Scalable, High Performance Fourier Domain Optical Coherence Tomography: why FPGAs and not GPGPUs

Jian Li, Marinko V. Sarunic, and Lesley Shannon
School of Engineering Science
Simon Fraser University, Burnaby BC, Canada
Email: {jla193, msarunic, lshannon}@sfu.ca

***Abstract*—Fourier Domain Optical Coherence Tomography (FD-OCT) is an emerging biomedical imaging technology featuring ultra-high resolution and fast imaging speed. Due to the complexity of the FD-OCT algorithm, real time FD-OCT imaging demands high performance computing platforms. However, the scaling of real-time FD-OCT processing for increasing data acquisition rates and 3-dimensional (3D) imaging is quickly outpacing the performance of general purpose processors.**

Our research analyzes the scalability of accelerating FD-OCT processing on two potential implementation platforms: General Purpose Graphical Processing Units (GPGPUs) and Field Programmable Gate Arrays (FPGAs). We implemented a complete FD-OCT system using a NVIDIA GPGPU as co-processor, with a speed up of *6.9x* over general purpose processors (GPPs). We also created a hardware processing engine using FPGAs with a speed up of *15.5x* over GPPs for a single pipeline, which can be replicated to further increase performance. Our analysis of the performance and scalability for both platforms shows that, while GPGPUs offer an easy and low cost solution for accelerating FD-OCT, FPGAs are more likely to match the long term demands for real-time, 3D, FD-OCT imaging.

***Keywords*-GPGPU, FPGA, FD-OCT**

I. Introduction

Clinicians are quickly coming to rely on high-quality medical imaging technology and fast processing speeds. A popular form of medical imaging is tomography, which produces two-dimensional (2D) images of the internal structure of a solid object by measuring the differences between the energy passing through the object and its echos/reflections. Different tomography technologies, such as Computed Tomography (CT), Positron Emission Tomography (PET) and Magnetic Resonance Imaging (MRI), have been adopted for various diagnostic purposes. Fourier Domain Optical Coherence Tomography (FD-OCT) is based on optical interferometry, which is rapidly gaining popularity for cross-sectional imaging of biological tissues and materials. As it provides ultra-high image resolution at a micrometer scale along with fast imaging speeds, FD-OCT is well suited for ophthalmic imaging in both clinical and research environments.

Fast processing speed and high image quality are crucial for real-time FD-OCT systems. However, FD-OCT systems implemented on general purpose processors (GPPs) cannot scale with the increasing data resolution and acquisition speeds from newer cameras. Currently, FD-OCT software is able to deliver a throughput rate of 30MB/s on a quad core Intel i7 3GHz with 3GB of RAM [1]. Data acquisition using new imaging equipment is expected to demand 8GB/s throughput rates [2]. Moreover, real-time 3D volumetric rendering, achieved by stacking multiple 2D FD-OCT images in real time, will require even greater processing speeds. Therefore, alternative platforms with better potential for performance scaling are needed by FD-OCT researchers and clinicians to meet these requirements.

Previous work to accelerate FD-OCT using either General Purpose Graphical Processing Units (GPGPUs) or Field Programmable Gate Arrays (FPGAs) does exist [3], [4]. However, it focused on improving the quality of data acquisition and imaging algorithms, as opposed to how these two platforms may satisfy the future processing demands of faster data acquisition rates and real-time, 3D, volumetric rendering. ASIC solutions for FD-OCT have not been considered due to the low production volume for FD-OCT applications. Also, FD-OCT researchers prefer platforms that provide reconfigurability for tuning different optical parameters.

The FD-OCT algorithm is highly suited to parallel processing. GPGPUs provide many processing cores and a large on-chip cache that can be used to exploit coarse-grained parallelism. Conversely, FPGAs have less on-chip memory, but enable fine-grained parallelism (i.e. pipelining). In this paper, we evaluate the potential of these two platforms to meet the requirements of real-time FD-OCT by identifying their limiting factors to scaling performance. The specific contributions of this work are:

- A complete FD-OCT system with GPGPU accelerated processing: the GPGPUs provide a maximum throughput of 527MB/s, whereas the maximum throughput for the overall system is limited to 207MB/s due to the overhead incurred by data transfers.
- A fully-pipelined, FPGA-based, FD-OCT processing engine on Xilinx Virtex 5 devices; the maximum processing throughput of one pipeline is 465MB/s, which can be increased by replicating it.
- An analysis of the FD-OCT algorithm's performance on GPGPUs that demonstrates that due to their discrete

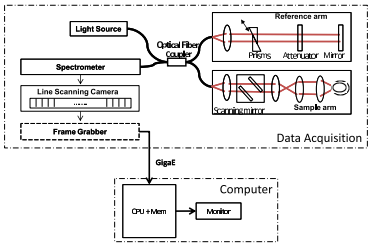

Figure 1. Diagram of our Real-time FD-OCT system.

