Computing Performance Benchmarks among CPU, GPU, and FPGA

MathWorks

Authors:
Christopher Cullinan
Christopher Wyant
Timothy Frattesi

Advisor:
Xinming Huang

Abstract

In recent years, the world of high performance computing has been developing rapidly. The goal of this project was to conduct computing performance benchmarks on three major computing platforms, CPUs, GPUs, and FPGAs. A total of 66 benchmarks were evaluated. GPUs outperformed the other platforms in terms of execution time. CPUs outperformed in overall execution combined with transfer time. FPGAs outperformed for fixed algorithms using streaming. The team made several recommendations for further research in this area.

Acknowledgements

The successful completion of this project would not have been feasible without the help of several key individuals. First, we would like to express our gratitude to our advisor, Xinming Huang, who met with us weekly throughout the course of the project offering advice and wisdom. In addition, we would like to thank our sponsor at MathWorks for their financial support of this project.

Authorship

Christopher Cullinan

Christopher C. was responsible for the CPU Multicore portion of this project. This included gathering and testing 34 benchmarks on the AMAX machine. In addition, he wrote the CPU sections for the background, benchmark, and results along with the future work and executive summary sections of this report.

Timothy Frattesi

Timothy was responsible for the GPU portion of this project. This included gathering and testing 24 benchmarks on the GeForce GTX 460 and GeForce 9800 GTX+ NVIDIA graphics cards. In addition, he wrote the GPU sections for the background, benchmark, and results of this report along with the future work and executive summary sections.

Christopher Wyant

Christopher W. was responsible for the FPGA portion of this project. This included gathering and testing of 8 benchmarks using the ISim program of Xilinx on a Virtex-5 board. Christopher W. wrote the FPGA sections of the background, benchmark, and results section of the report. In addition, he was responsible for writing the abstract, acknowledgements, introduction and conclusion of this report.

Executive Summary

Ever since the beginning of modern day computing, engineers and developers have been trying to squeeze every ounce of performance into their devices. How do we test for performance though? What quantifiable measurements can be made to justify the superiority of one device over another? These are the types of questions this project aims to answer. In this project, we investigated the performance abilities of current generation multiprocessing hardware. Through looking at multicore CPUs, general purpose GPU computing and an FPGA, we compared device capabilities to determine which platform future investments should be focused towards and why.

To begin our endeavor, we used benchmarking to accentuate the strengths and weaknesses of these three devices. Benchmarking is the technique of using crafted programs in order to attach quantifiable performance metrics to targeted computer subsystems. By using cross platform, as well as individual, benchmarks developed across a plethora of computational necessities, we determined which device would be best suited towards specific tasks. For this project, we tested our benchmarks across two Intel Xeon 5650 CPUs, the Virtex-5 FPGA and NVIDIA's GeForce GTX460 and 9800 GTX+ GPUs.

Realizing the sophistication early on in this project, we decided to use already written benchmarking suites to conduct our tests. A benchmarking suite is nothing more than a compilation of individual benchmarks with specific intent. In total, we used seven benchmarking suites. For the FPGA, we used cores designed by Xilinx Core Generator and MATLAB Simulink HDL Coder, which contained

benchmarks encompassing mathematical algorithms and encryption. For the CPUs, we used three benchmarking suites; SPEC CPU2006, Rodinia, and John Burkardt. Lastly, for the GPUs, we used the Parboil, Rodinia, and SHOC benchmarking suites. The CPU and GPU suites tested mathematical algorithms, high performance simulation, and common computational necessities such as compression and sorting.

To further enhance the findings of this project, we discussed several future recommendations at the end of this report. These recommendations include testing a broader spectrum of benchmarks capable of running across all three platforms. Another possibility is to look into newer technologies, such as an Accelerated Processing Unit (APU). We believe that by including these recommendations, a conclusion of greater impact can be reached.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

The world of high performance computing is a rapidly evolving field of study. Many options are open to businesses when designing a product. GPUs can provide astonishing performance using the hundreds of cores available. On the other hand, FPGAs can provide computational acceleration to many signal and data processing applications. The question arises as to what level each platform performs at for different benchmark algorithms.

To determine computing levels of each device we implemented existing benchmark suites for each device. We tested several applications to see which computing method was fastest for the various applications. While the benchmarks did not completely lineup between the different processors the information gathered laid a good foundation between different devices.

In order to explain the information in a clear manner, we broke up the information into several sections. Presented first is the background of the devices, both general and specific. The second section outlines all of the benchmarks we tested so that the reader can understand the limitations of the project. The third section discusses the results we gathered as well as a discussion of what they mean. Lastly, we discuss some recommendations we would make for similar projects in the future and some closing remarks.

2. Background

The background gives an overview of the different device we used in our project. Within each section are the history of the device and a general overview of how they work. This information gives a brief but necessary background into the platforms used for this project.

2.1. CPU

2.1.1. A Brief History

The history of the Central Processing Unit (CPU) is in all respects a relatively short one, yet it has revolutionized almost every aspect of our lives. In the early 1970, if I were to ask someone what a CPU was, they would have most likely responded "A what!" Yet just over 40 years later, CPUs have become an integral part of our lives. From desktop computers to cell phones, most of us do not go more than a few hours without somehow interacting with a CPU. Despite its indisputable popularity, most do not know how this hype all started.

In 1971, the 4-bit Intel 4004 was the first in the legacy of the CPU. It was the first commercially available CPU on chip, made possible by the all-new silicon gate technology. The max CPU clock rate of this revolutionary hardware was 740 kHz, an astonishing speed at the time. This little guy could execute 92,000 instructions per second with a single instruction cycle of 10.8 microseconds and a transistor count of 2,300. At the time, this device was truly a feat in computing technology, which paved the road for much more innovation to come. [1]

Intel dominated CPU infancy, coming out with several subsequent CPU designs including the 8086 (1978), 8088 (1979), 80186 (1980). Then, in 1993, one of the most popular names in the history of the CPU surfaced, the Intel Pentium processor. This legendary device operated at a whopping 60 MHz and 100 Millions of Instructions per Second (MIPS). The trend of innovation from Intel continued for several years until another major competitor in today's market made their first competitive appearance with the AMD AM5x86 in 1995. A fierce competition between Intel and AMD has continued since. [2]

The next milestone in the CPU history was the commercial release of the first 1GHz processor. This achievement was reached by the AMD Athlon in 1999 and then by the Intel Pentium III just two days later after. For this reason, "Athlon" was fitting name for AMD's milestone processor because it is the Greek word for "Champion/trophy of the games". The AMD Athlon is an x86-compatible processor containing 22 million transistors in a slim size of 184 mm^2. [3]

Nowadays, it is a common occurrence to see CPUs clocked well above 1GHz in devices as small as our cell phones. In just over 40 years, we have gone from 740 kHz to the GHz level (over a 1300 % increase) and increased the count of on chip transistors from 2,300 to more than a billion (over a 434,000 % increase). We are now producing CPUs with multiple cores on the same chip, which are capable of support an increasingly important feature known as parallel computing, which we will talk about in more detail later in this report. [2]

2.1.2. CPU Design

In a nutshell, CPU design is the design engineering process of creating a central processing unit to be used in a computing system. Many factors go into designing a CPU, especially with the level of sophistication in modern day CPUs. There are six primary focuses that designers must account for when creating a CPU, and they are; data paths, control unit, memory components, clock circuitry, pad transceiver circuitry, and logic gate cell library. [4]

A data path by definition is, "A collection of functional units, such as arithmetic logic units or multipliers, that perform data processing operations". [5] Intuitively from its name, data paths provide routes for data to traverse between the components of a CPU. These routes are typically known as "Buses". The majority of CPUs include both a data path and a control unit, where the control unit specializes in regulating data path and main memory interaction. [5]

Most modern day CPUs have several types of memory modules on chip. Two of the most popular are register memory and cache, both of which are normally high speed SRAM. Registers are the memory cells built directly into the CPU since they contain specific data vital to CPU operations. Cache is the next portion of memory in a CPU and is usually, in more complex processors, divided into L1 (level one) and L2 (level two) cache. Both L1 and L2 cache are there to store data that is most often used by the CPU and is typically SRAM as well. [6]

The CPU clock is the sinusoidal frequency reference signal typically created by a crystal oscillator. This sinusoidal waveform is first translated into a square waveform of the same frequency by internal circuitry and then used to synchronize

the internal components of the CPU. The clock signal traverses to the various CPU components via a clock distribution network. [7]

Lastly, the logic gate cell library is the collection of logic gates used to implement computational logic in the CPU. The logic library collection consists of low-level logic functions including AND, OR, INVERT gates as well as flip-flops, latches, and buffers. A vital feature of these libraries is that they are fixed height and variable width, meaning they can be placed in organized rows. This makes the process of automated digital layout of these components possible and efficient. [8]

To give you an idea of a simple CPU design, the following figure shows the AMD K10 Architecture of 2007. This processor is slightly outdated but is good for our purposes to show the anatomy of a modern day CPU.

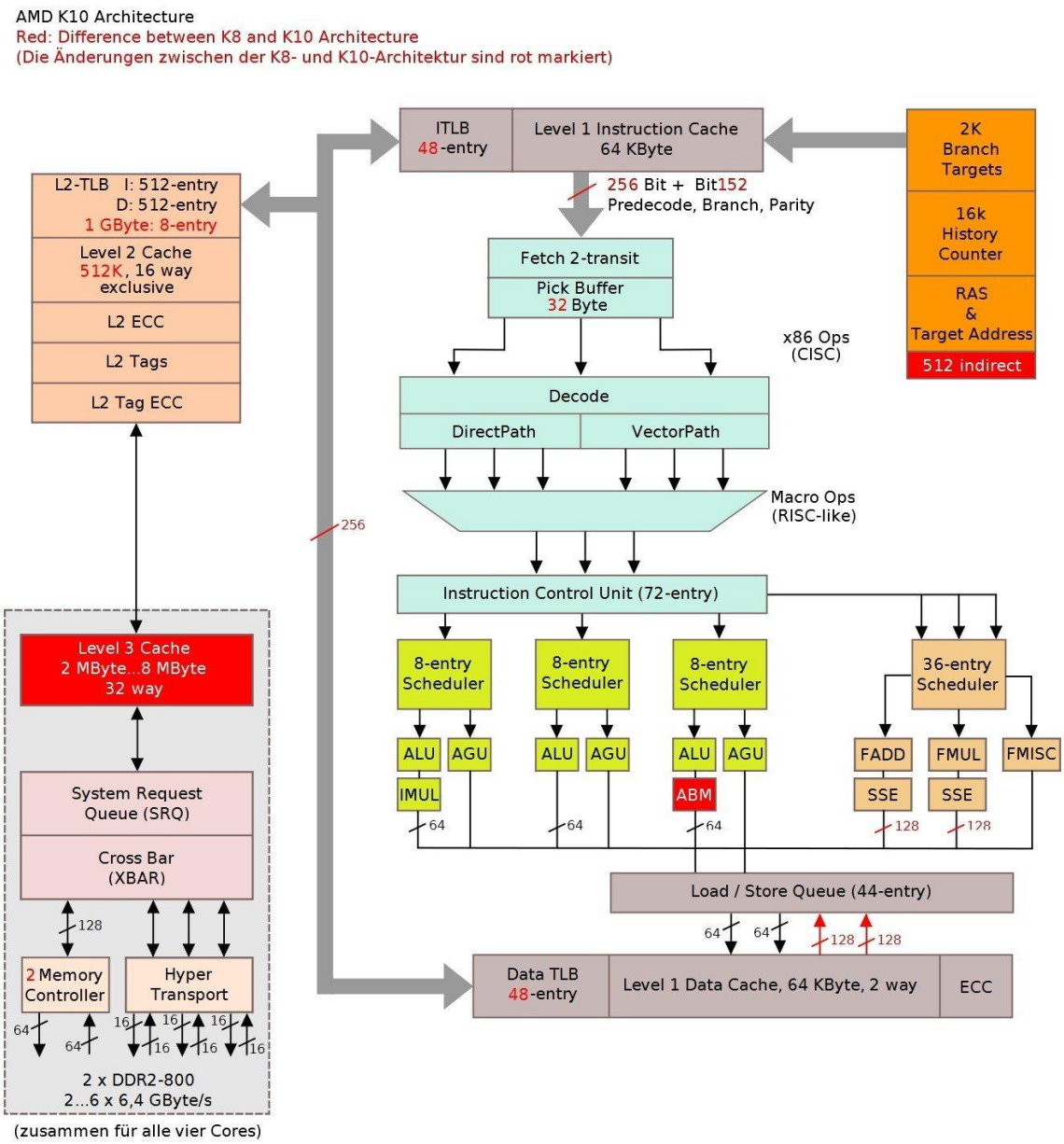

Figure 1 - AMD K10 Architecture

2.1.3. CPU Multicore

The term "multi-core" refers to a multiple core processor that is simply an integrated circuit where two or more processors have been attached for increased performance via parallel processing. [9] Parallel processing is a type of computation

where many calculations are performed simultaneously. This method of computation is based on the principle that large problems can be solved faster by breaking them down into smaller pieces and then solving those pieces concurrently. Because of this basic principle, parallel computing has become the dominant standard in computer architecture in the most popular form of multicore processing. As an example of a multicore processor design, the following figure shows the internals to an Intel I7 950 Quad Core Processor.

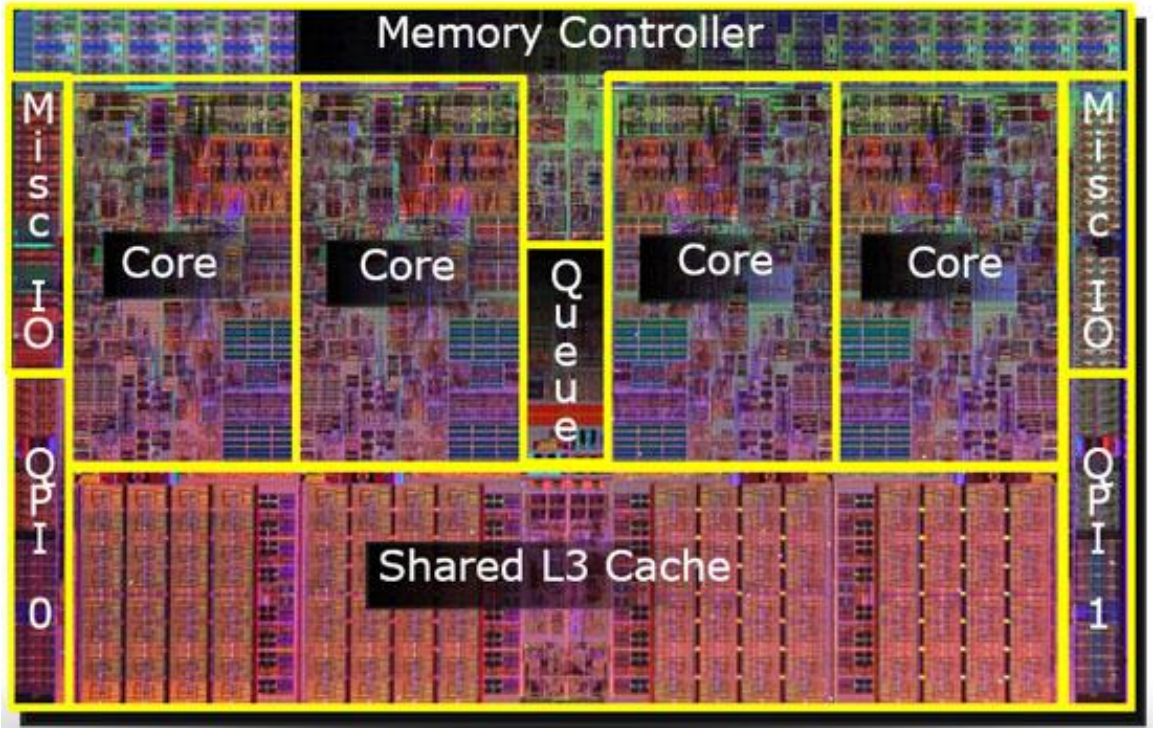

Figure 2 - Intel I7 950 Quad Core Processor Design

In the real world, parallel processing is not limited to integrated circuits. Virtually everything in our natural universe uses the principles of parallel processing. A few examples are galaxy formation, planetary movements, weather and ocean patterns, automobile assembly lines, rush hour traffic, and even ordering a hamburger at a fast food restaurant. Within all of these phenomena, numerous

complex and interrelated events occur simultaneously to achieve a common goal. [10]

In the past, parallel computing was attributed mainly to high end computing and was used to solve complex mathematical problem in various areas of study. A few of these areas included Atmospheric studies, Physics, Mechanical Engineering, Electrical Engineering, and Seismology. Parallel computing is still used in those areas today, but the rise of commercial applications has been a major contributor to both the need for faster computing and the dispersion of parallel computing into common electronic devices such as phones, desktops, laptops, etc. Database mining, web search engines, medical imaging, and advanced graphics are just a few of the applications that utilize parallel computing. [10]

At the beginning of this section, we briefly discussed the incentives to use parallel computing; now we will delve deeper to justify the use of parallel computing. Parallel computing saves time and money by both shortening the time to the outcome and because parallel components are cheap. Second, larger problems can be solved through the use of parallel computing that are not possible by using a single computing resources. Finally, there are many limitations to serial computing. These limitations include transmission speeds, limits to miniaturization, and economic limitations. To get away from these confines, modern computer architectures are heavily relying on multiple execution units, pipelined instructions, and multi-core at the hardware level to increase performance. [10]

2.2. Graphics Processing Unit

As stated by Prof. Jack Dongarra, "GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs." [11] From this comment one can see one of the many thoughts on where the future or high performance computing is headed.

2.2.1. GPU History

Much like computers in general, GPU's have progressed rapidly over the last 30 years since their introduction to the market. As GPU's have progressed over the years their core functions have remained the acceleration and processing of images.

The introduction of graphics units came early in the 1980's where both Intel and IBM brought specialized products to the market. Other companies such as Commodore and Texas Instruments also added simple graphics capabilities either on chip or using an external card. These cards had simplistic functionality and were relatively expensive. Functions such as filling an area, shape drawing, and modification of simple images were all that these early processors could support.

Figure 3 - S386C911 (1991)

Figure 4 - NVIDIA GeForce 256 (1998)

The 1990's were the real beginning as far as the takeoff of GPUs. From the beginning in 1991, S3 rolled out their 86C911 card which was one of the first standards for the GPU industry. Two dimensional graphic processing had made its way into almost every system by the mid 90's and the race was on to move towards 3D processing. Two notable chip sets in the race for dedicated 3D graphics include the 3dfx Voodoo and Rendition's Verite. Until the late 90's all 3D rendering was done with the assistance of CPUs, also known as hardware assisted 3D graphics which we still see in lower end laptops today. [12]

To assist in the commonality of graphics processing several “languages” were brought about in the late 90's including both OpenGL and Direct. Throughout the 90's OpenGL prospered as the software’s capability was usually ahead of Direct and it was capable of being used across cards and platforms. Towards the end of the 90's these two API's introduced support for transform and lighting (T&L) which provided a huge jump in GPU processing. T&L allowed for easier mapping of 3D images to a 2D plane while incorporating the lighting all into one. By this time there were only a few competing companies; NVIDIA, ATI, 3dfx, and S3. The end of the

90's saw the NVIDIA GeForce 256, the first readily available commercial card, bringing 3D graphics, NVIDIA, and Direct to their own level. [12]

Figure 5 - GeForce 6600 (2004)

Figure 6 - GeForce GTX 560 (2011)

Through 2010 and to today we continue to see significant advancements in the 3D rendering abilities of the GPU. On the front of programming the most notable improvements include programmable shading and floating point abilities. ATI and NVIDIA hold the majority of today's market share in graphics processing and thus have been major forces in shaping how these units improve.

One of the most significant advancements of the past decade is general purpose computing for GPU's. Due to the highly parallel structure of modern graphics cards it is possible to use them to perform research and analysis, often times competing or surpassing modern CPUs. While this can be done with almost any modern card, NVIDIA's introduction of the Compute Unified Device Architecture (CUDA) from NVIDIA this idea has become standardized. OpenCL is also a common language for performing GPU computation, but it does not support as many programming languages or have the same amount of industry support. CUDA

architecture is the main advancement that is allowing our society to take high performance computing from CPUs and FPGAs and move it to a quicker paralleled set of computations on thousands of threads instead of tens of threads. [13]

2.2.2. GPU Architecture and Parallelism

Here we will look at the both the CUDA architecture and the hardware architecture that corresponds to it. NVIDIA has created this specialized architecture to achieve the massively parallel systems that we have today.

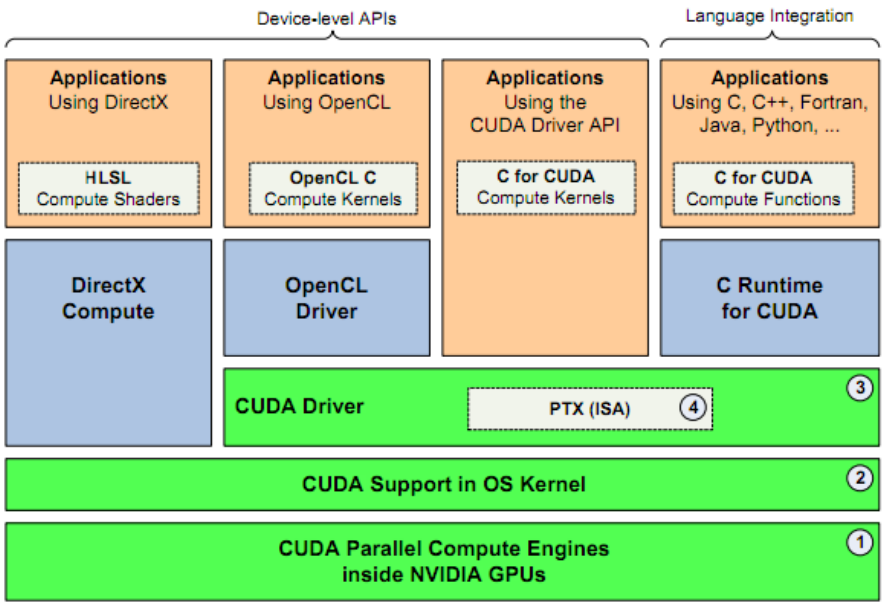

Figure 7 - CUDA Software Architecture

Looking backwards from computer to device we start with the Parallel Thread Execution (PTX) instruction set. This specially optimized instruction set allows for special optimization specifically designed for parallel processing on NVIDIA cards. The PTX instruction set allows NVIDIA to set a standard across multiple generations of GPUs as well as provide a common set of instructions for

both optimization and developer programming. This instruction set rests within the CUDA drivers which are provided dating back to NVIDIA's GeForce 8800 series. [14]

The next level up, CUDA Support in OS is provided through the CUDA Toolkit which is currently on version 4.1. This toolkit provides all of the necessary components to write and run CUDA code in an IDE such as Visual Studio or simply in a text editor. Currently supported are both a LLVM based and the standard nvcc compiler. The move to the LLVM based compiler has shown 10% increases in speed, but is geared more towards the new Fermi architecture and beyond. With the release of the new version of the CUDA toolkit, NVIDIA also provided a much larger base of functions for image processing.

The base level is the actual CUDA Parallel Computing engines that are the basis of all modern NVIDIA cards. The specialized hardware architecture in these cards is what allows for such successful parallelism in general purpose computing. On the current generation of CUDA capable hardware, Fermi, one can setup and process 65,535 simultaneous threads in grids of up to 1024x1024x64. These grid sizes have doubled and allowed for a third dimension since CUDAs initial release in 2006. The number of cores capable of providing floating point and integer functionality has also increased six fold. In the following figure the hardware advancements can be seen in overview as the architecture has changed. Further improvements can all be seen in the tables attached in Appendix X. [15] [16]

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

Figure 8 - Improvements in CUDA GPU Architecture

Parallelism is generated through the use of threads on the GPU. CUDA has the ability to run a particular kernel across multiple threads and multiple cores. Picture the newest card in NVIDIA's lineup has 1024 cores per card. If each core can provide up to 1024 threads per 65535 blocks, there is a possibility to run upwards of 60 billion instances of a single kernel per card in a system. Spanning this number across multi-card systems, or even multiple systems connected together, it is easy to see how parallelism prevails and allows for faster processing. This type of parallelism is known as single instruction multiple data (SIMD). CUDA devices are also capable of running thousands of small programs simultaneously as well.

