APPROVAL

Name: Jian Li
Degree: Master of Applied Science
Title of Thesis: Accelerating Fourier Domain Optical Coherence Tomography Using General Purpose Graphics Processing Units and Field Programmable Gate Arrays

Examining Committee: 

Dr. Albert Leung (P.Eng),
Professor of Engineering Science
Chair

Dr. Lesley Shannon (P.Eng),
Assistant Professor of Engineering Science
Senior Supervisor

Dr. Marinko V. Sarunic (P.Eng),
Assistant Professor of Engineering Science
Supervisor

Dr. Glenn Chapman (P.Eng),
Professor of Engineering Science
Internal Examiner

Date Approved: February 9 2011
Declaration of Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the “Institutional Repository” link of the SFU Library website <www.lib.sfu.ca> at: <http://ir.lib.sfu.ca/handle/1892/112>) and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author’s written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, BC, Canada
Abstract

Fourier Domain Optical Coherence Tomography (FD-OCT) is an emerging biomedical imaging technology that provides ultra high resolution and a fast imaging speed. The complexity of the FD-OCT algorithm demands high processing power from the underlying platform. However, the scaling of faster data acquisition rates and 3-dimensional (3D) imaging on real time FD-OCT systems is quickly outpacing the performance growth of General Purpose Processors (GPPs).

Our research investigates the scalability of two potential platforms for accelerating real time FD-OCT imaging — 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 GPPs. We also created a hardware processing engine using FPGAs, which can deliver more than twice the throughput rate over the GPGPU platform with 1024-point FFT. Our analysis on 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.
Acknowledgments

I would like to thank my supervisor Dr. Lesley Shannon for her guidance throughout my graduate study, for her kindness and patience, and most importantly for giving me the opportunity to pursue higher education and a better life in Canada. I would also like to thank my co-supervisor Dr. Marinko V. Sarunic for helping me with my thesis project and sorting out many of the concepts and ideas that were exotic to me.

I would like to thank Jason Lee, without your motivation I would never had finished my thesis. I would like to thank my lab mates in Reconfigurable Computing Lab — Farnaz Gharibian, who is always willing to listen to my problems and share her experiences as an international student; David Dickin, for teaching me how to use ChipScope by walking through the tedious step-up steps, and for inviting me to my first Canadian Halloween party; Eric Metthews, for treating me with his super delicious homemade deserts, pastries and ice cream; Aws Ismail, for being another Macintoch fan in the lab; Kevan Thompson, for being the cheer leader who brings fun and high spirit to the lab; and Zia Jalali, for staying late chatting with me in the lab.

Last but not least, I want to thank my wife, Lu Song, for being my sole strength and will to survive overseas alone; and my parents, who are always there for me.
Contents

Approval ii
Abstract iii
Acknowledgments iv
Contents v
Glossary viii
List of Tables x
List of Figures xi

1 Introduction 1
  1.1 Motivation ................................................. 2
  1.2 Objective ................................................ 2
  1.3 Contributions .......................................... 3
  1.4 Thesis Organization .................................... 4

2 Background 5
### 2.1 Real Time FD-OCT System

- 2.1.1 Data Acquisition Stage ........................................... 5
- 2.1.2 Processing Stage .................................................... 8
- 2.1.3 Display Stage .......................................................... 12

### 2.2 Accelerating FD-OCT Processing .................................... 12

### 2.3 Previous Work ................................................................ 13

- 2.3.1 GPGPUs and FPGAs for FD-OCT ................................. 13
- 2.3.2 Other Works on GPGPU and FPGA ............................ 15

### 3 GPGPU Implementation .................................................... 17

- 3.1 Introduction to GPGPU ................................................... 17
  - 3.1.1 The GPGPU Architecture ......................................... 17
  - 3.1.2 Framework for GPGPU ............................................. 18
- 3.2 Mapping FD-OCT onto GPGPU ...................................... 20
- 3.3 Experimental Setup .................................................... 20
- 3.4 GPGPU Results ............................................................ 23

### 4 FPGA Implementation ....................................................... 32

- 4.1 FPGA Architecture ..................................................... 32
- 4.2 Mapping FD-OCT onto FPGA ...................................... 34
- 4.3 Processing Modules for the FPGA Processing Engine .......... 35
  - 4.3.1 DC Removal ......................................................... 35
  - 4.3.2 Resample ............................................................ 36
  - 4.3.3 Dispersion Compensation ........................................ 37
  - 4.3.4 Fast Fourier Transform .......................................... 38
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog Digital Converter</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BEE3</td>
<td>Berkley Emulation Engine 3</td>
</tr>
<tr>
<td>BRAM</td>
<td>Block RAM</td>
</tr>
<tr>
<td>CT</td>
<td>Computed Tomography</td>
</tr>
<tr>
<td>CUDA</td>
<td>Computed Unified Device Architecture</td>
</tr>
<tr>
<td>FD-OCT</td>
<td>Fourier Domain Optical Coherence Tomography</td>
</tr>
<tr>
<td>FF</td>
<td>Flip-flop</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-In-First-Out</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Arrays</td>
</tr>
<tr>
<td>FSL</td>
<td>Fast Simplex Links</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>GigaE</td>
<td>GigaBit Ethernet</td>
</tr>
<tr>
<td>GPGPU</td>
<td>General Purpose Graphics Processing Unit</td>
</tr>
<tr>
<td>GPP</td>
<td>General Purpose Processor</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>IPP</td>
<td>Intel Performance Primitives</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-Up Table</td>
</tr>
<tr>
<td>MPPA</td>
<td>Massively Parallel Processor Arrays</td>
</tr>
<tr>
<td>MRI</td>
<td>Magnetic Resonance Imaging</td>
</tr>
<tr>
<td>OpenCL</td>
<td>Open Computing Language</td>
</tr>
<tr>
<td>PET</td>
<td>Positron Emission Tomography</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
</tr>
<tr>
<td>XPS</td>
<td>Xilinx Platform Studio</td>
</tr>
</tbody>
</table>
# List of Tables

3.1 Specifications for the GPGPUs in discussion. .......................... 21
3.2 Comparing to previous work using GPGPU. .......................... 28

4.1 Resource Usage for the Processing Blocks on BEE3. ................. 42
4.2 Resource Usage for the Processing Blocks on a Virtex 6 LX130T ... 42
4.3 FPGA: Comparing to previous works. .................................. 44

5.1 Different data types used in the GPP implementation. ............... 47
5.2 Percent Difference: GPGPU vs. GPP (Independent). ................ 49
5.3 Percent Difference: GPGPU vs. GPP (Integrated). .................. 49
5.4 Percent Difference: FPGA vs. GPP (Independent). .................. 56
5.5 Percent Difference: FPGA vs. GPP (Integrated). .................... 56
List of Figures

2.1 A spectrometer based real-time FD-OCT system. . . . . . . . . . . . . . . . . . . . 6
2.2 A typical swept-source FD-OCT system. . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.3 The FD-OCT processing flow. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
3.1 Pseudo code for stream processing. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
3.2 Diagram of a Real-time FD-OCT system. . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.3 The GPGPU algorithm flowchart. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
3.4 Line Throughput Rates versus Increasing Line Batch Sizes. . . . . . . . . . . . . . 25
3.5 The percentage of the GPGPU function runtime. . . . . . . . . . . . . . . . . . . . . . 27
3.6 Processing-only line rate over different line batch sizes on two GPUs. . . . . . . . 29
3.7 Two sample retina images from GPP and GPGPU. . . . . . . . . . . . . . . . . . . . . 31
4.1 A simplified block diagram of FPGA. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.2 Replicated FD-OCT Processing-Pipeline. . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.3 The block diagram of DC Removal module. . . . . . . . . . . . . . . . . . . . . . . . . 36
4.4 The block diagram of Resample module. . . . . . . . . . . . . . . . . . . . . . . . . . . 36
4.5 The block diagram of Dispersion Compensation. . . . . . . . . . . . . . . . . . . . . . 37
4.6 The block diagram of FFT module. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
4.7 The block diagram of Logarithmic Scaling module. . . . . . . . . . . . . . . . . . . . . 39
Chapter 1

Introduction

Tomography is an imaging technology that produces two-dimensional (2D) images of internal structures by measuring the difference between the energy passing through the object and its echo or reflection, and is widely used for clinical and biomedical imaging. For instance, Computed Tomography (CT), Positron Emission Tomography (PET) and Magnetic Resonance Imaging (MRI) are different tomography technologies that have been adopted for diagnostic purposes. Fourier Domain Optical Coherence Tomography (FD-OCT) is a tomography technology based on optical interferometry, which is rapidly gaining popularity for cross-sectional imaging in biological tissues and materials. FD-OCT provides a fast imaging speed and ultra high image resolution in a micrometer scale, it is therefore regarded as ideal for ophthalmic imaging by clinicians and researchers.
CHAPTER 1. INTRODUCTION

1.1 Motivation

Fast processing speed and high image quality are crucial for real-time FD-OCT systems. For biomedical imaging, it is essential that data acquisition is performed at high speed in order to avoid image motion artifacts due to subjects’ movements. Furthermore, clinicians and researchers are interested in creating three-dimensional (3D) volumetric rendering from 2D FD-OCT images in real time.