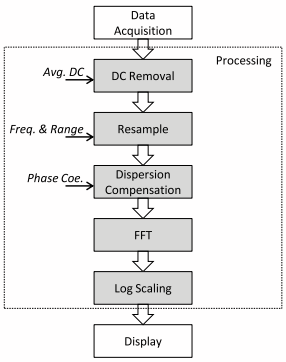

Figure 2. High-Level FD-OCT Algorithm flow.

memory model they *cannot* scale to meet future data acquisition rates, whereas FPGAs can.

This analysis will be used in future work to develop a scalable computing platform to enable real-time, volumetric rendering for future FD-OCT technologies.

The remainder of the paper is organized as follows. Section II provides an overview of FD-OCT and related works. The GPGPU and FPGA hardware implementations are described in Sections III and IV, respectively. Section V summarizes our conclusions and avenues for future work.

II. FD-OCT

Creating an FD-OCT image requires 3 steps: data acquisition, processing, and displaying the image. Figure 1 shows the FD-OCT system we use to acquire data. First, a broadband *light source* casts laser beams into both the *reference arm* and the *sample arm*. An interferometric pattern is generated from the light interference reflected from both arms. The optical signals of this pattern are then converted into the spectrum domain using a *spectrometer*. Our current system uses a CCD *line scanning camera* to scan and record individual lines from the spectrometer's output, where each "line" has 1024 pixels that fall into different locations of spectrum. As each line of data is independently acquired and processed in an FD-OCT system, their throughput rates are typically reported as the "line rate" (the number of lines that can be processed per second). These individual lines are coallated into a single frame using a *frame grabber* to reduce the overhead of data transfers to the processing system by transferring data in batches. Each assembled frame is then transferred to the host computer over Gigabit Ethernet (*GigaE*) for processing, during which time the next frame is constructed by the frame grabber.

Figure 2 summarizes our FD-OCT algorithm, highlighting the key functions required to process the data. Once the FD-OCT data is acquired, the first processing phase is a *DC Removal* procedure that subtracts the average DC levels from each scanning line. The average DC levels (shown as *Avg. DC* in Figure 2) are the average DC components across multiple lines; these are obtained prior to real time processing as the DC components remain relatively constant across different acquisitions. Next, the camera's output data is *Re-sampled* into linear wave-number (k) space. The output data set from the camera is evenly sampled in wavelength (λ) space; however, the FD-OCT algorithm processes data in wave-number (k) space [1]. *Re-sampling*, therefore, is used to transform the data into the desired form. The third phase of processing uses the Hilbert Transform to perform *Dispersion Compensation* [5], [6]. As the frequency of broadband light is dependent on the material through which it propagates, it is necessary to match the dispersion caused by differences between the sample and reference arms (see Figure 1) to consistently achieve the high resolution images [5]. Next, a 1024-point Fast Fourier Transform (*FFT*) is performed on each line of data to reproduce the pattern of interferometric fringes from the spectral signal. The final processing stage uses *Logarithmic Scaling* of the data to increase the visibility of details to improve image quality. The remainder of this section summarizes work using GPGPUs and FPGAs to accelerate FD-OCT algorithm.

A. Related Work

As the individual lines of data are acquired independently and can be processed independently, this parallelism can be used to accelerate the FD-OCT algorithm to increase throughput. GPGPUs are becoming a popular method of accelerating FD-OCT, however not all of our processing steps have been included in these previous works. For example, Watanabe et al. [3] used a NVIDIA GTX285 GPGPU to accelerate a FD-OCT system with a 2048-pixel line size to achieve a real time display of 27.9 frames/sec. Although they use a 2048-point FFT, their only processing steps are the DC removal, FFT and logarithmic scaling (recall Figure 2). More recently, Zhang et al [7] used a FX5800 GPGPU to accelerate an FD-OCT system similar to ours (but with no dispersion compensation). They achieved line rates of 680kHz for 1024 pixels and 320kHz for 2048 pixels. However, these line rates only reflect the GPGPUs

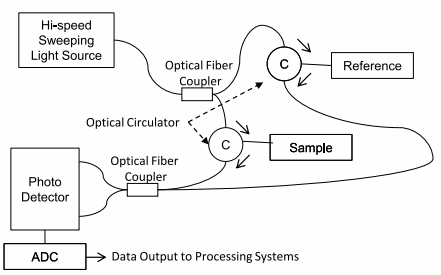

Figure 3. A typical swept-source FD-OCT system [9].

processing time, and drop to 128kHz (1024 pixels) and 70kHz (2048 pixels) respectively for the system's actual throughput. Zhang et al's system also used 3D real-time volumetric rendering to the display and achieved a refresh rate of 10 frames/second [7]. As a minimum refresh rate of 20 frames/second is required for real time imaging, significantly higher throughput rates are required.

