JEOM Research Summary Paper

Compute Unified Device Architecture (CUDA) Fast Fourier Transform (FFT) benchmarks on the NVIDIA Tesla c2070 graphics-processing unit (GPU)

Mathew R.W. Yamasaki, Greg Newby, Chris Fallen
University of Alaska Fairbanks Arctic Region Supercomputing Center

Abstract

Within the last several years, there has been an increased interest in processing computationally intensive applications such as the discreet Fourier transform (DFT) and its faster sibling, the Fast Fourier Transform (FFT), on commodity graphics processing units (GPUs) because of their high-processing power and relatively low-cost and power consumption when compared to their supercomputing counterparts. However, previous to the release of a proprietary GPU architecture by the NVIDIA Corporation, GPU programming has been difficult, introducing layers of additional complexities to already complex problems.

The project given to me was conducted at the University of Alaska Fairbanks (UAF) Arctic Region Supercomputing Center (ARSC) under the mentorship of ARSC Director, Dr. Greg Newby, and Research Assistant Professor, Dr. Chris Fallen. My mentors have been involved with experiments that use GPU-accelerated FFTs to process very large amounts of complex data received by the Modular UHF Ionosphere Radar (MUIR) at the High-frequency Active Auroral Research Project (HAARP) in Gakona, Alaska [1].

This research paper contains the experimental design, methods used, and benchmarking results of various sized one-dimensional (1D) and two-dimensional (2D) GPU accelerated FFTs. Testing was also conducted to observe the speedups achieved in comparison to FFTs executed on a uniprocessor.

I. Introduction

In November 2006, the NVIDIA Corporation introduced a new, massively parallel GPU architecture model called "*CUDA*" which stands for Compute Unified Device Architecture [4]. CUDA enabled GPUs use NVIDIA's proprietary programming language, *CUDA C*, which consists of familiar C programming language extensions to simplify programmability. This allows programmers to use the GPU as a co-processor to the central processing unit (CPU), thereby creating a heterogeneous computing model where sequential processing is performed on the CPU, and computationally intensive processing is performed to the GPU.

With the release of the CUDA software development kit (SDK), NVIDIA included an FFT package called the *CUFFT Library*, or *CUDA Fast Fourier Transform Library*. CUFFT automatically optimizes the run configuration of the FFT to be executed on the GPU to produce the fastest parallel execution times. This paper will describe the tools and programming methods used to benchmark the execution times of various sized 1D and 2D FFTs.

As a basis for comparison with non-GPU accelerated FFTs, CUFFT will be compared to FFTs executed on a single CPU processor. For this, the FFTW, which stands for "*Fastest Fourier Transform in the West*" will be used. The FFTW is generally accepted to be the fastest, publicly available CPU FFT software package [1]. Matteo Frigo and Steven G. Johnson [2] developed this package at the Massachusetts Institute of Technology (MIT). FFTW is the basis from which the CUFFT application-programming interface (API) is modeled [3].

B. Research Objectives

The massively parallel processing power of the GPU in comparison to the CPU is indisputable. This paper will not attempt to challenge this fact, but will instead describe the research effort undertaken to determine specific FFT sizes that benefit from GPU processing, and the average execution times in which they execute. I expect that smaller sized 1D and 2D transforms will have similar performance.

GPU processing incurs additional execution time costs as a consequence of required data transfers between the CPU and the GPU. For example, input data must first be created on the CPU, transferred to the GPU for processing, and the result transferred back to the CPU. I will determine where and to what degree this additional overhead, which is not incurred by CPU processing, affects overall GPU processing time.

II. Experiment

A. Software Used

- CUDA SDK
  - GPU-accelerated FFT library (CUFFT)
  - Code examples
- NetBeans 6.9.1 integrated development environment (IDE)
- JuxtaCore CUDA-plugin
- FFTW package

All software development and testing was done on NetBeans 6.9.1[1] integrated development environment (IDE). This is the most current version of NetBeans with a stable CUDA-plugin[2].

Many previous attempts were made to develop CUDA programs with Eclipse, another popular IDE with a CUDA-plugin, but I have found NetBeans to be much easier to setup and use.

B. Hardware Used

I performed GPU testing on an NVIDIA Tesla™ c2070 GPU on a Linux-based remote workstation codenamed "*BEACH*". This GPU has a clock rate of 1.15 GHz and uses CUDA driver version 3.2 with a compute capability of 2.0. There are 14 multiprocessors, each having 32 cores for a total of 448 cores.