However, current software FD-OCT systems on general purpose processors cannot keep up with the increasing imaging speed and the growing output data size from newer acquisition devices. Our current software FD-OCT system (from the Biomedical Optics and Research Group of Simon Fraser University) is able to deliver a throughput rate of about 30 MB/s [1], but it has been demonstrated that the processing demand from the latest data acquisition technology would require a throughput speed-up of approximately 330 times [2]. Moreover, 3D volumetric rendering in real time will push the demands for speed even higher. For this reason, alternative platforms capable of scaling for newer and faster image acquisition technologies are of great interest to FD-OCT researchers.

1.2 Objective

While previous works that use either General Purpose Graphical Processing Units (GPGPUs) or Field Programmable Gate Arrays (FPGAs) to accelerate the FD-OCT algorithm exist [3, 4], they mainly focus on improving imaging algorithms while pursuing faster processing speed. Little has been done to determine how these two platforms may satisfy future processing demands such as real time, 3D volumetric rendering.
CHAPTER 1. INTRODUCTION

The objective for this work is to evaluate how GPGPUs and FPGAs can be used with real-time FD-OCT processing by identifying their limiting factors including performance and scalability. The analysis of the strengths and weaknesses of these platforms will be helpful in proposing a new platform that is scalable with future FD-OCT technologies.

1.3 Contributions

The contributions of this work are as follows,

- a GPGPU accelerated software implementation of a complete FD-OCT system with dispersion compensation. We demonstrate that, for a complete FD-OCT system with a zero-padded 2048-point FFT, the GPGPU is able to achieve an overall system throughput of 207MB/s (maximum line rate of 110kHz), a 6.9x speed up over the software implementation. For an FD-OCT system with a 1024-point FFT, the overall throughput on GPGPU is 224MB/s (maximum line rate of 115kHz).

- a customized FPGA hardware implementation of a FD-OCT processing engine on Xilinx Virtex 5 devices that is fully pipelined; the maximum throughput rate for a single pipeline is 465MB/s (max line rate 238kHz) for complete FD-OCT with 1024-point FFT, and can be scaled for higher speeds by replicating the logic on newer and larger devices.

- an analysis of the FD-OCT algorithm’s performance on GPGPUs and FPGAs, and the scalability of both architectures for future data acquisition rate.
• a quantitative analysis on the output differences of both the GPGPU and FPGA implementations when compared to the original GPP implementation.

1.4 Thesis Organization

This thesis is organized as follows, Chapter 2 outlines the basic concepts of FD-OCT algorithm, as well as summarizing previous research related to FD-OCT acceleration. Chapter 3 describes the GPGPU implementation, while the FPGA hardware implementation is discussed in Chapter 4. Chapter 5 analyzes the output image quality across the three platforms. Finally, Chapter 6 concludes the thesis and comments on future work.
Chapter 2

Background

This chapter describes the basic components of a real time FD-OCT system, and then briefly summarizes the previous work related to our study.

2.1 Real Time FD-OCT System

This section describes the basic components of a real time FD-OCT system. The throughput rate of real-time FD-OCT systems is typically reported in terms of the maximum line rate (the maximum number of lines that can be acquired and processed per second). If a FD-OCT system supports a maximum line rate of 115kHz, with 1024 pixels per line and 16-bit per pixel, the processing throughput rate would be $115kHz \times 1024 \times 2\text{Byte} = 224MB/s$.

2.1.1 Data Acquisition Stage

Depending on the technology used in the data acquisition stage, FD-OCT systems can be subdivided into spectrometer based FD-OCT and swept-source based FD-OCT
Figure 2.1: A spectrometer based real-time FD-OCT system.

**Spectrometer Based FD-OCT**

Figure 2.1 shows a typical spectrometer based FD-OCT system that is used in the current GPP based setup on general purpose processors. In spectrometer based FD-OCT, a broadband light source is used to cast laser beams into both the reference arm and the sample arm. Then an interferometric pattern is generated by the reflected light from both arms. The optical signals of this pattern are then converted into the spectral domain using a spectrometer. A digital line scanning camera is used to scan and record the output of the spectrometer — a “line” of pixels that fall into different locations of spectrum. Multiple scanning lines are collected to form a frame by a frame grabber, which is used to reduce the overhead by transferring data in batches. Following the assembly of one whole frame, the data of this frame is transferred over an *GigaBit Ethernet* (GigaE) interface to the PC for processing, during which time,
a new frame can be assembled. Finally, the resulting data is rendered and displayed onto the screen. Spectrometer based systems typically better suited to acceleration using GPGPUs because they assemble multiple lines of data using the frame grabber before transferring the data for processing. This can be leveraged more efficiently by the GPGPUs’ larger on board memory as will be discussed in Section 3.2.

Swept Source Based FD-OCT

A swept-source system, shown in Figure 2.2, employs a narrow-band, *sweeping light source* to rapidly sweep over a broadband spectrum, producing interference fringes at each instant wavelength [5]. A *photo-detector* converts the interference signals into a series of voltages at a rate of up to 5.2MHz of line rate, which is the fastest scanning speed so far [2]. An Analog-to-Digital Converter (*ADC*) is then used to convert these voltages into digital signals for processing. The recent maximum speed for ADC has reached over 10 Giga samples per second [6], and we expect to see significant increase
in maximum scanning rate from swept-source based FD-OCT systems. As swept-source based systems acquire and output data in a serial fashion, the acquired data can be processed once it is available. Comparing to the spectrometer based FD-OCT where typically multiple lines need be stored in memory for processing, swept-source FD-OCT require smaller amounts of memory for processing and are better suited to FPGAs.

2.1.2 Processing Stage

This section will discuss the processing stage of the system, shown as Processing box in Figure 2.3.

![Figure 2.3: The FD-OCT processing flow.](image)

During the processing stage, the following procedures are executed as individual
functions to obtain the final images.

**DC Removal**

The DC removal procedure subtracts the average DC levels from each scanning line. The DC levels (shown as Avg. DC in Figure 2.3) are the average of the DC components across multiple lines, which can be obtained prior to real time processing because the DC components remains relatively constant across different acquisitions. The DC removal function performed using vector subtraction.

**λ-to-\( k \) Resampling**

This procedure resamples the output data into linear wave-number (\( k \)) space. The output data set from FD-OCT acquisition in both Figure 2.1 and Figure 2.2 is in wavelength (\( \lambda \)) space, which is the common setup provided by most commercial vendors \[7\]. However, in FD-OCT algorithm, data is processed in wave-number (\( k \)) space \[1\]; resampling therefore is needed to re-arrange the data into the desired form. Linear interpolation is used to resample the required data set \( S'[k] \), shown in Equation (2.1)

\[
S'[k] = S[n] + \frac{l'[k] - l[n]}{l[n+1] - l[n]} \cdot (S[n + 1] - S[n])
\]

(2.1)

where \( S[n] \) is the linear data set calibrated to the wavelength value series \( \lambda[n] \) of the camera, and \( l[n] \) is a non-linear series where \( l[n] = \frac{2\pi}{\lambda[n]} \). Finally \( l'[k] \) is a linear series with the same wavelength range as that of \( l[n] \), constructed by

\[
l'[k] = \frac{2\pi}{\max(\lambda[n])} + \frac{2\pi k}{L - 1} \cdot \left( \frac{1}{\min(\lambda[n])} - \frac{1}{\max(\lambda[n])} \right)
\]

(2.2)
where \( L \) is the length of the vector and \( k = 0, 1 \ldots L - 1 \). \( l'[k] \) should be in the interval of \( l[n] < l'[k] < l[n+1] \).

Also, because the sampling frequency and its range (shown as \textit{Freq. & Range} in Figure 2.3) for a specific image acquisition is fixed, the ratio

\[
\frac{l'[k] - l[n]}{l[n+1] - l[n]} \tag{2.3}
\]

can be obtained prior to the real time FD-OCT processing. As a result, Equation (2.1) can be expressed as:

\[
S'[k] = S[n] + c[k] \cdot (S[n+1] - S[n]) \tag{2.4}
\]

, where

\[
c[k] = \frac{l'[k] - l[n]}{l[n+1] - l[n]} \tag{2.5}
\]

is a vector with values that are constant to a specific acquisition. The \( \lambda \)-to-\( k \) resampling function contains vector add, multiplication, as well as re-ordering.