There has also been some work on FD-OCT accelerators that use FPGAs. These systems typically leverage a different data acquisition setup than the one shown in Figure 1, called a *swept-source based FD-OCT system*, shown in Figure 3. Swept-source systems use a narrow-band, *sweeping light source* to rapidly sweep over a broadband spectrum, producing interference fringes at each wavelength. The output is detected by a *photo-detector*, which converts the interference signal into a voltage at rates of up to 5.2 MHz per line [8], $\sim$100 times faster than a line scanning camera acquires a single line. However, unlike camera-based systems, photo-detectors *serially* convert the spectral frequencies to voltages. These voltages are then converted by an Analog-to-Digital Converter (*ADC*) for processing. As these systems acquire individual lines of data for processing (and not frames), they require less memory for processing the data and are better suited to FPGAs.

Desjardins et al. [10] used two unspecified Virtex 2 devices integrated into their data acquisition rates to perform their processing and achieved throughput rates of 5MB/s. Precise details on their hardware implementations for each device are not provided. Ustun et al. [4] implemented an FD-OCT accelerator on a more modern Virtex 4 FX12 using Simulink. Their implementation required 95% resource usage and achieved a throughput of 27MB/s. We have been able to achieve a significant speed up over this result ($>$17x), likely due to our hardware mapping (e.g. using more embedded multipliers, etc); however, again, the authors do not provide a detailed discussion of their implementation, so we can provide no real analysis.

In summary, the main focus of all this previous work that uses GPGPUs and FPGAs to accelerate FD-OCT is on the experimental setups and imaging algorithms to improve image quality, while trying to speed up processing. However, our work analyzes how the underlying algorithm maps to the technology to determine what is the most appropriate platform for the future requirements of increased data acquisition speeds and real time, 3D rendering.

```
/* Vector Add Function, same as:
 * ---------------------------------------
 * for (int i=0; i<HEIGHT; i++)
 *    for (int j=0; j<WIDTH; j++)
 *        Sum[i][j] = A[i][j] + B[i][j];
 * ---------------------------------------
 */
kernel vector_add (matrix A, matrix B, matrix Sum)
{
    thread_index i, j;  /* use concurrent threads */
    Sum[i][j] = A[i][j] + B[i][j];
}

/* main function call */
int main (void)
{
    matrix a, b, s;
    vector_add(a, b, s); /* kernel call */
}
```

Figure 4. Pseudo code example for stream processing.

III. GPGPU Implementation

This section discusses our software implementation of FD-OCT using the Computed Unified Device Architecture (CUDA) language [11] from NVIDIA Corp and analyzes its performance and scalability.

A. Mapping FD-OCT onto GPGPU for Parallel Processing

The underlying GPGPU architecture combines hundreds of processing cores, allowing thousands of tasks to be scheduled and executed in parallel. GPGPUs feature a Single-Instruction, Multiple-Data (SIMD) model to map threads of execution to these processing cores, enabling efficient parallel processing of vector operations. *Stream processing* divides the data into subsets, called streams, to exploit data parallelism. Similar *kernel* operations can be performed on each stream independently and concurrently. Each instance of a kernel's execution is called a *thread*. Figure 4 shows a pseudo code example of stream processing, where each element of the matrix is one stream. A kernel `vector_add` is defined to perform an *add* operation on each corresponding element in A and B. Each kernel execution has a unique *thread index*, comprised of i and j, to identify it. Finally, the `vector_add` is performed on each element (stream).

We leverage the large on-chip cache and device memory of the GPGPU, as mentioned in Section I, to provide coarse-grained parallelism by transferring and processing multiple frames. As the FD-OCT algorithm is mainly comprised of vector operations, substantial speed-up can be achieved if the potential parallelism is properly mapped onto the GPGPU. The two main opportunities identified for parallel processing in the FD-OCT algorithm are:

- Each line of the image is a stream that can be concurrently processed with other lines during all the different phases of algorithm outlined in Figure 2.

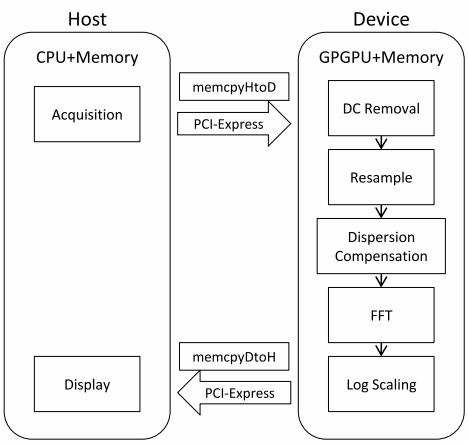

Figure 5. The GPGPU algorithm flowchart.

- Each scanning line can be divided into individual elements (individual streams) for parallel processing during some of these processing phases (e.g. the vector add and subtraction in DC Removal, the vector multiplication in λ-to-k re-sampling, etc.).