CPU benchmarks were performed on an Intel® Core™ i7 processor with a clock rate of 3.4 GHz (also located on BEACH). This processor has a total of 4 cores.

An Apple MacBook Pro featuring 4 GB of memory, OS X version 10.6.7, and a 2.53 GHz Intel Core 2 Duo processor was used to access the BEACH system.

C. Program Design

1. CUFFT

The GPU code I wrote was modeled after various examples from NVIDIA's CUDA software development kit (SDK) and CUFFT Library documentation. Additionally, the *NVIDIA CUDA C Programming Guide* and *CUDA Toolkit Reference Manual* were essential tools.

2. FFTW

Writing the FFTW application was very straightforward, as CUFFT is based on the FFTW library. Code was written based on examples from the FFTW online manual. I decided to write the CUFFT application before the FFTW to account for the additional CUDA research requirements.

D. Testing Methodology

For testing uniformity and consistency, Dr. Fallen recommended that I populate 1D and 2D arrays with complex numbers of the form $a + bi$ where a is represented by 1 and b is represented by 0 ($1 + 0i$). Using matrices of 1's for input data allowed us to quickly verify the accuracy of the program without adversely affecting the performance benchmark, as the result would be matrices of 0's.

The data types used, `cufftComplex` for CUFFT programs, and `fftw_complex` for FFTW programs, are single-precision, floating-point complex data types consisting of interleaved real and imaginary parts.

---

1 This is available for download from http://netbeans.org/downloads/6.9.1/.

2 The CUDA-plugin is available from NetBeans (http://plugins.netbeans.org/plugin/36176/cuda-plugin) or JuxtaCore (http://www.juxtacore.com/community/).

As shown in Table 2, I selected 1D input arrays with sizes that were powers of 2 (2^n) to provide CUFFT and FFTW with increasing transform sizes of uniform increments. For 2D FFTs, I used square arrays where N is the size in both dimensions and the total number of elements is given by N^2.

Table 2: CUFFT and FFTW benchmark sizes.

1D		2D	
2^n	Total Elements	N	Total Elements (N^2)
2^{10}	1,024	8	64
2^{11}	2,048	16	256
2^{12}	4,096	32	1,024
2^{13}	8,192	64	4,096
2^{14}	16,384	128	16,384
2^{15}	32,768	256	65,536
2^{16}	65,536	512	262,144
2^{17}	131,072	1,024	1,048,576
2^{18}	262,144	2,048	4,194,304
2^{19}	524,288	4,096	16,777,216
2^{20}	1,048,576	8,192	67,108,864
2^{21}	2,097,152		
2^{22}	4,194,304		
2^{23}	8,388,608		
2^{24}	16,777,216		

The average of 20 iterations for each 1D and 2D transform size was used as the final result for making the comparisons shown in Table 3. Testing was performed both with and without data transfers between the GPU and CPU, or what CUDA C defines as the *device* and *host*, respectively. Program iterations were accomplished by placing CUFFT and FFTW "execute" functions within a `for` loop that iterates 20 times (Figure 4).

Table 3: CUFFT and FFTW execution time comparisons performed. *DT* = data transfers between the CPU and GPU.

I.	1D CUFFT with DT
II.	1D CUFFT without DT
III.	2D CUFFT with DT
IV.	2D CUFFT without DT

For timing program executions without device-to-host data transfers, a timer was initialized and started before the program entered the loop, and was stopped immediately after the loop. I then divided the total elapsed time by the number of iterations (20), to obtain an average elapsed time in milliseconds (ms).

In timing program executions that included device-to-host data transfers, I initialized and started the timer immediately before the input data was transferred from the host-to-device. The timer was stopped after the result was transferred from the device to the host.

CUFFT program timing (for measuring execution times without data transfers) was performed using native device timing functions. For the FFTW programs (which occur

exclusively on the host), I used a high-resolution timer program[3] based upon the Linux `gettimeofday()` function.

Timing resolutions between the CPU and GPU differ by approximately ±0.000001 ms with the GPU having the higher resolution. This difference is the primary reasoning behind my choosing to perform 20 iterations per transform sizes in one and two dimensions, as this would increase the accuracy of CPU timing.

