TDT4590
Complex Computer Systems,
Specialization Project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Rune Johan Hovland

Latency and Bandwidth Impact on GPU-systems

Trondheim, Norway, December 17, 2008

NTNU
Norwegian University of Science and Technology

Abstract

General-purpose computing on graphics processing units (GPGPU) has become a hot topic for computer scientists. Especially within the High-Performance Computing (HPC) community, these massively parallel graphics processors have received much attention, and with good reason. Benchmarking of common problems solved on the newest GPUs from NVIDIA and other vendors shows a speedup of up to 70 times that of a single CPU implementation. Among the heart of all this innovation is the NVIDIA Tesla architecture, capable of scaling beyond that of modern CPUs, reaching a total of 240 streaming processors with its latest models. Much attention has been given to the architecture and implementation details, but little or no focus has been given upon the components surrounding the GPU, namely the computer and its components.

The process of moving data to and from the GPU is influenced by two important metrics, latency and bandwidth. For small transfers, the latency can severely impact the performance, while for larger transfers the bandwidth comes more and more into play. These metrics are therefore the metrics that will be used throughout this report to judge the important of various properties of the host system. These properties include processor clock frequencies, chipsets, memory frequencies and architecture, as well as the PCI Express bus.

Our measurements performed shows how the PCI Express bus is a major bottleneck for the transfers to and from the GPU, making overclocking this bus an action worth considering. The CPU clock frequency which one would assume to have great influence, proved not to affect the bandwidth at all, and affected the latency only to a small extent. The architecture of the CPU, however proved to be a crucial aspect. The Intel CPU we tested, greatly outperformed the AMD counterparts on all metrics.

Finally, note that there is still one important fact that has not yet been mentioned, and that is that the GPU is still not capable of performing all the tasks required of a realistic application. This makes the high-end general CPU still a necessity to achieve peak performance. However as more and more computations are moved over to the GPU, the trade-of between cost and performance can soon make the investment in high-end computers unacceptably large for a marginal improvement.

Preface

This report is the product of a student project at HPC-group at Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU). It was first intended as a pre-study for my master thesis, but has over the duration evolved into an independent report.

To be able to complete this report, there are many people who deserve my acknowledgement. First and foremost I would like to thank Associate Professor Anne C. Elster, who through her guidance and supervision made this report possible. Had it not been for her dedication towards building the HPC-lab at NTNU, this report would not have been possible. In that regard I must also send my gratitude to NVIDIA Corporation for donating most of the graphics cards used throughout this report, through Elsters Professor Partnership with NVIDIA. I would also like to thank Rune Erlend Jensen for beating even Google in quickly answering my many small questions. He and the rest of the HPC-group at NTNU has been a valuable source of information and support during this period.

Rune Johan Hovland

Contents

Chapter 1

Introduction

The field of High Performance Computing (HPC) is a large industry with many large installations trying to maximize the computing power accessible to the researchers. The modern processors suffers from physical constraints, and are today near their maximum frequency achievable with existing production techniques [4]. This combined with the development of programmable pipelines on the Graphics Processing Unit (GPU) has become the origin of a new research field for General-Purpose computing on the GPU (GPGPU). The graphics processor is a massively multi-core processor optimized for floating-point operations done in parallel. These properties suites the HPC community since a single GPU may hence provide computing power equivalent to small supercomputers [26].

Bigger is better is a statement which well has described the HPC community for the last decades. The eagerness to achieve higher and higher number of flops has been a metric for success. But with GPU, the landscape has changed; leaving the computers as mere supporting devices which only purpose is to facilitate for the graphics processors. The performance of a GPU is the same regardless of the surrounding system with respect to pure calculations. This has reduced the computer to a device whose purpose is to provide the GPU with the data required to perform the calculations. For at this key area, the GPU has yet to venture, since the GPU is not an alternative to a CPU, but rather an accelerator card. With this in mind, this report will put the focus on the surrounding system to determine wetter the high-end systems are still required or if a mere budget computer would suffice at achieving peak performance from the GPU.

GPGPU receives much attention from HPC-researchers due to its possibilities, but many properties of this field have yet to be unveiled. In this report the influence of some of these properties will be determined. By conducting these measurements and checking for the comformance with the model given in Section 3.2, this report seeks to provide the GPGPU developers and system administrators with one additional metric when considering computer systems and their suitability for GPGPU applications. There is a vast number of individual properties which may affect the performance of the GPU, This report focuses primarily on the external properties such as CPU frequency, memory bandwidth and others. Through isolating each of these properties in designated test-systems, the properties can be adjusted to measure the impact of different settings. This approach is somewhat limited in the sense that there might be combination of settings which may give unexpected results, but the vast num-

ber of combinations makes it impossible to test all these under the scope of this report.

1.1 Outline

This report will start with a brief introduction to the graphics processor and GPGPU programming in Chapter 2, and then continue on to describe the various components in a computer system which have properties that can affect the performance with respect to GPGPU applications.

Chapter 3 contains an explanation of latency and bandwidth, as well as a description of the methods which are used to test these two metrics. It then describes a theoretical model which seeks to give an abstraction of a GPGPU system. After that, the various system used in this report to isolate the different properties are described, including a brief introduction to the HPC-Lab located at Department of Computer and Information Science at the Norwegian University of Science and Technology, which was set up during the writing of this report. This chapter also includes a description of the proprietary software developed to perform the measurements used in this report, and finally a discussion of the various potential sources of error.

The results of the measurements and discussion of these are presented in Chapter 4, starting first with a test of the various graphics cards available, followed by a presentation of the properties tested with its results and a short comment on the impact these properties may have on GPGPU applications.

Finally, the conclusion and recommendations regarding various properties for GPGPU-systems, as well as an outline of future work will be given in Chapter 5.

Chapter 2

Background

This chapter will give a brief introduction to main concepts which are used in this report. First it starts with an overview of the field of benchmarking. Later it continues with a overview of the GPGPU and the NVIDIA Tesla architecture as well as CUDA which is used to develop for the Tesla architecture. Finally, the different hardware components tested in this report are described.

2.1 Benchmarking

A key aspect of choosing the right computing system for a calculation is knowing which system will perform the calculations fastest. But running the calculations on several machines just to test this would be pointless since it would require more time than running the calculations. To overcome this obstacle, a set of benchmarking tests can be performed. These tests seek to give a certain number of metrics to which on can derive the expected performance of a given calculation on the specified system. There exists many ways to perform these benchmarks, but common for most of them is that they give no complete view of the performanceof the system. As described by [13], the benchmarking can be biased with respect to the various metrics. In his case, he describes the relation between bandwidth and latency. These two metrics will be described more thoroughly in Section 3.1.1 and 3.1.2.

2.1.1 Hockney model

The two metrics latency and bandwidth are both part of a simple computer system model developed by Hockney [13]. The model describes in its simplest form how the bandwidth and latency affects the time it takes to transfer a given set of data. This relationship is described in Equation 2.1, where the time T is given as a function of the latency α, bandwidth β and transfer size m in bytes.

$$T = \alpha + \beta * m \tag{2.1}$$

2.1.2 Dwarfs and Motifs

Another approach to benchmarking a system is to consider how the system would perform on a set of problems considered to be representative of the full

set of problems which could be solved at the given system. These problems where first described by Phil Colella under the name dwarfs [4]. The seven dwarfs where later expanded to 13 problems and renamed to Motifs by a group of scientists in the important *Berkeley view* article [4] by Patterson et al. This extension where made to include the many new areas computers are used today, such as *machine learning*, *database software* and *computer graphics*.

These reference problems try to give metrics for a system without considering the details of the system, but rather focus on the performance. This approach also removes the hardware details from the metrics, allowing the metrics to be compared between a vast number of different computer architectures. Recently a group of researchers and students at the University of Virginia used some of these reference problems to measure the use performance of GPGPU with CUDA compared to single and multi-core CPU implementations [7].

2.2 Graphics Processors

The GPUs were primarily designed to output a 2D surface which can be displayed on screen. This has influenced the architecture of the graphics processors. Due to the highly parallelizable nature of this task the graphics processor has developed into a parallel processor which consists of several hundred streaming processors [24] [2].

2.2.1 General-purpose computing on graphics processing units

With the introduction of programmable vertex and fragment-shaders in the graphics processor, the possibility to manipulate the graphics calculations emerged [6]. The fact that still the graphics processor was foremost a graphics processor and thus only had a instruction set intended for graphics forced the general purpose calculations to be masked as graphics calculations. This was done by mapping the calculations into polygons and textures, and passing them through the graphics pipeline. The results could then be read from the resulting textures and images produced [12] [28]. This process required a great *creativity* of the programmers, and was thus not suited for the general market. These camouflaged calculations were done using shading languages which were used to manipulate the appearance of polygons and its textures. While the shading languages originally developed for just this purpose was not suited for general purpose calculations, and new languages had to be developed which hid the graphical computations from the developers. One of these attempts is CUDA which will be described more in Section 2.2.3.

Because of the vast number of parallel processors located on the GPU, it is such a sought after computing medium. Modern GPUs have up to 800 streaming processing units working in parallel [2]. Being able to utilize this massive parallelism would enable the scientist to have access to a personal mini-supercomputer. Utilizing this power properly can result in great speedups for calculations as shown by a group of researchers a University of Virginia [7] which achieved speedups ranging from 3 to 70 times that of a modern single core processor.

2.2.2 The Tesla Architecture

In November 2006, NVIDIA released its Tesla architecture which enabled general-purpose computations on the graphics cards [21]. Other vendors such as ATI also has similar architectures. The Tesla architecture is used in all of NVIDIAs resent graphics processors as of GeForce 8xxx. The intention is to create a scalable, general-purpose architecture which can be used in a wide range of graphics processors. This is achieved through a SIMT (single-instruction, multiplethread) multiprocessor with 16 streaming processors, designed to handle 32 threads in parallel, called a *warp*. These 32 threads are executed 8 at a time, and all 32 threads execute the same instruction. On the graphics processor the multiprocessor is duplicated to achieve the desired number of streaming processors.

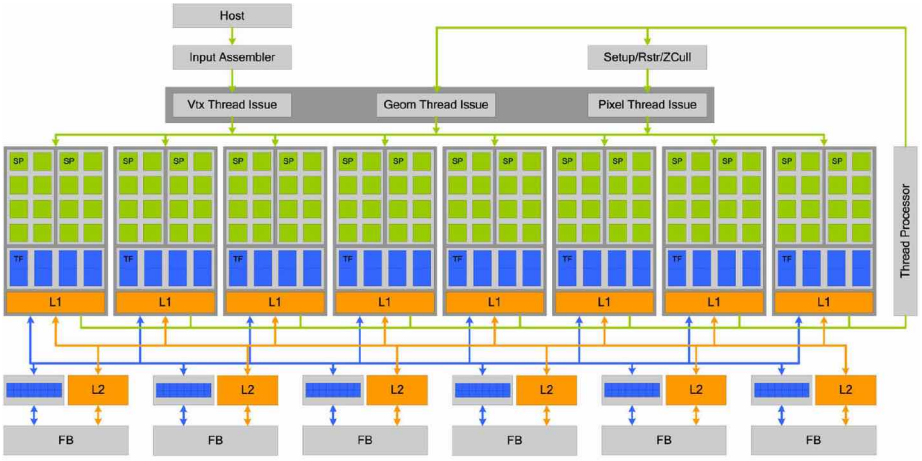

Figure 2.1: Architecture of modern NVIDIA graphics card [6].

2.2.3 CUDA

CUDA is a programming language extension created by NVIDIA Corporation. It allows the developer access to performing calculations on the graphics processor implementing the Tesla Architecture without having to camouflage the calculations as graphical calculations. The language extension is developed for the C language, but there has also been developed support for Python [16], and more languages will probably follow. CUDA allows the developer to write kernels which will be performed in parallel on the streaming processors on the GPU.

2.3 Computer architecture

Even though this report seeks to enhance the performance of GPGPU systems, it does not focus on the graphics processor itself since there are many other components within the computer which also play a role in the performance of the graphics card. The influence of many of these components will be tested

throughout this report, and a brief description of the various components is given here.