In order to achieve this number of parallel threads there is a complex setup of memory. Each and every thread an individual piece of memory that is unshared. This contains data such as program counters and individual registers. From there we move up to thread blocks which share memory amongst themselves and then up to sixty five thousand blocks sharing memory per core. These sets of blocks are

called grids and can share a set of application memory for use by smaller threads with global memory. All of the threads running a particular instance of a kernel are kept in synchronization through the use of special code functions that wait for all threads to be completed before reading or writing large changes from global memory. The memory hierarchy described here is important because each smaller level allows for quicker access to data; thus like the L cache of a CPU, there is less need to constantly read and write to slower global memory. [16] [17]

To handle all of these threads processing at the same time there is a unique feature called a warp handler. A warp is a group of 32 threads; the smallest data size for a SIMD setup. When programming in CUDA, users work with blocks so it is up to the warp handler to determine how to divide the instructions. The Fermi architecture has a dual warp scheduler, allowing it to process and divide up two sets of instructions at a time. With 32 bit mathematics it is also possible to dispatch two of a single type of instruction or a mix at one time. Since this setup works in sets of 32 in order to achieve the peak performance on CUDA capable GPUs is to run kernels in sets divisible by 32. [17]

Along with the efficient thread hierarchy, the Fermi architecture relies on NVIDIA's third generation of stream multiprocessing (SM) for its hardware architecture. In this revision of SM, there are 32 CUDA cores per multiprocessor, giving each card 16 to 32 SMs; with each core having it's out floating point and arithmetic logic units. Figure 9 below is a simplified example of third generation SM. Pictured is a single multiprocessor, with the hardware available in each SM. In the figure the 32 individual processing cores can be seen along with the 16

load/store units and the 4 special functions units. These units are accessed once the warp schedulers divide tasks amongst the cores. Following these processing units are the standard graphical processing hardware units such as those that perform tessellations and texturing. It is also noted that in this version each individual core has its own integer and floating point units. [16] [17]

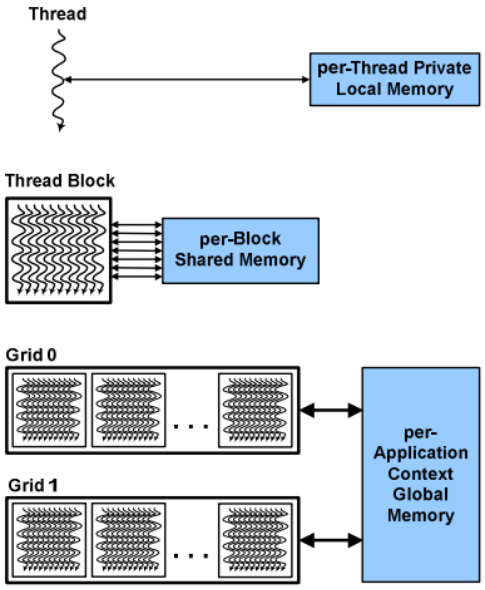

Figure 9 - Thread Hierarchy

Figure 10 - Stream Multiprocessor

Added support for the new IEEE floating point standard has given these new cards the ability to use a fused add and multiply in one step. Data precision has also been improved so that the integer units provide 64 bit support while the floating point units finally provide full 32 bit support. The sixteen load and store and four special function units allow for up to sixteen thread address to be calculated at a

time and four instances of functions such as sine or square roots. With most of these instructions executing with one instruction per clock cycle having so many SMs on a single card allows for distribution of complex equations and faster execution. [16]

2.3. FPGA Background

2.3.1. History of FPGAs

Ross Freeman, co-founder of the company Xilinx, invented Field Programmable Gate Arrays (FPGA) in 1894 while working for the company Zilog. After inventing the FPGA Freeman left Zilog with his Patent (patent 4,870,302) to found Xilinx. While Xilinx is a multi-billion dollar company today, Freeman did not live to see this become a reality passing away in 1989. He was honored in 2009 by being inducted into the National Inventor's Hall of Fame for his work on FPGAs. [18]

The first FPGAs released to the market had only several thousand gates and had several disadvantages to their counterparts, ASICs. They were slower, consumed more power, and had limited functionality. The industry of FPGAs grew slowly through the 1990s. In 1992 the U.S. Naval Surface Warfare department completed a project on FPGAs that implemented 600,000 logic gates. During this time, the main applications for FPGAs were networking and telecommunications.

By the late 90s, the number of gates on a single FPGA reached the millions and many of the disadvantages compared to ASICs were diminishing. FPGAs began entering many other industries because of the low time from development to market introduction. Much money could be generated by being the first to the market. [19]

Today, FPGAs can cater to many different applications. Different series and families are application specific and have additional logic to support faster processes. FPGAs have a high capacity for parallelization and pipelining processes. Often, they are used as peripherals to CPUs to carry out specific processes that a CPU has trouble handling.

2.3.2. Early Programmable Devices

The idea behind FPGAs originated from two devices, Programmable Logic Arrays (PLA) and Complex Programmable Logic Devices (CPLDs). PLAs were introduced during the early 1970s as one- time programmable chips to implement logic functions. The AND gates and OR gates were connected with a communication matric that could be programmed by burning fuses to implement a truth table. The limiting factors were the number of inputs, AND gates, and OR gates. [20]

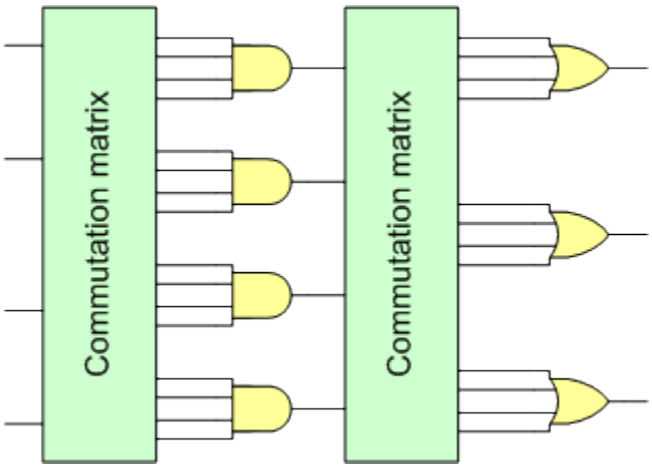

Figure 11 - PLA Architecture

CPLDs built upon the idea of PDAs with an interconnection matrix connecting all of the inputs and outputs. The connection matrix was formed of on-chip Flash

memory to configure macrocells. These macrocells are very similar in structure to a PLA. CPLDs are very similar to FPGAs as the main difference is in the underlying architecture. [20]

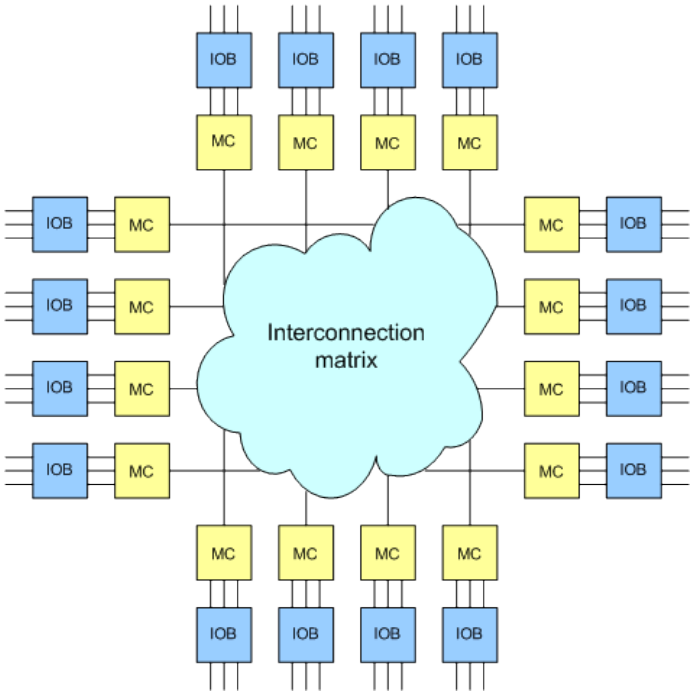

Figure 12 - CPLD Architecture

2.3.3. FPGA Architecture

Field Programmable Gate Array is a semiconductor device comprised of many logic blocks with configurable interconnections between these. The logic blocks are capable of acting as simple logic gates, such as AND and XOR. In addition to the logic gates, there are routing channels that run between each logic block. These channels are programmable and enable different logic blocks to talk to each other. In recent years, more specific circuits are implemented on FPGAs for

application specific purposes. These can include multipliers and DSP circuits, which speed up processing for those applications.

The main component of FPGAs is the logic block. Millions of these are replicated in a network throughout the chip. They are implemented in a Lookup Table (LUT) usually consisting of four input pins. The LUTs have small piece of memory attached that is programmed for output logic depending on the input. Essentially, a truth table defined for that piece of logic. Some designers are increasing the number of input pins to six to increase speed.

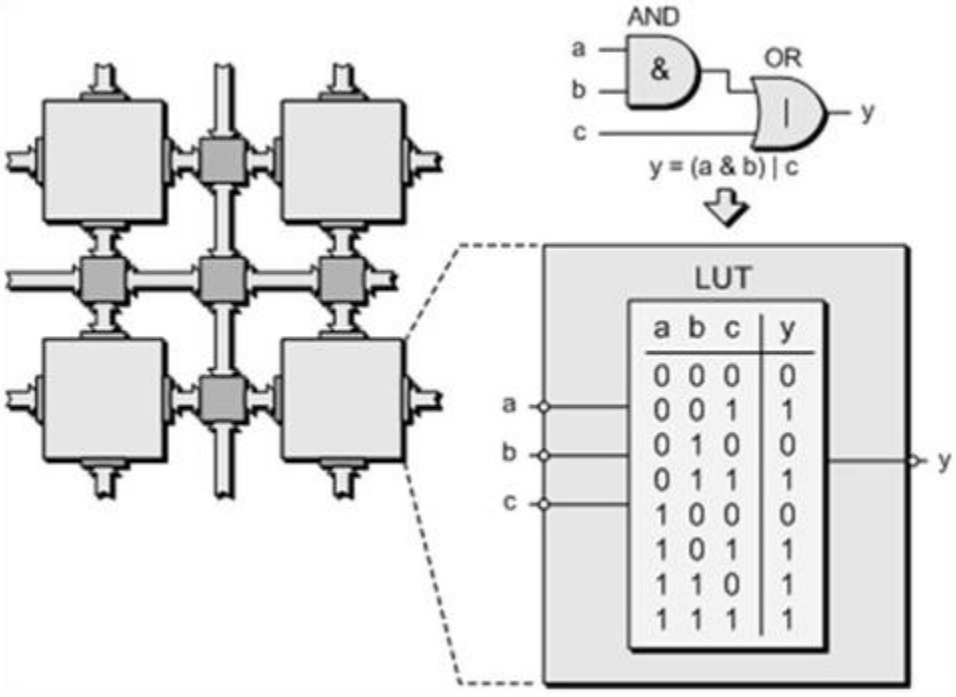

Figure 13 - Three Input LUT

Each LUT has only one output. This output can then be stored in a flip flop to preserve values over a clock cycles, or it can run to other LUTs to further implement logic. The Virtex-5, which the benchmarks in the report are based on, uses six input LUTs.

The routing channels, which run between logic blocks, are used to connect various LUTs together. The routing channels are controlled by switch blocks that control connections between crossing wires. An example of this is seen below in Figure 14.

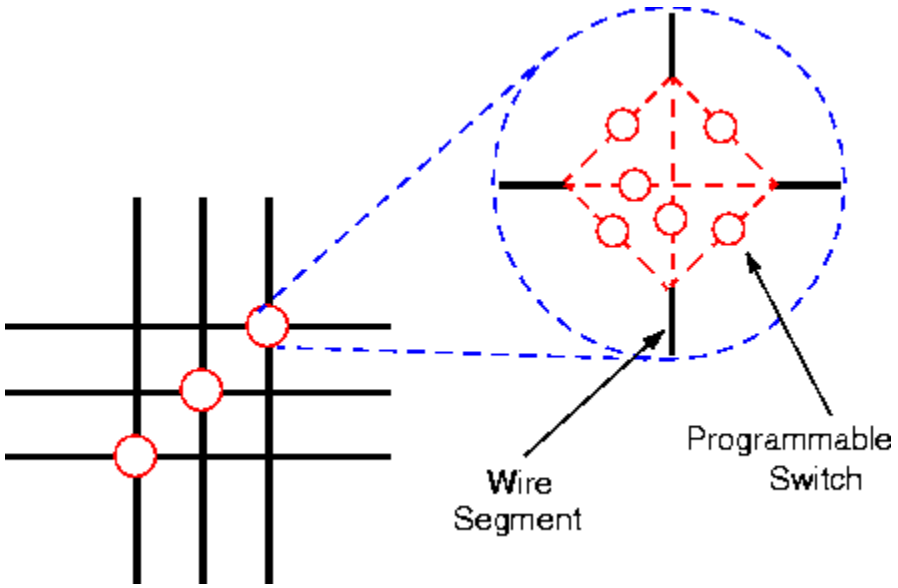

Figure 14 - Switch Block

These connections allow for an immense amount of configurable logic allowing an FPGA to carry out its functionality. Figure 15 below shows the layout of blocks throughout a network. [21]

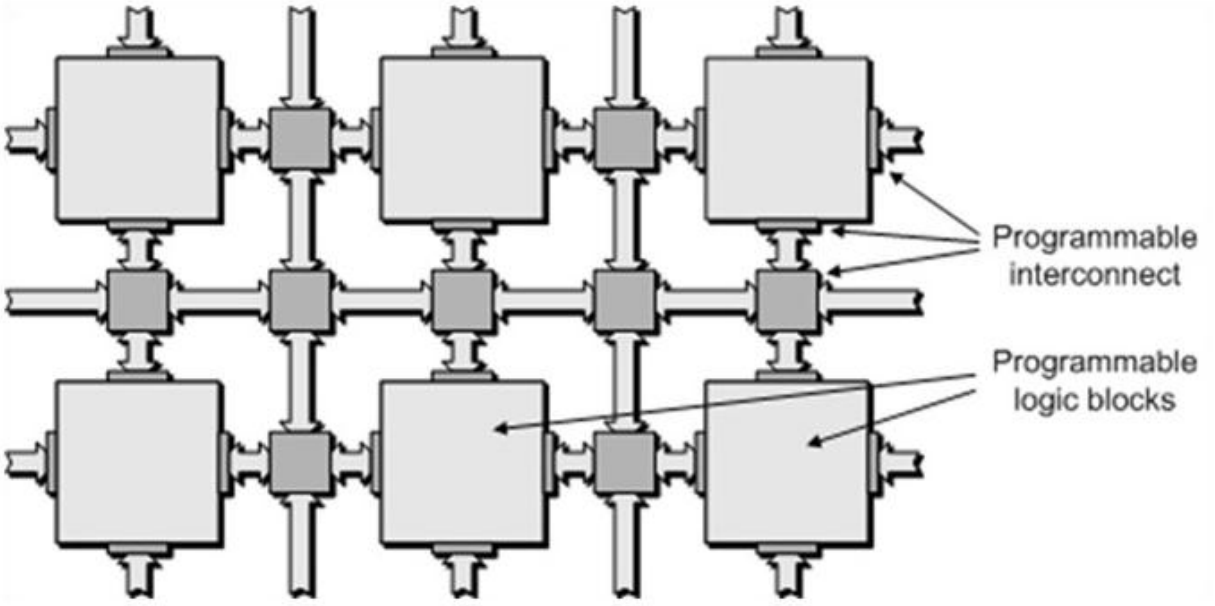

Figure 15 - FPGA Routing

In newer FPGA series other blocks are implemented for application specific functionality. These specialized blocks, or slices, run their specific functionality faster than can be implemented using LUTs and routing wires. These slices also drastically cut down the number of LUTs used. The two main blocks are the multipliers and DSP slices. To implement the multiplication of two 32 bit numbers would require more than 2000 operations for a single multiply. Different FGPA series have different types and quantities of specialized blocks based on designed applications. [22]

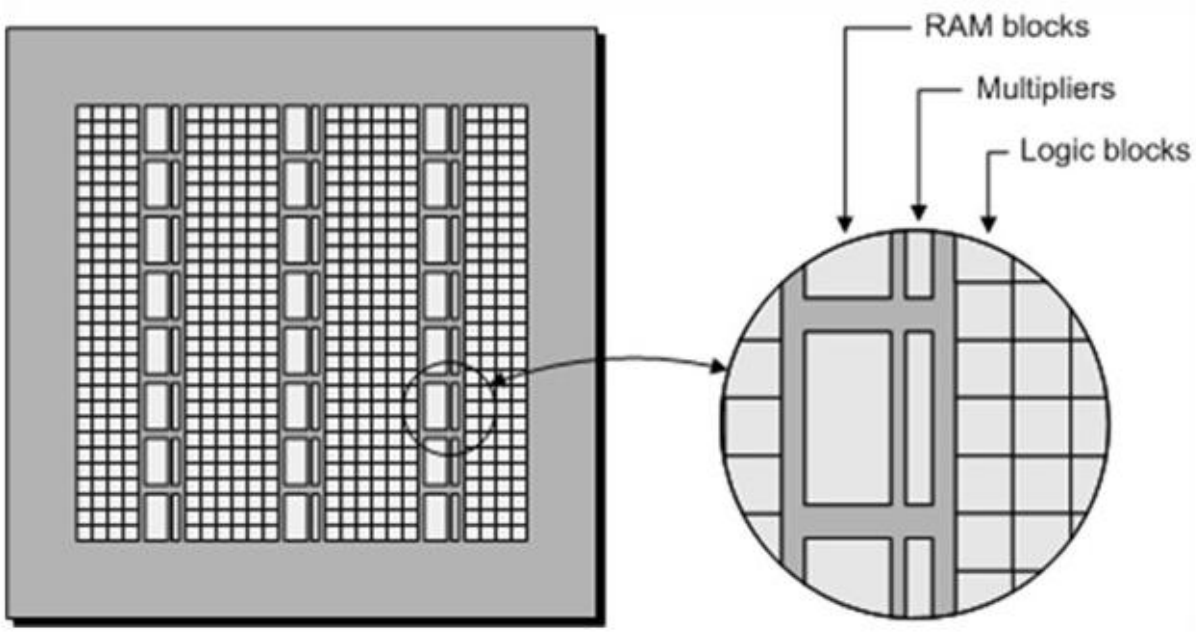

Figure 16 - Specialized Slices

The latest FPGA is the Virtex-7 from Xilinx. This model has increased computing power and efficiency. The architecture for this model can be seen below in Figure 17.

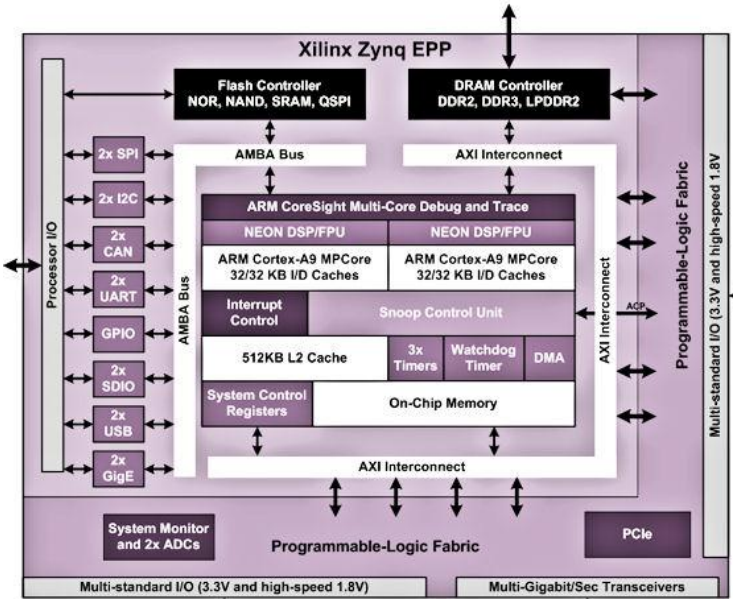

Figure 17 – Virtex-7 Architecture

3. Benchmarks

Benchmarking has been around for years and is widely used as the standard to which we base our computing processing power today. Programs available from many different packages are used to measure the metrics of new processors daily. In the beginning, performance was measured by various system specifications, such as clock rate. Many designers and consumer realized that while this may give some indication of the processing power of a machine, it did not incorporate the entire scope of the situation. Many benchmarks are used today to apply stress to processors through different processes for different applications. Benchmarks come in four different types, each having its own strength and weaknesses: real applications, small benchmarks, benchmark suites, and synthetic benchmarks.

Real applications are benchmarks bases pre-existing programs. They are comprised of a typical user's workload during the day. The advantages to running real applications are that they directly translate over to improved performance times on programs and very accurately mirror everyday workloads. However, these benchmarks are usually very big and require more time to run and transfer to other machines. In addition, it can be very hard to pinpoint a processing bottleneck to discover what instruction types need to be improved for the greatest performance increase.

The next type of benchmark is the small benchmark. It consists of a very small code segment that exists in many other applications. For example, the following C code could be a small benchmark.

```
for (j = 0; j<8; j++)
    S = S + A_j × B_{i-j};
```

This code runs very fast and does not take very long to compile or transfer between systems. In addition, it can give developers a very good idea about what portion of their process is bottlenecking the systems in order to make improvements and can be easy to simulate during design to test functionality. However, many designers can take advantage of the limited instructions used to design a system specialized for that particular loop. This abuses the benchmark and does not report accurate data to both designers and customers.

Benchmark suites are compilations of different benchmarks from different industries that together represent a variety of computing loads on a machine. Standard Performance Evaluation Corporation (SPEC) provides one of the main suites available to developers. SPEC began in 1989 with SPEC89 CPU intensive benchmark suite. Many companies came together and agreed on a set of programs that represented a user's typical workload. Suites are very useful in covering a diverse set of parameters and characteristics; however, they are still susceptible to abuse. In addition, they require periodic updates to change the applications as typical workloads change.

Lastly, some programmers advocate the use of synthetic benchmarks. These programs attempt to mirror the characteristics of other applications while using much less space and processing time. In reality, they do not perform any functional task on the processor, but do give an accurate representation of processing power.

It is always important to run many iterations of a benchmark. Timings will change and an average of all iterations should be used to provide a more accurate picture for comparisons. Often manufacturers can find one or more benchmarks that their platform particularly excels at and abuse this standard for advertisement. The best benchmarks to look at for different computers are those posted by non-profit organizations which are unbiased. [23]

3.1. FPGA Benchmarks

Benchmarking is not traditionally done for FPGAs. Rather, the datasheets for different FPGAs have details about maximum clock cycle that they can operate at. The user then looks at the program that they want to run and they can see number of clock cycles it takes to produce a result. The throughput does not change between iterations on the same machine. The following benchmarks that were run on the Virtex-5 series FPGA were compiled using either the Xilinx Core Generator or MATLAB Simulink HDL Coder.

The Xilinx CORE Generator is where most of the applications are from. It is built into the Xilinx program. When opening a core you have several options that can be selected based on your needs such as extra pin I/O, processing type, and size. The Simulink HDL coder is a new addition to the program. Rather than designing our own code, the current demos available in the Simulink program were used to generate HDL code, which is then tested in the Xilinx program. The HDL Coder is also capable of generating testbenches for the application.

Fast Fourier Transform (FFT)

A discrete Fourier transform that operates with reduced computational power. The number of computations needed is reduced from 2N^2 to 2N Log2 N where N is the number of points necessary for the computation. The FFT used in this study is a 1024 point FFT.

Finite Impulse Response Filter (FIR)

FIR Filters are designed to be simple to implement for Digital Signal Processing (DSP). FIR filters are more commonly used then IIR filters because of the advantages offered such as fractional arithmetic and fewer practical problems. The FIR filter used for performance testing in this study is a low pass filter.

Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data. The AES algorithm is a symmetric block cipher that is capable of encrypting (encipher) and decrypting (decipher) information. Encryption converts data to an unintelligible form called cipher text; decrypting the cipher text converts the data back into its original form, called plaintext. The AES algorithm is capable of using cryptographic keys of 128, 192, and 256 bits to encrypt and decrypt data in blocks of 128 bits. (http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf)

Double Precision Floating Point Multiplication

This block takes in two double floating point numbers for multiplication. This is similar to a small benchmark for CPUs. A single FPGA is able to implement multiple instances of this application allowing for large amounts of parallel processing.