\textbf{Dispersion Compensation}

This procedure is optional but is crucial to obtaining images with high resolution. Broadband light has frequency dependence in the materials it propagates, it is therefore necessary to match the dispersion caused by differences between the sample and reference arms in order to achieve optimal resolution consistent to the broadband light used for FD-OCT [8].

The main component of this function is a Hilbert Transform [8, 9], which is used
to construct a complex number representation $\hat{S}[k]$ from the sampled data $S'[k]$, so that $\hat{S} = |S'|e^{i\phi(\omega)}$. The compensation is achieved by tuning the phase coefficient $a_2$ and $a_3$ (shown as Phase Coe. in Figure 2.3) of the phase function $\phi(\omega)$ [8],

$$\phi(\omega) = -a_2(\omega - \omega_0)^2 - a_3(\omega - \omega_0)^3 \quad (2.6)$$

The Hilbert Transform is implemented using a forward Fast Fourier Transform (FFT), vector multiplication in complex numbers and an inverse FFT (IFFT).

**Fast Fourier Transform**

A Fast Fourier Transform is used to convert the interferometric fringes $S'[k]$ into $s(z)$, which represents the distance or depth information of the layers. $s(z)$ is a symmetric vector where the second half of the vector (from $N/2$ to $N-1$) mirrors the first half [10]. For example, if the raw data for FD-OCT processing is a line of 1024 samples in $k$-space, the output of a 1024-point FFT only gives unique information in 512 output samples along the $z$-axis. The output data is uniformly distributed on the $z$-axis, with a separation between pixels, $dz$, related to the total spectral bandwidth of the raw data, and the number of samples. In order to increase the precision of the distance measurement in the output signal, the original spectral data can be zero padded. The zero padding step does not change the Point Spread Function (resolution of the FD-OCT measurement), but smooths out the line profile in output and improves image quality. The GPP implementation provides the option to zero pad to raw data to 2048 points, and only one half of the symmetric Fourier transformed output, 1024 points, are preserved for display.

While FFTs are performed in floating point in GPGPU, they used fixed point
representations (int16 to be exact) in the GPP and FPGA implementations. Scaling factors are used for the fixed point FFTs in both GPP and FPGA platform to preserve the required precision, because the internal data width of FFT increases with each multiplication and summation, but the final output data width remains unchanged as it uses the same int16 as the input.

**Logarithmic Scaling**

Logarithmic scaling is the final processing step before the image display to obtain a better image quality especially for the visibility of the details.

### 2.1.3 Display Stage

Finally, the processed data is organized in frames or volumes for displaying. For the human brain to perceive a moving 2-dimensional image, a refresh rate of at least 20 frames/s is required, which translates to approximately a throughput of 20MB/s (or 10kHz in terms of line rate) for images with dimension of 1024 by 512 pixels. 3D volumetric real-time displaying requires a throughput that is significantly higher.

### 2.2 Accelerating FD-OCT Processing

FD-OCT can be accelerated by processing data in parallel. As discussed in Section 2.1.2, FD-OCT processing mainly involves vector operations in complex numbers. Significant speed up can be achieved by processing individual lines in parallel, because there is no data correlation between any two scanning lines (vectors). Current software FD-OCT systems on general purpose processors, even equipped with
vector processing primitives on the most up to date processor, are not able to exploit the parallelism of FD-OCT efficiently, because the architecture of general purpose processors is for a serial programming model.

GPGPUs and FPGAs are able to accelerate the algorithm by leveraging both spatial parallelism and pipelining. GPGPU architectures exploits spatial parallelism with their massively parallel processing cores. The Single Instruction Multiple Data (SIMD) feature of GPGPU provides an efficient way to process vector operations in parallel. Spatial parallelism can be achieved on FPGAs by replicating the processing logic to process different lines simultaneously. The circuit designs can be pipelined so that data can be processed at every clock cycle.

2.3 Previous Work

Both GPGPUs and FPGAs have been used to accelerate FD-OCT, however, little has been done regarding the two platforms on performance comparisons or scalability analysis.

2.3.1 GPGPUs and FPGAs for FD-OCT

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 and achieved a real time 2-D display of 27.9 frames/sec. But their only processing steps are the DC removal, FFT and Logarithmic Scaling
(recall Figure 2.3). Zhang et al [7] used a FX5800 GPGPU to accelerate an FD-OCT system similar to ours (but without Dispersion Compensation). They achieved line rates of 680kHz for 1024 pixels and 320kHz for 2048 pixels. However, these line rates only reflect the GPGPUs 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. More recently, Sylwestrzak et al. [11] implemented all the FD-OCT processing steps, and is able to achieve a real time 2-D display frame rate of 100 frames per second (frame size 1024 lines by 1024 pixels) using the same GPGPU as in [3]; the authors also reported a 3-D real-time rendering speed of 9 volumes per second, with a volume size of 100 frames (frame size 100 lines by 1024 pixels).

There has also been some work on FD-OCT accelerators that use FPGAs. Desjardins et al. [12] used two unspecified Virtex 2 devices integrated into their data acquisition system to perform 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, all these previous works that use GPGPUs or FPGAs to accelerate
FD-OCT mainly focus on the experimental setups and imaging algorithms, 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.

2.3.2 Other Works on GPGPU and FPGA

Comparisons between GPUs and FPGAs as accelerators have been performed on a variety of applications. Cope et al. evaluate FPGAs and GPUs for accelerating video processing, in which the strength and weakness from both platforms are discussed, as well as the grounds for comparing the two platforms [13].

Xue et al. [14] explored the performance and accuracy of using GPGPUs and FPGAs for solving a sample 2D/3D CT reconstruction problem, and demonstrated that at least an order of magnitude of speed-up could be achieved from these platforms, while maintaining similar feature clarity and noise levels to the GPP. However, the study is towards off-line image reconstruction and based on relatively small image sizes therefore does not have strict constraints on real-time system throughput, while our work focuses on real-time system performance.

Che et al. [15] presented a comparative study on implementing three computationally intensive applications (Gaussian Elimination, Data Encryption Standard and Needleman-Wunsch) on GPGPUs and FPGAs, and proposed suggestions on how to choose between these two platforms based on the application behaviors. The authors did not make special effort to optimize the performance of each application on either platform, and rely solely on cycle counts for performance comparison. Moreover, the FPGA implementation details (resource usage, maximum clock frequency) are not
given in the paper, therefore the performance comparison is more qualitative than quantitative.

Thomas et al. [16] gives a comprehensive study for comparing four platforms, CPU, GPGPU, FPGA and Massively Parallel Processor Arrays (MPPAs), on generating random numbers in three kinds of distributions (uniform, Gaussian, and exponential). The authors first selected the most appropriate algorithm on each platform for generating random numbers for each distribution, and then compared the peak generation rate (throughput) and power efficiency across all platforms over three kinds of generators. The results show that FPGA provides one order of magnitude more performance per joule than any other platform.

To summarize, both GPGPUs and FPGAs are being actively exploited as low cost, commercially available accelerators for various applications. We believe that our study on comparing GPGPUs and FPGAs for accelerating FD-OCT will not only be useful to the FD-OCT community, but will be beneficial to researchers who are interested in exploiting parallelism from these two platforms in general.
Chapter 3

GPGPU Implementation

This chapter discusses the software implementation of FD-OCT using Computed Unified Device Architecture (CUDA) from NVIDIA Corp., one of the frameworks for GPGPUs [17].

3.1 Introduction to GPGPU

GPGPU stands for “General Purpose Graphics Processing Unit,” where high level programming languages are used to leverage the computation power from commodity video cards to solve non-graphics specific problems [18].

3.1.1 The GPGPU Architecture

Driven originally by processing demands from graphics and gaming applications, the development of GPGPUs is gaining a tremendous momentum to another direction, where their computational power is harnessed for general computations. GPGPUs are designed for graphics applications with the goal that millions of pixels can be
manipulated at the same time, and the architecture of GPGPUs adopts many features optimized for this specialized task. Because of the specialized architecture of GPGPU, it is easier to add more transistors to scale for the computation power than General Purpose Processors (GPPs) [19]. Also for the same reason, GPGPUs have seen a tremendous growth in their computation capabilities faster than Moore’s Law had predicted [19][20].