2.3.1 Processors

The Central Processing Unit (CPU) is traditionally the main computational resource in the computer. It is responsible for interpret and execute the instructions in a program, and thus has a finger involved in all that happens during program execution. The clock frequency of the processor determines how fast the processor is able to process each instruction, and can thus play a crucial part in the overall performance of the system. The increase in CPU clock frequency has stagnated in the last couple of years due to the *brick wall* discussed in the *Berkeley view* article [4]. As a least-effort solution to increase system performance, the CPU manufacturers have begun creating multi-core CPUs instead of trying to increase the clock frequency. As of 2008, most processors are either dual or quad-core processors. This focus on parallelism forces the developers to redesign their programs for parallelism to achieve speedups.

In this report three recent processors are used during the tests, Intel Core 2 Quad Q9300, Intel Core 2 Quad Q9550 and AMD Phenom X4 9850 Black Edition. The two Intel processors [14] are both quad-core processors of the same series, but with different cache size and processor speed. The AMD processor [3] is also a quad-core. The key difference between the Intel and AMD processor is that the AMD has the memory-controller integrated on the chip instead of on the chipset, and also has lower cache size than the Intel counterparts. The Intel Q9300 and the AMD processors both have the same clock frequency, while the Intel Q9550 have a higher clock frequency.

2.3.2 Chipsets

The chipset is the infrastructure in the modern computer. It is responsible for connecting the processor(s) to the memory and peripheral units such as graphics cards. Vital in this infrastructure is the buses allowing data to be transported between the various locations. The chipset is normally divided into two parts, Northbridge and Southbridge. The Northbridge connects the high speed demanding units such as processors, memory and PCIe x16 ports, while the Southbridge connects the units with lower bus speed requirements such as PCI x1 ports, PCI, USB and so on. The Southbridge is connected to the Northbridge giving the chipset a hierarchical layout, where the peripheral units are connected on the layer providing the required bandwidth, without requiring peek-performance buses unless necessary. This approach reduces the cost of the chipsets.

During these tests, three different chipsets will be used, Intel X48[15], NVIDIA nForce 790i [25] and AMD 790FX [1]. These chipsets have somewhat different architecture and performance characteristics, but these details are beyond the scope of this report. And vital difference however is which processors the chipsets are designed for. While the NVIDIA chipset support Intel, the two other chipsets only supports its own vendors processors naturally. The AMD processor supported by the 790FX chipset has an integrated memory controller, and this is thus removed from the chipset as well.

2.3.3 Random Access Memory

The memory is the backbone of the data storage of a modern computer. To overcome the high access time for data located on harddrives, Random Access Memory (RAM) has become a vital level in the cache-hierarchy designed to camouflage these access times. The most common RAM used as of 2008 is Double-Data-Rate Synchronous Dynamic RAM (DDR-SDRAM or DDR for short). The DDR modules can operate on many different clock frequencies, and this in correlation with the timings [30] determines how fast the module can supply the computer with the required data. Once a request for data has been made to the memory module, the module requires a number of clock cycles to prepare and fetch the first piece of a set of data, and then usually a lower number of clock cycles to fetch the following pieces. The number of cycles required is given by the timings of the memory module. How long time a cycle takes is determined by the clock frequency the memory operates on and thus a speedup of the memory can be achieved both by reducing the timings and by increasing the clock frequency. However, an increase in the clock frequency may also require an increase in the timings of the module, and wetter a clock frequency increase would speed up the performance is situation dependent [17]. Most modern motherboards would automatically adjust the timings to a suitable level based on the clock frequency.

2.3.4 PCI Express bus

The PCI Express bus [5] technology was aimed to replace the aging PCI and AGP bus, due to the increased demand for bandwidth over the bus, mainly by the graphics cards producers. The design of the bus is modular with the possibility to change the number of lanes for each connection. The standard numbers of lanes are 1, 2, 4, 8 and 16. 32 lanes are also allowed but rarely seen. The notation for a connector is PCIe x16, where the number specifies the number for lanes. There are also possible to create connections with high number of lane connectors, while only using a lower number of bus-lanes. The extension card would then operate on a bus with a lower bandwidth than a card placed in a PCI Express connector with a full number of bus-lanes. This mode is denoted as PCIe x16 (x8 mode), where the number inside the parentheses gives the actual number of lanes used on the bus. This reduction of lanes is common for the second PCIe x16 port for budget-motherboards.

The first version of the PCI Express was capable of delivering 250 MB/s per lane, giving a total of 8 GB/s for a 16 lane bus [29]. The version mostly in used for graphics cards as of 2008 are version 2.0 which doubles this bandwidth. A version 3.0 is under development, promising another doubling of the bandwidth giving a total of 32 GB/s for a 16 lane bus.

Chapter 3

Methodology and models

This chapter will discuss the various properties to test, and the hardware and software used to perform these tests. The validity of the test and how they should behave according to the theoretical model defined in this chapter is also discussed.

3.1 Properties to test

The intention of this paper is to investigate the properties exterior to the graphics processor which may affect its performance. Due to this focus the only measurable properties are bandwidth and latency when transferring data to and from the GPU. These two metrics are the key metrics in the Hockney model described in 2.1.1, and will be described in more detail in the following two sections.

3.1.1 Latency

Latency is the time between a given task is initiated and the task is beginning to execute. In terms of data-transfer, the latency is the time between the instructions to transfer is given and the data-transfer actually commence. This delay may occur due to instruction decoding, memory latency, waiting for bus-access and other causes. The latency may have a great impact on performance when transferring small amounts of data, since the latency is a constant startup cost, which must be paid regardless of how many items are to be transferred.

Testing the latency is not a trivial task to test correctly, but there are ways to approximate the measurements by measuring transfer times. The latency is as given the startup time of a transfer, and can thus be measured by transferring as small as possible amount of data and use the timing as an approximation of the latency. Due to the possibility of computer components being optimized for transfers of data in blocks, the latency measurements will be done on several sizes and choosing the lowest time, under the assumption that this is the measurements closest to the actual latency. In this particular case the test will be performed for 1, 2, 4, ..., 1024 floats (32 bit), and averaging the time measurement over 10 000 transfers.

3.1.2 Bandwidth

Bandwidth is the amount of data which can be transfer in a given amount of time. This property has great impact on the performance of a graphic processor since all data which shall be used in the computation must be copied to the graphics processor. To measure the bandwidth one has to take into account the latency, since each transfer will have a period of time before the actual transfer commence. The formula for calculating the bandwidth is given in Formula 3.1 as a function of size, time and latency.

$$bandwidth = \frac{size}{time - latency} \tag{3.1}$$

Measuring the bandwidth will be done by conducting the test for a wide range of sizes and measuring the time each transfer took. For the tests in this report the sizes and number of transfers for each test is given in Table 3.1.

Table 3.1: Transfer sizes and number of transfers used for bandwidth tests.

Transfer size	4KB	16KB	64KB	256KB	1MB	4MB	16MB
# of transfers	5718	3270	1870	1069	611	349	200

3.2 Model

The computer is a complex machinery, with numerous connections and components. To try to encompass all this in a model over the common GPPGU system and how memory is copied between host and device-memory would render the results to complex to interpret. To overcome this obstacle, a simplified model has been constructed. This model as given in Figure 3.1 generalizes the system to such a coarse model that each single property can be isolated and tested.

The first important assumption used in this model is that the system complies with Hockneys model [13]. The Hockney model describes the relationship between the time needed to transfer a given amount of data between two places based on the parameters latency and bandwidth. To adapt this model to the model used in this report, the transfer can be thought of as a chain of transfers between the components on the path between the memory and GPU. This gives that the latency would be the sum of latencies on the chain of transfers, and that the bandwidth would be dictated by the transfer with the lowest bandwidth in the chain.

The model used in this report assumes the use of Direct Memory Access (DMA), since this is used by the graphics card. In case of copying from a non-DMA accessible memory location, the graphics card requires the data to be copied to a DMA-accessible location. To encompass this into the model, the reduction in performance will be view upon as a lower bandwidth from a non-DMA accessible location due to the extra memory copying. Also the buses between the memory, memory controller and chipset are considered to offer the required bandwidth and thus not affect the performance. When considering latency for these buses, this latency is considered to be a part of the latency inflicted by the chipset.

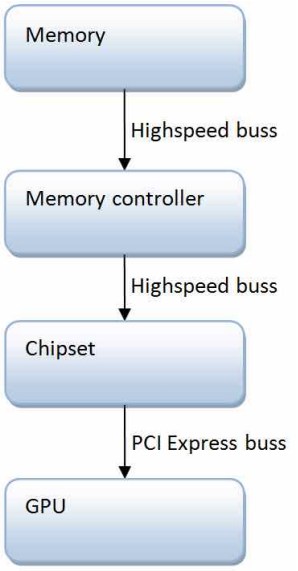

Figure 3.1: Proposed model of GPGPU system.

The use of DMA would require the CPU only to decode and interpret the copy-instruction, and signal the CPU to perform the copy. Thus the CPU should only be active during the initialization of the transfer, and thus only affect the latency and not the bandwidth. For the CPU, the only parameter adjustable in this model is the clock frequency.

The memory is considered to be a single memory module with equal access time regardless of location. This abstraction could be false for few test-runs, but due to the high number of runs the memory would statistically behave in accordance with the given abstraction. The timings of the memory module are considered to be directly linked to the clock frequency and are thus not tunable, but rather set by the system itself based on the given clock frequency. In addition to the clock frequency, both the size and architecture (DDR2 or DDR3) are tunable parameters. The memory‘s parameters are expected to affect both latency and bandwidth, with the exception of the size which should only affect the latency.

The chipset does in real life consist of many components with numerous interconnects between, but in this model these are abstracted down to a single component with no tunable parameters. The only alteration which can be made is choosing different chipsets. This abstraction is done due to the inability to adjust the low-level parameters of the chipset to such a degree that it can be isolated as independent properties. This, as well as the fact that the chipset is merely a transporting stage in the transfer; this abstraction will suffice for this report. As for the influence of the chipset towards the performance, it is considered to affect both the bandwidth and latency due to its involvement in routing packets between the various components connected to the chipset.

The transfer between the chipset and the GPU is conducted over the PCI Express bus. The bandwidth of this bus is expected to be a bottleneck and

is therefore emphasized as a separate component in the model. Packet switching and setup time are considered to be parts of the bandwidth and latency respectively, and the PCI Express bus will therefore be able to affect both.

The GPU are in this model considered to behave in the same way as a memory module with respect to memory transfers, but without its tunable parameters. These parameters are fixed for each graphics card model, and are not tunable. Even though graphics cards can be tuned for these parameters, this operation is considered to be outside the scope of this report. The various graphics cards models can however be changed. In the same way as the memory, the graphics card is expected to affect both latency and bandwidth.

3.3 Test hardware

The test systems are all various machines located at the HPC-Lab [9] [10] at the Department of Computer and Information Science at the Norwegian University of Science and Technology. The lab and its machines were assembled during fall 2008 to facilitate for research within GPGPU and other HPC activities. The participated in the construction of this lab has been a part of this project, trying to assure a wide range of systems which enables the user to develop applications for a wide range of systems. With this intention in mind the lab is also highly customizable, allowing the students and researchers to customize the hardware to fit their current needs. Each machine has been labeled HCPx and the default configuration for each system used in this report is given in Appendix A. In the following sections the reason for using each of the systems will be given. During the tests many alterations were conducted to the machines and these alterations are described in a separate section along with the test in question. The important differences are listed in Table 3.2.

Table 3.2: Comparison of important aspects of the machines.

	Processor	Chipset	Memory
HPC2	AMD Phenom, 2.50GHz, 4MB cache	AMD 790FX, DDR2	DDR2, 4GB, z 1066MH
HPC3	Intel Core 2 Quad, 2.50GHz, 6MB cache	Intel X48 Express	DDR2, 4GB, 1066MHz
HPC4	Intel Core 2 Quad, 2.83GH, 12MB cache	Intel X48 Express	DDR2, 4GB, 1066MHz
HPC5	Intel Core 2 Quad, 2.83GH, 12MB cache	Intel X48 Express	DDR3, 4GB, 1600MHz
HPC6	Intel Core 2 Quad, 2.83GH, 12MB cache	NVIDIA nForce 790i Ultra SLI	DDR3, 4GB, 1600MHz

For the tests conducted the NVIDIA GeForce GTX 280 graphics card were used since it was at the time the most powerful GPU available at the lab. At a later time the Tesla C1060 were also made available giving a wide range of available graphics card available at the lab, as can be seen in Table 3.3.