B. Experimental Setup

Our experimental setup is the same as Figure 1, except that the computer also includes an NVIDIA dual-GPGPU GeForce GTX295 co-processor connected via PCI-Express bus. Our current implementation uses one GTX295 GPGPU, with 240 processing cores running at 576MHz with 896 MB of DDR3 Memory. As the related works [3], [7] use only a single GPGPU, this enables a fair throughput comparison. The GTX295 GPGPU board communicates with the CPU via the 16 lanes PCI-Express v2.0 bus (similar to [3] and [7]).

We used version 2.3 of CUDA to develop and debug the FD-OCT algorithm on the GTX295 GPGPU. Figure 5 shows the algorithm flowchart, as well as the data flow between the *Host* (CPU and main memory) and the *Device* (GPGPU and device memory). In this GPGPU accelerated system, only data acquisition and display are performed on the GPP. Pending availability, one or more frames of data are transferred as a batch to the GPGPU for processing to amortize the cost of the transfer. It should be noted that this memory transfer from the CPU to the GPGPU (*memcpyHtoD*) is not due to the GPGPU's situation on a PCI-Express daughter-card. Instead, the memory transfer is inherent to the GPGPU architecture. Even integrated GPGPUs (e.g. the NVIDIA GeForce 9400M), which reside with CPU on the same motherboard, will require these memory transfers from host to device memory.

While we wrote most of the processing kernels for the FD-OCT algorithm, NVIDIA's optimized CUFFT library [12] for the FFT operations in the Dispersion Compensation and FFT processing phases. As with the GPP implementation of the FD-OCT algorithm, the user is able to choose between a 1024 and 2048-point FFT (so that the fringes in the displayed results are easier to observe). For the 2048-point FFT, we zero pad the 1024 data samples and use NVIDIA's 2048-point FFT function. Finally, as our GTX295 accelerator cannot directly display from the GPGPU memory onto the screen, the GPGPU's processed data is copied back to the CPU (*memcpyDtoH*) for rendering and display[1]. Figure 6 shows a sample retina image with a resolution of 1024x512 processed using our system.

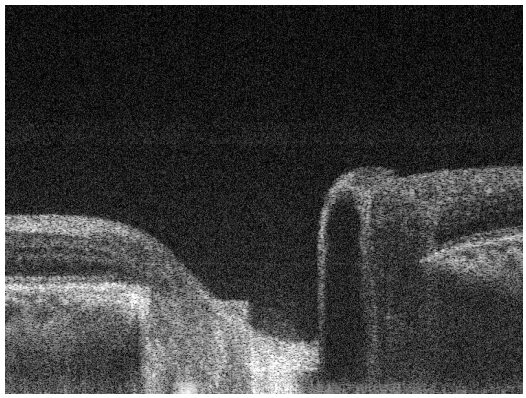

Figure 6. A sample retina image using GPGPU.

C. GPGPU Results

We measured the total FD-OCT processing time including the memory transfer time between host and device using *cudaprof* [13], the program profiler provided by NVIDIA. The throughput of the system is calculated by using *cudaprof* to measure the overall run time of all the processing steps for one whole frame, plus the additional data transfer time for the frame between the host and device.

As GPGPUs are configured as co-processors, without direct access to the main memory of the host processor, we wanted to ensure that we transferred sufficient data in a single memory copy to amortize the cost of the transfer during processing[2]. We presented the overall system throughput versus different data transfer sizes in [2] and showed that the overall throughput increased with batch size up to 2048 lines where it plateaus, achieving the maximum throughput at a batch size of 8192 lines. We also demonstrated in [2] that the overall performance could be improved by 22%, assuming an integrated GPGPU (physically sharing memory with the CPU) with the same processing power of GTX295.

Figure 7 illustrates the percentage of the FD-OCT algorithm's run-time as attributed to its various component

[1]Ideally, the GPU can directly display the post-processed data without this additional device-to-host copy, which we are currently investigating.

[2]Data Sets are "batched" into larger blocks to amortize the overhead of initiating data transfers between device and host memory [11].

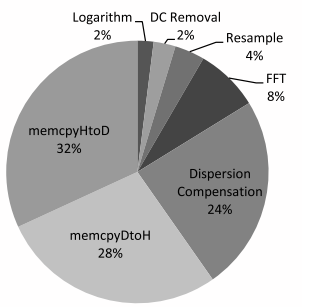

Figure 7. The percentage of the GPU functions runtime.

functions for a batch size of 8192 lines [2]. Similar results were seen in Watanabe et al. [14], but they didn't include dispersion compensation, a key component in high resolution FD-OCT. Figure 7 shows that the processing phases of the FD-OCT algorithm (DC-Removal, Resample, Dispersion Compensation, FFT and Logarithmic Scaling) account for ∼40% of the total run-time time, while the memory (data) transfers (host-to-device and device-to-host) require approximately 60% of the time. For a frame with 1024 by 512 pixels, the total processing and data transfer time is 4.82 ms, resulting in an overall system throughput of approximately 207MB/s or a line rate of 110kHz. When compared to a throughput rate of 30MB/s on a Intel i7 3GHz with 3GB DDR2, this equals a 6.9x increase.