3.2. SPEC CPU2006

Standard Performance Evaluation Corporation (SPEC) CPU 2006 is a benchmark suite that contains over twenty-five different benchmarks. These industry-standardized benchmarks were created to stress a system's processor, memory architecture, and compilers. This particular suite is divided into two sub categories, CINT2006 and CFP2006. The CINT2006 portion consists of twelve total benchmarks that measure integer operation performance, while the CFP2006 portion consists of seventeen benchmarks that measure floating point operation performance. A brief description of each SPEC CPU2006 benchmark can be seen below. [24]

3.2.1. CPUINT2006

400.perlbench

"400.perlbench is a cut-down version of Perl v5.8.7, the popular scripting language. SPEC's version of Perl has had most of OS-specific features removed. In addition to the core Perl interpreter, several third-party modules are used:

- SpamAssassin v2.61
- Digest-MD5 v2.33
- HTML-Parser v3.35

- MHonArc v2.6.8
- IO-stringy v1.205
- MailTools v1.60
- TimeDate v1.16" [25]

401.bzip2

"401.bzip2 is based on Julian Seward's bzip2 version 1.0.3. The only difference between bzip2 1.0.3 and 401.bzip2 is that SPEC's version of bzip2 performs no file I/O other than reading the input. All compression and decompression happens entirely in memory. This is to help isolate the work done to only the CPU and memory subsystem." [25]

403.gcc

"403.gcc is based on gcc Version 3.2. It generates code for an AMD Opteron processor. The benchmark runs as a compiler with many of its optimization flags enabled.

403.gcc has had its inlining heuristics altered slightly, so as to inline more code than would be typical on a UNIX system in 2002. It is expected that this effect will be more typical of compiler usage in 2006. This was done so that 403.gcc would spend more time analyzing its source code inputs, and use more memory. Without this effect, 403.gcc would have done less analysis, and needed more input workloads to achieve the run times required for CPU2006." [25]

429.mcf

"429.mcf is a benchmark that is derived from MCF, a program used for single-depot vehicle scheduling in public mass transportation. The program is written in C. The benchmark version uses almost exclusively integer arithmetic.

The program is designed for the solution of single-depot vehicle scheduling sub-problems occurring in the planning process of public transportation companies. It considers one single depot and a homogeneous vehicle fleet. Based on a line plan and service frequencies, so-called timetabled trips with fixed departure/arrival locations and times are derived. Each of these timetabled trips has to be serviced by exactly one vehicle. The links between these trips are so-called dead-head trips. In addition, there are pull-out and pull-in trips for leaving and entering the depot." [25]

445.gobmk

"The program plays Go and executes a set of commands to analyze Go positions." [25]

456.hmmer

"Profile Hidden Markov Models (profile HMMs) are statistical models of multiple sequence alignments, which are used in computational biology to search for patterns in DNA sequences.

The technique is used to do sensitive database searching, using statistical descriptions of a sequence family's consensus. It is used for protein sequence analysis." [25]

458.sjeng

“458.sjeng is based on Sjeng 11.2, which is a program that plays chess and several chess variants, such as drop-chess (similar to Shogi), and 'losing' chess.

It attempts to find the best move via a combination of alpha-beta or priority proof number tree searches, advanced move ordering, positional evaluation and heuristic forward pruning. Practically, it will explore the tree of variations resulting from a given position to a given base depth, extending interesting variations but discarding doubtful or irrelevant ones. From this tree the optimal line of play for both players ("principle variation") is determined, as well as a score reflecting the balance of power between the two.

The SPEC version is an enhanced version of the free Sjeng 11.2 program, modified to be more portable and more accurately reflect the workload of current professional programs.” [25]

462.libquantum

“Libquantum is a library for the simulation of a quantum computer. Quantum computers are based on the principles of quantum mechanics and can solve certain computationally hard tasks in polynomial time. In 1994, Peter Shor discovered a polynomial-time algorithm for the factorization of numbers, a problem of particular interest for cryptanalysis, as the widely used RSA cryptosystem depends on prime factorization being a problem only to be solvable in exponential time. An implementation of Shor's factorization algorithm is included in libquantum.

Libquantum provides a structure for representing a quantum register and some elementary gates. Measurements can be used to extract information from the system. Additionally, libquantum offers the simulation of decoherence, the most important obstacle in building practical quantum computers. It is thus not only possible to simulate any quantum algorithm, but also to develop quantum error correction algorithms. As libquantum allows adding new gates, it can easily be extended to fit the ongoing research, e.g. it has been deployed to analyze quantum cryptography." [25]

464.h264ref

"464.h264ref is a reference implementation of H.264/AVC (Advanced Video Coding), the latest state-of-the-art video compression standard. The standard is developed by the VCEG (Video Coding Experts Group) of the ITU (International Telecommunications Union, http://www.itu.int) and the MPEG (Moving Pictures Experts Group, http://mpeg.chiariglione.org) of the ISO/IEC (International Standardization Organization, http://www.iso.ch). This standard replaces the currently widely used MPEG-2 standard, and is being applied for applications such as the next-generation DVDs (Blu-ray and HD DVD) and video broadcasting." [25]

471.omnetpp

"The benchmark performs discrete event simulation of a large Ethernet network. The simulation is based on the OMNeT++ discrete event simulation system (www.omnetpp.org), a generic and open simulation framework. OMNeT++'s

primary application area is the simulation of communication networks, but its generic and flexible architecture allows for its use in other areas such as the simulation of IT systems, queuing networks, hardware architectures or business processes as well. The Ethernet model used in this benchmark is publicly available from the address given in the References." [25]

473.astra

"473.astar (pronounced: A-star) is derived from a portable 2D path-finding library that is used in game's AI. This library implements three different path-finding algorithms: First is the well-known A* algorithm for maps with passable and non-passable terrain types. Second is a modification of the A* path finding algorithm for maps with different terrain types and different move speed. Third is an implementation of A* algorithm for graphs. This is formed by map regions with neighborhood relationship. The library also includes pseudo-intellectual functions for map region determination." [25]

483.xalancbmkg

"This program is a modified version of Xalan-C++, an XSLT processor written in a portable subset of C++. You use the XSLT language to compose XSL style sheets. An XSL style sheet contains instructions for transforming XML documents from one document type to another document type (XML, HTML, or other). In structural terms, an XSL style sheet specifies the transformation of one tree of nodes (the XML input) into another tree of nodes (the output or transformation result)." [25]

3.2.2. CFP2006

410.bwaves

"410.bwaves numerically simulates blast waves in three dimensional transonic transient laminar viscous flow.

The initial configuration of the blast waves problem consists of a high pressure and density region at the center of a cubic cell of a periodic lattice, with low pressure and density elsewhere. Periodic boundary conditions are applied to the array of cubic cells forming an infinite network. Initially, the high pressure volume begins to expand in the radial direction as classical shock waves. At the same time, the expansion waves move to fill the void at the center of the cubic cell. When the expanding flow reaches the boundaries, it collides with its periodic images from other cells, thus creating a complex structure of interfering nonlinear waves. These processes create a nonlinear damped periodic system with energy being dissipated in time. Finally, the system will come to an equilibrium and steady state.

The algorithm implemented is an unfactored solver for the implicit solution of the compressible Navier-Stokes equations using the Bi-CGstab algorithm, which solves systems of non-symmetric linear equations iteratively." [26]

416.gamess

"A wide range of quantum chemical computations are possible using GAMESS. The benchmark 416.gamess does the following computations for the reference workload:

- Self-consistent field (SCF) computation (type: Restricted Hartree-Fock) of cytosine molecule using the direct SCF method
- SCF computation (type: Restricted open-shell Hartee-Fock) of water and cu2+ using the direct SCF method
- SCF computation (type: Multi-configuration Self-consisted field) of triazolium ion using the direct SCF method" [26]

433.milc

"The program generates a gauge field, and is used in lattice gauge theory applications involving dynamical quarks. Lattice gauge theory involves the study of some of the fundamental constituents of matter, namely quarks and gluons. In this area of quantum field theory, traditional perturbative expansions are not useful. Introducing a discrete lattice of space-time points is the method of choice." [26]

434.zeusmp

"434.zeusmp is based on ZEUS-MP, a computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (NCSA, University of Illinois at Urbana-Champaign) for the simulation of astrophysical phenomena. ZEUS-MP solves problems in three spatial dimensions with a wide variety of boundary conditions.

The program solves the equations of ideal (non-resistive), non-relativistic, hydrodynamics and magnetohydrodynamics, including externally applied gravitational fields and self-gravity. The gas can be adiabatic or isothermal, and the

thermal pressure is isotropic. Boundary conditions may be specified as reflecting, periodic, inflow, or outflow." [26]

435.gromacs

"435.gromacs is derived from GROMACS, a versatile package that performs molecular dynamics, i.e. simulation of the Newtonian equations of motion for systems with hundreds to millions of particles.

The benchmark version performs a simulation of the protein Lysozyme in a solution of water and ions. The structure of a protein is normally determined by experimental techniques such as X-ray crystallography of NMR spectroscopy. By simulating the atomic motions of these structures, one can gain significant understanding of protein dynamics and function, and, in some cases, it might even be possible to predict the structure of new proteins." [26]

436.cactusADM

"CactusADM is a combination of Cactus, an open source problem solving environment, and BenchADM, a computational kernel representative of many applications in numerical relativity (ADM stands for ADM formalism developed by Arnowitt, Deser and Misner). CactusADM solves the Einstein evolution equations, which describe how space-time curves as response to its matter content, and are a set of ten coupled nonlinear partial differential equations, in their standard ADM 3+1 formulation. A staggered-leapfrog numerical method is used to carry out the update." [26]

437.leslie3d

"437.leslie3d is derived from LESlie3d (*L*arge-*E*ddy *S*imulations with *L*inear-*E*ddy Model in *3D*), a research-level Computational Fluid Dynamics (CFD) code. It is the primary solver used to investigate a wide array of turbulence phenomena such as mixing, combustion, acoustics and general fluid mechanics.

For CPU2006, the program has been set up to solve a test problem which represents a subset of such flows, namely the temporal mixing layer. This type of flow occurs in the mixing regions of all combustors that employ fuel injection (which is nearly all combustors). Also, this sort of mixing layer is a benchmark problem used to understand physics of turbulent mixing." [26]

444.namd

"The 444.namd benchmark is derived from the data layout and inner loop of NAMD, a parallel program for the simulation of large biomolecular systems. Although NAMD was a winner of a 2002 Gordon Bell award for parallel scalability, serial performance is equally important to the over 10,000 users who have downloaded the program over the past several years. Almost all of the runtime is spent calculating inter-atomic interactions in a small set of functions. This set was separated from the bulk of the code to form a compact benchmark for CPU2006. This computational core achieves good performance on a wide range of machines, but contains no platform-specific optimizations." [26]

447.dealII

"The SPEC CPU2006 benchmark 447.dealII is a program that uses deal.II, a C++ program library targeted at adaptive finite elements and error estimation. The library uses state-of-the-art programming techniques of the C++ programming language, including the Boost library. It offers a modern interface to the complex data structures and algorithms required for adaptivity and enables use of a variety of finite elements in one, two, and three space dimensions, as well as time-dependent problems.

The main aim of deal.II is to enable development of modern finite element algorithms, using among other aspects sophisticated error estimators and adaptive meshes. Writing such programs is a non-trivial task, and successful programs tend to become very large and complex." [26]

450.soplex

"450.soplex is based on SoPlex Version 1.2.1. SoPlex solves a linear program using the Simplex algorithm." [26]

453.povray

"POV-Ray is a ray-tracer. Ray-tracing is a rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world but it does so backwards. In the real world, rays of light are emitted from a light source and illuminate objects. The light reflects off of the objects or passes through transparent objects. This reflected light hits the human eye or a camera lens. As the

vast majority of rays never hit an observer, it would take forever to trace a scene. Thus, ray-tracers like POV-Ray start with their simulated camera and trace rays backwards out into the scene. The user specifies the location of the camera, light sources, and objects as well as the surface textures and their interiors." [26]

454.calculix

"454.calculix is based on CalculiX, which is a free software finite element code for linear and nonlinear three-dimensional structural applications. It uses the classical theory of finite elements described in books such as the work by O.C. Zienkiewicz and R.L. Taylor, "The Finite Element Method", Fourth Edition, McGraw Hill, 1989. CalculiX can be used to solve a variety of problems such as static problems (bridge and building design), buckling, dynamic applications (crash, earthquake resistance) and eigenmode analysis (resonance phenomena)." [26]

459.GemsFDTD

"GemsFDTD solves the Maxwell equations in 3D in the time domain using the finite-difference time-domain (FDTD) method. The radar cross section (RCS) of a perfectly conducting (PEC) object is computed. GemsFDTD is a subset of the code GemsTD developed in the General ElectroMagnetic Solvers (GEMS) project." [26]

465.tonto

"Tonto is an open source quantum chemistry package, designed by Dylan Jayatilaka and Daniel J. Grimwood. Objectives include simplicity and portability;

aspects not seen in many quantum chemistry codes. The code is easily extendable by chemists with limited programming skills and time, and is easy to understand and use.

Tonto is written within an object oriented design, in Fortran 95. It uses derived types and modules to represent classes. Classes range from integers and text files, through to atoms, space groups and molecules. There is a "self" variable in most routines, which should be familiar from many OO languages. Tonto uses dynamic memory instead of common blocks, and uses array operations where possible." [26]

470.lbm

"This program implements the so-called "Lattice Boltzmann Method" (LBM) to simulate incompressible fluids in 3D as described in. It is the computationally most important part of a larger code which is used in the field of material science to simulate the behavior of fluids with free surfaces, in particular the formation and movement of gas bubbles in metal foams. For benchmarking purposes and easy optimization for different architectures, the code makes extensive use of macros which hide the details of the data access. A visualization of the results of the submitted code can be seen below (flow through a porous medium, grid size 150x150x150, 1000 time steps)." [26]

481.wrf

"481.wrf is based on the Weather Research and Forecasting (WRF) Model, which is a next-generation mesocale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs.

WRF features multiple dynamical cores, a 3-dimensional variational (3DVAR) data assimilation system, and a software architecture allowing for computational parallelism and system extensibility. The parallel portions of the code have been turned off for SPEC CPU2006 as the interest here is in single processor performance." [26]

482.sphinx3

"Sphinx-3 is a widely known speech recognition system from Carnegie Mellon University. The 482.sphinx3 benchmark focuses on the CPU-intensive portion of this speech recognition system." [26]

3.3. Rodinia Suite

The following benchmarks were taken from the Rodinia Suite.

Leukocyte

The leukocyte application detects and tracks rolling leukocytes (white blood cells) in in vivo video microscopy of blood vessels. The velocity of rolling leukocytes provides important information about the inflammation process, which aids biomedical researchers in the development of anti-inflammatory medications.

In the application, cells are detected in the first video frame and then tracked through subsequent frames. Detection is accomplished by computing for every pixel in the frame the maximal Gradient Inverse Coefficient of Variation (GICOV) score across a range of possible ellipses. The GICOV score for an ellipse is the mean gradient magnitude along the ellipse divided by the standard deviation of the gradient magnitude. The matrix of GICOV scores is then dilated to simplify the process of finding local maxima. For each local maximum, an active contour algorithm is used to more accurately determine the shape of the cell. [27]

LU Decomposition

LU Decomposition is an algorithm to calculate the solutions of a set of linear equations. The LUD kernel decomposes a matrix as the product of a lower triangular matrix and an upper triangular matrix. [27]

SRAD

SRAD (Speckle Reducing Anisotropic Diffusion) is a diffusion method for ultrasonic and radar imaging applications based on partial differential equations (PDEs). It is used to remove locally correlated noise, known as speckles, without destroying important image features. SRAD consists of several pieces of work: image extraction, continuous iterations over the image (preparation, reduction, statistics, computation 1 and computation 2) and image compression. The sequential dependency between all of these stages requires synchronization after each stage (because each stage operates on the entire image). [27]

K-means

K-means are a statistical analysis of clusters where each value is placed into a group where it has the most similar mean. Initial data points are chosen and the subsequent iterations shuffle the data around until there is convergence. K-means is often considered a complex calculation, but over time modified algorithms have improved speed. [27]

Heart Wall

The heart wall benchmark uses several types of processing to create a benchmark testing "braided parallelism". By using image despeckling and edge detection a new image is produced to detect shapes. Once this process is complete ellipses are added to the process and finally the entire image is tracked from frame to frame. This allows for testing parallelism of both multiple tasks and massive amounts of data. [27]

Hot Spot

This benchmark runs a simulation of processor power and temperature and how cells affect their neighbors. These calculations are done by a series of differential equations. The differential equations are run on a temperature map by taking power usage into account until the entire map has been normalized. [27]

Needleman

This benchmark is a nonlinear optimization of DNA. It finds the optimal path to particular cells. Based on cells surrounding elements, initially filled by the program, backwards calculation is done to find proper alignment. The larger the calculated score the closer to a match there is. Scores are calculated by looking at the north, west, and north western cells and points deducted to missing elements. [27]

Particle Filter

This benchmark estimated locations in noisy environments by looking at the location and path of an object. This is done by making guesses, checking their probability, normalizing the guesses, and updating the location of the object. This implementation of a Particle Filter looking at the speed up provided by GPU parallelism in order to make this application possible in real time applications. For our purposes, trials were run on 1, 2 and 5 million data points in 16, 32 and 64 sized processing grids. [27]

3.4. John Burkardt Benchmarks

John Burkardt is a computer programmer that has been working in the computer science field for many years. His first job was at the Pittsburgh Supercomputing Company from 1988 – 1992. Up until current day, he has had a robust career and worked at several places including Bell Helicopter, the Mathematics Department at Iowa State University, Virginia Tech, and is currently working at the Department of Scientific Computing at Florida State University. Mr.

Burkardt has written many applications, mostly for educational purposes, and two of them were used in this project for their parallel computing attributes using the OpenMP library. These two applications are an FFT benchmark and a Prime Number Counting benchmark. [28]

FFT

This benchmark is a C program that computes a Fast Fourier Transform using parallelism via OpenMP. Included in this program is the ability to change the number of threads used. This allows the user to compare the execution time of different numbers of threads. [28]

Prime Number Counting

This benchmark is a C program that counts the number of primes between 1 and *N*. In the default case, *N* is set to 131,072 but can be changed in the source code. For the sake of this project, we only used the default case. This program also allowed the user to change the number of active threads running for comparison of execution times. [28]

3.5. SHOC Suite

Scalable Heterogeneous Computing Benchmark (SHOC) provides benchmarks that are setup to run across GPUs, CPUs, and cluster computing through Message Passing Interface (MPI). While originally designed for use with OpenCL, the current version 1.1.2 supports NVIDIA's CUDA language and MPI. SHOC

provides both stress and performance tests, coving areas from mathematical problem and linear algebra to image processing.

Lennord Jones

This benchmark, based on molecular dynamics, performs nbody computations using the Lennord Jones potential. N atoms are spread randomly over a cubic domain for this particular execution. [29]

Reduction

As the name suggests, this benchmark is massive sum reduction performed on floating point data. Sets of data points are reduced in individual threads before being looked over again until a single number is reached. [29]

Chemical Modeling (S3D)

Using a three dimensional grid, with one thread per point, this benchmark is a massively parallel calculation of chemical rates using S3D. S3D is a direct numerical solver based on the Navier-Stokes equation. This benchmark relies heavily on floating point calculations, an estimated 10 kFLOPS per thread. [29]

Parallel Prefix Sum

Also known as a scan, this benchmark uses the addition of previous smaller summations to yield a new overall result. With the addition of multiple parallel threads, increasing the number of starting values make this harder to run on modern CPU based systems. [29]

Radix Sort

This benchmark is a radix sorting algorithm where sorting is performed based on individual locations within a number. For example sorting can be based first on the highest digit, such as the hundreds place for example, and then proceed to look at the next level down. Sorting as the algorithm processes each level allows for the list to be sorted quickly in parallel. [29]

2D Stencil Operation

A 9 point stencil computation where each value is updated in turn based on the X-point pattern defined by the algorithm. The standard pattern that updates this algorithm remains the same during this benchmark; however, multiple iterations show speedup in larger systems where multiple points can be processed at once. [29]

Vector Dot Product

This benchmark calculates and checks the bandwidth of a given device while performing a vector dot product operation. Vector dot product results in calculations to determine of the vectors are orthogonal by performing normalization and then looking at the angle between the two vectors. [29]

Device Information

These benchmarks include calculating data about the particular card that is being used to execute other benchmarks. These statistics include the download speed of the device, the devices memory capabilities, and the max flops possible on the device. Since these numbers could all be limiting factors in the calculation of the performance of any one device it is important to know where these numbers lie so that they can be taken into account. [29]

3.6. Parboil Suite

Parboil is a suite designed by the IMPACT group, Illinois Microarchitecture Project utilizing Advanced Compiler Technology, mainly for GPUs. Designed to focus on the massively parallelism systems of today's GPUs, modeling and complex mathematics form the basis of the benchmarks. While the IMPACT group also provides a CPU simulation compiler for CUDA, it is used solely for GPU testing in the context of our project.

Breadth First Search

Breadth First Searching a style of graph searching algorithm that goes through every node in a tree until the correct result is found. This algorithm basically looks through the tree layer by layer, finding children for its next round as it goes. It will not skip to any of the found children until every node on the current level is finished being examined. With this type of exhaustive searching large trees can require large amounts of space and time to compute. Benchmarking with a

breadth first searching algorithm will allow us to see the advantages, if any, to more modern multiprocessor multithreaded systems. [30]

Dense Matrix Multiplication

In matrix multiplication, the dot product of a column by a row is performed. The limitations on this idea are that the two matrices must have similar dimensions, or at least have the x dimension of A equal to the y dimension of B. Dense matrix multiplication follows this same principle, except that the properties of a dense matrix are different than those of a normal matrix. Consider for example that every block in a matrix is connected to every other block in the matrix, so that if you change one you affect the entire matrix. This make multiplication of dense matrices a time consuming task for simple non parallel processors and an excellent way to benchmark the idea of parallel processing in modern hardware. [30]

FFT

The Fast Fourier Transform (FFT) is a very important algorithm for Digital Signal Processing (DSP). Simply put, FFT is a computationally efficient method for calculating the Discrete Fourier Transform (DFT). The DFT is a mathematical operation of transforming a time domain function of finite (discrete) size into a frequency domain function. There are several ways of computing the DFT, FFT being one of them. Without FFT, calculating the DFT can be a long and tedious process. The FFT simplifies this process by breaking the set of data into smaller and smaller chunks and then calculating the DFT. For example, say there were 32 data

points. Trying to calculate the DFT on all 32 points at once would be very difficult. FFT breaks down that data set into smaller and smaller sets until, in this case, there are 16 sets of 2. It is much faster and easier to calculate the DFT at a data size of two than at a data size of 30, especially when parallel computing is involved and several of those 16 calculations can be done at the same time.

Sum of Absolute Differences

Sum of absolute differences (SAD) is an algorithm often used in video encoding as it is a comparison between the current image and the next occurring frame. The algorithm simple finds the absolute value of the given image minus the next before proceeding to add them all together. The smallest set is the closest to the same image. SAD is often used across multiple platforms for compressing video, simple animation, and object recognition. [30]

Distance Cutoff Coulombic Potential

This benchmark computes the Coulombic Potential of each grid point in a 3D matrix. This benchmark relies on the calculation of the unequal dipole forces of a water molecule. Speedup is shown by splitting up the calculations among parallel threads. [30]

Saturating Histogram

Using a 2 dimensional matrix, this benchmark calculates a saturated histogram based on the input data. For our particular set of tests, the input data is that of a silicon wafer with a Gaussian representation of data. [30]

Lattice Boltzman

The Lattice Boltzman benchmark is a fluid dynamics simulation within an enclosed container. Within this benchmark, particle collision and general interaction is calculated using a discrete equation. This method for fluid dynamic simulation is easily ported to GPU benchmarking for its ease in parallelism. [30]

Sparse Matrix Dense Vector Multiplication

The SPMV benchmark is similar to the dense matrix multiplication in that all points are related to one another, thus when one point is acted upon it effects the results of other points. In this case the matrix itself is sparsely populated so there are not as many points to work with. This particular execution uses JDS format so it allows for padding with zeros and multiple alignments. [30]

Two Angular Correction Function

The TPACF benchmark performs a statistical analysis of a spatial distribution, often used when measuring astronomical bodies. It calculates a histogram of distances between every set of objects within the data set. Completion of this benchmark places distances, normally on an exponential curve, within reach of a straight sloped line. Parallelism allows for processing of a multitude of points at one time. [30]

4. Results

4.1. Devices

The following is the exact specifications for the devices that these benchmarks were tested on. This information includes: clock rates, system memory, and host operating system among other specifications.