On the other hand, the GPGPU architecture also poses challenges for general use — GPGPU’s inherent architectural differences from GPP’s make it impossible to share code between the two platforms, and the GPGPU programming environment is also tightly constrained by its architecture due to GPGPUs’ rapid architectural changes from generation to generation. Several programming frameworks have been developed for general programming on GPGPUs.

3.1.2 Framework for GPGPU

The Brook framework provides an open source compiler and runtime implementation to use modern graphics hardware for general purpose computations [21]. The CUDA framework is introduced by NVIDIA in 2007 on NVIDIA GPGPUs. ATI Technology (now owned by AMD) introduced the FireStream framework to provide general processing on ATI GPGPUs. Apple Inc. initiated the OpenCL (Open Computing Language) standard for cross-platform heterogeneous computing including GPGPUs and CPUs, and has been adopted and supported by NVIDIA, AMD and IBM. Microsoft also provides an application programming interface (API) that supports the use of GPGPUs on Microsoft Vista and Windows 7. Brook, CUDA and OpenCL all share the concept of stream processing to exploit parallelism on GPGPUs.
CHAPTER 3. GPGPU IMPLEMENTATION

/* 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 3.1: Pseudo code for stream processing.

Stream processing divides the data into subsets, called streams, on which similar operations can be performed independently and simultaneously. The term kernel is used to refer to such operations and each kernel’s execution is called a thread. Figure 3.1 shows the pseudo code for an 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 thread of execution of the add kernel is differentiated by the thread index i and j. Finally the kernel is called for vector addition on each element (stream). Stream processing exploits spatial parallelism.
3.2 Mapping FD-OCT onto GPGPU

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 the different phases of algorithm outlined in Figure 2.3.

- 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 $\lambda$-to-$k$ re-sampling, etc.).

3.3 Experimental Setup

Figure 3.2 shows the block diagram of the FD-OCT system used in this work, which is similar to Figure 2.1 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-lane PCI-Express v2.0 bus as in the comparative works. Table 3.1 summarizes the specifications for the GPGPUs used for in previous works (column 2 and 3) as well as in our study (column 4). We are
CHAPTER 3. GPGPU IMPLEMENTATION

Figure 3.2: Diagram of a Real-time FD-OCT system.

Table 3.1: Specifications for the GPGPUs in discussion.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Cores</td>
<td>240</td>
<td>240</td>
<td>2x240</td>
<td>480</td>
</tr>
<tr>
<td>Device Memory (MB)</td>
<td>4096</td>
<td>2048</td>
<td>2x898</td>
<td>1536</td>
</tr>
<tr>
<td>Core Frequency (MHz)</td>
<td>610</td>
<td>648</td>
<td>576</td>
<td>700</td>
</tr>
<tr>
<td>Shader Frequency (MHz)</td>
<td>1296</td>
<td>1476</td>
<td>1242</td>
<td>1407</td>
</tr>
<tr>
<td>Memory Frequency (MHz)</td>
<td>800</td>
<td>1242</td>
<td>999</td>
<td>1848</td>
</tr>
<tr>
<td>Price (USD)</td>
<td>3000</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

currently investigating the latest Fermi architecture GTX480 (column 5), which will be discussed in the result section.

We used the CUDA (version 2.3) [22] to develop and debug the FD-OCT algorithm on the GTX295 GPGPU. Figure 3.3 shows the algorithm flowchart, as well as the data flow between the Host (CPU and main memory) and the Device (GPGPU and device memory). Pending availability, one or more frames of data is transferred over to the GPGPU (memcpyHtoD, memory copy from host to device) for processing as a batch to amortize the cost of the transfer. These memory transfers are not due to the GPGPU’s situation on a PCI-Express card, instead, the memory transfers
are inherent to the GPGPU architecture [22]. Even integrated GPGPUs (e.g. the NVIDIA GeForce 9400M), which reside on the same motherboard as the CPU, will require these memory transfers between host and device memory.

While we created most of the kernels used in the FD-OCT processing, we used the CUFFT library from NVIDIA [23] to process the FFT in two of the processing steps, specifically the Dispersion Compensation and Fast Fourier Transform procedures. Finally, as our GTX295 accelerator cannot directly display the results from the GPGPU memory onto the screen, the GPGPU’s processed data is copied back to
CHAPTER 3. GPGPU IMPLEMENTATION

the CPU (*memcpyDtoH*) for rendering and display\(^1\).

### 3.4 GPGPU Results

We measured the total FD-OCT processing time including the memory transfer time between host and device using *cudaprof* [24], 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 previously mentioned, GPGPUs are configured as co-processors, without direct access to the main memory of the host processor in a typical workstation. As such, we wanted to ensure that we transferred sufficient lines for processing in a single memory copy to amortize the cost of the data transfer when producing this profile\(^2\). Figure 3.4 shows the maximum system throughput in two scenarios — when a zero-padded 2048-point FFT is selected (Figure 3.4a) and when a 1024-point FFT is selected (Figure 3.4b). The y-axis is the system throughput in terms of line rate, while the x-axis represents the number of lines that are copied as a batch (i.e. *Batch Size*) using the memory copies to and from the GPGPU.

Three system configurations are plotted in each scenario. In all cases, the system’s throughput rate increases with the batch size, but plateaus after reaching the batch size of 2048. The maximum throughput rate is achieved at 8192 lines. The system

\(^{1}\)Ideally, the post-processed data should be directly copied into the frame buffer on the GPGPU for display without this additional copy from the device to the host, thus we are currently investigating the possibility.

\(^{2}\)Due to the overhead incurred from initiating data transfers between device and host memory, data sets need to be “batched” into larger blocks to amortize this cost [22].
configuration with the highest throughput rate per batch size for both scenarios is \textit{Memcpy Excl.}, which excludes the time required for the memory copy and represents only the processing throughput achievable by the GPGPU. The system with the lowest throughput rate per batch size, \textit{Memcpy Incl.}, accounts for both memory copies via the 16-lane PCI Express Bus used in our system. As the data transfer time between the CPU and GPGPU requires most of the processing time, there is greater variation in the results obtained for different batch sizes in the \textit{Memcpy Excl.} plot than in the \textit{Memcpy Incl.} plot.
Figure 3.4: Line Throughput Rates versus Increasing Line Batch Sizes.
The third system configuration we plotted in Figures 3.4a and 3.4b is an extrapolation of the potential throughput rate that our GPGPU implementation would have if the GTX295 were integrated on the motherboard with the host CPU (Intg. GPU). As previously mentioned, GPGPUs integrated on the motherboard with the CPU are available. However, GPGPUs sold in these configurations are low cost, low power solutions with fewer processing cores, and do not have the processing power of the GTX295 we are using. Therefore, for this third plot, we assumed the processing time was the same as that found using the GTX295 in the Memcpy Excl. plot. We then used an integrated NVIDIA GeForce 9400M, which also requires CPU data to be “transferred” to the device memory on the same memory chip, to measure the time needed for data transfers between the CPU and GPGPU. The extrapolated plot, Intg. GPU, represents the summation of the data transfer time on the NVIDIA GeForce 9400M plus the processing time on the GTX295. Interestingly, this extrapolation demonstrates that it is the memory functions (memcpyHtoD and memcpyDtoH), and not the data transfer via the PCI-Express Bus, that accounts for the majority of the performance loss relative to the pure GPGPU processing time (Memcpy Excl.). In fact, as seen in Figure 3.4, this extrapolation demonstrates that the integrated GPGPU only improves the effective line rate of our actual system (Memcpy Incl.) by 22% when a 2048-point FFT is selected and by 28% when a 1024-point FFT is selected.

Figures 3.5a and 3.5b illustrate the percentage of the FD-OCT algorithm’s run-time for both 2048-point FFT and 1024-point FFT as attributed to its various component functions for a batch size of 8192 lines. Similar results were seen in Watanabe et al. [25], but they didn’t include dispersion compensation, a key component in high
resolution FD-OCT. Figure 3.5 shows that the processing phases of the FD-OCT algorithm (DC-Removal, Resample, Dispersion Compensation, FFT and Logarithmic Scaling) account for approximately 40% of the total run-time time when a 2048-point FFT is selected, while the memory (data) transfers (host-to-device and device-to-host) require approximately 60% of the time. When a 1024-point FFT is selected, the processing phases runtime is 37% while the percentage for the memory transfer time goes up to 63%. For a frame with 1024 by 512 pixels processed with a zero-padded 2048-point FFT, 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 throughput rate available on GPPs, 30MB/s, this translates into a 6.9x increase in throughput. When processed with a 1024-point FFT, the overall system throughput is approximately 224MB/s or a line rate of 115kHz.