Table 3.3: List of available graphics card at the HPC-Lab as of December 2008.

Graphics card	Quantity
NVIDIA GeForce 8400 GS	1
NVIDIA GeForce 8600 GT	1
NVIDIA GeForce 8800 GTX	1
NVIDIA Tesla C870	2
NVIDIA Quadro FX 5600	1
NVIDIA GeForce 9800 GTX	1
NVIDIA GeForce 9800 GX2	2
NVIDIA GeForce GTX280	2
NVIDIA Tesla C1060	2

3.3.1 HPC2

HPC2 is the only AMD machine used in this series of testing, and is included only to be able to test the Intel processor versus the AMD processor, and the effect integrated memory controller might have on transfers between host and device-memory.

3.3.2 HCP3

The HPC3 machine is not particularly interesting hardware vise, but it is used in these tests to compare against HPC2 in the processor architecture tests. The reason for choosing HPC3 over one of the others is the default frequency of the processor in HPC3 which matches the AMD processor in HPC2.

3.3.3 HPC4

HPC4 is included in these tests due to the DDR2 memory. It is the only difference compared to the HPC5 machine, and thus enables the testing of different memory architecture.

3.3.4 HPC5

HPC5 is the main machine used in these tests. The reason for this choice is due to this machine being the one which can most easily be compared to the other machines and thus giving easy access to several properties desirable to test. It is also a machine that in my opinion closely resembles a standard machine configured to run GPGPU applications on.

3.3.5 HPC6

This machine is included in the tests mainly because of its NVIDIA nForce chipset which enables tests of different chipsets.

3.4 Test software

Even though there without doubt exists several programs capable of testing the properties which are to be tested throughout this report, the extra time needed to locate and verify the fitness of these programs would exceed the time required to write the program. Thus, the program used for this report was written from scratch using C++ and Cuda. Due to the crude testing methods the program is a simple ping pong program with timings for a group of transfers, In addition there has been created a program which simulates load on the host-memory by conducting copies of data between various memory locations.

3.4.1 Test program

The test program is an implementation of the latency and bandwidth test described in 3.1. It was implemented in C++ using CUDA to communicate with the graphics processor. The program contains a vital method *testTransfer* which accepts two memory locations and a transfertype-enum, as well as data size, number of iterations, which allows the method to perform the test as given by the parameters and return the average time per run. This complies with the remark in 3.1, which stated that the latency test is just a bandwidth test with neglectable amount of data. The method is located in *common.cu* along with other usefull code-snppets. The *measureMalloc.cu* and *measureCudaMallocHost.cu* is responsible for initiating the test of testing with *malloc* and *cudaMallocHost*, and also writes the results to the output file. The complete code can be found in Appendix B.

3.4.2 Load simulator

The load simulator simulates memory load by allocating four memory locations of a given size, and copying data between these locations in a circle. By doing so the memory's bandwidth is being used, and would potentially affect the performance of the host to device transfers. The constant copying of memory would also require the processor to run the program and oversee the data copy. The program may have several instances of it running at the same time, and takes as an optional parameter the number of kilobytes which are to be copied between the memory locations. The code can be seen in Appendix C.

3.5 Potential sources of error

Even though the systems described in the previous sections were put together to provide the possibility to be able to alter single properties with the system for testing purposes, there are several potential sources of error.

3.5.1 Test methods

The testing methods used in these experiments are simple tests designed to give an indication of the properties. To be able to fully and correctly test the properties tested in this report, more accurate testing methods must be developed. This however would include testing done in hardware, not in software

as is done her, and is beyond both my level of expertise and the scope of this report.

3.5.2 Multi-property variations

The machines used for these tests were selected because they were in many ways similar so the changes in hardware between two systems would be as few as possible. This enabled the possibility to make configurations where only single properties where different. However there might be other properties which are not accounted for which differentiates the systems and affect the results. For instance between the HPC4 and HPC5 machines there might be other differences between the machines other than the memory architecture due to the fact that they have different motherboards even if the motherboards uses the same chipsets. These hidden properties are hard to remove from the tests and must be taken into account when reviewing the results.

3.5.3 Background noise

These tests were performed in a controlled environment assuring the machines were unchanged between each group of tests. But due to the need for an operating system to run the tests on there are possibilities for background processes having affected the performance. These as well as process scheduling are factors which add indeterminism to the execution of the tests, but statistically would even out in the long run. To counter this, the tests are performed several times averaging the results over the multiple measurements. This approach reduces the likelihood of one of these factors to greatly influence the result, but does not remove it completely.

Chapter 4

Benchmarks

Through this chapter several different properties which can affect the performance of data transfer to and from the GPU will be examined. The first section covers the selection of a suitable graphics card to conduct the tests on. Further on the various properties are tested. Giving first a description of the configuration of the system which the tests are conducted, and the a review of the results for the latency and bandwidth tests, this chapter tries to give the reader an insight in how each of the properties affect the performance of GPGPU applications.

4.1 Graphics cards

Of course, when running GPGPU applications the graphics card which these applications are vital to the performance. However, this report does not focus on the graphics card itself, but rather on the properties of the system which are external to the graphics card. The following comparison of various graphics cards is merely to determine a fit graphics card to use throughout the tests conducted in this report.

4.1.1 NVIDIA G80 series

The G80 series from NVIDIA introduced the Tesla architecture, and is thus the first CUDA-enabled graphics cards. In that occasion NVIDIA also introduced the Tesla product lines, which are computational versions of its graphics counterparts. This is reality is basically a graphic cards without monitor connectors, and with extra device-memory. As can be seen in Figure 4.1, the GeForce 8800GTX and the Tesla c870 performs equally well. Also from the G80 series is the Quadro FX5600, which is a combination of the GeForce and the Tesla, providing large memory and graphical output.

4.1.2 NVIDIA G90 series

The G90 series did not provide a new card for the Tesla series, but provided a new type of card for the GeForce series, the GX2 cards as well as PCI Express 2.0 support. The 9800GX2 card [19] is basically two 9800GTX cores on the same graphics cards. They share the PCI Express connection, but besides that

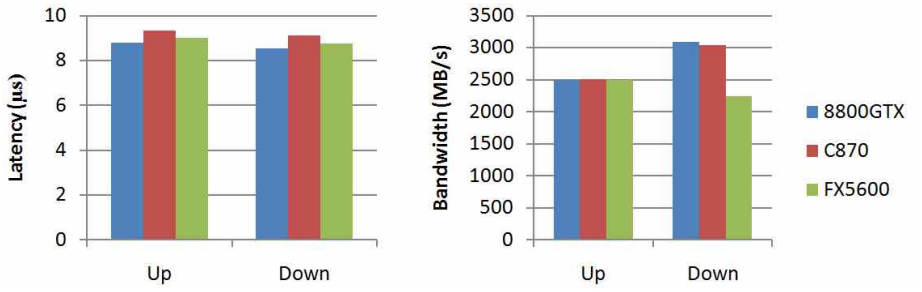

Figure 4.1: Performance of the cards of the G80 series.

operate independently as two separate cards. This sharing of the PCI Express bus gives a performance reduction in some cases as can be seen in Figure 4.2. Other cards tested from the G90 series is the 9800GTX [18].

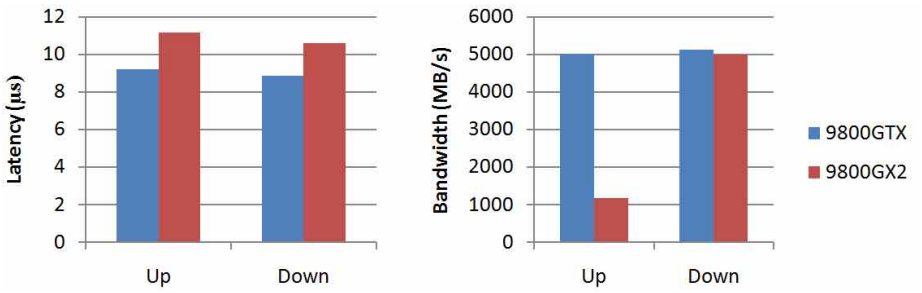

Figure 4.2: Performance of the cards of the G90 series.

4.1.3 NVIDIA G200 series

The G200 series is the newest of the NVIDIA graphics cards series. It does, like its predecessors, conform to the Tesla architecture, but provides a total of 30 Streaming Multiprocessors giving it a total of 240 Streaming processors. The G200 series does as the G90 series support PCI Express 2.0 over its connector. New with this series is the support for double precision, even though there is not fully supported due to the way double precision is implemented. For each Streaming Multiprocessor, there is added a single double precision unit, forcing a performance reduction when using double precision.

The G200 series provides cards for the GeForce, Tesla and Quadro product lines from NVIDIA with its GeForce GTX280 [20], Tesla c1060 [23] and Quadro FX5800 [22]. The GeForce card was the only card accessible at the lab when this report was started and was therefore the card of choice for the tests conducted. At the end of the period the Tesla c1060 was also made available so it is included in the tests of the various G200 series cards, but not used in the other tests. As the tests in Figure 4.3 show, there are only minor differences between the two cards, and the only difference between the two cards in respect to these tests would be the memory size, which in any case is large enough for both cards. The

Quadro FX5800 was regrettably not available for testing at the time of writing. For the remainder of this report, the GeForce GTX280 card will be used.

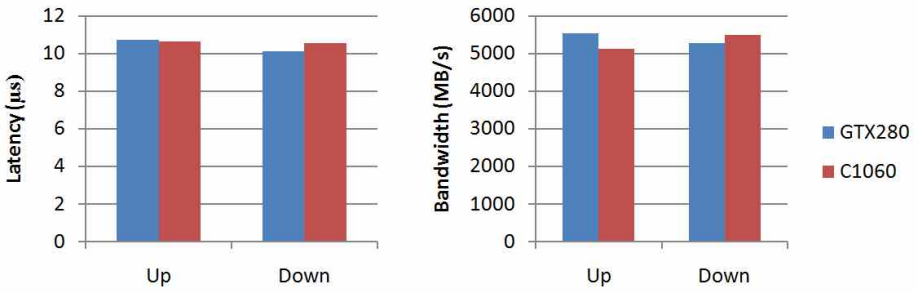

Figure 4.3: Performance of the cards of the G200 series.

NVIDIA has also announced some new models which will be available from January 2009. These models continue the trend from the earlier series, providing a high performance, low production technique version of the GTX280, named GTX285, as well as a new dual core card named GTX295. If NVIDIA has implemented their bus-sharing better than on the 9800GX2 cards, the GTX295, may prove to be a potent addition to any CUDA environment, as well as for the gaming enthusiasts.

4.2 Memory allocation

When using CUDA to do calculations on the GPU, the data on which to perform the calculations must be copied to the GPU. The location in memory of the data to be copied can affect the performance, and thus this may be an important aspect to consider. CUDA provides its own method *cudaMallocHost* in addition to C/C++'s *malloc*. It will be shown how the new CUDA method performs relative to *malloc* in both latency and bandwidth.

4.2.1 Setup

These tests were conducted on the HPC5 machine, with standard DDR3 memory running at 1600MHz.

4.2.2 Latency

While one would assume the latency should be the same, since the memory is allocated on the same memory modules, the test shows significant difference in latency as seen in Figure 4.4. The memory allocated which *cudaMallocHost* have one important feature, and that is that it is page-locked [21]. Why this is important is the way the NVIDIA GPUs transfer memory between the host-memory and device-memory. All transfers must be using DMA, and this requires that the memory is not pageable. To use DMA with pageable memory, the data must be copied to a location accessible to the DMA before it can be copied to the GPU [27]. While this effect should not affect the latency of the transfer, the

measurements show a reduced latency for *cudaMallocHost* allocated memory. This effect is an artifact of the way the latency is measured, since the measurement actually is a measurement of transfer time for a neglectable amount of data. Due to this way to measure the latency, each measurement of latency for the *malloc* allocated memory also include the overhead of a *memcpy*-instruction.