III. Results

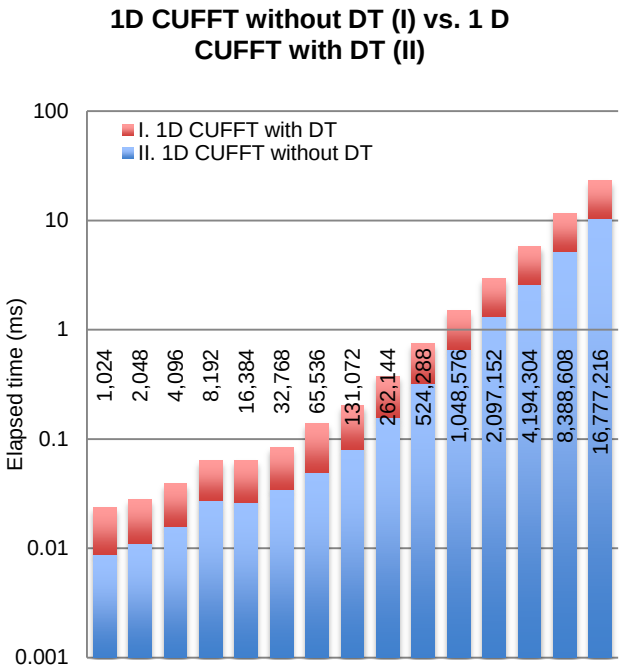

Figure 1: 1D sizes show an overall consistent but minimal overhead associated with DT.

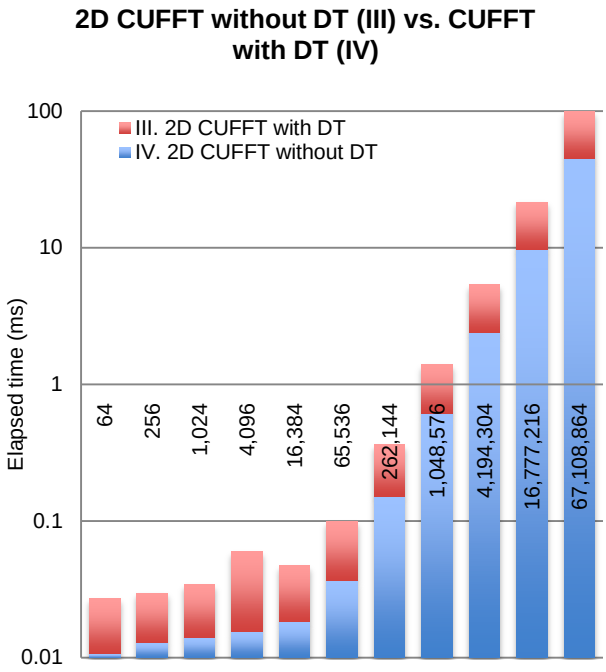

Figure 2: Smaller 2D sizes showing large DT overhead.

CUFFT 1D FFTs sizes benchmarked show consistent DT times. For 2D FFT sizes smaller than 16,384, the DT exceeded the CUFFT execution time (Figure 2). Overall, the DT costs incurred are negligible, increasing 1D and 2D processing times by an average of 0.32 ms and 1.12 ms, respectively.

C. Speedup Analysis

Speedup was calculated using the following equations:

$$S = \frac{1D\ FFTW}{1D\ CUFFT\ with\ DT} \quad \text{and} \quad S = \frac{2D\ FFTW}{2D\ CUFFT\ with\ DT}$$

[3] This program was created by Song Ho Ahn and is available from http://www.songho.ca/misc/timer/ timer.html.

Results are shown in Figures 2 & 3 and Tables 4 & 5.

