plaks, editor, *Proceedings of the International Conference on Engineering Reconfigurable Systems and Algorithms*, June 2005.

[4] E. Haacke et al. *Magnetic Resonance Imaging-Physical principles and sequence design*. Wiley-LISS, 1999.

[5] P. Hagmann, J. Thiran, L. Jonasson, P. Vandergheynst, S. Clarke, P. Maeder, and M. R. DTI mapping of human brain connectivity: statistical fibre tracking and virtual dissection. *Neuroimage*, 19(3):545–54, 2003.

[6] Intel Corporation. `http://www.intel.com`.

[7] S. Kim. *White Matter Tractography Using Diffusion Tensor-Magnetic Resonance Imaging*. PhD thesis, University of Southern California, 2003.

[8] S. Makeig, S. Debener, J. Onton, and A. Delorme. Mining event-related brain dynamics. *Trends in Cognitive Science*, 8(5):204–210, 2004.

[9] S. Makeig, A. Delorme, M. Westerfield, T.-P. Jung, J. Townsend, E. Courchesne, and T. J. Sejnowski. Electroencephalographic brain dynamics following visual targets requiring manual responses. *Public Library of Science Biology*, 2(6), 2004.

[10] Mentor Graphics ModelSim. `http://www.model.com`.

[11] S. Mori, W. Kaufmann, G. Pearlson, B. Crain, B. Stieltjes, M. Solaiyappan, and P. van Zijl. In vivo visualization of human neural pathways by magnetic resonance imaging. *Annals of Neurology*, 47(3):412–414, 2000.

[12] Nallatech. `http://www.nallatech.com`.

[13] C. Pierpaoli, P. Jezzard, P. Basser, A. Barnett, and G. Di Chiro. Diffusion tensor MR imaging of the human brain. *Radiology*, 201(3):637–48, 1996.

[14] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1999.

[15] M. Singh, W. Sungkarat, and K. Veera. Evaluation of mri dti tractography by tract-length histogram. *Progress in Biomedical Optics and Imaging: Physiology, Function and Structure for Medical Images*, 5746(1):138–147, 2005.

[16] E. von dem Hagen and R. Henkelman. Orientational diffusion reflects fiber structure within a voxel. *Magnetic Resonance in Medicine*, 48(3):454–59, 2002.

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

[18] L. Zhuo, G. R. Morris, and V. K. Prasanna. Designing scalable FPGA-based reduction circuits using pipelined floating-point cores. In *Proceedings of the 12th Reconfigurable Architectures Workshop*, Denver, CO, April 2005.