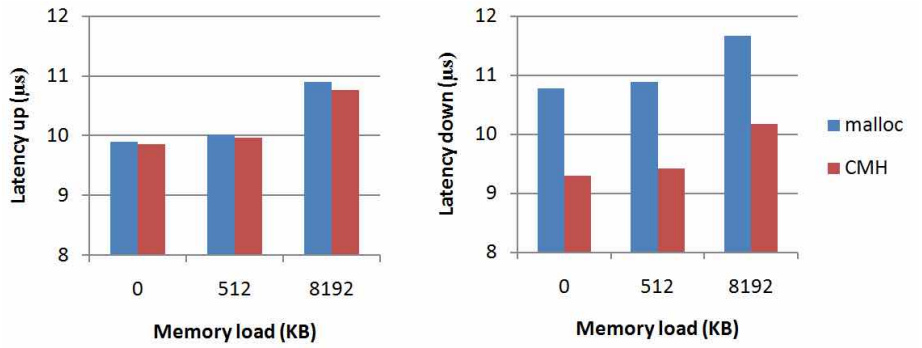

Figure 4.4: Latency when transferring to and from memory allocated using *malloc* and *cudaMallocHost* (CMH).

4.2.3 Bandwidth

Since the first obstacle one encounters in a GPGPU enabled application is the bandwidth on transfers to and from the GPU, this is an important issue. As mentioned in 4.2.2, there is a requirement for page-locked memory when transferring to the GPU. The extra copying of data required to fulfill this requirement when using *malloc* should have a noticeable effect on the measured bandwidth when copying to and from the GPU. As one can see in Figure 4.4, the bandwidth when using *cudaMallocHost* is at the best 300 percent better than the bandwidth achievable with *malloc*-allocated memory. The interesting aspect of this measurement is the fact that the transfers from *cudaMallocHost*-allocated memory are seemingly not affected by the memory-load on the host-system. On the other hand, *malloc*-allocated memory is greatly affected with a 50 percent reduction in bandwidth when there is heavy memory-load on the system.

4.3 Memory architecture

The preferred memory technology is in a change, since the introduction of DDR3, which aims to supersede DDR2. Both technologies are part of the Synchronous Dynamic Random Access Memory (SDRAM) family.

DDR3 provides several improvements over the DDR2 memory [8]. Most noticeable is the increased speed which the DDR3 can offer. This improvement comes from the change in hardware architecture for the DDR3. The DDR2s architecture does not permit higher speeds due to mechanical reasons, even though it is theoretically possible, but the mechanical technology is not just good enough to maintain the memory's integrity at increased speeds. The architecture used in DDR3 however is more suited for higher speeds, and according

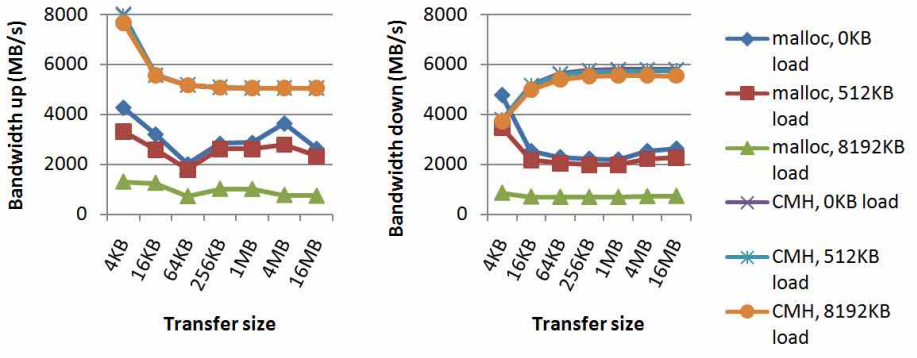

Figure 4.5: Bandwidth to and from memory allocated using *malloc* and *cudaMallocHost* (CMH).

to the specifications its speeds can reach twice that of the DDR2. Another improvement for the DDR3 is the decrease in power consumptions. As while the DDR2 required 1.8V as a base power, the DDR3 has reduced this requirement to 1.5V. This reduction may not seem important to a normal consumer, but to the large installations such as super computers and server farms, the power consumptions is one of the main financial burdens.

One of the drawbacks with DDR3 compared to DDR2 is the increased cost of purchase. Another vital difference is the lack of compatibility between DDR2 and DDR3 which forces a substitution of motherboard as well if one is to consider DDR3.

4.3.1 Setup

To test DDR2 and DDR3 against each other, the two systems HPC4 and HPC5 proved to be the most similar with only the memory being different. To allow the two memory architecture to compete on equal terms, the speed of the DDR3 memory on HPC5 were reduced to 1066MHz which is the standard running speed for the DDR2 memory on HPC4.

4.3.2 Latency

When it comes to latency, DDR3 is said to improve the latency compared to DDR2 but the initial DDR3 memory had the same latencies or worse as equivalent DDR2 memory [11]. This however is expected to change as the production techniques for DDR3 improves. The DDR3 memory contained in HPC5 has higher latencies than the ones contained in HPC4.

Measurement of latency when transferring between memory allocated with cudaMallocHost and the GPU however shows that the use of DDR3 memory gives on average a 0.4 percent reduction in latency. The specific latencies for each measurement can be seen in Figure 4.6. This reduction varies across the different level of memory load. On zero background load on the memory, the DDR2 memory outperformed the DDR3 memory as expected since the DDR3 memory has a higher latency. When the background load on the memory increases the

DDR3 performs better. The average reduction on latencies with background memory load is 0.7 percent. This may be due to the increased bandwidth, which makes the DDR3 memory perform the memory operations faster and thus reducing the latency.

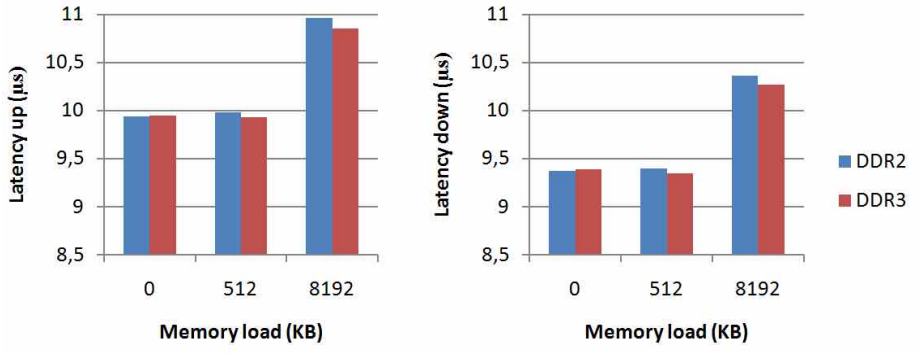

Figure 4.6: Latency to and from memory allocated on DDR2 and DDR3.

4.3.3 Bandwidth

The improvement DDR3 provides over DDR2 is the possibility for increased speed, which in turn give an increased bandwidth. However the intention with this test is not to check whether increased bandwidth gives increased bandwidth with use of CUDA. This test seeks to determine the difference in bandwidth with CUDA when using DDR3 compared to DDR2 with equal speed. The highest common speed achievable for both memory architectures was 1066MHz, which is slower than the default speed for the DDR3 modules. It is no ideal to alter the speed for the module, but due to the lack of a module with the correct speed, this is the alternative which gives the most correct test environment. The measurement of the bandwidth for DDR3 shows the same bandwidth as for the DDR2 with low or no memory load in the background. This is most likely because most of the memory load does not require the usage of the memory module due to caching. However, for high background memory load, the DDR3 memory outperforms the DDR2 memory by 4.4 percent when transferring to the GPU, and 7.6 percent when transferring from the GPU as seen in Figure 4.7. As the DDR2 and DDR3 in theory should have the same bandwidth, this difference may be due to the reduced latency of the DDR3 memory.

4.4 Memory speed

There is a direct relation between the speed of the memory and the bandwidth, and then since bandwidth is an important issue for GPGPU applications, this should be an important property of the computing system. Through these tests the impact on both bandwidth and latency shall be measured for various memory speeds, to determine the importance of high-speed memory for GPGU applications.

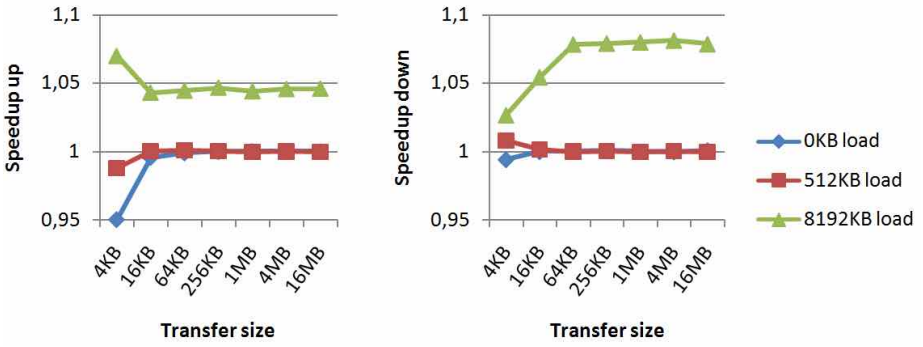

Figure 4.7: Speedup of bandwidth for DDR3 memory versus DDR2 memory

4.4.1 Setup

These tests are performed on HPC5 which has a DDR3 memory with a default speed of 1600MHz. The speeds 800MHz, 1066MHz, 1333MHz and 1600MHz are tested. The timings for the various speeds are determined by the system automatically to allow suitable timings for each memory speed. Adjusting the timings manually would require fine-tuning of the parameters and could lead to overclocking the memory which is unwanted in these tests.

4.4.2 Latency

The timings of a memory module affect latency by giving how many cycles it takes from the request for the data is made to the data is available at the pins of the memory. By increasing the memory's speed these cycles becomes shorter, and thus giving a lower latency. As one can see from Figure 4.8, this effect is noticeable. However there is not a one-to-one correlation between the speed and the latency, and this is due to the fact that higher memory speeds require higher timings to guaranty the correctness of the signal placed on the memory-pins, but still one can see an improvement in the latency for higher memory speeds.

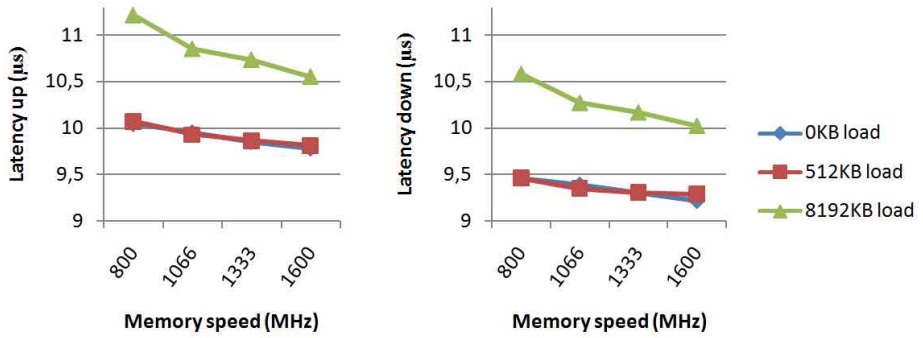

Figure 4.8: Latency up and down for increasing memory speeds.

4.4.3 Bandwidth

The bandwidth should, in the same way as the latency, be greatly related to the speed of the memory, since the bandwidth is related to the timings of the memory module, and thus also related to the speed. However, as with the latency, the bandwidth is not expected to have a one-to-one relationship due to the increase in timings for higher memory speeds. The measured bandwidths are shown in Figure 4.9, where the bandwidth is the average over the various transfer sizes. As one can see from the graphs the speed of the memory affects the bandwidth when transferring from the GPU, but the same effect is not visible for transfers to the GPU. This is most likely due to the fact that the memory is not the limiting factor in the transfer.