4.1.1. GPU

All of the GPU benchmarks were run on the same system. Since transfer time and driver commands are issued by the CPU we must look at the whole system. The test system is running Ubuntu 10.04 with the Linux kernel version 2.6.32-38-generic. Other system specifications are as follows:

- CPU: AMD Athlon 64 x2 5200+ running at 2.611GHz dual core
- 3GB of system memory, DDR2 800MHz
- GPU 1: NVIDIA GeForce GTX 460
    - CUDA Capability 2.1
    - 336 CUDA capable cores
    - 1024MB of global onboard memory
    - GPU clocks are as follows:
        - Graphics 675MHz
        - Memory 1800MHz
        - Processor 1350MHz
- GPU 2: NVIDIA GeForce 9800 GTX+
    - CUDA Capability 1.1
    - 128 CUDA capable cores
    - 512MB of global onboard memory
    - GPU clocks are as follows:

        - Graphics 738MHz
        - Memory 1100MHz
        - Processor 1836MHz
- System PCIE version 1.1

4.1.2. CPU

All of the CPU benchmarks were also run on the same system know at Worcester Polytechnic Institute as the AMAX machine. This system has the following specifications:

- 2x – Intel Xeon X5650 processors running at 1.6 GHz. The processor is capable by factory default to run at 2.67GHz. These processors have 6 cores and 12 threads a piece, totaling 12 cores and 24 threads using Hyper Threading.
- The total memory of this machine is 24.6 GB.
- The operating system running on this machine is Linux version 2.6.18-274.7.1.e15.
- The compiler is GCC, G++, GFortran versions 4.3.4.
- Intel C++ Compiler XE version 12.0.1.116 build 2010116

4.1.3. FPGA

The FPGA benchmarks were performed using simulations on a Virtex-5 family board. All simulations were performed using the ISIM program through the Xilinx program. Virtex-5 boards come in many different series and are specialized for the following applications taken from the Xilinx datasheet:

- Virtex-5 LX: High-performance general logic applications
- Virtex-5 LXT: High-performance logic with advanced serial connectivity

- Virtex-5 SXT: High-performance signal processing applications with advanced serial connectivity
- Virtex-5 TXT: High-performance systems with double density advanced serial connectivity
- Virtex-5 FXT: High-performance embedded systems with advanced serial connectivity [31]

4.2. All Devices

As a comparison between all three platforms we looked to the Fast Fourier Transform. The FFT benchmark was chosen since it was easily portable between all three devices. For the CPU the benchmark was acquired from John Burkardt, while the GPU one came from the Parboil Suite and the FPGA core came from MATLAB HDL Coder. While the benchmark came from different sources, the FFT implementation was similar. A factor for concern could come from the use of the different style compilers, making different optimizations to the code. The timings from this benchmark can be seen here in Table 1.

FFT Benchmark 262,155 Points							
Benchmark	CPU					GPU	FPGA
	2 Threads	4 Threads	8 Threads	12 Threads	24 Threads	Max Threads	Virtex-5
FFT	76.17 ms	45.41 ms	31.63 ms	27.85 ms	31.36 ms	8.13 us (Execution)	2.59 ms

Table 1 - FFT Results

Using these timings it is obvious that the GPU processed the FFT the fastest. The GPU throughput at an FFT this size is 311 times more than that of the FPGA and 3351 times faster than the CPU Multicore with 12 threads. What is missing from this data is the transfer time for the GPU to send and receive the data. This time is 69 ms for this size data. With this time added in the GPU actually becomes the slowest of all three platforms. Now, the FPGA is the fastest at 26.67 times faster than the GPU and 10.76 times faster than the CPU.

What these results can infer is that depending on the hookup of your GPU to minimize transfer time you could be better off using another platform. As GPU transfer times become smaller and smaller, it is evident that in terms of speed, your best choice will be a GPU.

4.3. CPU & GPU

In this section, we will discuss how the CPU and GPU compared against each other. The three benchmarks that we were able to test across both of these platforms were all from the Rodinia benchmark suite. The Leukocyte, LU Decomposition, and Speckle Reducing Anisotropic Diffusion (SRAD) benchmarks were the three that were executed and the average timings from three runs can be seen as a comparison in Table 2 below.

Benchmark	CPU					GPU
	2 Threads	4 Threads	8 Threads	12 Threads	24 Threads	Max Threads
Leukocyte	11.70 s	6.11 s	3.65 s	2.16 s	1.67 s	0.157 s
LU Decomposition	242.82 ms	130.00 ms	76.40 ms	69.23 ms	145.20 ms	7.38 ms
Speckle Reduction	638.14 ms	415.07 ms	306.41 ms	283.93 ms	492.26 ms	282.93 ms

Table 2 - CPU & GPU Results

Looking at the above data in most cases the GPUs beats out CPUs in raw processing. Raw processing is something to note because, for the GPU iterations, data is all moved to the GPU's global memory before testing. This keeps the GPU from having to ask for data over the slower system bus.

Looking at the times required for the Leukocyte benchmark there is a 10.6x speedup between the 24 Hyper threads of the AXAM machine and the CUDA based GTX 460. Again this is looking at raw processing as even in the benchmark papers it is stated that "Although the overall kernel executed in slightly less than a second, the memory allocation and copying overheads add more than eleven seconds to the overall runtime." [32]

The ultimate success of CUDA results from the many code optimizations and compacting the code into a single kernel; thus, removing much of the extra overhead that decreases speed.

Moving onto the LU Decomposition, it is easy to see that the GPU was much faster than the CPU running this benchmark. This is most likely due to the functionality of LU Decomposition, which is purely linear algebra mathematics.

Since this benchmark is purely mathematical based, the 336 cores and consequently the numerous ALUs of the GPU would be must faster in this type of computation than the 12 cores of the CPU. The only place that the CPU would be able to make up for its slower computation would be in the data transfer portion of this benchmark. In this study though, we are only looking at raw execution and not data transfer, so the GPU is much faster than the CPU in the computation of the LU Decomposition benchmark.

The Speckle Reduction benchmark uses mostly partial differential equations in its computations. As described in the background section, this benchmark consists of several, sequential parts, making synchronization of these parts a necessary feature. This necessity for synchronization is what most likely allowed the CPU execution time to equal that of the GPU. In the flat out computational portions of this benchmark, the GPU would most likely beat the CPU due to its sheer number of available cores. Advancements in CPU pipelining, however, allow the CPU to begin the next instruction while the current one is executing. This basically means that the CPU has the ability to synchronize and start its next instruction at the same time. We believe that is the uniqueness of CPU pipelining that allows it to compete with the GPU in execution time for this benchmark.

4.4. Individual Results

4.4.1. GPU Specific

For any GPU today the biggest area of a setback is in the area of data transfer. Until recently, and still the majority, GPUs were only on dedicated cards requiring a

link to the CPU and memory over a data bus. The limitation of these busses to transfer the computed data severely limits the abilities of GPUs in high performance computing. While bus speeds have come a long way since the original PCI and AGP graphics interfaces, even the newest version of PCIE limits speeds to 32 GB/sec assuming full utilization of every channel both upload and download.

PCIE Version	Raw Bitrate (GT/sec)	Total Bandwidth (GB/sec)
1.0a	2.5	8
1.1	2.5	8
2.0	5	16
2.1	5	16
3.0	8	32
4.0 (theoretical)	16	X

Table 3 - GeForce Specification

Just looking at the specifications for the GeForce GTX460 we can see that its maximum memory bandwidth is clocked at 115.2 GB/sec which is far in excess of the 32 GB/sec we have with the current transfer standard. Looking at the SHOC Benchmark Suite we can view download speeds and the speed of the onboard memory of the tested cards as well. In the following table it is clear that larger file sizes allow for use of more bandwidth, but the graphics cards simply are not capable of utilizing the full bandwidth. The maximum data rate we see below is 3.278 GB/sec whereas the host systems capability was 4 GB/sec in both the upload and download channels. With small file sizes, such as the 1kB file, the graphics card barely reaches a tenth of its potential. As the file size increases past the 512kB point, the time required to move the data increases in a linear fashion with a factor of two.

Focusing on the transfer capabilities of new generations of GPUs would allow for much faster processing of large data sets by significantly reducing transfer times.

Size of Data Chunk (kB)	GB / second	Time (milliseconds)
1	0.0968	0.01057
2	0.1919	0. 01067
4	0.3399	0. 01204
8	0.6074	0. 01348
16	1.0987	0.01491
32	1.5839	0.02069
64	2.1322	0.03074
128	2.5777	0.05084
256	2.8825	0.09094
512	3.0872	0.1698
1024	3.1711	0.3306
2048	3.1773	0.6602
4096	3.1769	1.3211
8192	3.2272	2.5995
16384	3.2552	5.1541
32768	3.2357	10.369
65536	3.2696	20.525
131072	3.2733	41.004
262144	3.2781	81.887
524288	3.2783	163.768

Table 4 - Data Size vs. Data Rate

In contrast to moving data to and from the device, moving data around between the internal memory levels is a faster process. Looking below at Table 5 gives us another look at how slow global memory transfers are in comparison to the internal capabilities of the GPU. The internal memory movement is on the order of a hundred to three hundred GB/sec.

Read / Write	Memory Block Size	Speed (GB/sec)
Local	32	190 / 181
Local	64	288 / 328
Local	128	299 / 388
Local	256	289 / 385
Local	512	277 / 368
Global	32	7.4 / 3.7
Global	64	5.8 / 3.5
Global	128	4.8 / 3.4
Global	256	4.3 / 3.4
Global	512	4.1 / 3.3

Table 5 – Global vs. Local Memory Speeds

As we can see in the table of timings below, from the Parboil Suite of Benchmarks, large portions of time during any given benchmark is transferring data to the GPU or the result back to main memory. While transferring data is not necessarily the bulk of the time, a more efficient method for transferring data is needed to help speed up the overall computation times. As seen in

arboil	Average - 460	Average - 9800	Percent Diff
CUTCP			
GPU	0.037495333	0.036806	-0.92775365
Copy	0.006528333	0.006477667	-0.38956379
FFT			
GPU	0.000812667		
Copy	0.069062		
LBM - long			
GPU	30.15880433	93.86147667	51.36472182
Copy	0.354906667	0.334240333	-2.99882802
LBM - short			
GPU	1.006141333	3.124029333	51.2784621
Copy	0.340840333	0.332132333	-1.29396043
MM - long			
GPU	0.008865333	0.010750667	9.61120174
Copy	0.010738333	0.010793667	0.256981856
SAD			
GPU	0.001495667	0.002197667	19.00722022
Copy	0.115957	0.064151	-28.7638528
SPVM - large			
GPU	0.000219333	0.000319333	18.56435644
Copy	0.076647333	0.054017333	-17.3191426
SPVM - medium			
GPU	0.000117	0.000114	-1.2987013
Copy	0.051963667	0.047360667	-4.6343125
SPVM - small			
GPU	4.73333E-05	4.93333E-05	2.068965517
Copy	0.049963667	0.045377	-4.81081875
TPACF			
GPU	1.361817	1.343129333	-0.69087015
Copy	0.081934	0.051611333	-22.7058976

Table 6 the smallest transfer time is still on the order of milliseconds (6.528 ms); magnitudes larger than a single cycle of execution for today's modern CPUs and GPUs (0.5 nanosec). An interesting note comes from the comparison of the older 9800 card to the current 460; the copy times actually increased by an average of

9.18% for the new architecture. This may be attributed to the added memory banks or new hierarchy.

Parboil	Average - 460	Average - 9800	Percent Diff
CUTCP			
GPU	0.037495333	0.036806	-0.92775365
Copy	0.006528333	0.006477667	-0.38956379
FFT			
GPU	0.000812667		
Copy	0.069062		
LBM - long			
GPU	30.15880433	93.86147667	51.36472182
Copy	0.354906667	0.334240333	-2.99882802
LBM - short			
GPU	1.006141333	3.124029333	51.2784621
Copy	0.340840333	0.332132333	-1.29396043
MM - long			
GPU	0.008865333	0.010750667	9.61120174
Copy	0.010738333	0.010793667	0.256981856
SAD			
GPU	0.001495667	0.002197667	19.00722022
Copy	0.115957	0.064151	-28.7638528
SPVM - large			
GPU	0.000219333	0.000319333	18.56435644
Copy	0.076647333	0.054017333	-17.3191426
SPVM - medium			
GPU	0.000117	0.000114	-1.2987013
Copy	0.051963667	0.047360667	-4.6343125
SPVM - small			
GPU	4.73333E-05	4.93333E-05	2.068965517
Copy	0.049963667	0.045377	-4.81081875
TPACF			
GPU	1.361817	1.343129333	-0.69087015
Copy	0.081934	0.051611333	-22.7058976

Table 6 – Parboil Suite Timings

Continuing to look at the Parboil Suite, there are other noticeable improvements with the new Fermi architecture. The full table of results can be seen in the appendices. Using the new architecture, the time spent on interactions between the CPU and GPU decreased by a noticeable amount. CPU computation time decreased by 8.77% and the time the CPU spent handling GPU commands

decreased by 2.21%. These increases show how newer GPUs are capable of handling more commands by themselves and, while still reliant on CPUs, are moving towards being able to compute by themselves.

Moving to the data from the Rodinia suite other observations on GPU processing can be made. Once again, as with the Parboil suite, we can see that the time taken to move memory around with the newer GTX 460 is still slower. Overall this effect could be due to the test system's PCIE bus. We did not have an intermediate GPU to test floating point speedup with the new architecture, but using the 9800 GTX+ we could observe native increase. Looking at the table below we see that the older GPU beat the new architecture by a very slight margin; an average increase of -0.97% for the GTX 460. Since the older architecture was optimized for standard arithmetic and did not support floating point, this result is reasonable.

Rodinia	Average - 460	Average - 9800	Percent Diff
Particle Float (non-float)			
A			
GPU execution	0.000117667	0.000116889	-0.3315964
B			
GPU execution	0.000129333	0.000129444	0.042936883
C			
GPU execution	0.000126333	0.000123444	-1.15658363
D			
GPU execution	0.000120333	0.000116111	-1.78571429
E			
GPU execution	0.000125333	0.000122111	-1.30220027
F			
GPU execution	0.000123	0.000121	-0.81967213
G			
GPU execution	0.000123667	0.000119889	-1.55109489
H			
GPU execution	0.000129667	0.000127556	-0.82073434

Table 7 - Rodinia Particle Filter Results

4.4.2. CPU Results

SPEC CPUINT2006 Results

There were two instances of the CPUINT2006 benchmark suite run on the CPU for this project. The first run through was done on only a single core and single thread using the GCC, G++, and GFortran 4.3.4 (GNU) compiler. The second was done using the Intel C++ Compiler XE. The GNU compiler run through had the auto parallel feature off, meaning it used only a single thread. The Intel compiler run through had the auto parallel feature enabled, meaning it was using multicore parallelism to complete the benchmark. The results of these two runs can as well as the overall speedup percentages can be seen in the following tables.

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
400.perlbench	398.00	396.00	396.00
401.bzip2	582.00	582.00	581.00
403.gcc	375.00	375.00	375.00
429.mcf	373.00	373.00	371.00
445.gobmk	510.00	511.00	511.00
456.hmmer	869.00	869.00	869.00
458.jeng	595.00	612.00	595.00
462.libquantum	508.00	506.00	506.00
464.h264ref	710.00	706.00	709.00
471.omnetpp	374.00	373.00	374.00
473.atar	494.00	495.00	494.00
483.xalancbmk	268.00	260.00	259.00

Table 8 - SPEC CPUINT2006 w/ GCC, G++, GFortran compiler w/o auto parallel

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
400.perlbench	410.00	411.00	411.00
401.bzip2	539.00	539.00	538.00
403.gcc	330.00	331.00	334.00
429.mcf	184.00	184.00	184.00
445.gobmk	600.00	601.00	607.00
456.hmmer	213.00	214.00	214.00
458.jeng	508.00	508.00	507.00
462.libquantum	12.20	14.30	13.00
464.h264ref	959.00	959.00	1031.00
471.omnetpp	339.00	340.00	339.00
473.atar	350.00	349.00	349.00
483.xalancbmk	214.00	214.00	214.00

Table 9 - SPEC CPUINT2006 run with Intel compiler in auto parallel

	Iteration #1 Speedup	Iteration #2 Speedup	Iteration #2 Speedup
400.perlbench	-3.02%	-3.79%	-3.79%
401.bzip2	7.39%	7.39%	7.40%
403.gcc	12.00%	11.73%	10.93%
429.mcf	50.67%	50.67%	50.40%
445.gobmk	-17.65%	-17.61%	-18.79%
456.hmmer	75.49%	75.37%	75.37%
458.jeng	14.62%	16.99%	14.79%
462.libquantum	97.60%	97.17%	97.43%
464.h264ref	-35.07%	-35.84%	-45.42%
471.omnetpp	9.36%	8.85%	9.36%
473.atar	29.15%	29.49%	29.35%
483.xalancbmk	20.15%	17.69%	17.37%
Average Increase Per Benchmark	21.72%	21.51%	20.37%
Total Average Increase		21.20%	

Table 10 - Speedup percentages from the GNU run to the Intel run

From the tables above, you can see that the multicore, auto parallel run through generally yielded a quicker execution time than the run through with only a single thread. There were, however, a few of the benchmarks that actually ran better using only a single thread. This is probably due to the fact that the work load of these specific benchmarks was better optimized for only a single thread. Overall, the multicore run through produced a 21.2% increase in speed over than of single thread run.

SPEC CFP2006 Results

Once again, there were two instances of the SPEC CPU2006 floating point suite benchmarks run on the CPU. There were two benchmarks in this suite, *bwaves* and *wrf*, which were left out of these runs because of an invalid run error. Both of these benchmarks would build successfully, but every time we tried to run them, we would get this invalid run error that we could not figure out how to fix. Otherwise, all the other benchmarks in the floating point suite ran fine and there timings can be seen in the following two tables. The first table shows the timings for the run on the GCC, G++, and GFortran compiler with the auto parallel feature off (single threaded). The subsequent table shows the timings of the run on the Intel compiler with the auto parallel feature on (multi-threaded).

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
416.gamess	937	940	938
433.milc	479	463	489
435.gromacs	579	579	578
436.cactusADM	1372	1441	1338
437.leslie3d	604	604	603
444.namd	496	497	496
447.dealII	430	429	430
450.soplex	270	270	283
453.povray	236	235	237
454.calculix	1484	1484	1484
459.GemsFDTD	517	517	517
465.tonto	652	649	652
470.lbm	378	379	378
482.sphinx3	632	630	632
434.zeusmp	623	625	625

Table 11 - SPEC CFP2006 w/ GCC, G++, GFortran compiler w/o auto parallel

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
416.gamess	1238	1185	1197
433.milc	190	189	190
435.gromacs	485	489	482
436.cactusADM	60.9	53.1	50.3
437.leslie3d	87.7	90.5	95.8
444.namd	457	456	457
447.dealII	293	293	293
450.soplex	296	263	286
453.povray	191	191	190
454.calculix	382	292	375
459.GemsFDTD	119	122	121
465.tonto	469	462	469
470.lbm	49.9	49.9	50.1
482.sphinx3	528	544	514
434.zeusmp	93.9	93	90.8

Table 12 - SPEC CFP2006 run with Intel compiler in auto parallel

Though there were a few benchmarks that produced a negative speed increase from the single threaded to the multi-threaded run, most benchmarks

displayed a significant increase in speed. The speedup of each benchmarking iteration, as well as the average total speedup of all the run-throughs, can be seen in the following table.

	Iteration #1 Speedup	Iteration #2 Speedup	Iteration #3 Speedup
416.gamess	-32.12%	-26.06%	-27.61%
433.milc	60.33%	59.18%	61.15%
435.gromacs	16.23%	15.54%	16.61%
436.cactusADM	95.56%	96.32%	96.24%
437.leslie3d	85.48%	85.02%	84.11%
444.namd	7.86%	8.25%	7.86%
447.dealII	31.86%	31.70%	31.86%
450.soplex	-9.63%	2.59%	-1.06%
453.povray	19.07%	18.72%	19.83%
454.calculix	74.26%	80.32%	74.73%
459.GemsFDTD	76.98%	76.40%	76.60%
465.tonto	28.07%	28.81%	28.07%
470.lbm	86.80%	86.83%	86.75%
482.sphinx3	16.46%	13.65%	18.67%
434.zeusmp	84.93%	85.12%	85.47%
Average Increase Per Benchmark	42.81%	44.16%	43.95%
Total Average Increase		43.64%	

Table 13 - Speedup percentages from the GNU run to the Intel run

As you can see, there was an average total speedup of 43.64% for the SPEC CPU2006 floating point suite. This is over two times the speedup we saw from the CPU2006 integer suite. We believe that such an increase in speedup is due to the complexity of floating point operations. A single thread would be able to run simpler integer operations faster than complex floating point operations. Thus, you would see a more significant speed increase when using the multi-threaded

capabilities of a CPU to calculate floating point operations as opposed to integer operations.

Rodinia / John Burkardt Results

Each of the Rodinia and Burkardt benchmarks was run five different times at three iterations a piece using 2, 4, 8, 12, and 24 threads. Since the CPU we were testing on has two processors, totaling 12 cores, we believed that this range of thread counts would cover the most practical multithreading situations. The following table shows the average execution time for three iteration of each benchmark on the various thread counts. The first five benchmarks in this table are from the Rodinia suite and the other two are from Burkardt suite.

	2 threads	4 threads	8 threads	12 threads	24 threads
Leukocyte (s)	11.70	6.11	3.65	2.16	1.67
LU Decomposition (ms)	242.82	130.00	76.40	69.23	145.20
Speckle Reduction (ms)	638.14	415.07	306.41	283.93	492.26
Means (s)	3.35	3.67	2.91	2.16	1.67
Stream Clusters (s)	47.08	25.18	14.61	11.91	10.40
FFT (ms)	76.17	45.41	31.63	27.85	31.36
Primes (s)	2.04	1.17	0.64	0.44	0.36

Table 14 - Rodinia/Burkardt average execution time on 2, 4, 8, 12, 24 threads

As you can see, the execution times change when transitioning to a different number of threads. The total speedup percentage change between thread counts can be seen in the following table.

	2-4 threads	4-8 threads	8-12 threads	12-24 threads
Leukocyte (s)	47.78%	40.26%	40.82%	22.69%
LU Decomposition (ms)	46.46%	41.23%	9.38%	-109.74%
Speckle Reduction (ms)	34.96%	26.18%	7.34%	-73.37%
kmeans (s)	-9.55%	20.71%	25.77%	22.69%
Stream Clusters (s)	46.52%	41.98%	18.48%	12.68%
FFT (ms)	40.38%	30.35%	11.95%	-12.60%
Primes (s)	42.65%	45.30%	31.25%	18.18%
Average Increase Between Threads	35.60%	35.14%	20.71%	-17.07%

Table 15 - Speedup between thread counts

Several interesting observations can be made from the information in the table above. First, we saw that three out of the seven benchmarks had a decrease in speed when transitioning between 12 and 24 threads. This is most likely due to the hardware limitations of our processor. At twelve threads, our processor can dedicate one core to each thread because it has a total of 12 cores. Once we transition to 24 threads though, 12 of the 24 threads become "virtual" threads that are implemented at the software level. This basically means that each core is handling two threads apiece. While this technology may be good for some applications, running two threads on a single core can sometimes produce slower execution times than using a single thread per core.

The other interesting observation was that the average speed increase from 2 to 4 and 4 to 8 hovered around a 35% increase while the increase between 8 to 12 dropped down to 20%. This shows that there is a significant speed increase up until 8 threads, but beyond that, the speedup begins to become less prevalent.

4.4.3. FPGA Results

As is evident from the benchmarks run on the FPGA, they do not usually perform as complex tasks as the CPU and GPU benchmarks show. The applications they are used for are generally specific and are used to enhance applications for other processes.

Benchmark	Clk Period (MHz)	Clk Cycles	Throughput (ns)	Delay for valid data (Clock Cycles)	Delay (ns)
FFT	101	1	9.87	12	118
AES	376	1	2.66	1	2.66
FIR	710	1	1.41	8	1.13
FP Mul	550	9	16.4	9	46.4
FIR Core	550	11	20.0	20	36.4

Table 16 - FPGA Results

As the Table above shows, the majority of applications run on the FPGA do not take much time between outputs, but there is usually a larger delay before the output is actually available. All of these benchmarks were designed using a pipelining implementation. This allowed the FPGA to use the ability to break up tasks and use internal storage to speed up the overall throughput.

FPGAs are very useful to high performance computing, however on an application specific basis. The advantage of having a higher processing power

compared to CPU and the ability to customize the data transfer method to meet your needs is great. However, it is vital in today's computing to find the board that has the proper computational slices to meet the needs of your application.