To compare with the most recent previous work, we exclude the dispersion compensation and data transfers and compare processing-only throughput for lines with 2048 pixels. Whereas Zhang et al.'s system had a maximum line rate of 320 kHz on the FX5800 GPGPU [7], our system's maximum line rate is 680 kHz on the GTX295 GPGPU. Therefore, our system is able to achieve a significantly faster line rate on a GPGPU with the same number of processing cores, but less memory and a slower clock rate. As the dispersion compensation phase accounts for more than 50% of the overall processing time, it slows our maximum processing-only line rate down to 320 kHz, which is only 14% slower than [7] (which excludes this extra processing step).

While excluding the memory transfer time allows us to compare our system with previous work, to properly evaluate the real-time performance of a GPGPU accelerated implementation, the data transfer time must also be included. This overhead for existing GPGPU architectures reduces the maximum camera line rate for the complete system implementation to ∼110 kHz, or 207MB/s.

Therefore, to significantly increase the maximum line rate, there needs to be a change to the underlying memory architecture of the GPU to limit the number of memory transfers between the CPU and GPGPU. Ideally, the GPGPU would be able to share memory with the CPU and then directly render its processed data to the display without the memory copies. However, even if the discrete model is maintained, the ability to directly render the data to the display would significantly impact the line rate of the system, increasing it nominally by ∼1.3x to a line rate of ∼143kHz.

We also implemented the algorithm on NVIDIA's latest Fermi architecture, the GTX480. We are currently investigating how to support the new features of the Fermi architecture, such as duplex data transfers between the host and device, in our implementation. The GTX480 exhibits performance versus data transfer size trends similar to the GTX295, but GTX480 delivers more than twice the processing throughput of the GTX295 [2]. Moreover, as the GTX480 allows duplex data transfers, it could reduce the impact of the memory copy by writing post-processed lines back to the CPU while new data for processing is read onto the GPU. Furthermore, we are pursuing the use of multiple host threads to exploit the additional spatial parallelism available on multi-GPU devices so that while one device processes the latest data, the other device can be used to render 3D images to the display. In general, the key to improving the throughput rates for real time processing of FD-OCT data on GPGPUs will be to minimize and/or hide the cost of the data transfers between the CPU and GPGPU.

However, even the GPGPUs maximum processing-only line rate on the new GTX480 (∼610 kHz [2]) is still insufficient to meet the current maximum data acquisition rates swept-source based systems (5.2 MHz) by a factor of 8.5. Based on the current scaling of GPGPU processing power and data acquisition rates alike, GPGPUs will not likely be able to keep pace as they are unable to leverage sufficient parallelism from the algorithm.

IV. FPGA Implementation

This section describes our hardware implementation of the FD-OCT processing algorithm (recall Figure 1), and analyzes its performance and scalability.

A. Mapping FD-OCT onto FPGA

FPGAs have less on-chip memory than GPGPUs and thus cannot process large amounts of data concurrently (i.e entire frames) However, FPGAs can leverage fine-grained parallelism using pipelining and replicated pipelines to process multiple lines in parallel. To achieve the maximum possible throughput rates for the FPGA implementation of FD-OCT, we assume that input is provided serially from a swept-source based system (recall Figure 3). This allows the lines to be processed individually without requiring intermediate memory to store a complete frame. As with the original GPP software implementation, each sample is assumed to be a 16-bit word and all calculations are performed using fixed point arithmetic. Unlike the GPGPUs, which used single precision floating point, this requires an additional scaling

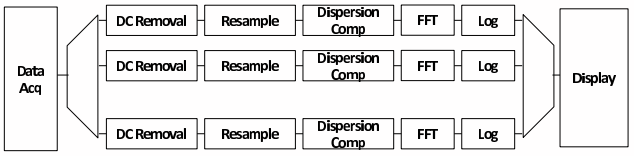

Figure 8. Replicated FD-OCT Processing-Pipeline.

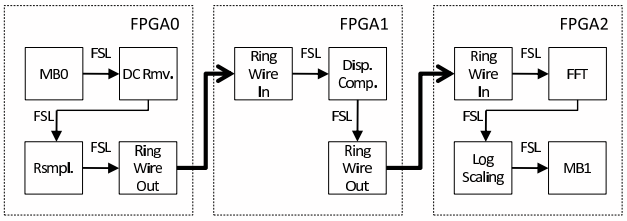

Figure 9. FD-OCT Design Partitioned onto BEE3.

factor be used during some of the processing stages (e.g. FFT). Finally, unlike the GPP and GPGPU based systems, we do not allow the user to select the size of the FFT (1024 vs 2048 point) and instead hard code 1024-point FFTs into our design.