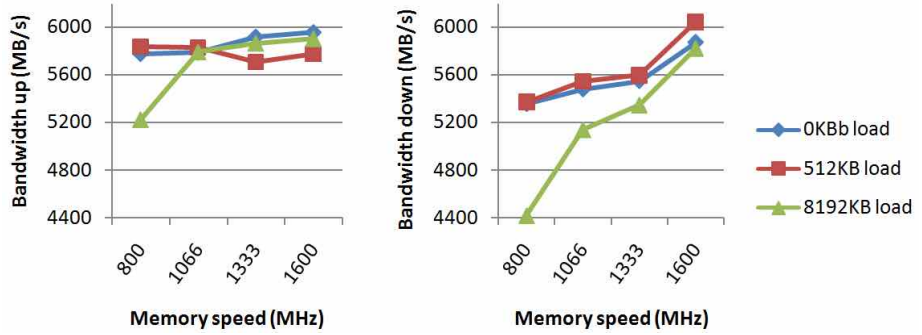

Figure 4.9: Average bandwidth up and down for increasing memory speeds.

4.5 Memory size

When configuring a computer system, one usually has the possibility to insert up to four memory modules to achieve a maximum size of the memory. These memory modules share two buses giving the situation where two and two modules share a bus if all four slots are occupied. Given this limitation, using more memory modules than necessary for the task to compute could affect the performanceboth in a positive and negative direction, and this test will try to shed some light on the effects of inserting all four memory modules compared to just using two.

4.5.1 Setup

The system used in this test is HPC5 with 4 and 8GB of DDR3 RAM. For the 4GB test two OCZ DDR3 4 GB Platinum EB XTC Dual Channel memory modules are used. When testing with 8GB, two additional modules of the same type are inserted.

4.5.2 Latency

The effect this alteration in memory configurations should have on the latency should if any be due to the fact that the bus to the memory is now shared

between two memory modules. Since none of the tests exceed the available storage on the memory modules, the memory size itself should have no affect on the performance. As seen in Figure 4.10, the latency is unaffected at no memory load, but at higher load the latency improves slightly for the 8GB configuration. However, if this effect is due to CUDA utilizing the memory better, or Ubuntu allocating the memory in a different way is difficult to determine based on this simple test.

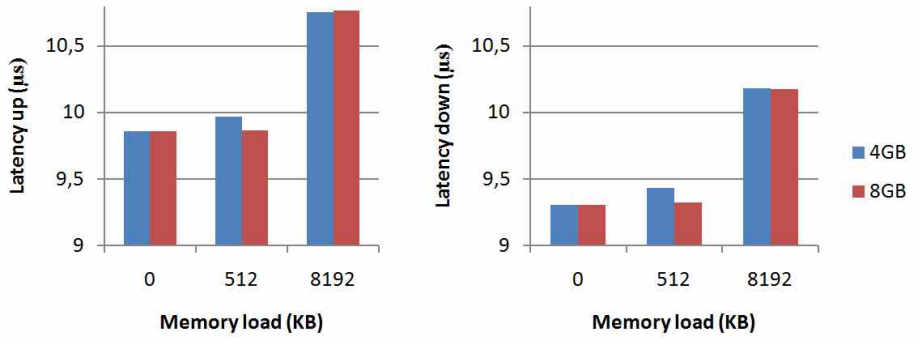

Figure 4.10: Latency up and down for memory size of 4GB and 8GB.

4.5.3 Bandwidth

As for the latency, the bandwidth may be affected both ways from the increase in memory size, and the results show the same. As one can see from Figure 4.11, there are cases which favors both the 4GB and 8GB configuration, and based on these results it is difficult to determine wetter the extra memory had a positive or negative effect. However, added memory may benefits other parts of the applications, so the added memory may not be a wasted investment after all.

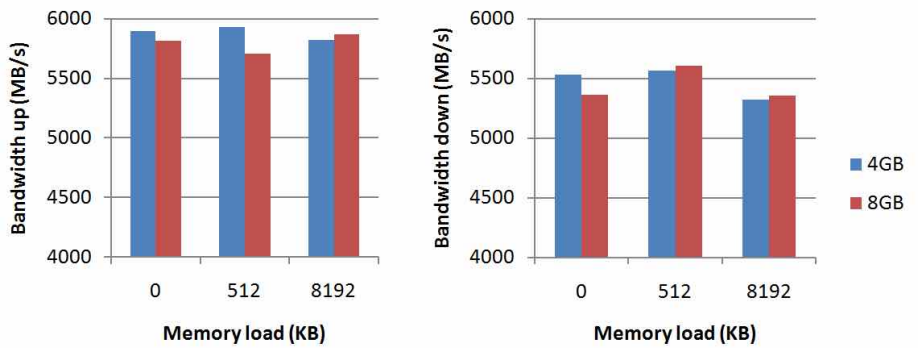

Figure 4.11: Bandwidth up and down for memory size of 4GB and 8GB.

4.6 CPU Architecture

The two companies Intel and AMD each produce state-of-the-art processors for the same instruction-set, but these processors and their related chipsets are different in architecture. Since the transfer from the host-memory to the device-memory also includes the interference of CPU, this is a property of a GPGPU system to be looked upon. As the two processors require different chipsets, this test will not be an entirely correct view upon the performance of the processors itself, but will also contain any difference in the performance of the two chipsets.

4.6.1 Setup

This test is performed using the AMD based HPC2 and Intel based HPC3. The use of HPC3 instead of HPC4 is to eliminate the need to reduce the speed of the processor, and thus adding another uncertainty to the test. Due to a lower cache size on both the Intel and AMD processor, the performance is not measured for 512KB memory load. Other tests however show that the increase in memory load does not affect the performance noticeable before it exceeds the available cache, and this should therefore not reduce the value of this test. Since there is no DDR3 enabled AMD machine available for testing, this test is conducted on two machines with DDR2 memory to allow an impartial comparison.

4.6.2 Latency

Since the transfer between the host-memory and device-memory is conducted using DMA, one would assume that the transfer is not to a great extent affected by the architecture of the CPU. This however proves to be incorrect as can be seen in Figure 4.13. The latency on the Intel-based system is half of the one measured on the AMD-based system. An important difference for between the AMD and Intel processors is the use of integrated memory controller for the AMD processor. This makes the data transfer pass through the processor on its way to the GPU, thus increasing the length of the wires as shown in Figure 4.12, and possibly reducing the performance. The benefit of having an on-board memory controller is the increased performance for memory operations performed by the CPU, which shows in Figure 15 where one can see the latency increases for the Intel processors and decreases for the AMD processor when the memory load on the system increases.

4.6.3 Bandwidth

The bandwidth is in the same way as the latency affected by the increased length as discussed in 4.6.2. This does reduce the bandwidth for the AMD processor versus the Intel processor. What is worth noticing is the reduction for the AMD processor when there is heavy memory load as can be seen in Figure 4.14. As opposed to the Intel processor, the AMD processor experiences a performance hit in this case.

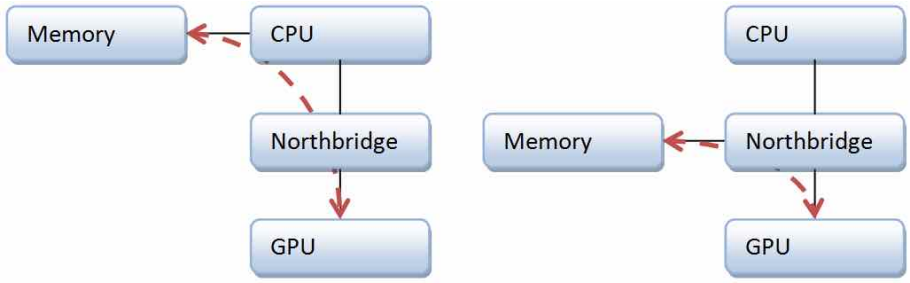

Figure 4.12: Paths for data transfer between memory and GPU on AMD (to the left) and Intel (to the right) processors.

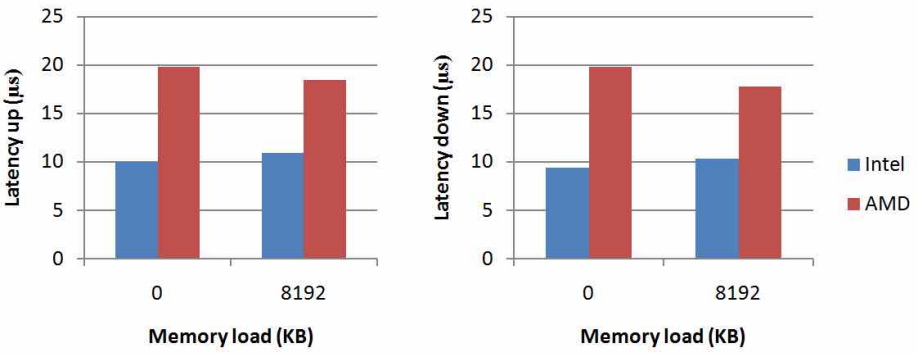

Figure 4.13: Latency up and down for Intel and AMD processors.

4.7 CPU Speed

Due to the use of DMA when transferring between host and device-memory, the speed of the processor should not have an impact on the bandwidth of the transfer. However, the speed may contribute to the latency, and this will be shown in Section 4.7.2.

4.7.1 Setup

The machine used in this test is HPC5, which has a Intel Core 2 Quad Q9550 processor with a speed of 2.83 GHz. To test the various speeds the multiplier of the processor were adjusted between 6 and 9.5. This gave the following speeds to test 2.00GHz, 2.33GHz, 2.50GHz, 2.67GHz and 2.83GHz. The computer did not manage to stabilize at 2.17GHz, and is thus not included in the test.

4.7.2 Latency

Even though the transfers between host and device-memory are conducted using DMA, there are still involvement from the CPU in terms of initialization and thus can the processors speed have an effect on the latency. However this

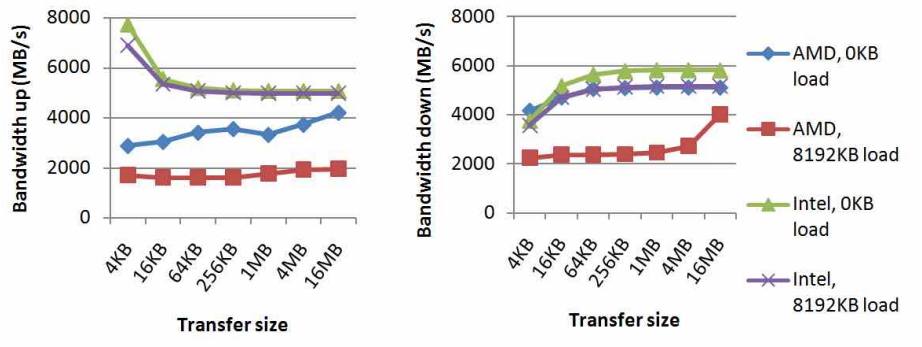

Figure 4.14: Bandwidth up and down for Intel and AMD processors.

improvement should not be very significant due to the low involvement of the CPU. This is also shown by the measurements presented in Figure 4.15, where one can see that a 40 percent increase in processor speed only yields a 3 percent reduction of latency in the best case.

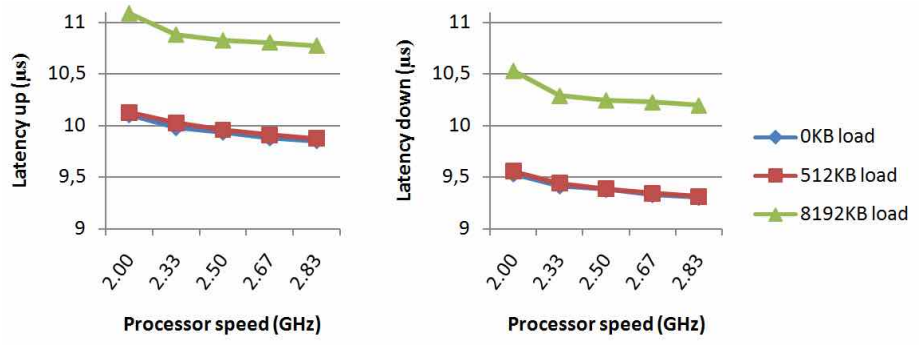

Figure 4.15: Latency up and down for various processor speeds.

4.7.3 Bandwidth

The bandwidth for the transfer between host and device-memory should however not be affected by the CPU speed since as mention earlier, the transfer is conducted using DMA. This implies that once the transfer is initialized, the CPU has no more involvement in the transfer, and should in theory have no effect on the transfer. Having measured the bandwidth for various memory loads under different CPU-speeds and different transfer sizes, it is evident that the speed of the processor does not affect the bandwidth. The results can be viewed in Figure 4.16 where the averages of the bandwidth for different memory loads are displayed. This generalization is done since there were no alterations in the ratio between the performances under different memory loads for the different CPU speeds.