5. Future Work

For future projects similar to this one, the main topics to focus on would be a broader spectrum of benchmarks capable of running across all three platforms as well as possibly looking into newer technologies, such as an Accelerated Processing Unit (APU). Benchmarking of clusters or testing a single benchmark using multiple platforms for speedup would also be of use in future research.

While most of today's benchmarks lie in the realm of scientific and mathematical algorithms, it would be beneficial to create cross platform benchmarks across other types of general processing. Examples of general processing benchmarks could be ones that handle word processing, weather tracking, molecule design, encryption, and data compression, all with the capability of running on all three platforms.

Newer technologies have allowed designers to put both CPUs and GPUs on the same die, the concept behind APUs. This decreases data transfer times and allows for newer instruction sets to incorporate both units. These new devices have the ability to provide substantial performance increases to the processing world. With this is mind, it would be beneficial to benchmark these new platforms against their predecessors.

6. Conclusion

High Performance Computing is a rapidly growing field that will require more research to understand. The technology surrounding CPUs, GPUs, and FPGAs is still rapidly evolving and will continue in future years. Benchmarking will be a constant process to stratify different systems as well as different devices. With the information in this report, some light is shed on the processing power for different applications between CPUs, GPUs, and FPGAs.

Overall, 66 benchmarks were investigated over eight suites and sources to gather information. These results are useful to compare the three systems discussed as well as in comparison with other devices during future studies. The world of High Performance Computing is a constantly evolving field that will play a significant role in the years to come in many diverse fields.

7. Appendices

7.1. Parboil Results 9800 GTX+

Parboil	Iteration 1	Iteration 2	Iteration 3	Average	SD
CUTCP					
IO	0.039047	0.047604	0.042384	0.0430117	0.00431289
GPU	0.036803	0.036811	0.036804	0.036806	4.3589E-06
Copy	0.006495	0.006459	0.006479	0.0064777	1.8037E-05
Driver	0.000139	0.000141	0.000141	0.0001403	1.1547E-06
Compute	0.199751	0.199162	0.199347	0.19942	0.00030121
CPU Overlap	0.036803	0.036811	0.036804	0.036806	4.3589E-06
LBM - long					
IO	0.040944	0.040932	0.040948	0.0409413	8.3267E-06
GPU	93.829116	93.867864	93.88745	93.861477	0.02968691
Copy	0.335708	0.333089	0.333924	0.3342403	0.00133785
Driver	77.793763	77.834447	77.81375	77.813987	0.02034303
Compute	1.016244	1.009388	1.009609	1.011747	0.00389608
CPU Overlap	78.004436	78.050671	78.00946	78.021522	0.02536816
LBM - short					
IO	0.049419	0.051679	0.051344	0.050814	0.00121966
GPU	3.121433	3.125452	3.125203	3.1240293	0.00225193
Copy	0.331451	0.332959	0.331987	0.3321323	0.00076443
Driver	0.000734	0.000679	0.000699	0.000704	2.7839E-05
Compute	0.920874	0.976684	0.968332	0.9552967	0.03010198
CPU Overlap	0.124557	0.125872	0.12585	0.1254263	0.00075295
MM - long					
IO	3.084938	3.099593	3.09249	3.0923403	0.00732865
GPU	0.01076	0.010734	0.010758	0.0107507	1.4468E-05
Copy	0.010821	0.010749	0.010811	0.0107937	3.9004E-05
Driver	0.000104	0.000124	0.000121	0.0001163	1.0786E-05
Compute	0.051372	0.051629	0.051597	0.0515327	0.00014006
CPU Overlap	0.000146	0.000169	0.000159	0.000158	1.1533E-05
GFLOPS	1.55E-13	1.55E-13	1.55E-13	1.554E-13	1.4673E-16
SAD					
IO	0.166941	0.20198	0.198754	0.189225	0.0193658
GPU	0.00221	0.002185	0.002198	0.0021977	1.2503E-05

Copy	0.063778	0.064767	0.063908	0.064151	0.00053742
Driver	0.00003	0.000031	0.000029	0.00003	0.000001
Compute	0.000727	0.00849	0.007983	0.0057333	0.00434302
CPU Overlap	0.000042	0.000042	0.000041	4.167E-05	5.7735E-07
SPVM - large					
IO	0.142549	0.087202	0.113094	0.1142817	0.02769261
GPU	0.000319	0.000319	0.00032	0.0003193	5.7735E-07
Copy	0.053893	0.054192	0.053967	0.0540173	0.00015573
Driver	0.000063	0.000062	0.000062	6.233E-05	5.7735E-07
Compute	0.004965	0.004981	0.004978	0.0049747	8.5049E-06
CPU Overlap	0.000079	0.000079	0.00008	7.933E-05	5.7735E-07
SPVM - medium					
IO	0.024611	0.02542	0.02499	0.025007	0.00040477
GPU	0.000117	0.000112	0.000113	0.000114	2.6458E-06
Copy	0.047665	0.047027	0.04739	0.0473607	0.00032001
Driver	0.000064	0.00006	0.000061	6.167E-05	2.0817E-06
Compute	0.002742	0.002828	0.00279	0.0027867	4.3097E-05
CPU Overlap	0.00008	0.000075	0.000079	0.000078	2.6458E-06
SPVM - small					
IO	0.021669	0.000503	0.02007	0.0140807	0.01178575
GPU	0.000051	0.000048	0.000049	4.933E-05	1.5275E-06
Copy	0.045435	0.045306	0.04539	0.045377	6.5475E-05
Driver	0.000043	0.000041	0.000041	4.167E-05	1.1547E-06
Compute	0.002261	0.002294	0.002278	0.0022777	1.6503E-05
CPU Overlap	0.000054	0.000052	0.000052	5.267E-05	1.1547E-06
TPACF					
IO	1.129532	1.067352	1.07802	1.0916347	0.03325068
GPU	1.343114	1.343151	1.343123	1.3431293	1.9296E-05
Copy	0.051403	0.051871	0.05156	0.0516113	0.00023819
Driver	0.000093	0.000089	0.00009	9.067E-05	2.0817E-06
Compute	0.019158	0.0188	0.019022	0.0189933	0.00018071
CPU Overlap	0.000137	0.000129	0.000131	0.0001323	4.1633E-06

7.2. Parboil Results GTX 460

Parboil	Iteration 1	Iteration 2	Iteration 3	Average	SD
CUTCP					
IO	0.024785	0.026053	0.026624	0.025820667	0.000941257
GPU	0.037782	0.03735	0.037354	0.037495333	0.000248269
Copy	0.006324	0.006336	0.006925	0.006528333	0.000343576
Driver	0.000138	0.000148	0.000155	0.000147	8.544E-06
Compute	0.198731	0.199476	0.199039	0.199082	0.000374357
CPU Overlap	0.037782	0.03735	0.037354	0.037495333	0.000248269
FFT					
IO	0.052376	0.040751	0.041094	0.044740333	0.006614905
GPU	0.000813	0.000812	0.000813	0.000812667	5.7735E-07
Copy	0.074595	0.066348	0.066243	0.069062	0.004792006
Driver	0.000047	0.000048	0.000047	4.73333E-05	5.7735E-07
Compute	0.000353	0.000467	0.000369	0.000396333	6.17198E-05
CPU Overlap	0.000062	0.000064	0.000064	6.33333E-05	1.1547E-06
Histogram					
IO	0.152658	0.156614	0.158582	0.155951333	0.003017083
GPU	0.140838	0.140679	0.140491	0.140669333	0.000173702
Copy					
Driver	0.139518	0.139357	0.139178	0.139351	0.000170079
Compute	0.00043	0.000496	0.000457	0.000461	3.31813E-05
CPU Overlap	0.140838	0.140679	0.140491	0.140669333	0.000173702
LBM - long					
IO	0.047836	0.049771	0.049561	0.049056	0.001061756
GPU	30.16735	30.14701	30.16205	30.15880433	0.010548376
Copy	0.363583	0.352151	0.348986	0.354906667	0.007678761
Driver	24.9618	24.94686	24.94136	24.95000833	0.01057388
Compute	0.961555	0.996269	1.013636	0.990486667	0.02651762
CPU Overlap	25.16581	25.14708	25.14502	25.152638	0.011457143
LBM - short					
IO	0.054089	0.060643	0.075741	0.063491	0.011103405
GPU	1.005346	1.004551	1.008527	1.006141333	0.002103939
Copy	0.345664	0.336826	0.340031	0.340840333	0.00447424
Driver	0.000834	0.000976	0.000836	0.000882	8.14125E-05
Compute	1.020029	0.9414	0.927952	0.963127	0.049735203
CPU Overlap	0.125006	0.123213	0.112586	0.120268333	0.006713225
MM - long					
IO	3.195248	3.154655	3.190209	3.180037333	0.022125664
GPU	0.008869	0.008867	0.00886	0.008865333	4.72582E-06

Copy	0.010757	0.010806	0.010652	0.010738333	7.86787E-05
Driver	0.000109	0.000132	0.000116	0.000119	1.17898E-05
Compute	0.092311	0.081318	0.080229	0.084619333	0.006683396
CPU Overlap	0.000153	0.000179	0.000158	0.000163333	1.37961E-05
GFLOPS	1.55E-13	1.55E-13	1.55E-13	1.55371E-13	1.15326E-17
SAD					
IO	0.213267	0.153	0.206935	0.191067333	0.033118952
GPU	0.001498	0.001493	0.001496	0.001495667	2.51661E-06
Copy	0.126214	0.102197	0.11946	0.115957	0.012385771
Driver	0.000038	0.000035	0.000035	0.000036	1.73205E-06
Compute	0.000981	0.000795	0.000721	0.000832333	0.00013396
CPU Overlap	0.000053	0.00005	0.000049	5.06667E-05	2.08167E-06
SPVM - large					
IO	0.146448	0.148342	0.148184	0.147658	0.001050864
GPU	0.000218	0.00022	0.00022	0.000219333	1.1547E-06
Copy	0.084568	0.075166	0.070208	0.076647333	0.007293707
Driver	0.000066	0.000068	0.000066	6.66667E-05	1.1547E-06
Compute	0.004543	0.004651	0.004879	0.004691	0.000171534
CPU Overlap	0.000081	0.000084	0.000082	8.23333E-05	1.52753E-06
SPVM - medium					
IO	0.023466	0.024767	0.024647	0.024293333	0.000719
GPU	0.000118	0.000116	0.000117	0.000117	1E-06
Copy	0.051357	0.05229	0.052244	0.051963667	0.000525892
Driver	0.000063	0.000061	0.000062	0.000062	0.000001
Compute	0.002235	0.002791	0.002609	0.002545	0.000283471
CPU Overlap	0.000078	0.000076	0.000077	0.000077	1E-06
SPVM - small					
IO	0.007575	0.014441	0.015238	0.012418	0.00421305
GPU	0.000048	0.000047	0.000047	4.73333E-05	5.7735E-07
Copy	0.049176	0.050157	0.050558	0.049963667	0.000710995
Driver	0.000043	0.000042	0.000043	4.26667E-05	5.7735E-07
Compute	0.001726	0.002104	0.00209	0.001973333	0.000214311
CPU Overlap	0.000055	0.000054	0.000054	5.43333E-05	5.7735E-07
TPACF					
IO	1.18402	1.18196	1.177699	1.181226333	0.003223734
GPU	1.361854	1.361822	1.361775	1.361817	3.97366E-05
Copy	0.095364	0.075245	0.075193	0.081934	0.01163075
Driver	0.000113	0.00011	0.0001	0.000107667	6.80686E-06
Compute	0.00904	0.01928	0.020788	0.016369333	0.006392015
CPU Overlap	0.000156	0.000152	0.000142	0.00015	7.2111E-06

7.3. Rodinia Results 9800 GTX+

Rodinia	Iteration 1	Iteration 2	Iteration 3	Average	SD
LUD	ms				
64	0.238	0.239	0.239	0.238666667	0.00057735
256	1.185	1.512	1.456	1.384333333	0.17488377
512	3.311	3.315	3.312	3.312666667	0.00208167
2048	34.445	34.534	34.511	34.49666667	0.04619885
Particle Float (naïve)	sec				
A					
send from GPU	0.02881	0.034164	0.032453	0.031809	0.00273448
send to GPU	0.037435	0.037895	0.03769	0.037673333	0.00023045
GPU execution	0.00016	0.000167	0.000166	0.000164333	3.7859E-06
Total	3.284789	3.280237	3.28295	3.282658667	0.00228994
B					
send from GPU	0.059425	0.060929	0.060115	0.060156333	0.00075285
send to GPU	0.074129	0.074013	0.074069	0.074070333	5.8011E-05
GPU execution	0.000162	0.000164	0.000164	0.000163333	1.1547E-06
Total	6.524812	6.702561	6.593282	6.606885	0.08965187
C					
send from GPU	0.146823	0.149165	0.148302	0.148096667	0.00118442
send to GPU	0.186539	0.184094	0.185983	0.185538667	0.00128163
GPU execution	0.000173	0.000169	0.00017	0.000170667	2.0817E-06
Total	16.691667	16.667685	16.682983	16.68077833	0.01214205
D					
send from GPU	0.029974	0.029582	0.029834	0.029796667	0.00019865
send to GPU	0.037564	0.037992	0.0376982	0.0377514	0.0002189
GPU execution	0.000154	0.000159	0.000156	0.000156333	2.5166E-06
Total	3.278011	3.289114	3.283495	3.28354	0.00555164
E					
send from GPU	0.058106	0.056998	0.057398	0.057500667	0.00056109
send to GPU	0.074384	0.073757	0.074287	0.074142667	0.0003375
GPU execution	0.000167	0.000159	0.000166	0.000164	4.3589E-06
Total	6.527779	6.501522	6.51213	6.513810333	0.0132089
F					
send from GPU	0.141125	0.139959	0.13997	0.140351333	0.00067004
send to GPU	0.184281	0.182941	0.183213	0.183478333	0.00070831
GPU execution	0.000169	0.000166	0.000168	0.000167667	1.5275E-06
Total	16.157624	16.205709	16.199371	16.187568	0.02612518
G					
send from GPU	0.029478	0.029737	0.029587	0.029600667	0.00013004
send to GPU	0.037714	0.037855	0.037729	0.037766	7.744E-05

GPU execution	0.000163	0.00016	0.000161	0.000161333	1.5275E-06
Total	3.286379	3.285156	3.285983	3.285839333	0.00062403
H					
send from GPU	0.058453	0.057608	0.057983	0.058014667	0.00042339
send to GPU	0.074478	0.074515	0.745983	0.298325333	0.38768291
GPU execution	0.000163	0.000173	0.001064	0.000466667	0.00051733
Total	6.700087	6.518035	6.690865	6.636329	0.10254933

7.4. Rodinia Results GTX 460

Rodinia	Iteration 1	Iteration 2	Iteration 3	Average	SD
Leukocyte	sec				
Detection					
computation	0.0185	0.01944	0.01946	0.019133333	0.000548574
dilation	0.01006	0.01068	0.01068	0.010473333	0.000357957
total	0.08415	0.09912	0.09981	0.09436	0.008848847
Tracking	sec				
computation	0.04268	0.04274	0.04282	0.042746667	7.02377E-05
evolution	0.01027	0.01027	0.01037	0.010303333	5.7735E-05
total	0.0655	0.06538	0.06544	0.06544	6E-05
TOTAL	4.01395	4.02195	4.02639	4.020763333	0.006304327
LUD	ms				
64	0.575	0.579	0.575	0.576333333	0.002309401
256	2.848	2.828	2.83	2.835333333	0.011015141
512	7.374	7.388	7.387	7.383	0.00781025
2048	115.623	115.375	117.59	116.196	1.213590953
Particle Float (float)	sec				
A					
send from GPU	0.484112	0.48184	0.480469	0.482140207	0.001840149
send to GPU	0.014158	0.013941	0.013862	0.013987	0.000153268
GPU execution	0.000251	0.000258	0.000255	0.000254667	3.51188E-06
Total	0.65792	0.584687	0.61418	0.618929	0.036846748
B					
send from GPU	0.977968	0.979106	0.973441	0.976838333	0.002996693
send to GPU	0.028367	0.027495	0.027456	0.027772667	0.000515077
GPU execution	0.000267	0.00024	0.000237	0.000248	1.65227E-05
Total	1.099265	1.099386	1.094044	1.097565	0.003049876
D					
send from GPU	21.25275	14.00893	30.37887	21.88018267	8.202989064

send to GPU	0.013946	0.013852	0.013902	0.0139	4.70319E-05
GPU execution	0.000265	0.00253	0.000255	0.001016667	0.001310595
Total	21.34469	14.10073	30.46716	21.97086	8.201161321
E					
send from GPU	8.556762	8.546939	8.539814	8.547838333	0.008509717
send to GPU	0.027564	0.027571	0.027614	0.027583	2.7074E-05
GPU execution	0.000234	0.000244	0.000259	0.000245667	1.25831E-05
Total	8.675124	8.666531	8.660525	8.667393333	0.007337603
G					
send from GPU	32.85077	13.84569	14.8465	20.51432133	10.69539136
send to GPU	0.013938	0.013869	0.013931	0.013912667	3.79781E-05
GPU execution	0.000251	0.000245	0.000248	0.000248	3E-06
Total	32.94073	13.93179	14.93657	20.603027	10.69656602
H					
send from GPU	8.535585	8.5487	8.544959	8.543081333	0.006756111
send to GPU	0.027881	0.027692	0.027521	0.027698	0.000180075
GPU execution	0.000245	0.000236	0.000241	0.000240667	4.50925E-06
Total	8.656592	8.669315	8.670502	8.665469667	0.007711159
Particle Float (naïve)	sec				
A					
send from GPU	18.03987	11.67335	11.65105	13.78808867	3.682170995
send to GPU	0.037767	0.037449	0.037533	0.037583	0.000164791
GPU execution	0.00012	0.00012	0.000113	0.000117667	4.04145E-06
Total	21.40222	14.9382	14.92392	17.08811367	3.736133375
B					
send from GPU	8.695787	9.222604	9.195602	9.037997667	0.296670494
send to GPU	0.075593	0.075548	0.0736	0.074913667	0.001137891
GPU execution	0.000129	0.000118	0.000141	0.000129333	1.15036E-05
Total	15.20682	15.67589	15.67246	15.518391	0.269830416
C					
send from GPU	14.00524	8.159942	8.216452	10.12720967	3.358587309
send to GPU	0.187186	0.183569	0.185398	0.185384333	0.001808539
GPU execution	0.000135	0.000118	0.000126	0.000126333	8.5049E-06
Total	30.2434	24.24978	24.24987	26.247683	3.460390697
D					
send from GPU	12.49726	12.42138	12.48212	12.46691867	0.040157217
send to GPU	0.038156	0.037788	0.037667	0.037870333	0.000254685
GPU execution	0.000133	0.000117	0.000111	0.000120333	1.13725E-05
Total	15.77098	15.77281	15.74104	15.76160833	0.017839721
E					
send from GPU	10.85223	10.2257	9.016069	10.031332	0.933383694
send to GPU	0.075725	0.075797	0.07437	0.075297333	0.000803901

GPU execution	0.000135	0.000117	0.000124	0.000125333	9.07377E-06
Total	46.67132	16.67096	15.6721	26.33812367	17.6161456
F					
send from GPU	14.11983	15.16998	14.26351	14.51777167	0.56937989
send to GPU	0.18705	0.185197	0.183663	0.185303333	0.001696002
GPU execution	0.000129	0.00012	0.00012	0.000123	5.19615E-06
Total	30.24967	31.24654	30.24503	30.58041333	0.576885553
G					
send from GPU	13.49624	12.51239	12.51785	12.84216133	0.566457901
send to GPU	0.037861	0.037656	0.037663	0.037726667	0.000116389
GPU execution	0.000135	0.000125	0.000111	0.000123667	1.20554E-05
Total	16.7578	15.77179	15.77775	16.102446	0.567560183
H					
send from GPU	9.182923	10.20361	9.200904	9.529146333	0.584173588
send to GPU	0.075046	0.075518	0.074179	0.074914333	0.000679141
GPU execution	0.000136	0.000123	0.00013	0.000129667	6.50641E-06
Total	15.67639	16.6652	15.67365	16.00507867	0.571680899

7.5. SHOC Max Flops GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
Add1-DP	Size:2097152	GFLOPS	37.7811	37.7811	6.28041e-05	37.781	37.7812	37.781	37.7812	37.7811	37.7812	37.781	37.7812	37.781	37.7811	37.7811	37.7811
Add1-SP	Size:2097152	GFLOPS	298.222	298.233	0.0364559	298.193	298.322	298.223	298.193	298.214	298.222	298.263	298.193	298.25	298.218	298.232	298.322
Add2-DP	Size:2097152	GFLOPS	37.7759	37.7759	4.89902e-05	37.7758	37.776	37.7759	37.7759	37.776	37.7759	37.776	37.776	37.7759	37.7759	37.7758	37.7759
Add2-SP	Size:2097152	GFLOPS	452.568	452.569	0.00851849	452.549	452.579	452.564	452.549	452.576	452.568	452.568	452.579	452.567	452.579	452.564	452.572
Add4-DP	Size:2097152	GFLOPS	37.7672	37.7672	5.91473e-05	37.7671	37.7673	37.7673	37.7672	37.7672	37.7671	37.7672	37.7672	37.7671	37.7672	37.7671	37.7671
Add4-SP	Size:2097152	GFLOPS	451.925	451.922	0.00596587	451.908	451.929	451.908	451.926	451.919	451.926	451.925	451.929	451.916	451.926	451.925	451.922
Add8-DP	Size:2097152	GFLOPS	37.7414	37.7414	0.00013747	37.7412	37.7417	37.7416	37.7415	37.7414	37.7414	37.7417	37.7415	37.7413	37.7412	37.7414	37.7414
Add8-SP	Size:2097152	GFLOPS	451.566	451.565	0.00516266	451.552	451.572	451.561	451.567	451.552	451.566	451.566	451.572	451.57	451.565	451.564	451.568
MAdd1-DP	Size:2097152	GFLOPS	75.5511	75.5511	8.77467e-05	75.5509	75.5513	75.5513	75.5511	75.5512	75.5512	75.5511	75.5512	75.5511	75.5509	75.551	75.5511
MAdd1-SP	Size:2097152	GFLOPS	595.109	595.112	0.122587	594.878	595.302	595.243	595.09	595.129	595.074	595.208	594.954	595.174	595.072	595.302	594.878
MAdd2-DP	Size:2097152	GFLOPS	75.5488	75.5488	5.76678e-05	75.5488	75.549	75.5489	75.5489	75.5488	75.5488	75.5488	75.549	75.5488	75.5488	75.5488	75.5489
MAdd2-SP	Size:2097152	GFLOPS	881.025	881.074	0.380338	880.478	881.587	881.574	880.715	881.037	881.587	880.609	880.934	881.4	881.395	881.013	880.478
MAdd4-DP	Size:2097152	GFLOPS	75.5051	75.5052	0.00023447	75.5049	75.5057	75.5053	75.5053	75.5057	75.505	75.5055	75.5051	75.5052	75.505	75.5051	75.5049
MAdd4-SP	Size:2097152	GFLOPS	885.019	885.01	0.0323639	884.918	885.037	884.918	885.021	885.032	885.018	885.013	885.037	885.022	885.025	885.016	885
MAdd8-DP	Size:2097152	GFLOPS	75.4606	75.4606	0.00021646	75.4603	75.461	75.4608	75.461	75.4603	75.4606	75.4608	75.4607	75.4604	75.4603	75.4606	75.4605
MAdd8-SP	Size:2097152	GFLOPS	869.364	869.232	0.275908	868.481	869.416	869.357	869.402	869.395	869.414	869.416	868.481	869.184	869.259	869.371	869.042
MAddU-DP	Size:4194304	GFLOPS	75.4371	75.4371	1.47598e-05	75.4371	75.4371	75.4371	75.4371	75.4371	75.4371	75.4371	75.4371	75.4371	75.4371	75.4371	75.4371
MAddU-SP	Size:4194304	GFLOPS	528.465	528.476	0.143183	528.253	528.696	528.696	528.317	528.408	528.343	528.253	528.554	528.422	528.616	528.648	528.509
Mul1-DP	Size:2097152	GFLOPS	37.7822	37.7822	1.90731e-05	37.7822	37.7823	37.7823	37.7823	37.7823	37.7823	37.7823	37.7822	37.7822	37.7822	37.7822	37.7822
Mul1-SP	Size:2097152	GFLOPS	300.036	300.037	0.0318387	299.989	300.1	299.989	300.048	300.032	299.997	300.027	300.016	300.04	300.054	300.1	300.072
Mul2-DP	Size:2097152	GFLOPS	37.7798	37.7798	3.16594e-05	37.7797	37.7798	37.7798	37.7798	37.7798	37.7798	37.7798	37.7798	37.7798	37.7798	37.7797	37.7798
Mul2-SP	Size:2097152	GFLOPS	449.894	449.967	0.122553	449.838	450.194	449.883	449.88	449.881	449.897	450.123	449.892	450.125	449.838	450.194	449.96
Mul4-DP	Size:2097152	GFLOPS	37.7696	37.7696	4.18636e-05	37.7695	37.7697	37.7697	37.7695	37.7697	37.7697	37.7696	37.7696	37.7696	37.7696	37.7696	37.7696
Mul4-SP	Size:2097152	GFLOPS	452.382	452.382	0.00149423	452.381	452.386	452.381	452.381	452.386	452.382	452.384	452.383	452.383	452.381	452.383	452.381
Mul8-DP	Size:2097152	GFLOPS	37.7576	37.7576	0.00013146	37.7574	37.7578	37.7575	37.7575	37.7574	37.7577	37.7576	37.7578	37.7578	37.7575	37.7576	37.7578
Mul8-SP	Size:2097152	GFLOPS	451.532	451.532	0.00179845	451.529	451.535	451.531	451.533	451.532	451.535	451.533	451.532	451.532	451.532	451.529	451.534
MulMAdd1-DP	Size:2097152	GFLOPS	56.6703	56.6703	4.64428e-05	56.6703	56.6704	56.6703	56.6704	56.6704	56.6704	56.6704	56.6704	56.6703	56.6703	56.6703	56.6703
MulMAdd1-SP	Size:2097152	GFLOPS	447.744	447.736	0.059606	447.638	447.857	447.857	447.644	447.638	447.778	447.728	447.752	447.736	447.72	447.753	447.754
MulMAdd2-DP	Size:2097152	GFLOPS	56.6687	56.6687	4.19358e-05	56.6686	56.6687	56.6687	56.6687	56.6687	56.6687	56.6687	56.6687	56.6686	56.6686	56.6687	56.6687
MulMAdd2-SP	Size:2097152	GFLOPS	671.839	671.837	0.00651623	671.826	671.844	671.831	671.836	671.844	671.843	671.839	671.839	671.826	671.844	671.826	671.841
MulMAdd4-DP	Size:2097152	GFLOPS	56.659	56.6589	0.000199583	56.6584	56.6591	56.659	56.659	56.659	56.659	56.6591	56.659	56.659	56.6584	56.6587	56.6591
MulMAdd4-SP	Size:2097152	GFLOPS	613.356	613.357	0.00715913	613.347	613.372	613.351	613.353	613.347	613.361	613.364	613.355	613.357	613.362	613.35	613.372
MulMAdd8-DP	Size:2097152	GFLOPS	56.6316	56.6316	0.000117022	56.6314	56.6318	56.6317	56.6314	56.6315	56.6316	56.6317	56.6316	56.6318	56.6316	56.6318	56.6315
MulMAdd8-SP	Size:2097152	GFLOPS	644.746	644.745	0.00695146	644.733	644.756	644.736	644.739	644.744	644.756	644.741	644.751	644.733	644.75	644.748	644.748
MulMAddU-DP	Size:4194304	GFLOPS	56.3124	56.3124	1.81655e-05	56.3123	56.3124	56.3124	56.3124	56.3123	56.3124	56.3124	56.3124	56.3124	56.3124	56.3124	56.3124
MulMAddU-SP	Size:4194304	GFLOPS	450.022	450.003	0.0502006	449.895	450.047	449.895	450.034	450.028	450.011	450.039	450.04	449.919	450.006	450.047	450.016