As discussed previously in Section III-A, parallelism can be exploited between the various processing phases as each line of data is independent. Furthermore, some computational sub-components of these phases are also independent (vector adds, subtracts, et.), providing further opportunities for parallel processing. Our design leverages both types of parallelism and is fully pipelined, able to receive new data at every clock edge so that it can operate at the same frequency as the data acquisition system. As some phases require the entire scanning line for processing (e.g. FFT), buffering mechanisms are provided between phases to ensure that individual samples in a scanning line can be stored without stalling the other pipeline stages and reducing throughput.

Assuming there are sufficient hardware resources, the throughput of the FPGA implementation can be increased by duplicating the pipeline. Figure 8 illustrates how the input is de-multiplexed into individual lines, which are then processed by one of the pipelines. After processing, these lines can be coallated into frames to be displayed.

Finally, along with Look-up Tables (LUTs) and Flipflops (FFs), modern FPGAs provide embedded DSP blocks and dual-ported Block RAMs (BRAMs) that have much greater area efficiency and clocking rates than their equivalent circuits mapped to LUTs and FFs. Given the nature of the algorithm, both of these resources can be leveraged to improve the design's throughput. For example, the DSP blocks are used in the multiply-accumulate operations required by the FFTs. Furthermore, instead calculating the logarithmic scaling at runtime, values are obtained from a look-up table stored in the BRAMs. Also, the Resampling module is required to reorder the input data as part of its processing. Therefore, we fix one port of the BRAM as the input port used to reorder the data when it is stored to the BRAM, and the other port is used to sequentially read out the reordered data for the next processing phase.

B. Experimental Setup

The FD-OCT algorithm is written in VHDL and synthesized using version 11.5 of Xilinx's Platform Studio (XPS) for Virtex 5 LX 155T devices. We use Xilinx's system generator to create the FFT IP cores (version 7.0) used in the Dispersion Compensation and FFT modules. As our design requires almost all of the DSP40E slices available on one of these devices, it does not place and route well on a single device. Therefore, we tested and verified the design on the Berkley Emulation Engine 3 (BEE3) platform [15], which has four Virtex 5 LX155Ts, allowing us to emulate much larger devices (e.g. the Virtex 6 SX475T and the Virtex 7 870T). By partitioning the design onto different FPGAs on the BEE3, we achieve significantly higher clock frequencies, and thereby better throughput.

Figure 9 shows the partitioning scheme for implementing our design on the BEE3 platform. *FPGA0* implements both the DC Removal and the Resample module, two of the simplest modules in the design. The next phase, Dispersion Compensation, is the most complex module and is implemented on its own VLX155T FPGA (*FPGA1*). Finally, the remaining two phases, the FFT and the Logarithm are implemented on *FPGA3*. As the FPGA-based processing engine is not currently hooked into a real system, both *FPGA0* and *FPGA2* also include one MicroBlaze processor for initiating the data transfer and receiving the results. The inter-FPGA connections are uni-directional, GPIO-based buses called *Ring Wire Buses* provided by the BEE3 platform. The direction of each of the inter-FPGA connections has been indicated in Figure 9 and are guaranteed to operate reliably for operating frequencies up to 400MHz [15].

As we currently do not have access to a swept-source based FD-OCT system, the FPGA design also includes two MicroBlazes, *MB0* and *MB1*, to emulate the data acquisition system and the display, respectively. Data acquired using our current camera based system is pre-loaded into *MB0*'s BRAM. *MB0* reads in this data and writes it directly to a Fast Simplex Link (FSL) bus, a FIFO provided by the MicroBlaze for integrating hardware accelerators. This FSL bus is connected to the first processing module, *DC Removal*. After the last processing phase has been completed (*Logarithmic Scaling*), the data is written to another FSL connected to *MB1*, which reads in the data and stores it to its own BRAM. The correctness of the design is verified by reading this data off the board onto a host PC, where it can be compared to the GPP software implementation.

Table I
RESOURCE USAGE FOR THE PROCESSING BLOCKS.