To compare with the most recent previous work (see Table 3.2), we exclude the dispersion compensation and data transfers and compare processing-only throughput
### Table 3.2: Comparing to previous work using GPGPU.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Removal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Resample</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Disp. Comp.</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FFT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Log. Scaling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Name</th>
<th>FX5800</th>
<th>GTX285</th>
<th>GTX285</th>
<th>GTX295</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>1250MB/s</td>
<td>N/A</td>
<td>N/A</td>
<td>1093MB/s (2656MB/s w/o Disp. Comp.)</td>
</tr>
<tr>
<td>(Proc. Only)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>256MB/s</td>
<td>109MB/s</td>
<td>200MB/s</td>
<td>207MB/s (265 MB/s w/o Disp. Comp.)</td>
</tr>
</tbody>
</table>

For lines with 2048 pixels. Whereas Zhang et al.’s system had a maximum throughput of 1250MB/s (320 kHz in terms of line rate) on the FX5800 GPGPU [7], our system’s maximum throughput is 2656MB/s (680 kHz in terms of line rate) 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 (recall Table 3.1). As the dispersion compensation phase accounts for more than 50% of the overall processing time, it slows our maximum processing-only throughput to 1093MB/s (line rate 280 kHz). Even with this extra processing step, our system is only around 13% slower than Zhang et al.’s system [7]. We use a GPGPU with slower clock frequency than Watanabe et al. [3], but we are able to achieve a significantly higher throughput rate with two more processing steps. The implementation by Sylwestrzak et al. [11] achieves much faster processing speed than Watanabe et al.’s using the same GTX285 GPGPU, and is only slightly slower
Figure 3.6: Processing-only line rate over different line batch sizes on two GPUs.

than our work. This is because Sylwestrzak et al. reduced the transfer overhead by implementing an extra GPGPU kernel for directly displaying the processed results on the screen without copying them back to the host. However, as the authors did not give a runtime profile of their implementation, we do not know the performance penalty induced by the new kernel versus the performance gained from reducing the transfer overhead.

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. It is an unavoidable overhead for existing GPGPU architectures, which reduces the maximum camera line rate for the complete system implementation to approximately 207 MB/s (110 kHz line rate) for FD-OCT with zero-padded 2048-point FFT, or 224 MB/s (115 kHz line rate) for FD-OCT with 1024-point FFT.
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, thus removing the memory copies altogether. However, even if the discrete model is maintained, the ability to directly render the data to the display would not significantly impact the line rate of the system, increasing it nominally by \( \sim 1.3x \) to a line rate of \( \sim 143kHz \) if a zero-padded 2048-point FFT is selected.

We are also currently investigating the latest Fermi architecture, the GTX480 GPGPU from NVIDIA, which supports duplex data transfers between the host and device. Figure 3.6 illustrates that using the the GTX480 GPGPU more than doubles the processing-only line rates for 2048-point FFT over that of our current GTX295. 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 GPGPU. Furthermore, we are pursuing the use of multiple host threads to exploit the additional spatial parallelism available on multi-GPGPU 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 latest Fermi architecture (\( \sim 610 \) kHz) is still insufficient to meet the current maximum data acquisition rates swept-source based systems (5.2 MHz [2]) 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.

Figures 3.7a and 3.7b show the final output images of a human retina using the GPP and GPGPU platforms respectively. The GPGPU platform is able to deliver comparable image quality to the original GPP implementation, with well defined features of the layers of the internal human retina.
Chapter 4

FPGA Implementation

This chapter first gives a brief introduction of Field Programmable Gate Arrays (FPGAs) and then describes our FPGA implementation of the FD-OCT processing algorithm (recall Figure 2.3). Finally, we discuss the performance and scalability of the FPGA processing engine.

4.1 FPGA Architecture

Figure 4.1 shows a simplified diagram of the FPGA architecture [26]. FPGAs consist of a giant array of configurable logic blocks, interconnected with each other by programmable routing fabrics. The basic elements of a configurable logic block typically include a look-up-table (LUT), a D-type Flip-flop (FF) and a multiplexer, as highlighted in the ellipse labeled \( a \) at the upper right corner of Figure 4.1. Different combinatorial and sequential functions can be implemented by programming the content of the look-up-table and the selecting pin of the multiplexer. The configurable
routing fabric, as shown in the circle labeled b in the bottom right corner of Figure 4.1, contain programmable multiplexers (shown as black dots) to determine how the inputs and outputs of the configurable logic blocks map onto different routing tracks, as well as how the vertical tracks are connected to the horizontal tracks.

In addition to configurable logic blocks, modern FPGAs also provide enhanced IP (Intellectual Property) blocks to improve system performance, as shown ellipse c in the left of Figure 4.1. For example, multipliers mapped onto DSP slices achieve faster clock rates than being mapped onto configurable logic blocks; memory structures, such as buffers, also obtain better area efficiency if mapped onto the Block RAMs (BRAMs) than on LUTs and FFs. Moreover, some FPGA series are embedded with processors blocks to facilitate digital system design.
4.2 Mapping FD-OCT onto FPGA

Based on the nature of the FD-OCT algorithm, we leverage the DSP slices and BRAMs embedded on modern FPGAs to achieve the maximum possible throughput rates for the FPGA implementation of FD-OCT. We also assume that input is provided serially from a swept-source based system (recall Figure 2.2), which allows the lines to be processed individually without requiring intermediate memory to store a complete frame. All calculations are performed using fixed point arithmetic on FPGA, and as with the original GPP software implementation, each sample is assumed to be a 16-bit word. Unlike the GPGPUs, which used single precision floating point, this requires an additional scaling 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 3.2, 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 spatial and temporal parallelism and is fully pipelined, able to receive new data at every clock edge. As some steps 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 4.2 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.

### 4.3 Processing Modules for the FPGA Processing Engine

This section discusses in detail each processing module of the processing engine as shown in the Process box in Figure 2.3. All processing modules use 16-bit fixed-point numbers for data representation, and the processing modules are connected via a pair of Fast Simplex Links (FSL), a unidirectional FIFO IP core for Xilinx FPGAs. The data width for the FSL buses is 32 bits (16 bits for real and 16 bits for imaginary).

#### 4.3.1 DC Removal

Figure 4.3 shows the block diagram for the DC Removal module. The DC Removal module subtracts the average DC levels from the input data. The Xilinx Subtractor IP core is used for the subtract function, which will be mapped onto high speed DSP slices on the FPGA to ensure a fast clock rate. In a real time FD-OCT system, the
average DC level can be acquired prior to the FD-OCT processing and remains within relatively small fluctuation amount across multiple scanning lines, for this reason, we currently assume a constant average DC level for the FPGA design, which is stored in a pre-initialized single ported ROM instantiated from a Xilinx BRAM.

### 4.3.2 Resample

The block diagram of the Resample Module is shown in Figure 4.4. ROM0 serves as a Look-Up-Table for the new order of the current input data sample, while ROM1
CHAPTER 4. FPGA IMPLEMENTATION

contains the vector of linear interpolation coefficients. The content for both ROM0 and ROM1 is fixed for a given data acquisition setup and can be pre-initialized prior to the real time processing. A buffering mechanism is required in the Resample module to ensure that data can be temporarily stored without stalling the entire processing line.

4.3.3 Dispersion Compensation

Figure 4.5 shows the block diagram for the Dispersion Compensation module. Two Xilinx FFT cores are used to implement the Hilbert Transform, and both are configured for 1024-point FFTs, with fully pipelined capabilities to process data at every clock cycle. Also, each FFT core can be configured to obtain optimal performance by mapping both the complex multiplication and butterfly arithmetic on to DSP slices.
However, due to the resource limitations of the FPGA device used for the implementation (Virtex 5 LX155T), we could not configure either FFT core’s butterfly arithmetic onto DSP slices, as this will cause mapping failures. As a result, only complex multiplication of the FFT cores are mapped onto the DSP slices, while the butterfly arithmetic is implemented using LUTs and FFs. Moreover, the phase coefficients are pre-initialized in the single-ported ROM, while a high-speed complex multiplier IP core from Xilinx is used to calculate the complex product from the Hilbert Transform output and the phase coefficients.

### 4.3.4 Fast Fourier Transform

The FFT core in this module (shown in Figure 4.6) is configured with the same 1024-point, fully-pipelined feature as those in the Dispersion Compensation module, except that the complex multiplication and butterfly arithmetic are both mapped onto DSP slices.

### 4.3.5 Logarithmic Scaling

We implement the Logarithmic Scaling module using a look-up table in order to reduce design complexity. The look-up table is implemented using a single-ported ROM, with
an address width of 16 bit (64K entries) and a data width of 16 bit (2 bytes). The 32-bit output from FFT module (16-bit real and 16 imaginary) are truncated to 16-bit as the address of the table, while the contents of the look-up table are calculated using MATLAB, which first computes the logarithmic scaling in floating point and then converts them into 16-bit integers.