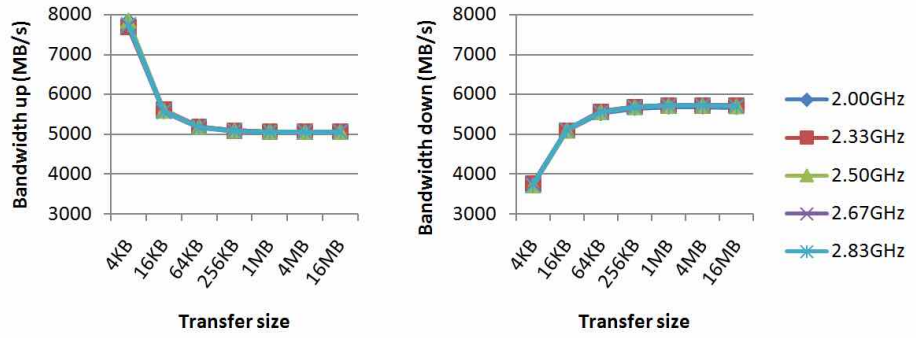

Figure 4.16: Average bandwidth up and down for various processor speeds.

4.8 Chipsets

The chipsets on the motherboard is the infrastructure of any computer. It contains the logic which processes requests for use of the bus and access to memory. This makes the implementation of the chipset an important property with respect to transfers between host and device memory.

4.8.1 Setup

In this test the two machines HPC5 and HPC6 were used to measure the differences between the Intel X48 chipset located In HPC5 and the NVIDIA nForce 790i chipset which is found in HPC6. Besides the difference in chipset these two systems are similar.

4.8.2 Latency

As part of latency, initialization of transfer in hardware is a vital part. The GPU has to request for memory access and use of the bus. This including the actual latency on the signal transmitted over the bus adds up to a significant factor in the latency. The PCI Express bus is also subject to differences on the various chipsets, in terms of transfer speeds and initialization latencies. As can be seen in Figure 4.17, the difference in latency is noticeable, and measuring the NVIDIA nForce 790i chipsets to have a much lower latency.

4.8.3 Bandwidth

As well as with the latency, bandwidth is also subject to influence by the implementation details of the chipset in terms of speed of the PCI express bus and load-handling. The load handling capabilities comes into play when there are several simultaneous requests for data located in memory and the chipset has to prioritize them. This also occurs for the busses on the motherboard, including the PCI Express bus. It is then up to the vendor to device algorithms which decides which request should have priority and how the resources should be timeshared. One noticeable result from this test is how the NVIDIA chipset suffers a performance penalty when there is heavy background memory

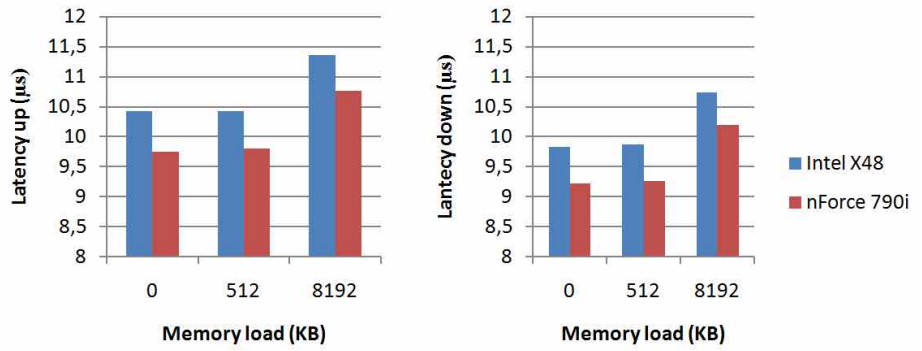

Figure 4.17: Latency up and down for Intel X48 and NVIDIA nForce 790i chipsets.

load, while the Intel chipset only has a minor reduction in bandwidth. Besides this effect, the NVIDIA chipset outperforms its counterpart from Intel, but the performance hit is worth having in mind when deciding between the various chipsets.

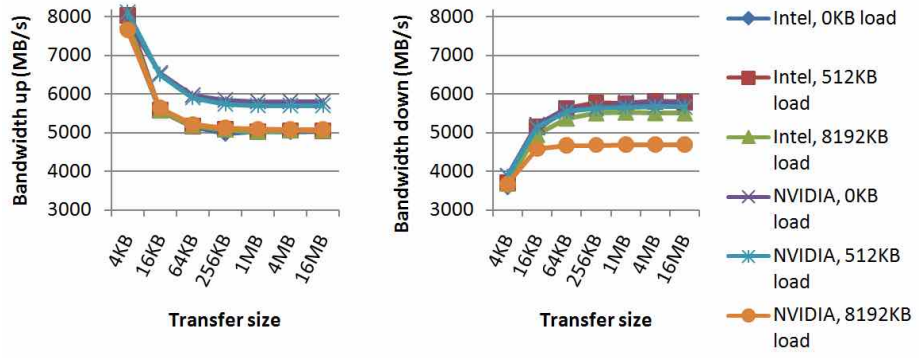

Figure 4.18: Average bandwidth up and down for Intel X48 and NVIDIA nForce 790i chipsets.

4.9 PCI Express bus clock frequency

Once the memory controller has allowed the transfer between the host-memory and device-memory, the transfer is conducted over the PCI-express bus. As one of many possible bottlenecks limiting the speed of the transfer, the PCI-express bus is one place where a performance gain could be achieved using overclocking should it prove to be the bottleneck. Overclocking however is not an exact science, and could prove damaging for the hardware, so this test will not overclock beyond what is possible by just changing the PCI-express bus clock frequency.

4.9.1 Setup

For this test, the HPC5 machine was used, due to the ability to adjust the clock frequency of the PCI Express bus. It was possible to achieve a stable system using clock frequencies ranging from 100 MHz to 115 MHz, so this test will measure the performance within this range for increases of 5 MHz for each step.

4.9.2 Latency

It is expected that the PCI Express bus would be one of the bottlenecks limiting the transfer to the GPU. This assumption proves correctly when one sees the measurements shown in Figure 4.19. Here one can see that a 15 percent increase in bandwidth yields a 7 percent reduction in latency. Most of this gain is achieved through the first 5 percent increase in clock frequency which yields a 5 percent reduction in latency. These results clearly show that the PCI Express bus is a key bottleneck for the data transfers between host and device memory.

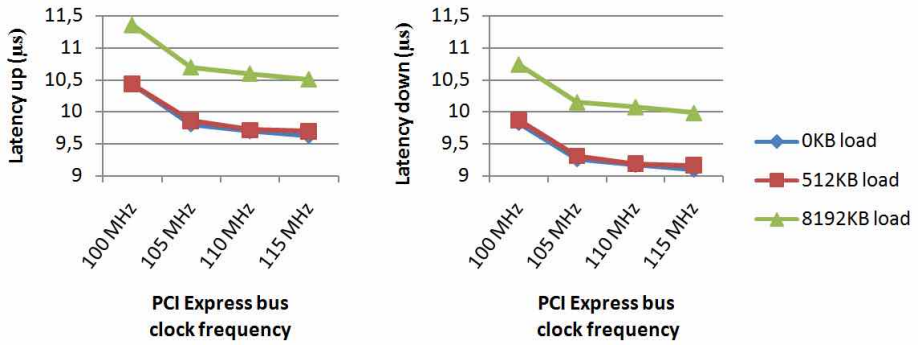

Figure 4.19: Latency up and down for various PCI Express bus clock frequencies.

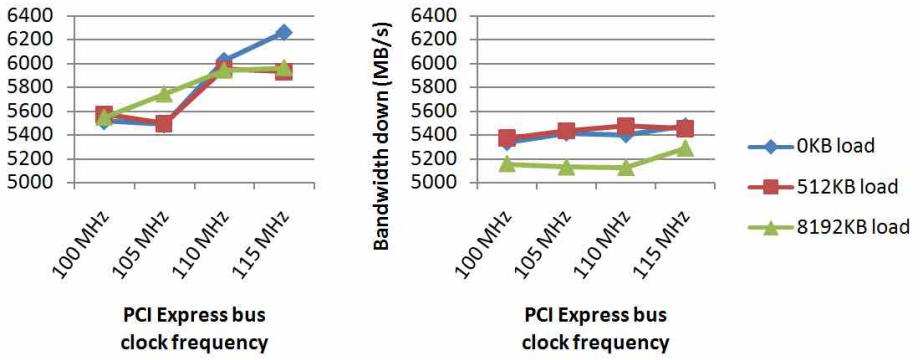

Figure 4.20: Bandwidth up and down for various PCI Express bus clock frequencies.

4.9.3 Bandwidth

After achieving such significant reductions in latency for adjustment of the clock frequency of the PCI Express bus, one would assume there to be equal improvements in bandwidth. If one look at the bandwidth for transfers from host to device-memory one improvements ranging from 6 to 13 percent as seen in Figure 4.20. For transfers the opposite way there are only minor improvements which suggests that the PCI Express bus is not the only bottleneck limiting the transfers in that direction.

Chapter 5

Conclusion and Future Works

This chapter summarizes the findings in this report and discusses the validity of the model defined in Chapter 3. It also gives a list of suggestions for future work, as well as gives some final thoughts on hardware configurations for GPGPU systems.

5.1 Our CPU-GPU System Model versus Test Results

Our model of general GPU - CPU systems, is a fairly simple one, but by modelling the GPU as a black box, allowed the examination of individual system properties such as memory bandwidth and latency. Expected property behavior of our model were provided and compared with actual benchmarks to verify the correctness of our assumptions.

Our model complied well with the results. regarding memory and memory lated properties such as size, clock frequency and memory architecture. The newer DDR3 architecture seems to improve the latency and bandwidth in the high-load cases, ragrdless of memory frequency. The clock frequency affects both latency and bandwidth greatly. However, the amount of available memory gave inconclusive results with regards to latency and bandwidth.

The properties regarding the CPU should not have too much influence on the performance due to the use of DMA. However, for the test of architecture and its influence, the tests gave huge influence. This test tries to put the Intel and AMD processors up against each other in a test designed to allow them to be compared on equal terms. However, due to the massive differences in both latency and bandwidth, the test must be further refined to give a conclusive result. In terms of clock frequency, the CPU performed exactly as predicted in the model, having no influence over the bandwidth, and gave a small reduction in latency.

The chipset proved to give the expected influence over the latency and the bandwidth. However, the two Intel and NVIDIA chipsets we tested probably had different design goals, given that one is mainly a processor manufacturer

whereas the other is primarily a graphics cards manufacturer. Our point was not to show that one is better than the other, but that one needs to pay attention to the chipset and ones goal when building a CPU - GPU system for HPC. However, performance for the two chipsets supports the model in the sense that the chipset are a key component in the GPU system.

One of our more clear results, was the influence of the PCI Express bus. Both the latency and bandwidth showed good improvements when overclocking the bus. Overclocking is not an exact science, and there might be additional effects from this alteration, but the results show an improvement.

5.2 Future works

While this report covered several CPU- GPU properties, many still remain. As mentioned earlier, there are possibilities for many dependent properties which have not been tested here due to the sheer number of such combinations. Many of these properties might also behave differently for other combinations of hardware. Therefore these tests must be further verified for other architectures and combinations of hardware.

Just a month prior to the completion of this report, the specifications for OpenCL where released. OpenCL is a language extentions designed to provide the HPC community with a language capable of addressing all forms of processing platforms ranging from multi-core CPU, Cell BE and GPUs. It also aims to be vendor independent which would allow it to utilize both NVIDIA and ATI graphics cards. The introduction of OpenCL would allow more tests to be developed and an easier comparing between NVIDIA and ATI, as well as GPGPU versus Cell BE for instance. However, the late release of OpenCL and its lack of support yet for graphics cards leaves this as a task for future work. Here one could draw upon the findings in this report and that of one of my fellow students who is assessing the usability of OpenCL compared to Cuda.