7.6. SHOC Bus Download Speed GTX 460

test	atts	units	median	mean	stddev	min	max	\trial0	trial1	trial2	trial3	trial4	trial5	trial6
DownloadSpeed	1kB	GB/sec	0.0968232	0.0944747	0.00719869	0.0728929	0.0972644	0.0728929	0.0966767	0.0972644	0.0969697	0.0963855	0.0969697	0.0969697
DownloadSpeed	2kB	GB/sec	0.191908	0.172397	0.0523309	0.0161575	0.193939	0.180791	0.189349	0.0161575	0.192771	0.179272	0.193353	0.193353
DownloadSpeed	4kB	GB/sec	0.339974	0.337767	0.00640744	0.321608	0.343164	0.331606	0.321608	0.343164	0.335079	0.339523	0.341333	0.342246
DownloadSpeed	8kB	GB/sec	0.607356	0.605821	0.00527627	0.591224	0.609524	0.608076	0.606635	0.606635	0.591224	0.609524	0.606635	0.609524
DownloadSpeed	16kB	GB/sec	1.09871	1.08759	0.0318089	0.996109	1.11063	1.07338	1.09871	1.09636	1.11063	0.996109	1.10108	1.09871
DownloadSpeed	32kB	GB/sec	1.58391	1.46017	0.362414	0.373586	1.59502	1.56815	1.59502	1.58269	1.54917	0.373586	1.58514	1.59253
DownloadSpeed	64kB	GB/sec	2.13223	2.13755	0.0215719	2.11789	2.19979	2.1289	2.12669	2.13556	2.11979	2.11789	2.12669	2.13779
DownloadSpeed	128kB	GB/sec	2.57772	2.39463	0.490279	0.936015	2.58586	2.57772	2.369	2.5826	0.936015	2.57286	2.58586	2.57772
DownloadSpeed	256kB	GB/sec	2.88248	2.75776	0.378052	1.62443	2.90393	2.89572	2.84741	2.88349	1.62443	2.89368	2.87741	2.89061
DownloadSpeed	512kB	GB/sec	3.08724	2.8824	0.373859	2.02697	3.09424	3.08841	2.02697	3.08841	2.26423	2.99689	3.08608	3.08957
DownloadSpeed	1024kB	GB/sec	3.17105	3.05058	0.218715	2.61433	3.18136	3.14291	2.61433	3.17519	2.6206	3.07334	3.17766	3.17827
DownloadSpeed	2048kB	GB/sec	3.17726	3.08654	0.161202	2.84481	3.23427	3.22933	2.93739	3.23012	2.84654	2.95847	2.84481	3.23427
DownloadSpeed	4096kB	GB/sec	3.17691	3.11632	0.150924	2.88317	3.26058	3.25655	3.9265	3.27214	2.97276	3.09673	2.99203	3.26058
DownloadSpeed	8192kB	GB/sec	3.22724	3.2079	0.0765973	3.01776	3.27385	3.25467	3.17608	3.27214	3.15981	3.18368	3.01776	3.27385
DownloadSpeed	16384kB	GB/sec	3.25523	3.24591	0.0397178	3.14735	3.27897	3.27698	3.23343	3.2767	3.22434	3.27854	3.23138	3.27897
DownloadSpeed	32768kB	GB/sec	3.23577	3.23798	0.0220416	3.20851	3.26634	3.21462	3.28235	3.28235	3.21581	3.20851	3.24159	3.24768
DownloadSpeed	65536kB	GB/sec	3.26963	3.27041	0.0122743	3.25417	3.28475	3.25417	3.28308	3.28475	3.26398	3.28321	3.27528	3.28464
DownloadSpeed	131072kB	GB/sec	3.27329	3.25049	0.0552959	3.09411	3.28578	3.28155	3.27094	3.26677	3.22422	3.28262	3.27564	3.28578
DownloadSpeed	262144kB	GB/sec	3.27809	3.27352	0.013592	3.2426	3.28625	3.26977	3.27759	3.2426	3.25421	3.28623	3.27855	3.28625
DownloadSpeed	524288kB	GB/sec	3.27825	3.27474	0.00933627	3.2542	3.28322	3.26876	3.28271	3.27434	3.28152	3.263	3.27812	3.2542
DownloadTime	1kB	ms	0.010576	0.0109184	0.00104359	0.010528	0.014048	0.014048	0.010592	0.010528	0.0106	0.010624	0.01056	0.01056
DownloadTime	2kB	ms	0.010672	0.0222968	0.0347864	0.01056	0.126752	0.011328	0.010816	0.126752	0.010624	0.011424	0.010592	0.010592
DownloadTime	4kB	ms	0.012048	0.012312	0.000237081	0.011936	0.012736	0.012352	0.012736	0.011936	0.012224	0.012064	0.012	0.011968
DownloadTime	8kB	ms	0.013488	0.013532	0.000119904	0.01344	0.013856	0.013472	0.013504	0.013504	0.013856	0.01344	0.013504	0.01344
DownloadTime	16kB	ms	0.014912	0.0150784	0.000473297	0.014752	0.016448	0.015264	0.014912	0.014944	0.014752	0.016448	0.01488	0.014912
DownloadTime	32kB	ms	0.020688	0.0274272	0.0200956	0.020544	0.087712	0.020896	0.020544	0.020704	0.021152	0.087712	0.020672	0.020576
DownloadTime	64kB	ms	0.030736	0.0306624	0.000302498	0.029792	0.030944	0.030784	0.030816	0.030688	0.029792	0.030944	0.030816	0.030656
DownloadTime	128kB	ms	0.050848	0.060176	0.026653	0.050688	0.140032	0.050848	0.053328	0.050752	0.140032	0.050944	0.050688	0.050848
DownloadTime	256kB	ms	0.090944	0.0979552	0.0211452	0.090272	0.161376	0.090528	0.092064	0.090912	0.161376	0.09056	0.091104	0.090688
DownloadTime	512kB	ms	0.169824	0.185818	0.0303219	0.16944	0.258656	0.16976	0.258656	0.16976	0.231552	0.174944	0.169888	0.169696
DownloadTime	1024kB	ms	0.330672	0.345712	0.0276447	0.3296	0.401088	0.333632	0.401088	0.33024	0.400128	0.341184	0.329984	0.32992
DownloadTime	2048kB	ms	0.660224	0.681354	0.0364573	0.648416	0.737184	0.649408	0.713952	0.649248	0.736736	0.708864	0.737184	0.648416
DownloadTime	4096kB	ms	1.32109	1.34913	0.0663402	1.28637	1.45475	1.28717	1.43322	1.28723	1.41091	1.35443	1.40182	1.28637
DownloadTime	8192kB	ms	2.5995	2.61652	0.064568	2.5823	2.77974	2.57741	2.64118	2.56365	2.65478	2.63488	2.77974	2.5823
DownloadTime	16384kB	ms	5.15414	5.1695	0.0642949	5.11661	5.33659	5.11971	5.18867	5.12016	5.2033	5.11728	5.19197	5.11661
DownloadTime	32768kB	ms	10.3698	10.3632	0.070276	10.2227	10.458	10.4381	10.2278	10.2227	10.4342	10.458	10.3512	10.3318
DownloadTime	65536kB	ms	20.525	20.5203	0.077016	20.4365	20.6224	20.4408	20.6224	20.4305	20.5604	20.44	20.4895	20.4311
DownloadTime	131072kB	ms	41.004	41.3039	0.725913	40.848	43.3784	40.9607	41.0334	41.0857	41.6279	40.8874	40.9746	40.848
DownloadTime	262144kB	ms	81.8878	82.0036	0.342271	81.6844	82.784	82.0961	81.9003	82.784	82.4886	81.6849	81.8762	81.6844
DownloadTime	524288kB	ms	163.768	163.945	0.468787	163.52	164.978	164.243	163.545	163.963	163.604	164.533	163.774	164.978

7.7. SHOC Device Memory GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
TextureRepeatedCacheHit	16kB	GB/sec	249.817	249.972	1.0569	248.319	252.22	248.319	249.606	249.638	248.922	249.995	249.154	250.473	252.22	250.888	250.501
TextureRepeatedCacheHit	64kB	GB/sec	680.911	677.807	10.5231	646.709	684.922	679.146	681.132	646.709	684.922	680.689	678.437	679.739	682.069	682.718	682.504
TextureRepeatedCacheHit	256kB	GB/sec	705.515	704.972	2.36999	700.024	708.999	702.573	704.752	703.304	706.43	706.768	705.208	708.999	705.822	705.84	700.024
TextureRepeatedCacheHit	1024kB	GB/sec	868.979	861.764	16.494	827.49	873.594	871.612	830.498	827.49	869.365	866.879	868.593	871.143	873.594	870.024	868.44
TextureRepeatedCacheHit	4096kB	GB/sec	908.485	907.896	2.04231	904.699	910.857	908.82	910.857	908.242	905.301	904.699	909.598	908.728	906.416	910.283	906.015
TextureRepeatedLinearAccess	16kB	GB/sec	68.73	68.7563	0.0747196	68.6649	68.8988	68.7206	68.6981	68.8313	68.7043	68.6649	68.792	68.8362	68.6773	68.7393	68.8988
TextureRepeatedLinearAccess	64kB	GB/sec	166.242	166.155	0.276027	165.414	166.452	166.452	166.194	165.414	166.148	166.243	166.309	165.951	166.318	166.242	166.28
TextureRepeatedLinearAccess	256kB	GB/sec	103.855	103.657	0.347234	103.032	103.96	103.896	103.032	103.923	103.135	103.814	103.762	103.96	103.896	103.904	103.245
TextureRepeatedLinearAccess	1024kB	GB/sec	121.317	121.14	0.429301	120.335	121.613	121.533	120.443	120.335	120.83	121.437	121.371	121.419	121.613	121.158	121.263
TextureRepeatedLinearAccess	4096kB	GB/sec	123.961	123.98	0.0820779	123.881	124.169	124.169	123.881	123.899	123.951	124.023	123.963	124.054	123.96	123.902	124.003
TextureRepeatedRandomAccess	16kB	GB/sec	36.8203	36.7924	0.103581	36.4856	36.859	36.4856	36.859	36.8269	36.8141	36.8026	36.8184	36.8511	36.8135	36.8222	36.8311
TextureRepeatedRandomAccess	64kB	GB/sec	50.5226	50.4675	0.181774	49.925	50.5699	50.5423	50.5699	50.5347	49.925	50.5164	50.5018	50.5143	50.5096	50.5323	50.5288
TextureRepeatedRandomAccess	256kB	GB/sec	48.4988	48.4683	0.135681	48.099	48.5867	48.5366	48.5525	48.5867	48.461	48.099	48.4503	48.4411	48.4247	48.571	48.5599
TextureRepeatedRandomAccess	1024kB	GB/sec	25.0025	25.0074	0.0185024	24.9785	25.0325	24.9844	24.9785	24.9945	25.0044	25.0325	25.0312	24.9998	25.0006	25.0281	25.02
TextureRepeatedRandomAccess	4096kB	GB/sec	18.3725	18.3726	0.00210641	18.3694	18.3757	18.3704	18.3757	18.3734	18.3708	18.3715	18.3738	18.3757	18.3713	18.3737	18.3694
readGlobalMemoryCoalesced	blockSize:032	GB/s	79.4664	79.4403	0.227365	79.0545	79.8856	79.5709	79.2872	79.0545	79.6659	79.2205	79.4665	79.8856	79.5026	79.4663	79.2827
readGlobalMemoryCoalesced	blockSize:064	GB/s	95.3313	95.3931	0.370008	94.8193	96.16	95.5424	95.1413	95.3639	95.2987	94.8193	95.1258	95.1563	95.4562	95.8673	96.16
readGlobalMemoryCoalesced	blockSize:128	GB/s	95.86	95.8892	0.146875	95.6285	96.1455	95.8593	96.0451	95.8026	95.962	96.024	96.1455	95.8154	95.8608	95.7491	95.6285
readGlobalMemoryCoalesced	blockSize:256	GB/s	95.3288	95.33	0.00923051	95.3187	95.3491	95.3357	95.3202	95.3491	95.3346	95.3276	95.32	95.3187	95.33	95.339	95.3248
readGlobalMemoryCoalesced	blockSize:512	GB/s	92.4885	92.4903	0.0102933	92.4789	92.5118	92.5118	92.5041	92.491	92.4863	92.4797	92.4789	92.4955	92.4812	92.4907	92.4842
readGlobalMemoryUnit	blockSize:032	GB/s	7.40374	7.40064	0.0176191	7.37095	7.43441	7.41345	7.41215	7.38937	7.40532	7.40459	7.39723	7.37095	7.37605	7.43441	7.40288
readGlobalMemoryUnit	blockSize:064	GB/s	5.8348	5.83442	0.0122876	5.81516	5.85731	5.84059	5.84253	5.84661	5.81516	5.83919	5.85731	5.82196	5.82501	5.83041	5.82548
readGlobalMemoryUnit	blockSize:128	GB/s	4.77861	4.78074	0.0104245	4.76814	4.80264	4.77545	4.76814	4.76849	4.78169	4.80264	4.78444	4.77553	4.77329	4.78429	4.79342
readGlobalMemoryUnit	blockSize:256	GB/s	4.31423	4.3133	0.0046518	4.30446	4.31935	4.31935	4.31413	4.31433	4.31593	4.31484	4.30446	4.30978	4.31401	4.31923	4.30698
readGlobalMemoryUnit	blockSize:512	GB/s	4.053	4.04979	0.0218373	3.99878	4.07445	4.05253	4.05213	4.02782	4.07445	4.05724	4.07177	4.07001	4.05348	3.99878	4.03971
readLocalMemory	blockSize:032	GB/s	190.042	190.05	0.095262	189.889	190.232	190.107	189.948	190.004	190.232	190.158	190.042	189.889	189.993	190.042	190.084
readLocalMemory	blockSize:064	GB/s	288.244	288.243	0.0196789	288.219	288.278	288.241	288.222	288.278	288.219	288.248	288.247	288.252	288.227	288.225	288.274
readLocalMemory	blockSize:128	GB/s	299.656	299.658	0.0666282	299.645	299.667	299.656	299.66	299.665	299.655	299.645	299.653	299.653	299.667	299.666	299.656
readLocalMemory	blockSize:256	GB/s	289.272	284.612	5.90403	277.221	289.743	277.437	289.511	277.221	277.431	289.272	289.271	289.743	277.446	289.513	289.277
readLocalMemory	blockSize:512	GB/s	277.151	277.239	0.183945	276.917	277.562	276.917	277.474	277.139	277.562	277.163	277.139	277.137	277.14	277.356	277.364
writeGlobalMemoryCoalesced	blockSize:032	GB/s	92.6863	92.7431	0.17521	92.518	93.1587	92.7906	93.1587	92.518	92.5952	92.6565	92.731	92.6758	92.6969	92.9414	92.6667
writeGlobalMemoryCoalesced	blockSize:064	GB/s	91.3822	91.3367	0.199681	90.9287	91.6185	91.171	91.4988	90.9287	91.108	91.3985	91.3659	91.3167	91.4691	91.6185	91.4923
writeGlobalMemoryCoalesced	blockSize:128	GB/s	92.7195	92.7199	0.0349922	92.6621	92.7808	92.7263	92.7632	92.6912	92.7808	92.7348	92.6621	92.7443	92.6902	92.6932	92.7126
writeGlobalMemoryCoalesced	blockSize:256	GB/s	92.8269	92.8266	0.0308942	92.777	92.886	92.788	92.886	92.8017	92.777	92.8242	92.8404	92.8268	92.8269	92.8335	92.8613
writeGlobalMemoryCoalesced	blockSize:512	GB/s	91.9003	91.8954	0.0322305	91.8444	91.94	91.8444	91.94	91.8551	91.8865	91.9152	91.9083	91.8586	91.8923	91.9172	91.9367
writeGlobalMemoryUnit	blockSize:032	GB/s	3.67295	3.67445	0.0143484	3.65556	3.70246	3.67774	3.66864	3.65556	3.66201	3.67331	3.6726	3.68569	3.69036	3.70246	3.65617
writeGlobalMemoryUnit	blockSize:064	GB/s	3.47009	3.46721	0.0176114	3.43398	3.49727	3.46598	3.47792	3.49727	3.45724	3.48	3.47419	3.43398	3.45849	3.48029	3.44674
writeGlobalMemoryUnit	blockSize:128	GB/s	3.44127	3.44122	0.00360111	3.43461	3.44946	3.44333	3.43936	3.44274	3.44108	3.43461	3.43923	3.44946	3.44145	3.4393	3.44167
writeGlobalMemoryUnit	blockSize:256	GB/s	3.42033	3.42198	0.0070682	3.41015	3.43686	3.41914	3.41686	3.43086	3.41858	3.43686	3.41015	3.42464	3.42104	3.41962	3.42206
writeGlobalMemoryUnit	blockSize:512	GB/s	3.31071	3.3137	0.023414	3.29073	3.37801	3.31299	3.32027	3.29147	3.2988	3.37801	3.3117	3.30971	3.29073	3.30758	3.31576
writeLocalMemory	blockSize:032	GB/s	181.766	181.757	0.0437495	181.689	181.82	181.717	181.77	181.82	181.743	181.689	181.815	181.768	181.695	181.765	181.791
writeLocalMemory	blockSize:064	GB/s	328.684	328.689	0.0293	328.649	328.752	328.703	328.673	328.702	328.649	328.653	328.716	328.691	328.676	328.752	328.678
writeLocalMemory	blockSize:128	GB/s	388.966	388.961	0.0411515	388.862	389.014	388.937	388.973	388.862	388.934	388.994	389.001	389.014	388.96	388.959	388.973
writeLocalMemory	blockSize:256	GB/s	385.716	385.713	0.0394536	385.626	385.781	385.781	385.626	385.718	385.714	385.703	385.738	385.675	385.735	385.701	385.738
writeLocalMemory	blockSize:512	GB/s	368.518	368.462	0.156346	368.007	368.584	368.517	368.452	368.527	368.007	368.453	368.541	368.584	368.519	368.535	368.485

7.8. SHOC SPMV GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4
CSR-Scalar-DP	10485_elements_1024_rows	Gflop/s	0.603019	0.602467	0.00163314	0.597629	0.603274	0.597629	0.602736	0.603069	0.603235	0.602414
CSR-Scalar-DP_PCIe	10485_elements_1024_rows	Gflop/s	0.145213	0.14508	0.000386159	0.144008	0.145399	0.144008	0.145166	0.144993	0.145259	0.145276
CSR-Scalar-SP 1	10485_elements_1024_rows	Gflop/s	0.738709	0.7375	0.00358396	0.726768	0.738988	0.726768	0.738322	0.738405	0.738455	0.738838
CSR-Scalar-SP_PCIe	10485_elements_1024_rows	Gflop/s	0.163498	0.162918	0.00166143	0.157957	0.163673	0.157957	0.163071	0.163319	0.163607	0.163503
CSR-Vector-DP	10485_elements_1024_rows	Gflop/s	1.11445	1.11463	0.000883064	1.11336	1.1162	1.11438	1.1162	1.11448	1.1152	1.11598
CSR-Vector-DP_PCIe	10485_elements_1024_rows	Gflop/s	0.163419	0.163423	1.89766e-05	0.163396	0.163457	0.163418	0.163457	0.16342	0.163435	0.163452
CSR-Vector-SP	10485_elements_1024_rows	Gflop/s	1.3298	1.33005	0.000984298	1.32859	1.33191	1.32964	1.32859	1.33153	1.33012	1.32969
CSR-Vector-SP_PCIe	10485_elements_1024_rows	Gflop/s	0.181337	0.181341	1.82905e-05	0.181314	0.181376	0.181334	0.181314	0.181369	0.181343	0.181335
ELLPACKR-DP	10485_elements_1024_rows	Gflop/s	0.544524	0.544062	0.00143224	0.539769	0.544655	0.539769	0.544442	0.544623	0.544573	0.544528
ELLPACKR-SP	10485_elements_1024_rows	Gflop/s	0.551276	0.550913	0.00122221	0.547257	0.551531	0.547257	0.551215	0.551239	0.551262	0.551355
Padded_CSR-Scalar-DP	16832_elements_1024_rows	Gflop/s	0.551077	0.551203	0.000423255	0.550719	0.552115	0.550719	0.550967	0.551663	0.551507	0.552115
Padded_CSR-Scalar-DP_PCIe	16832_elements_1024_rows	Gflop/s	0.187201	0.187194	7.61091e-05	0.187043	0.187299	0.187248	0.187043	0.187257	0.187139	0.187275
Padded_CSR-Scalar-SP	16832_elements_1024_rows	Gflop/s	0.73395	0.729391	0.0150044	0.684464	0.736592	0.733709	0.734314	0.734027	0.73416	0.733459
Padded_CSR-Scalar-SP_PCIe	16832_elements_1024_rows	Gflop/s	0.238387	0.23196	0.0110696	0.208988	0.238758	0.238397	0.238461	0.238376	0.237636	0.238587
Padded_CSR-Vector-DP	16832_elements_1024_rows	Gflop/s	1.83573	1.83565	0.000895882	1.83438	1.83746	1.8363	1.83589	1.83438	1.83467	1.8363
Padded_CSR-Vector-DP_PCIe	16832_elements_1024_rows	Gflop/s	0.245504	0.245502	1.60219e-05	0.24548	0.245535	0.245514	0.245506	0.24548	0.245485	0.245514
Padded_CSR-Vector-SP	16832_elements_1024_rows	Gflop/s	2.16307	2.14816	0.0453347	2.0122	2.16554	2.0122	2.16305	2.1623	2.16336	2.1631
Padded_CSR-Vector-SP_PCIe	16832_elements_1024_rows	Gflop/s	0.298411	0.298108	0.000917896	0.295355	0.298457	0.295355	0.29841	0.298396	0.298416	0.298411