4.4 Experimental Setup

The FD-OCT algorithm is written in VHDL and synthesized using version 11.5 of Xilinx Platform Studio (XPS) for Virtex 5 LX 155T devices. We use Xilinx’s Core Generator to create the FFT IP cores (version 7.0), as well as the subtractor core (version 11.0) and multiplier core (version 3.1). As our design requires almost all of the DSP48E 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 [27], 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 4.8 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 Logarithmic Scaling 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 (MB0) and receiving the results (MB1). 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 4.8 and are guaranteed to operate reliably for operating frequencies up to 400MHz [27].

Data acquired using our current spectrometer based system is pre-loaded into MB0’s BRAM. MB0 reads in this data and writes it directly to a FSL bus, which 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.

4.5 FPGA Results

This section provides an analysis for the throughput performance and scalability of the FPGA platform for FD-OCT.

4.5.1 FPGA Resource Usage

The resource usage for each individual processing module on a Virtex5-155T device is listed in Table 4.1. 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 significantly reduces its design complexity.
### Table 4.1: Resource Usage for the Processing Blocks on BEE3.

<table>
<thead>
<tr>
<th>FPGA ID</th>
<th>Modules</th>
<th>6-LUT</th>
<th>FF</th>
<th>BRAM</th>
<th>DSP48E</th>
<th>Max Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># Pct</td>
<td># Pct</td>
<td>KBit</td>
<td>Pct # Pct</td>
<td>(MHz)</td>
</tr>
<tr>
<td>DC Removal</td>
<td>71 1%</td>
<td>19 1%</td>
<td>18 1%</td>
<td>1 1%</td>
<td>246</td>
<td></td>
</tr>
<tr>
<td>FPGA 0</td>
<td>Resample</td>
<td>148 1%</td>
<td>178 1%</td>
<td>216 7%</td>
<td>3 2%</td>
<td>268</td>
</tr>
<tr>
<td>MB0</td>
<td>1237 1%</td>
<td>1388 1%</td>
<td>2048 27%</td>
<td>3 2%</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>FPGA 1</td>
<td>Disp. Comp.</td>
<td>9166 9%</td>
<td>10962 11%</td>
<td>324 4%</td>
<td>64 50%</td>
<td>244</td>
</tr>
<tr>
<td>FPGA 2</td>
<td>FFT</td>
<td>3548 3%</td>
<td>4304 4%</td>
<td>162 2%</td>
<td>56 43%</td>
<td>276</td>
</tr>
<tr>
<td>Log Scaling</td>
<td>112 1%</td>
<td>14 1%</td>
<td>1080 14%</td>
<td>- -</td>
<td>358</td>
<td></td>
</tr>
<tr>
<td>MB1</td>
<td>1237 1%</td>
<td>1388 1%</td>
<td>2048 27%</td>
<td>3 2%</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>FPGA 0,1,2 Overall</td>
<td>15519 -</td>
<td>18253 -</td>
<td>5896 -</td>
<td>130 -</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.2: Resource Usage for the Processing Blocks on a Virtex 6 LX130T

<table>
<thead>
<tr>
<th>Modules</th>
<th>6-LUT</th>
<th>FF</th>
<th>BRAM</th>
<th>DSP48E1</th>
<th>Max Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Pct</td>
<td># Pct</td>
<td>KBit Pct</td>
<td># Pct</td>
<td>(MHz)</td>
</tr>
<tr>
<td>DC Removal</td>
<td>32 0%</td>
<td>15 0%</td>
<td>18 0%</td>
<td>1 0%</td>
<td>-</td>
</tr>
<tr>
<td>Resample</td>
<td>156 0%</td>
<td>193 0%</td>
<td>216 2%</td>
<td>3 0%</td>
<td>-</td>
</tr>
<tr>
<td>Disp. Comp.</td>
<td>4406 5.5%</td>
<td>9393 5.9%</td>
<td>324 4%</td>
<td>118 24.6%</td>
<td>-</td>
</tr>
<tr>
<td>FFT</td>
<td>2095 2.6%</td>
<td>4503 2.8%</td>
<td>162 2%</td>
<td>56 11.7%</td>
<td>-</td>
</tr>
<tr>
<td>Log Scaling</td>
<td>90 0%</td>
<td>22 0%</td>
<td>1080 11.4%</td>
<td>- -</td>
<td>-</td>
</tr>
<tr>
<td>Overall</td>
<td>6779 8.1%</td>
<td>14136 8.7%</td>
<td>1800 19%</td>
<td>178 36.3%</td>
<td>238</td>
</tr>
</tbody>
</table>
Recalling from Section 4.2 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 2.2, where data can be acquired at every clock cycle, the maximum overall system throughput is 465MB/s or a line rate of 238kHz. Recalling that the maximum throughput of the GPGPU based system with 1024-point FFT is 224MB/s, the FPGA based system more than doubles the throughput rate over the GPGPU platform. 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, 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.

While Table 4.1 provides the maximum system throughput if the processing engine is implemented onto multiple FPGAs, Table 4.2 shows the resource usage if the entire pipeline is mapped onto one single device — the Virtex 6 LX130T. We are able to reconfigure both FFT cores in the Dispersion Compensation module to be fully optimized with all butterfly arithmetic being mapped onto DSP48E1 slices, an enhanced version of the DSP48E slice from the Virtex 5 family. For this reason, the LUT usage for the Dispersion Compensation module is reduced by 51% comparing to Table 4.1, while the DSP48E1 usage is 118. The percentages of used resource drop because the Virtex 6 LX130T is a larger device than the Virtex 5 LX155T. The maximum operating frequency for the FD-OCT engine is 238MHz on the Virtex 6 LX130T device, which translates to a overall system throughput of 454MB/s, almost identical to that from the BEE3 platform.
Table 4.3: FPGA: Comparing to previous works.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Removal</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Resample</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Disp. Comp.</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FFT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Log. Scaling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Device</strong></td>
<td>2 Virtex-2</td>
<td>Virtex-4</td>
<td>3 Virtex-5</td>
</tr>
<tr>
<td></td>
<td>FX12</td>
<td>LX155T</td>
<td>LX130T</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>5MB/s</td>
<td>27MB/s</td>
<td>465MB/s</td>
</tr>
</tbody>
</table>

Table 4.3 summarizes previous work that use FPGA for FD-OCT acceleration. Our design achieves significant higher throughput rate than both Desjadins’ [12] and Ustun’s [4], probably due to a better design mapping using the embedded IP blocks on larger and new devices. The slight throughput difference (<3%) between the Virtex 5 and Virtex 6 implementation is probably due to a change in the critical paths when the CAD tools place-and-route the design.

The next phase of this work will investigate a suitable scaling schedule for the FFT cores as well as for the entire system so as to better match the output from GPP platform. We are also interested in 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.

Figures 4.9a and 4.9b show two images of the human retina, processed using the GPP and FPGA platform respectively. The image output from the FPGA processing engine does not illustrate layers of the internal structure as clearly as the GPP output, and the outlines of the FPGA image are blurry and not well separated from the background. Furthermore, the image quality from our FPGA engine is not as good as that achieved in previous work [4][12] using FPGAs for FD-OCT acceleration. We will discuss the causes of the FPGA’s image quality degradation in the next chapter.
Chapter 5

Image Quality Analysis

This chapter presents a quantitative analysis on the output differences from both the GPGPU and the FPGA FD-OCT system when compared to the original GPP implementation. The final output images from each platform are also included for qualitative comparisons.

5.1 Quantitative Analysis: GPGPU vs. GPP

This section compares the GPGPU implementation’s output to the original GPP implementation’s output. The original software FD-OCT processing system is implemented using standard C++, where a range of data types are used in different processing steps, as shown in Table 5.1. While the main data structure for storing the processing data is maintained in an array of int16, the Resample and Dispersion Compensation steps used doubles and floats for processing. Using different data types in C/C++ requires typecasting, where all data types are first converted into
Table 5.1: Different data types used in the GPP implementation.

<table>
<thead>
<tr>
<th>Processing Step</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Removal</td>
<td>int16</td>
</tr>
<tr>
<td>Resample</td>
<td>int16, double</td>
</tr>
<tr>
<td>Dispersion Compensation</td>
<td>int16, float</td>
</tr>
<tr>
<td>FFT</td>
<td>int16</td>
</tr>
<tr>
<td>Logarithmic Scaling</td>
<td>int16</td>
</tr>
</tbody>
</table>

one data type with the highest precision before processing [28]. In the GPP’s Resample function, where a mix of double and int16 are used, the int16 type is first typecast into double type for processing. As the main data structure in the GPP is int16, the resulting data from the Resample is typecast back from double into int16.