5.3 Final thoughts

While this report has given the reader an indication of which components of the GPGPU system which affects the performance and which who does not, the reader must not forget that the system might be used for more than just pure GPGPU. Even if the system is intended for GPGPU, the graphics cards are not optimal for all types of tasks, and a GPGPU application might perform faster by ofloading some of the calculations to the CPU. With this in mind, the *Bigger is better* statement still holds for purchases of new equipment, but this report might have shifted the prioritization among the components.

Appendix A

Hardware specifications

Table A.1: HPC2 Specifications.

Processor		
	AMD Phenom X4 9850 Black Edition	
	Clock frequency	2.50 GHz
	L2 Cache	2 MB
	L3 Cache	2 MB
Motherboard		
	Gigabyte GA-MA790FX-DQ6 AMD790FX DDR2	
	Chipset	AMD 790FX
Memory		
	Mushkin DDR2 4096MB REDLINE extreme performance	
	Frequency	1066 MHz
	Size	2x 2048 MB

Table A.2: HPC3 Specifications.

Processor		
	Intel Core 2 Quad Q9300, 64 bit	
	Clock frequency	2.50 GHz
	L2 Cache	6 MB
	Bus speed	1333 MHz
Motherboard		
	ASUS Rampage Formula X48	
	Chipset	Intel X48 Express
Memory		
	Mushkin DDR2 4096MB REDLINE extreme performance	
	Frequency	1066 MHz
	Size	2x 2048 MB

Table A.3: HPC4 Specifications.

Processor		
	Intel Core 2 Quad Q9550, 64 bit	
	Clock frequency	2.83 GHz
	L2 Cache	12 MB
	Bus speed	1333 MHz
Motherboard		
	ASUS Rampage Formula X48	
	Chipset	Intel X48 Express
Memory		
	Mushkin DDR2 4096MB REDLINE extreme performance	
	Frequency	1066 MHz
	Size	2x 2048 MB

Table A.4: HPC5 Specifications.

Processor		
	Intel Core 2 Quad Q9550, 64 bit	
	Clock frequency	2.83 GHz
	L2 Cache	12 MB
	Bus speed	1333 MHz
Motherboard		
	ASUS P5E3 Premium	
	Chipset	Intel X48 Express
Memory		
	OCZ DDR3 4 GB Platinum EB XTC Dual Channel	
	Frequency	1600 MHz
	Size	2x 2048 MB

Table A.5: HPC6 Specifications.

Processor		
	Intel Core 2 Quad Q9550, 64 bit	
	Clock frequency	2.83 GHz
	L2 Cache	12 MB
	Bus speed	1333 MHz
Motherboard		
	EVGA nForce 790i Ultra SLI	
	Chipset	NVIDIA nForce 790i Ultra SLI
Memory		
	OCZ DDR3 4 GB Platinum EB XTC Dual Channel	
	Frequency	1600 MHz
	Size	2x 2048 MB

Appendix B

Test Program

Here follows the code used in the testing software

B.1 main.cu

```
1  /*
2  The MIT License
3
4  Copyright (c) 2008 Rune Johan Hovland
5
6  Permission is hereby granted, free of charge, to any
       person obtaining a copy of this software and
       associated documentation files (the "Software"), to
       deal in the Software without restriction, including
       without limitation the rights to use, copy, modify,
       merge, publish, distribute, sublicense, and/or sell
       copies of the Software, and to permit persons to whom
       the Software is furnished to do so, subject to the
       following conditions:
7
8  The above copyright notice and this permission notice
       shall be included in all copies or substantial
       portions of the Software.
9
10 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
       KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
       TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
       PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
       SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
       ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
       ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
       OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
       OR OTHER DEALINGS IN THE SOFTWARE.
11 */
12
```

```
13 #include <stdlib.h>
14 #include <stdio.h>
15 #include <string.h>
16 #include <math.h>
17 #include <cutil.h>
18 #include <common.cu>
19
20 #include "measureTransferMalloc.cu"
21 #include "measureTransferCudaMallocHost.cu"
22
23 int
24 main( int argc, char** argv)
25 {
26   if(argc > 0){
27     int core = atoi(argv[1]);
28
29     // measure bandwidth and latency  up and down with
       malloc
30     measureTransferMalloc(core);
31
32     // measure bandwidth up and down with cudaMalloc
33     measureTransferCudaMallocHost(core);
34   }
35 }
```

B.2 common.cu

```
1  /*
2  The MIT License
3
4  Copyright (c) 2008 Rune Johan Hovland
5
6  Permission is hereby granted, free of charge, to any
       person obtaining a copy of this software and
       associated documentation files (the "Software"), to
       deal in the Software without restriction, including
       without limitation the rights to use, copy, modify,
       merge, publish, distribute, sublicense, and/or sell
       copies of the Software, and to permit persons to whom
       the Software is furnished to do so, subject to the
       following conditions:
7
8  The above copyright notice and this permission notice
       shall be included in all copies or substantial
       portions of the Software.
9
10 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
       KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
       TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
       PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
```

```
    SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
    ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
    ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
    OR OTHER DEALINGS IN THE SOFTWARE.
11  */
12
13  #ifndef _common_
14  #define _common_
15
16  #include <stdio.h>
17  #include <stdlib.h>
18  #include <sys/time.h>
19  #include <cmath>
20  #include <cuda.h>
21
22  #define MB (1024 * 1024)
23  #define ITERATIONS_FEW 200
24  #define ITERATIONS_MANY 10000
25  #define LATENCY_LIMIT 4096
26
27  #define startTiming gettimeofday(&startTime, NULL);
28  #define endTiming gettimeofday(&endTime, NULL);
        measuredTime = timediff(endTime, startTime);
29  #define setDummy dummyMeasurement = true;
30  #define checkDummy if(dummyMeasurement){ dummyMeasurement
        = false; i-=2; continue;}
31  #define forLoop for(int i = 8; i < 24; i+=2){ int
        datasize = 1 << i; iterations = ITERATIONS_FEW * pow
        (1.32237785818103,22-i);
32  #define print(result, datasize, iterations) printf("%.15f
        \t%d\t%d\n",result,datasize,iterations);fflush(stdout)
        ;
33
34  int iterations;
35  struct timeval startTime, endTime;
36  double measuredTime;
37  bool dummyMeasurement;
38
39  // Calculate difference in time between two times.
40  double
41  timediff(timeval end, timeval start)
42  {
43    double ts = start.tv_sec + (start.tv_usec / 1000000.0);
44    double te = end.tv_sec + (end.tv_usec / 1000000.0);
45    return te - ts;
46  }
47
48  // Tests the time needed to perform the given
        transfertype and data size.
```

```
49 double
50 testTransfer(int datasize, int testcount, float* from,
       float* to, enum cudaMemcpyKind type)
51 {
52   // Allocate and create testdata
53   size_t size = datasize * sizeof(float);
54
55   // Allocate local memory and fill with data.
56   if(type == cudaMemcpyHostToDevice || type ==
         cudaMemcpyHostToHost)
57   {
58     for(int i = 0; i < datasize; i++)
59     {
60       from[i] = (float) i;
61     }
62   }
63
64   // Allocate memory on device and fill with memory.
65   else if(type == cudaMemcpyDeviceToHost || type ==
         cudaMemcpyDeviceToDevice)
66   {
67     float* data = (float*)malloc(datasize*sizeof(float));
68     for(int i = 0; i < datasize; i++)
69     {
70       data[i] = (float) i;
71     }
72     cudaMemcpy(data, from, size, cudaMemcpyHostToDevice);
73     free(data);
74   }
75
76   // Initialize and start timer
77   startTiming
78   for ( int i = 0; i < testcount; i++ )
79   {
80     cudaMemcpy(to, from, size, type);
81   }
82   endTiming
83
84   // return measured time
85   return measuredTime / ((double)testcount);
86 }
87
88 #endif
```

B.3 measureTransferMalloc.cu

```
1 /*
2 The MIT License
3
4 Copyright (c) 2008 Rune Johan Hovland
```

```
5
6  Permission is hereby granted, free of charge, to any
       person obtaining a copy of this software and
       associated documentation files (the "Software"), to
       deal in the Software without restriction, including
       without limitation the rights to use, copy, modify,
       merge, publish, distribute, sublicense, and/or sell
       copies of the Software, and to permit persons to whom
       the Software is furnished to do so, subject to the
       following conditions:
7
8  The above copyright notice and this permission notice
       shall be included in all copies or substantial
       portions of the Software.
9
10 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
       KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
       TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
       PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
       SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
       ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
       ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
       OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
       OR OTHER DEALINGS IN THE SOFTWARE.
11 */
12
13 #ifndef _measureTransferMalloc_
14 #define _measureTransferMalloc_
15
16 #include "common.cu"
17
18 void
19 measureTransferMalloc(int core)
20 {
21   printf("\n");
22   printf("
       +------------------------------------------------------->\
       n");
23   printf("| Measuring transfers with malloc.\n");
24   printf("|    (core %d)\n",core);
25   printf("
       +------------------------------------------------------->\
       n");
26   printf("\n");
27
28   float *from, *to;
29
30   // Measure latency up
31   printf("Latency up\n");
32   double meanTsUp = 100000;
```

```
33
34    for(int i = 1; i <= LATENCY_LIMIT; i*=2)
35    {
36      // Allocate memory.
37      from = (float*)malloc(i*sizeof(float));
38      cudaMalloc((void**)&to, i*sizeof(float));
39
40      // Perform test.
41      double meanTsUpTmp = testTransfer(i, ITERATIONS_MANY,
            from, to, cudaMemcpyHostToDevice);
42
43      if(meanTsUpTmp < meanTsUp){
44        meanTsUp = meanTsUpTmp;
45      }
46
47      // Free memory.
48      free(from);
49      cudaFree(to);
50      print(meanTsUpTmp, i, ITERATIONS_MANY);
51    }
52    printf("\n");
53
54    // Measure latency down
55    printf("Latency down\n");
56    double meanTsDown = 100000;
57
58    for(int i = 1; i <= LATENCY_LIMIT; i*= 2)
59    {
60      // Allocate memory.
61      cudaMalloc((void**)&from, i*sizeof(float));
62      to = (float*)malloc(i*sizeof(float));
63
64      // Perform test.
65      double meanTsDownTmp = testTransfer(i,
            ITERATIONS_MANY, from, to, cudaMemcpyDeviceToHost)
            ;
66
67      if(meanTsDownTmp < meanTsDown){
68        meanTsDown = meanTsDownTmp;
69      }
70
71      // Free memory.
72      cudaFree(from);
73      free(to);
74      print(meanTsDownTmp, i, ITERATIONS_MANY);
75    }
76    printf("\n");
77
78    // Measure bandwidth up
79    printf("Bandwidth up\n");
```

```
80    setDummy
81    forLoop
82      // Allocate memory.
83      from = (float*)malloc(datasize*sizeof(float));
84      cudaMalloc((void**)&to, datasize*sizeof(float));
85
86      // Perform test.
87      double betaTime = testTransfer(datasize, iterations,
            from, to, cudaMemcpyHostToDevice);
88
89      // Free memory.
90      free(from);
91      cudaFree(to);
92
93      // Calculate bandwidth.
94      double meanBeta = ( (betaTime - meanTsUp) / datasize
            ) * (MB / sizeof(float));
95      double bandwidth = 1 / meanBeta;
96
97      // Redo run if dummy-test.
98      checkDummy
99      print(bandwidth, datasize, iterations);
100   }
101   printf("\n");
102
103   // Measure bandwidth down
104   printf("Bandwidth down\n");
105   setDummy
106   forLoop
107     // Allocate memory.
108     cudaMalloc((void**)&from, datasize*sizeof(float));
109     to = (float*)malloc(datasize*sizeof(float));
110
111     // Perform test.
112     double betaTime = testTransfer(datasize, iterations,
            from, to, cudaMemcpyDeviceToHost);
113
114     // Free memory.
115     cudaFree(from);
116     free(to);
117
118     // Calculate bandwidth.
119     double meanBeta = ( (betaTime - meanTsDown) /
            datasize ) * (MB / sizeof(float));
120     double bandwidth = 1 / meanBeta;
121
122     // Redo run if dummy-test.
123     checkDummy
124     print(bandwidth, datasize, iterations);
125   }
```

```
126     printf("\n");
127 }
128
129 #endif
```