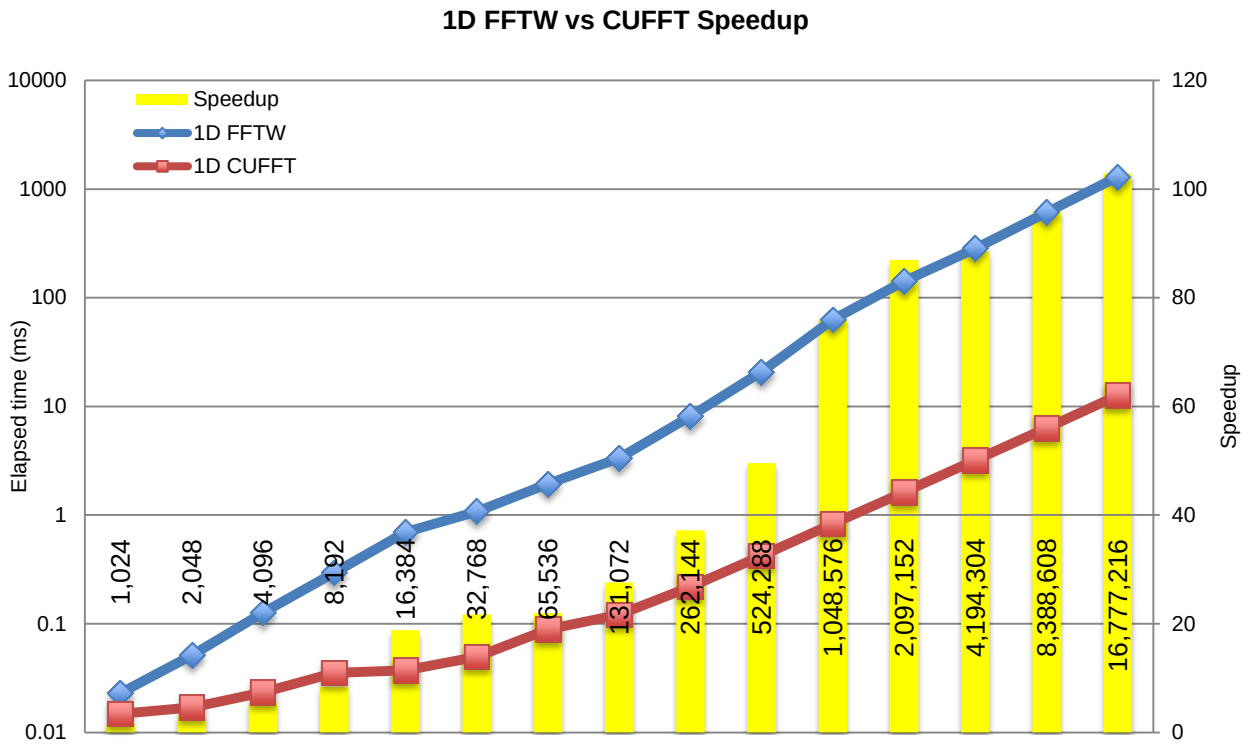

Figure 2: 1D CUFFT speedup compared to 1D FFTW.

Table 4: Execution times for 1D CUFFT and 1D FFTW benchmark sizes from Table 2. The last rows shows CUFFT speedup.

FFT Size:	1,024	2,048	4,096	8,192	16,384	32,768	65,536	131,072
1D FFTW:	0.023050	0.051150	0.126550	0.295700	0.699350	1.076550	1.942200	3.341350
1D CUFFT:	0.014850	0.016950	0.023350	0.035250	0.037150	0.049550	0.088500	0.121000
Speedup:	1.552189	3.017699	5.419700	8.388652	18.825034	21.726539	21.945763	27.614463
(Continued)								
FFT Size:	262,144	524,288	1,048,576	2,097,152	4,194,304	8,388,608	16,777,216	
1D FFTW:	8.103100	20.884600	63.104252	141.604599	283.558838	608.141663	1297.759521	
1D CUFFT:	0.218000	0.421350	0.829000	1.628900	3.205100	6.337550	12.616150	
Speedup:	37.170183	49.565919	76.120931	86.932653	88.471136	95.958480	102.864941	

Speedup increased significantly for 1D transform sizes larger than 524,288 elements, and was over 100 times faster than FFTW for an FFT of 16,777,216 elements. Dr. Fallen noted that these results contradict those achieved by Owens, Sengupta, and Horn [8], which showed that there are no significant speedups of GPU-accelerated 1D FFTs over CPU-based 1D FFTs. However, at the time their experiments were conducted, GPU programming was done without the availability of CUDA.