7.9. SHOC MD GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
MD-LJ	12288_atoms	GFLOPS	52.4619	52.4502	0.0402902	52.3673	52.5033	52.4019	52.3673	52.4339	52.4579	52.4712	52.4285	52.4659	52.4766	52.5033	52.4953
MD-LJ-Bandwidth	12288_atoms	GB/s	40.2452	40.2363	0.030908	40.1727	40.277	40.1992	40.1727	40.2238	40.2422	40.2524	40.2197	40.2483	40.2565	40.277	40.2709
MD-LJ-Bandwidth_PCIe	12288_atoms	GB/s	9.38313	9.38264	0.00168132	9.37918	9.38486	9.38063	9.37918	9.38197	9.38297	9.38352	9.38174	9.3833	9.38375	9.38486	9.38453
MD-LJ-DP	12288_atoms	GFLOPS	34.14	34.1355	0.0269753	34.0824	34.1751	34.14	34.0824	34.14	34.1649	34.1751	34.1536	34.096	34.1422	34.1321	34.1287
MD-LJ-DP-Bandwidth	12288_atoms	GB/s	45.8705	45.8645	0.036244	45.7932	45.9176	45.8705	45.7932	45.8705	45.9039	45.9176	45.8887	45.8114	45.8735	45.8599	45.8553
MD-LJ-DP-Bandwidth_PCIe	12288_atoms	GB/s	14.0789	14.0783	0.00341607	14.0716	14.0833	14.0789	14.0716	14.0789	14.082	14.0833	14.0806	14.0733	14.0791	14.0779	14.0774
MD-LJ-DP_PCIe	12288_atoms	GFLOPS	10.4785	10.478	0.00254247	10.473	10.4818	10.4785	10.473	10.4785	10.4808	10.4818	10.4797	10.4743	10.4787	10.4777	10.4774
MD-LJ-DP_Parity	12288_atoms	N	2.25811	2.25781	0.00178422	2.25431	2.26043	2.25811	2.25431	2.25811	2.25976	2.26043	2.25901	2.2552	2.25826	2.25759	2.25736
MD-LJ_PCIe	12288_atoms	GFLOPS	12.2314	12.2308	0.0021917	12.2263	12.2337	12.2282	12.2263	12.2299	12.2312	12.2319	12.2296	12.2316	12.2322	12.2337	12.2332
MD-LJ_Parity	12288_atoms	N	3.2891	3.28837	0.002526	3.28317	3.2917	3.28534	3.28317	3.28735	3.28885	3.28969	3.28701	3.28936	3.29003	3.2917	3.2912

7.10. SHOC Reduction GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
Reduction	262144_items	GB/s	62.8407	62.8136	0.166818	62.3359	62.996	62.3359	62.8365	62.8878	62.8431	62.996	62.8092	62.894	62.8384	62.8638	62.8318
Reduction-DP	131072_items	GB/s	61.6712	61.7126	0.102425	61.5786	61.8985	61.5786	61.8985	61.6492	61.6941	61.6819	61.7691	61.6442	61.6605	61.6551	61.8948
Reduction-DP_PCIe	131072_items	GB/s	2.87049	2.86534	0.0185665	2.81098	2.87957	2.81098	2.87026	2.87526	2.86781	2.87332	2.87957	2.87072	2.86673	2.87351	2.86523
Reduction-DP_Parity	131072_items	N	20.4912	20.5385	0.132249	20.4412	20.9064	20.9064	20.5655	20.4412	20.5126	20.4671	20.4508	20.4735	20.509	20.4564	20.602
Reduction_PCIe	262144_items	GB/s	2.88511	2.86744	0.0479907	2.72484	2.89162	2.72484	2.86923	2.89162	2.88389	2.87283	2.88916	2.88425	2.88642	2.88597	2.88615
Reduction_Parity	262144_items	N	20.7868	20.9113	0.327234	20.7396	21.8769	21.8769	20.9001	20.7483	20.7911	20.9282	20.7396	20.806	20.7703	20.7826	20.7701

7.11. SHOC S3D GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6
S3D-DP	4096_gridPoints	GFLOPS	14.9961	14.9782	0.0392359	14.8882	15.0237	14.8882	14.9493	15.0237	14.939	15.0023	15.0015	14.998
S3D-DP_PCIe	4096_gridPoints	GFLOPS	12.4294	12.4159	0.0302911	12.3435	12.4476	12.3435	12.396	12.4476	12.3907	12.4361	12.4329	12.4389
S3D-DP_Parity	4096_gridPoints	N	0.206389	0.20637	0.00036282	0.206556	0.20651	0.206155	0.205973	0.20651	0.205656	0.206353	0.206598	0.206598
S3D-SP	13824_gridPoints	GFLOPS	29.9089	29.8943	0.0727289	29.6901	29.964	29.6901	29.9068	29.8805	29.9111	29.9507	29.964	29.9339
S3D-SP_PCIe	13824_gridPoints	GFLOPS	24.9594	24.948	0.0530374	24.7982	24.9957	24.7982	24.9606	24.9384	24.9583	24.9903	24.9957	24.9763
S3D-SP_Parity	13824_gridPoints	N	0.198331	0.198263	0.00038098	0.197267	0.198768	0.197267	0.19816	0.198173	0.198445	0.198493	0.198768	0.198492

7.12. SHOC Scan GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6
Scan	262144items	GB/s	17.1424	17.1388	0.0120004	17.1068	17.1555	17.1068	17.1423	17.1555	17.1345	17.1448	17.1368	17.1448
Scan-DP	131072items	GB/s	13.8939	13.8921	0.00993379	13.8738	13.905	13.8839	13.905	13.8958	13.8945	13.8812	13.8933	13.8738
Scan-DP_PCIe	131072items	GB/s	0.00257527	0.00257527	3.41444e-10	0.00257527	0.00257527	0.00257527	0.00257527	0.00257527	0.00257527	0.00257527	0.00257527	0.00257527
Scan_DP_Parity	131072items	N	5394.13	5393.43	3.85667	5386.32	5398.43	5390.26	5398.43	5384.88	5394.35	5389.21	5393.9	5386.32
Scan_PCIe	262144items	GB/s	0.0025617	0.0025617	2.69126e-10	0.0025617	0.0025617	0.0025617	0.0025617	0.0025617	0.0025617	0.0025617	0.0025617	0.0025617
Scan_Parity	262144items	N	6681.74	6680.34	4.67752	6667.88	6686.85	6667.88	6681.71	6686.85	6678.65	6682.68	6679.55	6682.69

7.13. SHOC SGEMM GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
DGEMM-N	128	GFlops	50.4949	50.5374	0.293406	50.0896	51.0753	50.1567	50.0896	50.4511	50.6314	51.0753	50.7097	50.4026	50.4026	50.5387	50.9166
DGEMM-N_PCIe	128	GFlops	17.5706	17.5756	0.0354535	17.5212	17.6403	17.5294	17.5212	17.5653	17.5871	17.6403	17.5965	17.5594	17.5594	17.5759	17.6213
DGEMM-N_Parity	128	N	1.87383	1.87541	0.0108881	1.85879	1.89537	1.86128	1.85879	1.87221	1.8789	1.89537	1.88181	1.87041	1.87041	1.87546	1.88948
DGEMM-T	128	GFlops	50.6195	50.6161	0.259669	50.1999	50.976	50.9166	50.1999	50.3445	50.4948	50.7441	50.456	50.976	50.8622	50.7933	50.3736
DGEMM-T_PCIe	128	GFlops	17.5856	17.5851	0.0313531	17.5347	17.6285	17.6213	17.5347	17.5523	17.5706	17.6006	17.5659	17.6285	17.6148	17.6066	17.5559
DGEMM-T_Parity	128	N	1.87846	1.87833	0.00963617	1.86289	1.89169	1.88948	1.86289	1.86825	1.87383	1.88308	1.87239	1.89169	1.88747	1.88491	1.86933
SGEMM-N	256	GFlops	202.765	202.635	0.314203	202.038	203.026	202.233	202.741	202.976	202.486	202.389	203.026	202.79	202.8	202.038	202.868
SGEMM-N_PCIe	256	GFlops	69.8381	69.8226	0.0373269	69.7516	69.869	69.7748	69.8352	69.8631	69.805	69.7934	69.869	69.841	69.8422	69.7516	69.8504
SGEMM-N_Parity	256	N	1.90336	1.90213	0.00294943	1.89653	1.9058	1.89836	1.90313	1.90534	1.90074	1.89983	1.9058	1.90359	1.90368	1.89653	1.90433
SGEMM-T	256	GFlops	196.271	194.471	3.39911	188.001	197.101	197.101	195.996	196.611	195.539	196.547	192.629	188.288	196.934	188.001	197.064
SGEMM-T_PCIe	256	GFlops	69.0512	68.8222	0.431297	67.9988	69.1536	69.1536	69.017	69.0932	68.9603	69.0853	68.5949	68.0363	69.1331	67.9988	69.149
SGEMM-T_Parity	256	N	1.8424	1.8255	0.0319075	1.76477	1.85019	1.85019	1.83981	1.84559	1.83552	1.84499	1.80821	1.76746	1.84862	1.76477	1.84984

7.14. SHOC Sort GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6
Sort-Rate	2621441items	GB/s	0.987055	0.985707	0.0040278	0.974875	0.99003	0.974875	0.986676	0.987776	0.987628	0.99003	0.982667	0.98586
Sort-Rate_PCIe	2621441items	GB/s	0.580774	0.580171	0.00259622	0.572762	0.582444	0.572762	0.58037	0.581385	0.581477	0.582444	0.579365	0.580761
Sort-Rate_Parity	2621441items	N	0.698851	0.698998	0.00169846	0.69611	0.70206	0.70206	0.70008	0.699007	0.698479	0.699785	0.69611	0.69753

7.15. SHOC Stencil 2D GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6
DP_Sten2D	1000,768x768:16x16	s	0.35183	0.351729	0.000422792	0.351116	0.352377	0.352377	0.352262	0.351412	0.351324	0.351196	0.35181	0.351116
SP_Sten2D	1000,768x768:16x16	s	0.190368	0.195241	0.0148919	0.18994	0.239909	0.239909	0.19036	0.190376	0.189955	0.190759	0.190042	0.18994

7.16. SHOC Triad GTX 460

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5
TriadBwth	Block:00064KB	GB/s	2.61981	2.56891	0.125201	2.24194	2.66794	2.63963	2.6331	2.61855	2.64104	2.57481	2.43406
TriadBwth	Block:00128KB	GB/s	2.77941	2.73973	0.138379	2.4166	2.85603	2.78851	2.84799	2.4166	2.85603	2.76886	2.76479
TriadBwth	Block:00256KB	GB/s	2.86521	2.86948	0.0211908	2.84296	2.92228	2.8578	2.87374	2.87549	2.85272	2.88512	2.92228
TriadBwth	Block:00512KB	GB/s	2.93342	2.91206	0.0598474	2.79159	2.96307	2.82526	2.98307	2.87132	2.79159	2.92946	2.9422
TriadBwth	Block:01024KB	GB/s	2.99822	2.99813	0.0060999	2.98672	3.00802	2.99388	2.99418	3.00147	2.99293	2.99882	3.00565
TriadBwth	Block:02048KB	GB/s	3.00692	2.9714	0.0489781	2.89608	3.0161	2.89608	3.01593	3.00924	2.92156	3.0161	3.00972
TriadBwth	Block:04096KB	GB/s	3.00581	2.96346	0.0647738	2.87331	3.0224	2.99626	3.01669	3.01536	3.0224	2.89317	2.87797
TriadBwth	Block:08192KB	GB/s	2.90042	2.93374	0.0601725	2.87543	3.02881	2.89952	3.01966	3.02596	3.02881	2.90579	2.89683
TriadBwth	Block:16384KB	GB/s	2.76109	2.66166	0.217612	2.34058	2.91214	2.91214	2.88207	2.79908	2.83783	2.4044	2.34058
TriadFlops	Block:00064KB	GFLOP/s	0.436634	0.428152	0.0208668	0.373657	0.444657	0.444657	0.439938	0.43885	0.436425	0.440173	0.429136	0.405676
TriadFlops	Block:00128KB	GFLOP/s	0.463236	0.456622	0.0230631	0.402766	0.476005	0.422323	0.464919	0.474664	0.402766	0.476005	0.461476	0.460798
TriadFlops	Block:00256KB	GFLOP/s	0.477536	0.478247	0.00353013	0.473826	0.487047	0.476301	0.478957	0.479248	0.475718	0.475454	0.480853	0.487047
TriadFlops	Block:00512KB	GFLOP/s	0.488903	0.485343	0.00997456	0.465264	0.497178	0.470876	0.497178	0.49522	0.47779	0.465264	0.488243	0.490367
TriadFlops	Block:01024KB	GFLOP/s	0.499703	0.499688	0.00101667	0.497786	0.501337	0.49898	0.499031	0.500245	0.497786	0.498822	0.499803	0.500941
TriadFlops	Block:02048KB	GFLOP/s	0.501154	0.495249	0.00816301	0.48268	0.502683	0.48268	0.502656	0.501539	0.484147	0.486927	0.502683	0.50162
TriadFlops	Block:04096KB	GFLOP/s	0.500969	0.49391	0.0107956	0.478885	0.503733	0.499377	0.502782	0.50256	0.503718	0.503733	0.482195	0.479661
TriadFlops	Block:08192KB	GFLOP/s	0.483403	0.488957	0.0100288	0.479239	0.504802	0.483254	0.503276	0.504326	0.504802	0.483553	0.484299	0.482805
TriadFlops	Block:16384KB	GFLOP/s	0.460182	0.44361	0.0362687	0.390097	0.485356	0.485356	0.480346	0.466513	0.47291	0.473605	0.400734	0.390097

7.17. SHOC Max Flops 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
Add1-SP	Size:2097152	GFLOPS	231.047	231.04	0.0528887	230.949	231.117	230.949	231.03	231.099	230.964	231.063	231.117	231.052	231.043	230.997	231.085
Add2-SP	Size:2097152	GFLOPS	231.156	231.149	0.0308438	231.097	231.196	231.196	231.133	231.154	231.097	231.159	231.184	231.157	231.167	231.142	231.098
Add4-SP	Size:2097152	GFLOPS	231.109	231.106	0.0340844	231.058	231.159	231.148	231.062	231.058	231.086	231.073	231.159	231.126	231.126	231.126	231.093
Add8-SP	Size:2097152	GFLOPS	230.715	230.711	0.0174072	230.679	230.735	230.707	230.723	230.716	230.679	230.684	230.706	230.735	230.731	230.72	230.713
MAdd1-SP	Size:2097152	GFLOPS	309.282	309.283	0.0316217	309.238	309.344	309.344	309.305	309.298	309.238	309.262	309.284	309.28	309.259	309.317	309.247
MAdd2-SP	Size:2097152	GFLOPS	309.277	309.274	0.0140903	309.249	309.291	309.28	309.27	309.265	309.287	309.274	309.288	309.249	309.286	309.291	309.253
MAdd4-SP	Size:2097152	GFLOPS	309.065	309.057	0.0254775	308.985	309.079	308.985	309.079	309.066	309.055	309.066	309.061	309.064	309.069	309.048	309.077
MAdd8-SP	Size:2097152	GFLOPS	308.846	308.851	0.0110059	308.837	308.867	308.867	308.847	308.867	308.861	308.837	308.856	308.846	308.84	308.845	308.838
MAddU-SP	Size:4194304	GFLOPS	388.453	388.507	1.22614	387.006	389.915	389.653	387.147	387.498	387.448	389.72	387.006	389.9	389.408	387.379	389.915
Mul1-SP	Size:2097152	GFLOPS	298.097	298.055	0.141668	297.725	298.24	297.97	298.042	298.24	298.112	298.098	298.095	298.213	297.935	297.725	298.117
Mul2-SP	Size:2097152	GFLOPS	276.89	276.894	0.0219902	276.849	276.928	276.928	276.887	276.877	276.91	276.878	276.889	276.891	276.918	276.91	276.849
Mul4-SP	Size:2097152	GFLOPS	278.497	278.491	0.0406902	278.409	278.542	278.481	278.531	278.535	278.496	278.499	278.527	278.409	278.518	278.471	278.542
Mul8-SP	Size:2097152	GFLOPS	280.209	280.207	0.0264508	280.147	280.244	280.194	280.203	280.244	280.147	280.235	280.186	280.214	280.22	280.226	280.198
MulMAdd1-SP	Size:2097152	GFLOPS	389.945	389.954	0.0435699	389.875	390.031	389.929	389.934	389.93	389.973	390.031	390.023	389.875	389.95	389.951	389.939
MulMAdd2-SP	Size:2097152	GFLOPS	378.69	378.687	0.0293214	378.634	378.728	378.655	378.715	378.661	378.68	378.721	378.68	378.634	378.701	378.728	378.699
MulMAdd4-SP	Size:2097152	GFLOPS	377.373	377.37	0.0300483	377.325	377.413	377.325	377.336	377.334	377.355	377.413	377.41	377.388	377.392	377.382	377.363
MulMAdd8-SP	Size:2097152	GFLOPS	378.334	378.333	0.032578	378.277	378.387	378.277	378.312	378.387	378.346	378.376	378.328	378.34	378.355	378.301	378.309
MulMAddU-SP	Size:4194304	GFLOPS	495.103	495.13	0.110516	494.991	495.407	495.067	495.042	494.991	495.063	495.152	495.079	495.127	495.158	495.209	495.407

7.18. SHOC Bus Download Speed 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8
DownloadSpeed	1kB	GB/sec	0.0841	0.082579	0.0841	0.082579	0.0513956	0.0673684	0.0862534	0.0673684	0.0848806	0.0844327	0.0835509	0.0839895	0.0837696	0.0829016
DownloadSpeed	2kB	GB/sec	0.160225	0.160489	0.160225	0.160489	0.0770024	0.150588	0.170667	0.154217	0.170667	0.153846	0.150588	0.152745	0.166667	0.166234
DownloadSpeed	4kB	GB/sec	0.300121	0.299614	0.300121	0.299614	0.0418758	0.292237	0.30622	0.292237	0.304762	0.301176	0.299065	0.301176	0.296296	0.294931
DownloadSpeed	8kB	GB/sec	0.543527	0.542001	0.543527	0.542001	0.00686272	0.528926	0.551724	0.528926	0.54818	0.544681	0.54818	0.551724	0.533333	0.536688
DownloadSpeed	16kB	GB/sec	0.950549	0.952378	0.950549	0.952378	0.0317932	0.91592	0.998051	0.975238	0.924188	0.925859	0.91921	0.920863	0.978967	0.978967
DownloadSpeed	32kB	GB/sec	1.48729	1.4767	1.48729	1.4767	0.0223625	1.4382	1.50147	1.4382	1.50147	1.49489	1.46915	1.49927	1.44225	1.45661
DownloadSpeed	64kB	GB/sec	2.03984	2.03275	2.03984	2.03275	0.0201278	2	2.05829	2	2.05005	2.00784	2.03984	2.03175	2.00391	2.05829
DownloadSpeed	128kB	GB/sec	2.53309	2.53611	2.53309	2.53611	0.0101023	2.52372	2.5584	2.54568	2.53309	2.5284	2.53152	2.53309	2.54568	2.52372
DownloadSpeed	256kB	GB/sec	2.85517	2.84071	2.85517	2.84071	0.0336721	2.75361	2.87136	2.86233	2.83362	2.75361	2.87136	2.8258	2.86034	2.85336
DownloadSpeed	512kB	GB/sec	3.04847	2.76925	3.04847	2.76925	0.701862	0.680088	3.05672	3.04932	2.79162	3.03801	3.05273	3.05159	0.680088	3.04762
DownloadSpeed	1024kB	GB/sec	3.14696	2.79079	3.14696	2.79079	0.689764	1.38361	3.16967	3.1666	3.04875	3.06415	3.16752	3.16721	1.44467	3.16967
DownloadSpeed	2048kB	GB/sec	3.18725	2.94442	3.18725	2.94442	0.509087	1.91559	3.22885	3.2252	3.15608	3.15912	3.21539	3.22266	1.91559	3.22885
DownloadSpeed	4096kB	GB/sec	3.23401	3.0139	3.23401	3.0139	0.350765	2.46464	3.25701	3.25241	3.21499	3.21562	3.25402	3.25281	2.48341	3.25701
DownloadSpeed	8192kB	GB/sec	3.234	3.11465	3.234	3.11465	0.202179	2.78975	3.27275	3.24351	3.16435	3.22448	3.27275	3.26887	2.83252	3.27189
DownloadSpeed	16384kB	GB/sec	3.15697	3.12591	3.15697	3.12591	0.160862	2.87232	3.28117	3.04682	2.87232	3.26712	3.28086	3.27998	2.89636	3.28117
DownloadSpeed	32768kB	GB/sec	3.24847	3.19635	3.24847	3.19635	0.10431	2.98314	3.28498	3.13251	3.22434	3.2726	3.28498	3.28441	3.15989	3.28345
DownloadSpeed	65536kB	GB/sec	3.27914	3.259	3.27914	3.259	0.033918	3.20102	3.28686	3.28467	3.27361	3.20268	3.28553	3.28686	3.22488	3.28579
DownloadSpeed	131072kB	GB/sec	3.28507	3.27734	3.28507	3.27734	0.0134925	3.25166	3.2879	3.28777	3.28518	3.27168	3.28495	3.28786	3.25166	3.2879
DownloadSpeed	262144kB	GB/sec	3.28511	3.26469	3.28511	3.26469	0.0347326	3.18579	3.28822	3.28786	3.28359	3.18579	3.26253	3.28671	3.26703	3.28822
DownloadTime	1kB	ms	0.012176	0.0128	0.012176	0.0124576	0.000922028	0.011872	0.0152	0.0152	0.012064	0.012128	0.012256	0.012192	0.012224	0.012352
DownloadTime	2kB	ms	0.0128	0.0127904	0.0128	0.0127904	0.000612372	0.012	0.0136	0.01328	0.012	0.013312	0.0136	0.013408	0.012288	0.01232
DownloadTime	4kB	ms	0.013648	0.0136736	0.013648	0.0136736	0.000191492	0.013376	0.014016	0.014016	0.01344	0.0136	0.013696	0.0136	0.013824	0.013888
DownloadTime	8kB	ms	0.015072	0.0151168	0.015072	0.0151168	0.000192639	0.014848	0.015488	0.015488	0.014944	0.01504	0.014944	0.014848	0.01536	0.015264
DownloadTime	16kB	ms	0.017248	0.0172224	0.017248	0.0172224	0.00057347	0.016416	0.017888	0.0168	0.017728	0.017696	0.017824	0.017792	0.016736	0.016736
DownloadTime	32kB	ms	0.022032	0.0221952	0.022032	0.0221952	0.000339321	0.021824	0.022784	0.022784	0.021824	0.02192	0.022304	0.021856	0.02272	0.022496
DownloadTime	64kB	ms	0.032128	0.0322432	0.032128	0.0322432	0.000321022	0.03184	0.032768	0.032768	0.031968	0.03264	0.032128	0.032256	0.032704	0.03184
DownloadTime	128kB	ms	0.051744	0.0516832	0.051744	0.0516832	0.000205125	0.051232	0.051936	0.051488	0.051744	0.05184	0.051776	0.051744	0.051488	0.051936
DownloadTime	256kB	ms	0.091808	0.0922944	0.091808	0.0922944	0.00111494	0.091296	0.0952	0.091584	0.092512	0.0952	0.091296	0.092768	0.091648	0.091872
DownloadTime	512kB	ms	0.171984	0.234461	0.171984	0.234461	0.178897	0.17152	0.770912	0.171936	0.187808	0.172576	0.171744	0.171808	0.770912	0.172032
DownloadTime	1024kB	ms	0.333216	0.416013	0.333216	0.416013	0.163135	0.330816	0.757856	0.331136	0.343936	0.342208	0.33104	0.331072	0.725824	0.330816
DownloadTime	2048kB	ms	0.658032	0.742141	0.658032	0.742141	0.172952	0.649504	1.09478	0.65024	0.66448	0.66384	0.652224	0.650752	1.09478	0.649504
DownloadTime	4096kB	ms	1.29696	1.41294	1.29696	1.41294	0.182954	1.28778	1.70179	1.2896	1.30461	1.30435	1.28896	1.28944	1.68893	1.28778
DownloadTime	8192kB	ms	2.5939	2.70531	2.5939	2.70531	0.185456	2.56317	3.00694	2.58627	2.65098	2.60154	2.56317	2.56621	2.96154	2.56384
DownloadTime	16384kB	ms	5.32082	5.38166	5.32082	5.38166	0.282008	5.11318	5.84099	5.50646	5.84099	5.13517	5.11366	5.11504	5.79251	5.11318
DownloadTime	32768kB	ms	10.3299	10.5092	10.3299	10.5092	0.353181	10.2145	11.248	10.7117	10.4066	10.2532	10.2145	10.2163	10.6188	10.2193
DownloadTime	65536kB	ms	20.4655	20.5941	20.4655	20.5941	0.216152	20.4173	20.9649	20.4309	20.5	20.954	20.4256	20.4173	20.8097	20.4239
DownloadTime	131072kB	ms	40.8569	40.954	40.8569	40.954	0.169335	40.8217	41.2766	40.8234	40.8555	41.0241	40.8584	40.8222	41.2766	40.8217
DownloadTime	262144kB	ms	81.7128	82.2333	81.7128	82.2333	0.88031	81.6356	84.2601	81.6444	81.7506	84.2601	82.2783 8	1.673	82.1649	81.6356