We observe a casting difference on the GPP implementation when both the Resample and Dispersion Compensation are used in processing. Specifically, on the completion of the Resample step, and before the start of the Dispersion Compensation step, a type casting occurs to convert an array of doubles into an array of int16s. This changes the output values from the Resample such that the values processed in Dispersion Compensation step differ from the original values. Figures 5.1 show the same line of samples before and after casting respectively, where the values before casting range from 0 to approximately 2400, and the values after casting range approximately from -11000 to 4200. How these precise values are generated is unknown due to the lack of documentation on the typecast functions. However, if both arrays are normalized to 1, the percent difference is less than 0.1%. We suspect that this is because the GPP platform is trying to preserve the precision when casting the data format from a higher precision (double, 64-bit) into a lower precision (int, 16-bit).
However, as stated previously, we don’t know exactly how the casting is computed on the GPP platform as we could not find the relevant documentation.

On the other hand, the data structures and all the processing functions on the GPGPU platform use the float type, as required by the CUFFT library [23]. Type-casting is also used in the GPGPU, specifically, after the raw data array (int16) is copied from the Host to the Device, it is typecast into an array of floats; on the completion of all the processing functions (all in float), the float array is then converted back to int16, which is then copied back to the host.

5.1.1 Independent Processing Step Comparison

To independently evaluate each processing step, the same frame (1024 by 512 pixels) of data is processed on both the GPP and GPGPU implementations. The output values from each individual step on the GPGPU are compared to the same step on the GPP,
and the average percent differences from GPP are listed in Table 5.2. Overall, the individual output from each GPGPU’s processing steps is consistent with the GPP platform as the average percent differences are smaller than 1.2%.

5.1.2 Integrated Comparison

To evaluate the cumulative percent difference in values in a complete system, each of the processing functions on both platforms are incrementally included for processing, and the cumulative output values of the GPGPU functions are then compared to the corresponding GPP functions. The input for the comparison is the same 512 lines of 1024-pixels used in the independent test, and the results are listed in Table 5.3.

Due to the casting differences that we previously mentioned (recall Figure 5.1), the
input value ranges for GPP and GPGPU functions’ Dispersion Compensation function
differ significantly, but are within a 0.1% difference if normalized to 1. For this
reason, we normalize the output values from both platforms to 1 when comparing the
Dispersion Compensation, FFT and Logarithmic Scaling functions. While both the
DC Removal and Resample steps still deliver consistent output on both platforms, the
average difference increases to approximately 11% after the Dispersion Compensation
step. It should be noted that as the non-normalized Dispersion Compensation outputs
from both platforms contain both positive and negative values, normalizing them
into the range from 0 to 1 for comparisons will give lower percent differences because
the values are now “shifted” away from the zero point. However, for the purpose of
analyzing the percent difference to detect overflow and underflow errors, it is sufficient.
The average percent difference is amplified to 23.3% after the FFT step. Besides the
casting differences, this likely results from the different data types (int16 on GPP,
float on GPGPU), as well as different library functions used for the FFT on the
two platforms. In the GPP implementation, the FFT library functions from the
Intel Performance Primitives (IPP) are used for both the Dispersion Compensation
and FFT step, where a scaling factor is employed to ensure proper precision can be
maintained on the int16 array. In the GPGPU implementation, the CUFFT library
function requires floating point data types, therefore no scaling factor is necessary.
The percent difference decreases after the final step, as the output values are now in
logarithmic scale.

Figures 5.2 to 5.4 show the normalized cumulative output values of one sample
line from the Dispersion Compensation to the Logarithmic Scaling steps on both
platforms, as shown on the y-axis on the left. The average percent differences are
Figure 5.2: Integrated GPGPU vs. GPP: Real (top) and Imaginary (bottom) output after the Dispersion Compensation step.

shown on the y-axis on the right, and the x-axis is the sequence number of the samples within one line. Figures 5.3 and 5.4 show only the 512 unique values out of the 1024 output, as only these unique 512 samples are used for displaying (recall Section 2.1.2). All three figures show that the average output differences between the two platforms have a large variation across pixels. For example, the Dispersion Compensation output (Figure 5.2) has outliers with percent differences up to 500%, while percent differences for the majority pixels are much smaller.
Figure 5.3: Integrated GPGPU vs. GPP: Absolute value after the FFT step.

Figure 5.4: Integrated GPGPU vs. GPP: Output after the Logarithmic Scaling step.
Figure 5.5: A Histogram for Integrated GPGPU vs. GPP.
Although average percent differences can be used to indicate the consistency of the GPGPU output when compared to the GPP implementation, they are easily affected by outliers — a small number of pixels that give significantly larger values differences in output. To better understand the distribution of the percent differences across all data samples, we use a histogram to represent the distribution of percent differences for each processed pixels across the entire frame, as shown in Figure 5.5. In Figure 5.5, the y-axis is the percentage of the total samples (pixels) that falls in a specific percent difference range, whereas the x-axis represents the percent differences of the GPGPU output values over the GPP platform and is divided into eleven different percentile ranges. Each bin has five bars for each of the five processing steps, representing the percentage of total output samples whose percent differences that fall into the range of that bin.

Overall, more than 95% of the GPGPU output samples from each of the 5 processing steps are no more than 30% different from the GPP output. Specifically, almost 100% of the DC Removal and Resample output samples from the GPGPU platform are less than 10% different from the GPP output. However after Dispersion Compensation, the percentage of GPGPU’s real and imaginary output samples that are less than 10% different drops to 63% and 59% respectively, while 21% of the real and 22% and imaginary output samples are different from the GPP output by a percentage ranging from 11% to 20%. This is likely caused by the use of different data types and function libraries on the two platforms. Although Table 5.3 shows the average percent difference increases to over 23% after FFT step, over 95% of the GPGPU’s FFT output samples fall within the same bin, which represents the percent difference from 21% to 30%. This indicates that the integrated FFT outputs from both platforms
have a relatively “fixed” pixel-to-pixel difference across more than 95% of the total pixels. This concentration in the range of the percent difference in the FFT output would thus act as more an offset, whereas a distributed error would mask/destroy image characteristics. After Logarithmic Scaling, approximately 64% of the GPGPU output samples are no more than 10% different from the GPP output, while over 35% of the GPGPU output samples are different by a percentage ranging from 11% to 20%. This seemingly improvement is largely due to the fact the differences are now in a logarithmic scale.

5.2 Quantitative Analysis: FPGA vs. GPP

This section provides a quantitative comparison between the FPGA implementation and the original GPP implementation. Unlike both the GPP and GPGPU implementation, the FPGA uses fixed point number representation (int16 to be exact) for all the processing steps. As the bit-width of fixed point data increases over the pipeline after each operations such as multiplication, a scaling method needs to be used to maintain the bit-width. The FPGA design currently employs truncating for scaling.

5.2.1 Independent Processing Step Comparison

For the independent comparison between FPGA and GPP, the same frame of data (1024 by 512 pixels) is used as input for both platforms as in Section 5.1.1, and the average percent differences from the GPP platform for each step are listed in Table 5.4. Overall, all processing modules on the FPGA deliver output values that are on average no more than 8.1% different from the GPP output in the independent
Table 5.4: Percent Difference: FPGA vs. GPP (Independent).

<table>
<thead>
<tr>
<th>Module</th>
<th>Avg. Pct. Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Removal</td>
<td>0.0%</td>
</tr>
<tr>
<td>Resample</td>
<td>5.0%</td>
</tr>
<tr>
<td>Dispersion Compensation (Real)</td>
<td>8.1%</td>
</tr>
<tr>
<td>Dispersion Compensation (Imag)</td>
<td>7.5%</td>
</tr>
<tr>
<td>FFT (Magnitude)</td>
<td>3.0%</td>
</tr>
<tr>
<td>Logarithmic Scaling</td>
<td>1.5%</td>
</tr>
</tbody>
</table>

Table 5.5: Percent Difference: FPGA vs. GPP (Integrated).