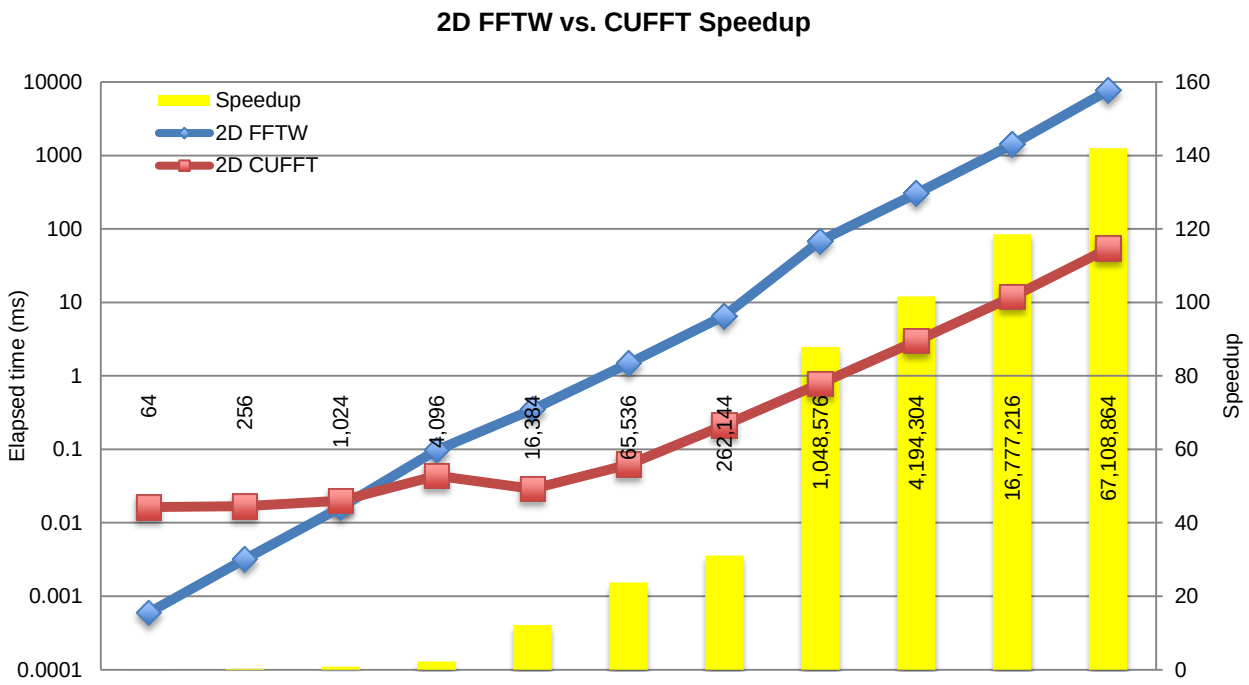

Figure 2: 2D CUFFT speedup compared to 2D FFTW.

Table 5: Execution times for 1D CUFFT and 1D FFTW benchmark sizes from Table 2. The last rows shows CUFFT speedup.

N:	8	16	32	64	128	256	512	1024
N^2:	64	256	1,024	4,096	16,384	65,536	262,144	1,048,576
2D FFTW:	0.000600	0.003200	0.016150	0.097250	0.349850	1.471450	6.598650	69.009750
2D CUFFT:	0.016300	0.016800	0.020000	0.043750	0.028850	0.062050	0.212100	0.785350
Speedup:	0.036810	0.190476	0.807500	2.222857	12.126516	23.713940	31.111033	87.871331
(Continued)								
N:	2048	4096	8192					
N^2:	4,194,304	16,777,216	67,108,864					
2D FFTW:	303.592194	1412.630737	7671.035156					
2D CUFFT:	2.986450	11.920850	54.048752					
Speedup:	101.656547	118.500840	141.928072					

CUFFT and FFTW were almost evenly matched for 2D transform sizes of 1,024 and smaller. There was a significant increase from 262,144 to 1,048,576, where speedup jumped from 31.1 to 87.9 times faster.

IV. Conclusion

It is impressive to observe the orders of magnitude speedup that GPU computing provides, especially for 2D FFTs. The Tesla c2070's performance increases with progressively larger FFTs, indicating that massive and efficient parallelization of CUDA dramatically increases throughput.

V. Impact of the Summer Research Experience

A. Personal Goals

I sought an experience that would be personally challenging and rewarding, and this internship provided be with much more than I had expected. I had no previous experience with most of the project requirements, and I was initially very doubtful of my ability to accomplish any of the project objectives.

My main objective for applying for and attending this internship was to prove to myself that I could accomplish this challenge, and reap the rewards of having a sense of accomplishment. I have been an online student at the University of Maryland University College (UMUC) for the past three years and have worked for BAE Systems as a munitions specialist/assistant crew chief for the past eight years. Much of overcoming this challenge stems from my desire to prove to myself that I can undertake and accomplish something far removed from what I had been doing for so long. I wanted to venture outside of my comfort-zone, and what I had been fearful of falling into: complacency. I knew that in participating in this internship, my comfort zone would begin to expand again.