B.4 measureTransferCudaMallocHost.cu

```
1  /*
2  The MIT License
3
4  Copyright (c) 2008 Rune Johan Hovland
5
6  Permission is hereby granted, free of charge, to any
       person obtaining a copy of this software and
       associated documentation files (the "Software"), to
       deal in the Software without restriction, including
       without limitation the rights to use, copy, modify,
       merge, publish, distribute, sublicense, and/or sell
       copies of the Software, and to permit persons to whom
       the Software is furnished to do so, subject to the
       following conditions:
7
8  The above copyright notice and this permission notice
       shall be included in all copies or substantial
       portions of the Software.
9
10 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
        KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
       TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
       PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
       SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
       ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
       ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
       OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
       OR OTHER DEALINGS IN THE SOFTWARE.
11 */
12
13 #ifndef _measureTransferCudaMallocHost_
14 #define _meausreTransferCudaMallocHost_
15
16 void
17 measureTransferCudaMallocHost(int core)
18 {
19   printf("\n");
20   printf("
       +------------------------------------------------------->\
       n");
21   printf("| Measuring transfers with cudaMallocHost.\n");
22   printf("|   (core %d)\n",core);
```

```
23    printf("
          +------------------------------------------------------------>\
          n");
24    printf("\n");
25
26    float *from, *to;
27
28    // Measure latency up
29    printf("Latency up\n");
30    double meanTsUp = 100000;
31
32    for(int i = 1; i <= LATENCY_LIMIT; i*=2)
33    {
34      // Allocate memory locations
35      cudaMallocHost((void**)&from, i*sizeof(float));
36      cudaMalloc((void**)&to, i*sizeof(float));
37
38      // Perform test.
39      double meanTsUpTmp = testTransfer(i, ITERATIONS_MANY,
            from, to, cudaMemcpyHostToDevice);
40
41      if(meanTsUpTmp < meanTsUp){
42        meanTsUp = meanTsUpTmp;
43      }
44
45      // Free memory.
46      cudaFreeHost(from);
47      cudaFree(to);
48      print(meanTsUpTmp, i, ITERATIONS_MANY);
49    }
50    printf("\n");
51
52    // Measure latency down
53    printf("Latency down\n");
54    double meanTsDown = 100000;
55
56    for(int i = 1; i <= LATENCY_LIMIT; i*= 2)
57    {
58      // Allocate memory locations
59      cudaMalloc((void**)&from, i*sizeof(float));
60      cudaMallocHost((void**)&to, i*sizeof(float));
61
62      // Perform test.
63      double meanTsDownTmp = testTransfer(i,
            ITERATIONS_MANY, from, to, cudaMemcpyDeviceToHost)
            ;
64
65      if(meanTsDownTmp < meanTsDown){
66        meanTsDown = meanTsDownTmp;
67      }
```

```
68
69      // Free memory.
70      cudaFree(from);
71      cudaFreeHost(to);
72      print(meanTsDownTmp, i, ITERATIONS_MANY);
73    }
74    printf("\n");
75
76    // Measure bandwidth up
77    printf("Bandwidth up\n");
78    setDummy
79    forLoop
80      // Allocate memory.
81      cudaMallocHost((void**)&from, datasize*sizeof(float))
          ;
82      cudaMalloc((void**)&to, datasize*sizeof(float));
83
84      // Perform test.
85      double betaTime = testTransfer(datasize, iterations,
          from, to, cudaMemcpyHostToDevice);
86
87      // Free memory.
88      cudaFreeHost(from);
89      cudaFree(to);
90
91      // Calculate bandwidth.
92      double meanBeta = ( (betaTime - meanTsUp) / datasize
          ) * (MB / sizeof(float));
93      double bandwidth = 1 / meanBeta;
94
95      // Redo run if dummy-test.
96      checkDummy
97      print(bandwidth, datasize, iterations);
98    }
99    printf("\n");
100
101   // Measure bandwidth down
102   printf("Bandwidth down\n");
103   setDummy
104   forLoop
105     // Allocate memory.
106     cudaMalloc((void**)&from, datasize*sizeof(float));
107     cudaMallocHost((void**)&to, datasize*sizeof(float));
108
109     // Perform test.
110     double betaTime = testTransfer(datasize, iterations,
          from, to, cudaMemcpyDeviceToHost);
111
112     // Free memory.
113     cudaFree(from);
```

```
114       cudaFreeHost(to);
115
116       // Calculate bandwidth.
117       double meanBeta = ( (betaTime - meanTsDown) /
              datasize )  * (MB / sizeof(float));
118       double bandwidth = 1 / meanBeta;
119
120       // Redo run if dummy-test.
121       checkDummy
122       print(bandwidth, datasize, iterations);
123     }
124     printf("\n");
125   }
126
127   #endif
```

Appendix C

Load-Simulator Software

Here follows the code used in the load simulator.

C.1 loadSimulator.cu

```
1  /*
2  The MIT License
3
4  Copyright (c) 2008 Rune Johan Hovland
5
6  Permission is hereby granted, free of charge, to any
       personobtaining a copy of this software and associated
        documentation files (the "Software"), to deal in the
       Software without restriction, including without
       limitation the rights to use, copy, modify, merge,
       publish, distribute, sublicense, and/or sell copies of
        the Software, and to permit persons to whom the
       Software is furnished to do so, subject to the
       following conditions:
7
8  The above copyright notice and this permission notice
       shall be included in all copies or substantial
       portions of the Software.
9
10 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
        KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
       TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
       PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
       SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
       ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
       ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
       OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
       OR OTHER DEALINGS IN THE SOFTWARE.
11 */
12
```

```
13 #include "stdio.h"
14 #include <cstdlib>
15
16 int
17 main( int argc, char** argv)
18 {
19   // Size of allocation.
20   int size = 2*1024*1024;
21
22   // change size if specified.
23   if (argc > 1)
24   {
25     size = atoi(argv[1]) * 1024;
26   }
27
28   // Allocate memory to copy between.
29   float* a = (float*)malloc(size);
30   float* b = (float*)malloc(size);
31   float* c = (float*)malloc(size);
32   float* d = (float*)malloc(size);
33
34   // Fill memory with data.
35   for (int i = 0; i < size / 4; i++)
36   {
37     a[i] = i;
38   }
39
40   // Inform user that loadsimulator has startet.
41   printf("Load simulation started (%db)\n",size);
42
43   // Copy memory between various locations forever.
44   while (true)
45   {
46     memcpy(b,a,size);
47     memcpy(c,b,size);
48     memcpy(d,c,size);
49     memcpy(a,d,size);
50   }
51 }
```

Bibliography

[1] Advanced Micro Devices (AMD), "AMD 790FX Chipset Specifications," [Cited: December 7, 2008.]. [Online]. Available: http://www.amd.com/us-en/0,,3715_15337_15354_15358,00.html#top

[2] ——, "ATI Radeon HD 4800 Series - GPU Specifications," [Cited: October 22, 2008.]. [Online]. Available: http://ati.amd.com/products/radeonhd4800/specs.html

[3] ——, "Key Architectural Features of AMD Phenom X4 Quad-Core Processors," [Cited: December 7, 2008.]. [Online]. Available: http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331_15332%5E15334,00.html

[4] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yellick., "The Landscape of Parallel Computing Research: A view from Berkeley," Electrical Engineering and Computer Sciences, University of California at Berkeley, Tech. Rep., 2006.

[5] A. V. Bhatt, "Creating a PCI Express Interconnect," [Cited: December 7, 2008.]. [Online]. Available: http://www.pcisig.com/specifications/pciexpress/resources/PCI_Express%_White_Paper.pdf

[6] D. Blythe, "Rise of the Graphics Processor," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 761–777, May 2008.

[7] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, October 2008.

[8] O. Coles, "DDR3 RAM: System Memory Technology Explained," May 10, 2008, [Cited: November 13, 2008.]. [Online]. Available: http://benchmarkreviews.com/index.php?option=com_content&task=view\&id=174\&Itemid=1

[9] A. C. Elster, "HPC Lab at IDI/NTNU," [Cited: December 10, 2008.]. [Online]. Available: http://www.idi.ntnu.no/~elster/hpc-lab/

[10] A. K. B. Ernes, "NTNU vil lage de kuleste pc-ene," October 29, 2008, [Cited: November 29, 2008.]. [Online]. Available: http://digi.no/php/art.php?id=792051

[11] W. Fink, "DDR3 vs. DDR2," May 15, 2007, [Cited: November 14, 2008.]. [Online]. Available: http://www.anandtech.com/memory/showdoc.aspx?i=2989

[12] M. Harris, "Mapping Computational Concepts to GPUs," in *GPU Gems 2*, M. Pharr and R. Fernando, Eds. Addison-Wesley Professional, 2005, pp. 493–508.

[13] R. W. Hockney, "The communication challenge for MPP: Intel Paragon and Meiko CS-2," *Parallel Computing*, vol. 20, no. 3, pp. 389–398, March 1994.

[14] Intel Corporation, "Intel Core 2 Quad Processor, Product Brief," [Cited: December 7, 2008.]. [Online]. Available: http://www.intel.com/products/processor/core2quad/prod_brief.pdf

[15] ——, "Intel X48 Express Chipset, Product Brief," 2008, [Cited: December 7, 2008.]. [Online]. Available: http://www.intel.com/Assets/PDF/prodbrief/319646.pdf

[16] A. Klckner, "PyCuda," [Cited: October 22, 2008.]. [Online]. Available: http://mathema.tician.de/software/pycuda

[17] J. Kullberg, "Tight Timings vs High Clock Frequencies," March 31, 2006, [Cited: December 7, 2008.]. [Online]. Available: http://www.tomshardware.com/reviews/tight-timings-high-clock-frequencies,1236.html

[18] NVIDIA Corporation, "GeForce 9800GTX Specifications," [Cited: December 7, 2008.]. [Online]. Available: http://www.nvidia.com/object/product_geforce_9800_gtx_us.html

[19] ——, "GeForce 9800GX2 Specifications," [Cited: December 7, 2008.]. [Online]. Available: http://www.nvidia.com/object/product_geforce_9800_gx2_us.html

[20] ——, "GeForce GTX280 Specifications," [Cited: December 7, 2008.]. [Online]. Available: http://www.nvidia.com/object/product_geforce_gtx_280_us.html

[21] ——, "NVIDIA CUDA Compute Unified Device Architecture, Programming Guide," [Cited: September 26, 2008.]. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf

[22] ——, "Quadro FX5800 Specifications," [Cited: December 7, 2008.]. [Online]. Available: http://www.nvidia.com/object/product_quadro_fx_5800_us.html

[23] ——, "Tesla c1060 Specifications," [Cited: December 7, 2008.]. [Online]. Available: http://www.nvidia.com/object/product_tesla_c1060_us.html

[24] ——, "NVIDIA GeForce 8800 GPU Architecture Overview," November 2006, [Cited: October 22, 2008.]. [Online]. Available: http://www.nvidia.com/object/IO_37100.html

[25] ——, "NVIDIA-based Motherboard Family for Intel," March 2008, [Cited: December 7, 2008.]. [Online]. Available: http://www.nvidia.com/docs/IO/35382/LC_NV_motherboards_INTEL_Mar08.pdf

[26] ——, "NVIDIA Tesla Makes Personal SuperComputing A Reality," November 19, 2008, [Cited: November 28, 2008]. [Online]. Available: http://www.nvidia.com/object/io_1227008280995.html

[27] nwilt, "Page-locked memory," February 1, 2008, [Cited: November 26, 2008.]. [Online]. Available: http://forums.nvidia.com/index.php?showtopic=58505\&st=0\&p=318592\&#entry318592

[28] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, May 2008.

[29] PCI Sig., "PCI Express 3.0 Frequently Asked Questions," [Cited: December 7, 2008.]. [Online]. Available: http://www.pcisig.com/news_room/faqs/PCI_Express_Gen3_faq_FINAL.pdf

[30] P. Schmid, "Ups and Downs: Memory Timings Put to the Test," January 19, 2004, [Cited: December 7, 2008.]. [Online]. Available: http://www.tomshardware.com/reviews/ups-downs,743.html