FPGA ID	Modules	6-LUT		FF		BRAM		DSPE48		Max Freq
		#	Percent	#	Percent	KBit	Percent	#	Percent	(MHz)
FPGA 0	DC Removal	71	1%	19	1%	18	1%	1	1%	246
	Resample	151	1%	115	1%	216	7%	3	2%	268
	MB0	1237	1%	1388	1%	2048	27%	3	2%	-
FPGA 1	Dispersion Compensation	8952	9%	10884	11%	324	4%	64	50%	244
FPGA 2	FFT	3548	3%	4304	4%	162	2%	56	43%	276
	Log Scaling	112	1%	14	1%	1080	14%	-	-	358
	MB1	1237	1%	1388	1%	2048	27%	3	2%	-
FPGA 0,1,2	Overall	15308	-	18112	-	5896	-	130	-	-

C. FPGA results

Figure 10 shows the image quality of the FPGA output is degraded compared to the GPGPU. This is largely due to the FPGA implementation using 16-bit fixed point numbers for data representation, so the current scaling method (truncation) causes overflow/underflow errors to propagate along the pipeline. Future work will aim to improve image quality from the FPGA by opting for a different scaling method and/or increasing the data representation width.

The resource usage for each individual processing module on a Virtex 5-155T device is listed in Table I. The usage of 6-input LUTs, FFs, BRAMs and DSP48E slices are listed from Columns 3 through 6 respectively. Each column contains two sub-columns indicating the number of each type of resource used and the percentage of that resource used on this device. Column 7 shows the maximum frequency of each module. All modules, except Dispersion Compensation, use a relatively small number of LUTs and FFs, no more than 4% of those available on the Virtex 5 LX155T. However, both the Dispersion Compensation and FFT modules use a high percentage of DSP48E slices to increase the speed of their multiply-accumulate operations. The Logarithmic Scaling module uses the most BRAM (14%) due to its look-up table architecture, which reduces its design complexity.

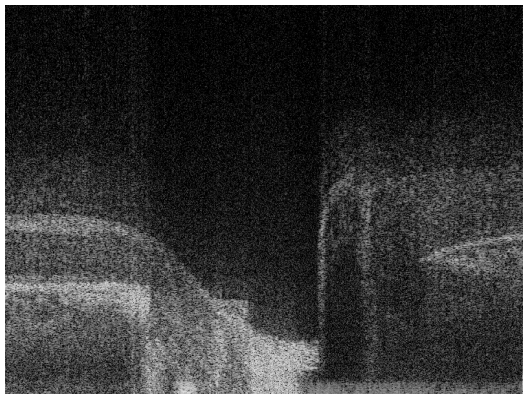

Figure 10. A sample retina image from FPGA.

Recalling from Section IV-A that our design is fully pipelined, the module with the lowest operating frequency dictates the maximum overall throughput of the processing system. Assuming the swept-source based data acquisition set up shown in Figure 3, where data can be acquired at every clock cycle, the maximum overall system throughput is 465MB/s or a line rate of 238kHz. Recalling the maximum throughput of the GPP based system, 30MB/s, the FPGA based system has increased throughput by 15.5x. Recalling that 5.2 MHz is the current maximum data acquisition rate for swept-source based systems, the FPGA design is 21.8x too slow. However, as ISE v11.5 supports Virtex 6 devices, we synthesized the entire design onto a single Virtex 6 LX130T, which has approximately 25% of the resources of the SX475T (the largest device in the Virtex 6 family). On the Virtex 6 LX130T, the maximum clock frequency is 238MHz, with a resource usage of 8.1% LUT, 8.7% FF, 36.3% DSP48E1 and 19% BRAM, suggesting that the pipeline could be easily replicated 22x on the SX475T. Furthermore, based on the data sheets posted for the Virtex 7 VH807T, a preliminary estimation suggests that there are sufficient resources to replicate the processing pipeline 30x, which is more than is currently required.

The next phase of this work will investigate integrating this processing system with a data acquisition system. Although our initial swept-source based system would not have a 5.2MHz line rate, we need to determine how to utilize the FPGAs high-speed I/Os to acquire input data at high speeds. We also need to confirm that the memory copy required to transfer the data back to the host PC for display can be performed in parallel with the FPGA processing new samples. Our preliminary analysis suggests that the FPGA based processing system will be able to continue processing and storing new data while host PC system copies the existing data back for display. Furthermore, we would also like this data to be copied to a GPU for 3D, volumetric rendering in the future.

V. Conclusions and Future Work

In this paper, we investigated how GPGPUs and FPGAs can be used to accelerate FD-OCT processing for current and future data acquisition rates. We implemented a complete FD-OCT system using a GPGPU as a co-processor and achieved an overall system throughput of 207MB/s. We also demonstrated a hardware FD-OCT processing engine on a BEE3 board, using three Virtex5-155Ts, with a maximum overall system throughput of 465MB/s. The GPGPU speed up over the original GPP implementation [1] is 6.9x; for our FPGA implementation, it is 15.5x, and able to scale with logic replication on Virtex 6 and 7 devices.