7.19. SHOC SPMV 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4
CSR-Scalar-SP	10485_elements_1024_rows	Gflop/s	0.492353	0.491852	0.00156929	0.487232	0.49285	0.487232	0.49285	0.491884	0.492394	0.492294
CSR-Scalar-SP_PCIe	10485_elements_1024_rows	Gflop/s	0.137159	0.136804	0.000837205	0.134479	0.137312	0.134479	0.137019	0.137231	0.136955	0.137234
CSR-Vector-SP	10485_elements_1024_rows	Gflop/s	0.261349	0.255234	0.0128315	0.223399	0.262433	0.23735	0.223399	0.260362	0.262245	0.261401
CSR-Vector-SP_PCIe	10485_elements_1024_rows	Gflop/s	0.109412	0.108255	0.00243434	0.102147	0.109601	0.104968	0.102147	0.109238	0.109568	0.109421
ELLPACKR-SP	10485_elements_1024_rows	Gflop/s	0.558802	0.558625	0.000974549	0.555872	0.559661	0.555872	0.558511	0.558687	0.559059	0.558854
Padded_CSR-Scalar-SP	16832_elements_1024_rows	Gflop/s	0.639735	0.639539	0.000408259	0.638691	0.640031	0.639339	0.640031	0.639747	0.639852	0.638691
Padded_CSR-Scalar-SP_PCIe	16832_elements_1024_rows	Gflop/s	0.210868	0.211135	0.00057479	0.210648	0.212292	0.212292	0.210751	0.210889	0.210774	0.210648
Padded_CSR-Vector-SP	16832_elements_1024_rows	Gflop/s	0.471938	0.471626	0.000862654	0.470233	0.472941	0.471965	0.472041	0.471018	0.471912	0.472682
Padded_CSR-Vector-SP_PCIe	16832_elements_1024_rows	Gflop/s	0.18873	0.18868	0.000141317	0.188456	0.18889	0.188734	0.188746	0.188583	0.188726	0.188849

7.20. SHOC MD 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
MD-LJ	12288_atoms	GFLOPS	38.3134	38.2883	0.220931	37.7012	38.5828	37.7012	38.5828	38.2593	38.2635	38.3632	38.3803	38.2564	38.3818	38.2366	38.4576
MD-LJ-Bandwidth	12288_atoms	GB/s	29.3915	29.3722	0.169483	28.9218	29.5982	28.9218	29.5982	29.35	29.3532	29.4297	29.4428	29.3478	29.4439	29.3326	29.5021
MD-LJ-Bandwidth_PCIe	12288_atoms	GB/s	8.62724	8.62553	0.0147116	8.58632	8.64497	8.58632	8.64497	8.62367	8.62395	8.63054	8.63167	8.62348	8.63176	8.62216	8.63676
MD-LJ_PCIe	12288_atoms	GFLOPS	11.2461	11.2438	0.0191774	11.1927	11.2692	11.1927	11.2692	11.2414	11.2418	11.2504	11.2518	11.2412	11.252	11.2395	11.2585
MD-LJ_Parity	12288_atoms	N	2.40682	2.40524	0.0138787	2.36836	2.42374	2.36836	2.42374	2.40342	2.40369	2.40995	2.41102	2.40324	2.41111	2.40199	2.41588

7.21. SHOC Reduction 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
Reduction	262144_items	GB/s	45.021	45.0116	0.042387	44.9105	45.0758	44.9105	44.968	45.0758	45.0202	45.0219	45.032	45.0407	45.0267	45.0052	45.0153
Reduction_PCIe	262144_items	GB/s	2.79718	2.7812	0.0489921	2.63522	2.80602	2.63522	2.7905	2.79448	2.80072	2.79929	2.80268	2.79507	2.80602	2.80186	2.78617
Reduction_Parity	262144_items	N	15.0988	15.1893	0.28622	15.0465	16.0424	16.0424	15.1147	15.1303	15.0745	15.0833	15.0675	15.1144	15.0465	15.0626	15.1567

7.22. SHOC S3D 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6
S3D-SP	13824 gridPoints	GFLOPS	23.0383	22.9121	0.32531	21.9898	23.1593	21.9898	23.0096	23.1074	23.0713	23.0669	22.8426	23.1593
S3D-SP_PCIe	13824 gridPoints	GFLOPS	19.4323	19.3408	0.234265	18.6752	19.5184	18.6752	19.4123	19.476	19.4553	19.4524	19.2929	19.5184
S3D-SP_Parity	13824 gridPoints	N	0.185562	0.184621	0.00254038	0.177486	0.18654	0.177486	0.185307	0.186453	0.185857	0.185817	0.183989	0.18654

7.23. SHOC SGEMM 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6	trial7	trial8	trial9
SGEMM-N	256	GFlops	227.254	226.987	0.752719	225.331	227.877	227.408	227.1	225.331	227.531	226.084	227.877	227.605	226.768	226.646	227.519
SGEMM-N_PCIe	256	GFlops	72.3799	72.3526	0.0766422	72.1837	72.443	72.3955	72.3642	72.1837	72.408	72.2608	72.443	72.4155	72.3306	72.3181	72.4067
SGEMM-N_Parity	256	N	2.13973	2.13722	0.00708732	2.12163	2.1456	2.14118	2.13829	2.12163	2.14235	2.12872	2.1456	2.14304	2.13516	2.13401	2.14223
SGEMM-T	256	GFlops	236.712	236.773	0.352543	236.299	237.436	237.436	236.539	236.299	236.365	237.261	236.819	236.619	236.579	236.806	237.007
SGEMM-T_PCIe	256	GFlops	73.3129	73.3187	0.0337872	73.2732	73.3822	73.3822	73.2962	73.2732	73.2796	73.3655	73.3231	73.3039	73.3001	73.3219	73.3411
SGEMM-T_Parity	256	N	2.22879	2.22936	0.00331941	2.2249	2.23561	2.23561	2.22716	2.2249	2.22553	2.23396	2.2298	2.22791	2.22754	2.22967	2.23156

7.24. SHOC Sort 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6
Sort-Rate	2621441tems	GB/s	0.666183	0.666404	0.00499752	0.659243	0.672937	0.660272	0.672771	0.67173	0.672937	0.659243	0.669185	0.668919
Sort-Rate_PCIe	2621441tems	GB/s	0.455566	0.455641	0.00258533	0.451393	0.459132	0.451393	0.458788	0.458525	0.459132	0.452575	0.456786	0.456691
Sort-Rate_Parity	2621441tems	N	0.463726	0.462551	0.00320461	0.45665	0.466411	0.462743	0.466411	0.464982	0.465673	0.45665	0.464987	0.46471

7.25. SHOC Stencil 2D 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5	trial6
SP_Sten2D	1000:768x768:16x16	s	0.973421	0.978881	0.0146955	0.972847	1.02273	1.02273	0.974375	0.973131	0.97814	0.972935	0.972955	0.972847

7.26. SHOC Triad 9800 GTX+

test	atts	units	median	mean	stddev	min	max	trial0	trial1	trial2	trial3	trial4	trial5
TriadBdwth	Block:00064KB	GB/s	0.77632	0.00488421	0.76938	0.782538	0.775907	0.777225	0.776422	0.782538	0.762938	0.77622	0.779149
TriadBdwth	Block:00128KB	GB/s	1.2345	0.00610622	1.22538	1.24678	1.22538	1.23093	1.23775	1.22726	1.23663	1.23306	1.24678
TriadBdwth	Block:00256KB	GB/s	1.76745	0.00893392	1.74641	1.77424	1.77246	1.76493	1.77115	1.74641	1.77424	1.77099	1.75064
TriadBdwth	Block:00512KB	GB/s	2.2517	0.00299578	2.24781	2.25618	2.25094	2.25557	2.24804	2.25082	2.25356	2.25245	2.24781
TriadBdwth	Block:01024KB	GB/s	2.60669	0.00300208	2.60081	2.61179	2.60908	2.60767	2.6035	2.61179	2.60879	2.60395	2.6071
TriadBdwth	Block:02048KB	GB/s	2.83006	0.00654738	2.81116	2.83431	2.81116	2.82291	2.83089	2.82943	2.83431	2.82988	2.82546
TriadBdwth	Block:04096KB	GB/s	2.95867	0.00896205	2.93846	2.96482	2.96482	2.95772	2.95961	2.9478	2.93846	2.96191	2.96308
TriadBdwth	Block:08192KB	GB/s	3.02942	0.0375629	3.02181	3.03457	3.02181	3.03236	3.03457	3.03397	3.02839	3.03114	3.0299
TriadBdwth	Block:16384KB	GB/s	2.92692	0.0197325	2.87615	2.92909	2.87615	2.92909	2.92537	2.8912	2.92819	2.92878	2.9267
TriadFlops	Block:00064KB	GFLOP/s	0.129387	0.00814035	0.127156	0.130423	0.129318	0.129538	0.129404	0.130423	0.127156	0.12937	0.129858
TriadFlops	Block:00128KB	GFLOP/s	0.20575	0.0010177	0.20423	0.207797	0.20423	0.206155	0.206291	0.204544	0.206106	0.205509	0.207797
TriadFlops	Block:00256KB	GFLOP/s	0.294575	0.00148899	0.291069	0.295707	0.29541	0.294154	0.295191	0.291069	0.295707	0.295164	0.291774
TriadFlops	Block:00512KB	GFLOP/s	0.375283	0.00049297	0.374635	0.37603	0.375157	0.375928	0.374673	0.375137	0.375594	0.375408	0.374635
TriadFlops	Block:01024KB	GFLOP/s	0.434449	0.00050346	0.433468	0.435298	0.434846	0.434612	0.433916	0.435298	0.434799	0.433992	0.434517
TriadFlops	Block:02048KB	GFLOP/s	0.471676	0.00109123	0.468527	0.472386	0.468527	0.470485	0.471815	0.471571	0.472386	0.471646	0.470909
TriadFlops	Block:04096KB	GFLOP/s	0.493111	0.00149367	0.489744	0.494136	0.494136	0.492953	0.493269	0.4913	0.489744	0.493652	0.493846
TriadFlops	Block:08192KB	GFLOP/s	0.504904	0.00062604	0.503635	0.505762	0.503635	0.505394	0.505762	0.505662	0.504732	0.505189	0.504984
TriadFlops	Block:16384KB	GFLOP/s	0.48782	0.00328875	0.479358	0.488181	0.479358	0.488181	0.487562	0.481867	0.488031	0.48813	0.487784

7.27. SPEC CPU2006 Integer Results (No Auto-Parallel)

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
400.perlbench	398.00	396.00	396.00
401.bzip2	582.00	582.00	581.00
403.gcc	375.00	375.00	375.00
429.mcf	373.00	373.00	371.00
445.gobmk	510.00	511.00	511.00
456.hmmer	869.00	869.00	869.00
458.jeng	595.00	612.00	595.00
462.libquantum	508.00	506.00	506.00
464.h264ref	710.00	706.00	709.00
471.omnetpp	374.00	373.00	374.00
473.atar	494.00	495.00	494.00
483.xalancbmk	268.00	260.00	259.00

7.28. SPEC CPU2006 Integer Results (Auto-Parallel Enabled)

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
400.perlbench	410.00	411.00	411.00
401.bzip2	539.00	539.00	538.00
403.gcc	330.00	331.00	334.00
429.mcf	184.00	184.00	184.00
445.gobmk	600.00	601.00	607.00
456.hmmer	213.00	214.00	214.00
458.jeng	508.00	508.00	507.00
462.libquantum	12.20	14.30	13.00
464.h264ref	959.00	959.00	1031.00
471.omnetpp	339.00	340.00	339.00
473.atar	350.00	349.00	349.00
483.xalancbmk	214.00	214.00	214.00

7.29. Speedup of SPEC CPU2006 Integer Results

	Iteration #1 Speedup	Iteration #2 Speedup	Iteration #2 Speedup
400.perlbench	-3.02%	-3.79%	-3.79%
401.bzip2	7.39%	7.39%	7.40%
403.gcc	12.00%	11.73%	10.93%
429.mcf	50.67%	50.67%	50.40%
445.gobmk	-17.65%	-17.61%	-18.79%
456.hmmer	75.49%	75.37%	75.37%
458.jeng	14.62%	16.99%	14.79%
462.libquantum	97.60%	97.17%	97.43%
464.h264ref	-35.07%	-35.84%	-45.42%
471.omnetpp	9.36%	8.85%	9.36%
473.atar	29.15%	29.49%	29.35%
483.xalancbmk	20.15%	17.69%	17.37%
Average Increase Per Benchmark	21.72%	21.51%	20.37%
Total Average Increase		21.20%	

7.30. SPEC CPU2006 Floating Point Results (No Auto-Parallel)

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
416.gamess	937	940	938
433.milc	479	463	489
435.gromacs	579	579	578
436.cactusADM	1372	1441	1338
437.leslie3d	604	604	603
444.namd	496	497	496
447.dealII	430	429	430
450.soplex	270	270	283
453.povray	236	235	237
454.calculix	1484	1484	1484
459.GemsFDTD	517	517	517
465.tonto	652	649	652
470.lbm	378	379	378

482.sphinx3	632	630	632
434.zeusmp	623	625	625

7.31. SPEC CPU2006 Floating Point Results (Auto-Parallel Enabled)

	Iteration #1 [s]	Iteration #2 [s]	Iteration #3 [s]
416.gamess	1238	1185	1197
433.milc	190	189	190
435.gromacs	485	489	482
436.cactusADM	60.9	53.1	50.3
437.leslie3d	87.7	90.5	95.8
444.namd	457	456	457
447.dealII	293	293	293
450.soplex	296	263	286
453.povray	191	191	190
454.calculix	382	292	375
459.GemsFDTD	119	122	121
465.tonto	469	462	469
470.lbm	49.9	49.9	50.1
482.sphinx3	528	544	514
434.zeusmp	93.9	93	90.8

7.32. Speedup of SPEC CPU2006 Floating Point Results

	Iteration #1 Speedup	Iteration #2 Speedup	Iteration #3 Speedup
416.gamess	-32.12%	-26.06%	-27.61%
433.milc	60.33%	59.18%	61.15%
435.gromacs	16.23%	15.54%	16.61%
436.cactusADM	95.56%	96.32%	96.24%
437.leslie3d	85.48%	85.02%	84.11%
444.namd	7.86%	8.25%	7.86%
447.dealII	31.86%	31.70%	31.86%
450.soplex	-9.63%	2.59%	-1.06%
453.povray	19.07%	18.72%	19.83%
454.calculix	74.26%	80.32%	74.73%
459.GemsFDTD	76.98%	76.40%	76.60%
465.tonto	28.07%	28.81%	28.07%
470.lbm	86.80%	86.83%	86.75%

482.sphinx3	16.46%	13.65%	18.67%
434.zeusmp	84.93%	85.12%	85.47%
Average Increase Per Benchmark	42.81%	44.16%	43.95%
Total Average Increase	43.64%		

7.33. Rodinia/Burkardt Benchmarks Average Execution Times

	2 threads	4 threads	8 threads	12 threads	24 threads
Leukocyte (s)	11.70	6.11	3.65	2.16	1.67
LU Decomposition (ms)	242.82	130.00	76.40	69.23	145.20
Speckle Reduction (ms)	638.14	415.07	306.41	283.93	492.26
Kmeans (s)	3.35	3.67	2.91	2.16	1.67
Stream Clusters (s)	47.08	25.18	14.61	11.91	10.40
FFT (ms)	76.17	45.41	31.63	27.85	31.36
Primes (s)	2.04	1.17	0.64	0.44	0.36

7.34. Rodinia/Burkardt Benchmarks Speedup between Thread Count

	2-4 threads	4-8 threads	8-12 threads	12-24 threads
Leukocyte (s)	47.78%	40.26%	40.82%	22.69%
LU Decomposition (ms)	46.46%	41.23%	9.38%	-109.74%
Speckle Reduction (ms)	34.96%	26.18%	7.34%	-73.37%
kmeans (s)	-9.55%	20.71%	25.77%	22.69%
Stream Clusters (s)	46.52%	41.98%	18.48%	12.68%
FFT (ms)	40.38%	30.35%	11.95%	-12.60%
Primes (s)	42.65%	45.30%	31.25%	18.18%
Average Increase Between Threads	35.60%	35.14%	20.71%	-17.07%

7.35. FPGA Results

Benchmark	Clk Period (MHz)	Clk Cycles	Throughput (ns)	Delay for valid data (Clock Cycles)	Delay (ns)
FFT	101	1	9.87	12	118
AES	376	1	2.66	1	2.66
FIR	710	1	1.41	8	1.13
FP Mul	550	9	16.4	9	46.4
FIR Core	550	11	20.0	20	36.4

Bibliography

[1] "The Methodology of Random Logic Design," [Online]. Available: http://www.intel4004.com/mrld.htm. [Accessed 20 January 2012].

[2] D. Risley, "PCMech," 23 March 2001. [Online]. Available: http://www.pcmech.com/article/a-cpu-history/10/. [Accessed 21 January 2012].

[3] "AMD," 5 June 2000. [Online]. Available: http://www.amd.com/us/press-releases/Pages/Press_Release_729.aspx. [Accessed 20 January 2012].

[4] C. S. a. D. Patterson, "Berkley," [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1982/CSD-82-106.pdf. [Accessed 20 January 2012].

[5] M. Schmalz, "University of Florida," [Online]. Available: http://www.cise.ufl.edu/~mssz/CompOrg/CDA-proc.html. [Accessed 21 January 2012].

[6] J. Tyson, "HowStuffWorks.com," 23 August 2000. [Online]. Available: http://computer.howstuffworks.com/computer-memory4.htm. [Accessed 21 January 2012].

[7] T. Soderstrom, "Tom's Hardware," 11 December 2006. [Online]. Available: http://www.tomshardware.com/reviews/overclocking-guide-part-1,1379.html. [Accessed 20 January 2012].

[8] G. Petley, "VlsiTechnology," 22 September 2008. [Online]. Available: http://www.vlsitechnology.org/. [Accessed 22 January 2012].

[9] "SeachDataCenter," October 2004. [Online]. Available: http://searchdatacenter.techtarget.com/definition/multi-core-processor. [Accessed 23 January 2012].

[10] B. Barney, "Lawrence Livermore National Laboratory," [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp/. [Accessed 23 January 2012].

[11] NVIDIA, "What is GPU Computing?," 2012. [Online]. Available: http://www.nvidia.com/object/GPU_Computing.html. [Accessed 18 1 2012].

[12] P. Lilly, "From Voodoo to GeForce: The Awsome History of 3D Graphics," 19 5 2009. [Online]. Available: www.maximumpc.com/article/features/voodoo_geforce_awsome_history_3d_graphics. [Accessed 26 1 2012].

[13] "Graphics Processing Unit," 12 1 2012. [Online]. Available:

en.wikipedia.oeg/wiki/Grpahics_processing_unit. [Accessed 19 1 2012].

[14] NVIDIA, "PTX: Parallel Thread Execution ISA Version 2.3," NVIDIA, 2011.

[15] "NVIDIA CUDA Compute Capability Comparative Table," 6 6 2010. [Online]. Available: www.geeks3d.com/20100606/gpu-computing-nvidia-cuda-compute-capability-comparative-table/. [Accessed 11 12 2011].

[16] NVIDIA, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi v1.1," NVIDIA, 2009.

[17] NVIDIA, "NVIDIA CUDA Architecture," NVIDIA, 2009.

[18] F. E. Allen, "The Greatest Inventors You've Never Heard Of," 2009.

[19] "FPGA Applications & Consulting Experts," [Online]. Available: http://fpgaace.com/index.php/fpga-history/. [Accessed 27 February 2012].

[20] "FPGA Central," 2011. [Online]. Available: http://www.fpgacentral.com/docs/fpga-tutorial/history-programmable-logic. [Accessed 28 January 2012].

[21] V. Betz, "University of Toronto," [Online]. Available: http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html. [Accessed February 1 2012].

[22] "Electrical Engineering Times," October 2008. [Online]. Available: http://www.eetimes.com/electrical-engineers/education-training/courses/4000134/Fundamentals-of-FPGAs. [Accessed 2 February 2012].

[23] B. C. L. a. A. E. Crews, "The Evolution of Benchmarking as a Computer Performance Evaluation Technique," *MIS Quarterly,* vol. 9, no. 1, pp. 7-16, 1985.

[24] "SPEC CPU 2006," 7 September 2011. [Online]. Available: http://www.spec.org/cpu2006/. [Accessed 22 January 2012].

[25] "SPEC," 24 August 2006. [Online]. Available: http://www.spec.org/cpu2006/CINT2006/. [Accessed 22 January 2012].

[26] "SPEC," 27 September 2006. [Online]. Available: http://www.spec.org/cpu2006/CFP2006/. [Accessed 22 January 2012].

[27] "Rodinia: Accelerated Compute-Intensive Applicaitons with Accelerators," 16 12 2011. [Online]. Available: https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Main_Page. [Accessed 18 12 2011].

[28] J. Burkardt, "Florida State University," 03 September 2011. [Online]. Available: http://people.sc.fsu.edu/~jburkardt/. [Accessed 23 January 2012].

[29] "Scalable HeterOgeneous Computing (SHOC) Benchmarking Suite," Oak Ridge National Laboratory, 11 11 2011. [Online]. Available: ft.ornl.gov/doku/shoc/start. [Accessed 19

12 2011].

[30] "Parboil Benchmark Suite," Illinois Microarchitecture Project utilizing Advanced Compiler Technology (IMPACT), [Online]. Available: impact.crhc.illinois.edu/parboil.php. [Accessed 18 12 2011].

[31] "Xilinx," 6 February 2009. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf. [Accessed 12 February 2012].

[32] "University of Virginia," [Online]. Available: http://www.cs.virginia.edu/!skadron/Papers/boyer_leukocyte_ipdps09.pdf. [Accessed 23 January 2012].

[33] M. Strickland, "HEARST Electronic Products," 1 March 2010. [Online]. Available: http://www2.electronicproducts.com/The_evolution_of_FPGA_coprocessing-article-FAJH_FPGA_Mar2010-html.aspx. [Accessed January 29 2012].

[34] J. Sanders and E. Kandrot, CUDA by Example, Boston, MA: Pearson Education, 2011.

[35] NVIDIA, "NVIDIA CUDA C Programming Guide," NVIDIA, 2011.

[36] W.-m. Hwu and D. Kirk, Programming Massively Parallel Processors: A Hands-on Approach, Burlington, MA: Morgan Kaufmann, 2010.

[37] A. Danalis, P. Roth, G. Marin, K. Spafford, C. McCurdy, V. Tipparaju, J. Meredith and J. Vetter, "The Scalabe HeterOgeneous Computing (SHOC) Benchmark Suite," Pittsburgh, PA, 2010.

[38] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," Dept Computer Science, UVA, 2009.

[39] F. Abi-Chahla, "A Few Definitions," 18 6 2008. [Online]. Available: www.tomshardware.com/reviews/nvidia-cuda-gpu,1945-7.html. [Accessed 27 1 2012].

[40] "National Instruments," 19 December 2011. [Online]. Available: http://zone.ni.com/devzone/cda/tut/p/id/6983. [Accessed 27 January 2012].