<table>
<thead>
<tr>
<th>Module</th>
<th>Avg. Pct. Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC Removal</td>
<td>0.0%</td>
</tr>
<tr>
<td>Resample</td>
<td>5.0%</td>
</tr>
<tr>
<td>Dispersion Compensation (real)</td>
<td>19.8%</td>
</tr>
<tr>
<td>Dispersion Compensation (img)</td>
<td>22.3%</td>
</tr>
<tr>
<td>FFT (Magnitude)</td>
<td>70.4%</td>
</tr>
<tr>
<td>Logarithmic Scaling</td>
<td>24.0%</td>
</tr>
</tbody>
</table>

The differences mainly come from the different scaling methods employed in both platforms, as the GPP platform utilizes the IPP library functions from Intel for the processing functions, while the FPGA implementation simply uses a truncation method for scaling, which is currently hard-coded into the FPGA design. Moreover, according to the IPP documentation [29] from Intel, higher precision numbers are used internally for fixed point computations, but no further detailed information is given in the documentation regarding to the internal data width of the higher precision number used. It is therefore difficult for the FPGA to reproduce the same output of the GPP.
5.2.2 Integrated Comparison

Similar to the integrated comparison in Section 5.1.2, we compare the cumulative differences of the two platforms by incrementally integrating each FPGA processing module into the pipeline and comparing the output values against the same GPP output. We use the same frame of data as input as with the GPGPU integrated comparison, and the average percent differences of the FPGA output values are listed in Table 5.5. Due to the casting differences on the GPP platform mentioned previously, the output from the Dispersion Compensation, FFT and Logarithmic Scaling modules on both platforms are normalized to 1 for comparison.

Both the DC Removal and Resample module show the same output difference as in the independent comparison. The cumulative differences for both the real and imaginary output of the Dispersion Compensation module increased dramatically to 19.8% and 22.3% respectively. As stated previously in Section 5.1.2, these differences would have been larger, had the Dispersion Compensation outputs not been normalized to the range from 0 to 1. The increased differences from the previous step mainly result from the casting differences on the GPP platform, which is likely the same cause as in the integrated GPGPU comparison; however the increase are much larger than seen in the GPGPU comparison. This is because the FPGA implementation uses int16 for data representation, which is more likely to have overflow/underflow error than the 32-bit float used in GPGPU. Moreover, the increased differences of the Dispersion Compensation step may also result from the different methods used for scaling, as the FPGA implementation uses a simple truncating method while the GPP uses IPP library functions with scaling factors. For the same reason, the differences increase is amplified up to approximately 70% after the FFT step, which is also significantly
CHAPTER 5. IMAGE QUALITY ANALYSIS

higher than the increase from GPGPU comparison. After the data is converted to logarithmic scale in Logarithmic Scaling step, the percent difference drops to around 24%.

Figure 5.6: Integrated FPGA vs. GPP: Real output (top) and imaginary output (bottom) after Dispersion Compensation.
CHAPTER 5. IMAGE QUALITY ANALYSIS

Figure 5.7: Integrated FPGA vs. GPP: Absolute value of the output after FFT stage.

Figure 5.8: Integrated FPGA vs. GPP: Output after Logarithmic Scaling stage.
CHAPTER 5. IMAGE QUALITY ANALYSIS

Using the same input line as in Figures 5.2 to 5.4, Figures 5.6 to 5.8 graph the normalized cumulative output values of this line for the same processing steps on both the GPP and FPGA platforms, where the x-axis and the two y-axises have the same meaning as in Figures 5.2 to 5.4. Figures 5.7 and 5.8 only show the 512 unique samples out of the 1024 samples similar to Figure 5.3 and 5.4 respectively. Overall, Figures 5.6 to 5.8 display an even larger variance over the percent differences than the GPGPU comparisons.

A histogram similar to Figure 5.5 is shown in Figure 5.9, where the x-axis and y-axis have the same meaning as in Figure 5.5. Comparing to Figure 5.5, only the DC Removal still keeps almost 100% of the output samples within 10% of difference over the GPP output, while the Resample module only has 88% of its output samples under 10% difference. The percent difference distribution of the Dispersion Compensation is more “spread out” than the GPGPU platform (Figure 5.5) — specifically, for samples that are under 20% difference from the GPP output, the FPGA only has less than 70% of the real output and less than 50% of the imaginary output within this range, while the GPGPU’s Dispersion Compensation has 85% and 80% respectively. For the FFT on the FPGA, more than 75% of the total output is less than 20% different from the GPP output, which suggests that the FFT’s average percent difference of approximately 70% in Table 5.5 is greatly affected by the outlier pixels. However, as the FPGA’s FFT does not show a relatively concentrated distribution in any of the bins of percent differences as the GPGPU’s FFT does, it means the differences from the previous steps are propagated through the pipeline. This can also be seen from the distribution of FFT’s Logarithmic Scaling, which has a similar “spread out” distribution to the FFT.
Figure 5.9: A Histogram for Integrated FPGA vs. GPP.
5.3 A Comparison using Images

This section presents the human retina image processed using the three platforms in the same page to provide an overview for the image quality across the three platforms. Figures 5.10a, 5.10b and 5.10c, which have been presented in Chapter 3 and 4, show the final image output from the GPP, GPGPU and FPGA platforms respectively. The FD-OCT processing includes all the processing steps and the raw data size is 512 lines by 1024 pixels. As the current FPGA implementation only allows 1024-point FFT, both the GPP and GPGPU implementation use the 1024-point FFT option for processing for fair comparisons. As only the 512 unique pixels are collected from each line, the output picture dimension therefore is 512 lines by 512 pixels.

All three platforms are able to reproduce the outlines of the human retina. The GPGPU image provides almost identical quality image features to the GPP image, with clear details and well defined layers of the internal structure. Looking at these three images, significant artefacts are apparent in the FPGA generated image. The layers on the FPGA image are not visible, and the separation between the retina structure and the background is not as well defined as the other two images.

The inferior image quality from the FPGA platform corresponds to the quantitative comparison results discussed in Section 5.2, where the current scaling scheme on the FPGA implementation cannot effectively maintain the required precision, and the overflow/underflow error propagates and accumulates along the pipeline and eventually causes information lost and degradation in image quality. Interestingly, even though the GPGPU’s average integrated percent difference from the GPP implementation at the FFT step reaches up to 23.2% (recall Table 5.3), the final output images from the two platforms are almost visually identical.
Figure 5.10: Image Comparison: GPP vs. GPGPU vs. FPGA
Chapter 6

Conclusions and Future Work

In this thesis, we investigated how GPGPUs and FPGAs can be used to accelerate FD-OCT processing to meet current and future data acquisition rates. We implemented a complete software FD-OCT system using a GPGPU as a co-processor. The overall system throughput, with a zero-padded 2048-point FFT, is 207MB/s (110kHz in terms of maximum line rate), a 6.9x speed up over the original implementation [1]; the throughput increases to 224MB/s (115kHz in terms of line rate) when a 1024-point FFT is applied. We also demonstrate a hardware FD-OCT processing engine on a BEE3 platform, using three Virtex5-155T devices, the maximum throughput is 465MB/s (234kHz in terms of line rate) for FD-OCT processing with 1024-point FFT, achieving more than 2x speed-up over the GPGPU platform.

To keep up with the increasing FD-OCT data acquisition rate, FPGA provides better scalability than GPGPU. The discrete memory model of the current GPGPU architecture imposes additional data transfers between Host and Device 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.

Currently, the GPGPU implementations is able to provide comparable image quality to the GPP implementation, but the image quality from the FPGA implementation is inferior to the other two platforms. This is due to the fact that the FPGA engine uses fixed point FFTs, in which the data bit width is increased along the pipeline and scaling is therefore required to maintain the 16-bit data width on FPGA. However, our current scaling method of truncation causes overflow/underflow errors in the 16-bit data, while the data types used on the other two platforms have higher resolution — the GPP uses higher bit widths internally and the GPGPU uses float. Moreover, while the GPGPU platform provides the same degree of flexibility as the GPP platform for adjusting the FD-OCT processing parameters in real time, the FPGA platform currently pre-initializes the parameters prior to processing. Finally, the design turn-around time and system integration time is much shorter for the software-based GPGPU platform than the hardware FPGA platform.

Future work on the GPGPU system will focus on the implementation of duplex data transfers, as well as directly displaying the processed data using the GPGPU without transferring it back the host. Future work on the FPGA processing engine will focus on improving the output image quality. By employing a better scaling method to reduce the overflow/underflow errors, and/or by increasing the data bit-width of the FD-OCT processing engine, it is possible for the FPGA processing engine to match the image quality from the other two platforms. We are also investigating how to display the FPGA’s output in real time, as well as how to integrate the
FPGA engine with high-speed swept-source based acquisition devices so as to build a complete FPGA-based FD-OCT system. Moreover, we would like to implement a FFT module capable of the zero-padding function used on the GPP and GPGPU platforms for our FPGA processing engine. The long term goal for both the GPGPU and the FPGA will be creating a hybrid FPGA-GPGPU platform that combines the strengths from both sides — a scalable, high performance FD-OCT system capable of real time, high quality 3D volumetric rendering.
Bibliography


BIBLIOGRAPHY