GPGPUs provide an easier programming model than FPGAs, able to provide moderate speedups to FD-OCT processing. Moreover, GPGPUs require a much shorter design turn-around time: the FD-OCT GPGPU implementation took $\sim$3 months, where as the the FPGA implementation took almost 4x longer. However, the discrete memory model of the current GPGPU architecture imposes additional data transfers between CPU and GPU memory; these transfers are the limiting factor for the GPGPU platform. For this reason, it will be extremely difficult to scale GPGPU implementations for future FD-OCT processing demands. Conversely, given a large enough/multiple FPGA devices, replicating our existing pipeline 22x will allow FD-OCT processing speeds to match the current fastest data acquisition rate of 5.2 MHz.

Work is currently underway to investigate the implementation of bi-directional data transfers using the new Fermi architecture GPU GTX480. We are also investigating how to best integrate the FPGA processing engine with a swept-source based system to form a complete FD-OCT system. Our immediate concern is how to transfer the post-processed data to the display without stalling processing of new samples on the FPGA. Long term, we are investigating how to integrate the FPGA into swept-source based systems with extremely high data acquisition rates.

Acknowledgements

The authors thank: the Canadian Microelectronics Corporation and Xilinx for their equipment donations; the Canadian National Science and Engineering Research Council, the Canadian Institute of Health Research and the Michael Smith Foundation for Health Research for funding this project; and Pavel Bloch and Jing Xu for their contributions.

References

[1] J. Xu, L. Molday, R. Molday, and M. Sarunic, "In vivo imaging of the mouse model of X-linked juvenile retinoschisis with fourier domain optical coherence tomography," *Investigative ophthalmology & visual science*, vol. 50, no. 6, p. 2989, 2009.

[2] J. Li, P. Bloch, J. Xu, M. Sarunic, and L. Shannon, "Performance and Scalability of Fourier Domain Optical Coherence Tomography Acceleration Using Graphics Processing Units," *To Appear in Applied Optics*, 2011.

[3] Y. Watanabe and T. Itagaki, "Real-time display on fourier domain optical coherence tomography system using a graphics processing unit," *Journal of Biomedical Optics*, vol. 14, no. 6, p. 060506, 2009.

[4] T. E. Ustun, N. V. Iftimia, R. D. Ferguson, and D. X. Hammer, "Real-time processing for fourier domain optical coherence tomography using a field programmable gate array," *Review of Scientific Instruments*, vol. 79, no. 11, p. 114301, 2008.

[5] M. Wojtkowski, V. Srinivasan, T. Ko, J. Fujimoto, A. Kowalczyk, and J. Duker, "Ultrahigh-resolution, high-speed, fourier domain optical coherence tomography and methods for dispersion compensation," *Opt. Express*, vol. 12, no. 11, pp. 2404–2422, 2004.

[6] J. Goodman, *Statistical Optics*. New York: Wiley, 2000.

[7] K. Zhang and J. U. Kang, "Real-time 4d signal processing and visualization using graphics processing unit on a regular nonlinear-k fourier-domain oct system," *Opt. Express*, vol. 18, no. 11, pp. 11 772–11 784, 2010.

[8] W. Wieser, B. R. Biedermann, T. Klein, C. M. Eigenwillig, and R. Huber, "Multi-megahertz oct: High quality 3d imaging at 20 million a-scans and 4.5 gvoxels per second," *Opt. Express*, vol. 18, no. 14, pp. 14 685–14 704, Jul 2010.

[9] Y. Yasuno, V. D. Madjarova, S. Makita, M. Akiba, A. Morosawa, C. Chong, T. Sakai, K.-P. Chan, M. Itoh, and T. Yatagai, "Three-dimensional and high-speed swept-source optical coherence tomography for in vivo investigation of human anterior eye segments," *Opt. Express*, vol. 13, no. 26, pp. 10 652–10 664, Dec 2005.

[10] A. Desjardins, B. Vakoc, M. Suter, S. Yun, G. Tearney, and B. Bouma, "Real-Time FPGA Processing for High-Speed Optical Frequency Domain Imaging," *Medical Imaging, IEEE Transactions on*, vol. 28, no. 9, pp. 1468–1472, 2009.

[11] (2009) NVIDIA CUDA Programming Guide. NVIDIA Corp. [Online]. Available: http://developer.nvidia.com/object/cuda_2_3_downloads.html

[12] (2009) CUDA CUFFT Library. NVIDIA Corp. [Online]. Available: http://developer.nvidia.com/object/cuda_2_3_downloads.html

[13] (2009) CUDA Visual Profiler. NVIDIA Corp. [Online]. Available: http://developer.nvidia.com/object/cuda_2_3_downloads.html

[14] Y. Watanabe and T. Itagaki, "Real-time display on SD-OCT using a linear-in-wavenumber spectrometer and a graphics processing unit," in *Optical Coherence Tomography and Coherence Domain Optical Methods in Biomedicine XIV*, J. A. Izatt, J. G. Fujimoto, and V. V. Tuchin, Eds., vol. 7554. SPIE, 2010, p. 75542S.

[15] (2009) BEE3 Hardware Platform User Manual, rev1.1. BEECube Inc.