I was very fortunate that Dr. Newby and Dr. Fallen selected me for this opportunity. Everyone that I have encountered at ARSC has been fantastic. It is truly a world-class academic institution not just in terms of its supercomputing capabilities, but also in terms of the quality of its people.

My personals goals were to experience new things, be challenged and overcome those challenges, and grow from those experiences. I define personal growth is a commitment to improving every aspect of myself. It is a process that never ends.

The JEOM internship and ARSC have provided the opportunity for me to learn and apply new skills that will give me the confidence to explore new and progressively challenging experiences where there will be more opportunities for personal enrichment.

B. Educational Goals

Although this internship will push my projected graduation date out by a few months, I consider this sacrifice to be insignificant when measured against everything I have learned here. I have not been a traditional student for many years. As an online student that works full-time and supports a large family, there is a high probability that this opportunity of study at an academic institution such as ARSC will not repeat itself.

I will graduate from UMUC with a B.S. in computer science and a minor in business administration. I plan to enter graduate school and pursue an M.S. in cyber-security and an MBA. This internship has provided me with invaluable, real-world experience working in my field of study. This internship has better prepared me to meet my educational goals by providing a preview of what can be expected from my remaining advanced coursework.

C. Career Goals

Ultimately, I would like to become a chief information officer (CIO). This internship has provided me with a great deal of insight into the functioning of a supercomputing facility. Although I have not had many opportunities to interact with others at work, I have learned a

great deal just by being in the environment, and observing the interactions among the people here.

More importantly, I have learned a great deal about project management. I will be receiving a certificate in business project management after the Fall 2011 semester. I have had what I consider to be many missteps throughout this project. I was working with subjects previously unknown, and did not possess sufficient knowledge to ask the right questions. There are things I learned that I later learned were not required in order for me to accomplish a task. Despite my lack of knowledge and experience, I was never in any way made to feel that I was incapable of completing the project objectives. Most of the time was spent learning the basics so I could get to a point where I could begin working with the tools that the project required.

I had initially created a project plan; however, I was unable to anticipate the relatively steep learning-curve that I would encounter in building a working knowledge of many new technologies within a specified timeframe. I found myself unable to adhere to the plan I had created, and was forced to change the plan almost on a daily basis. This internship as given me a project management experience that will help guide me throughout my personal, professional, and educational careers.

Acknowledgements

I would like to express my sincere thanks to the JEOM program for making this internship possible, especially Pam Gilliam for her professionalism and patience. I am also sincerely grateful to Dr. Newby and Dr. Fallen at UAF ARSC for their patience, guidance, and encouragement, without which, this experience would never have been possible. Many thanks also to all the supporting staff at ARSC.

References

[1] Fallen, C.T., Bellamy, B.V.C., Newby, G.B. & Watkins, B.J. (n.d.). GPU performance comparison for accelerated radar pulse processing. Retrieved from saahpc.ncsa.illinois.edu/presentations/fallen.pdf

[2] FFTW.org. (n.d.). FFTW home page.. Retrieved from http://www.fftw.org/

[3] FFTW. (n.d.). Retrieved from http://web.mit.edu/tlo/www/ industry/fftw-1.html

[4] NVIDIA Corporation. (2011). CUDA CUFFT Library. Retrieved from http:// developer.download.nvidia.com/compute/DevZone/docs/html/ CUDALibraries/doc/CUFFT_Library.pdf

[5] NVIDIA Corporation. (2011). CUDA C Programming Guide. Retrieved from http:// developer.download.nvidia.com/compute/DevZone/docs/html/C/ do/CUDA_C_Best_Practices_Guide.pdf

[6] NVIDIA Corporation. (February 2011). CUDA API reference manual. Retrieved from http:// developer.download.nvidia.com/compute/DevZone/docs/html/C/ doc/CUDA_Toolkit_Reference_Manual.pdf

[7] FFTW.org. (n.d.). FFTW User manual. Retrieved from http://www.fftw.org/fftw3_doc/

[8] Owens, John. D., Sengupta, Shubhabrata & Horn, Daniel. (2005). Assessment of graphics processing units (GPUs) for Department of Defense (DoD) digital signal processing (DSP) applications. Retrieved from www.idav.ucdavis.edu/func/return_ pdf?pub_id=866