GENERAL-PURPOSE COMPUTATION USING GRAPHICAL PROCESSING UNITS

Adrian Salazar, Texas A&M-University-Corpus Christi
Faculty Advisor: Dr. Ahmed Mahdy, Texas A&M-University-Corpus Christi

ABSTRACT

Graphical Processing Units (GPU) are mainly developed to process intense graphical operations. This has been the purpose for many years; however recent advancements in the architecture of GPUs and how they process data have led to a breakthrough in how we are able to process information. General Purpose Computation using GPUs is a new method of processing information using a GPU and its many processors. While programming to the GPU is not straightforward, there are platforms that have been developed with this in mind. Using stream processing to handle data in parallel on the GPU, we are able to substantially increase application performance as compared to standard CPU performance.

INTRODUCTION

Graphical Processing Units (GPUs) have evolved substantially since first being introduced in the Atari™ Game console. They are no longer limited to rendering images, they are now able to perform millions of calculations per second. This capability in addition to their parallel architecture has enabled the GPU to become a new processing tool for General-Purpose Computation using GPUs. GPGPU is a new technology that seeks to use the many processors available to the GPU to perform scientific calculations, no longer limiting a GPU to perform image processing. While writing to the GPU is fairly difficult, new platforms have been developed to facilitate this process. Two of the most prominent platforms readily available are produced by NVIDIA® Corporation and Rapid Mind® Inc. Each platform gives developers full control over the GPU through the use of C and C++. In this manner, developers are able to develop and execute complex algorithms such as sorting, searching as well as applications that rely on large amount of data such as computer vision and audio processing. While not all applications are capable of being processed in parallel by a GPU, processing largely independent data has a positive effect in increasing performance time.

GPU EVOLUTION AND ARCHITECTURE

Graphical Processing Units have long been known for their speed when it comes to image rendering. Their primary role at the time was to use Direct Memory Access (DMA) to help reduce the host load time when writing to the television screen; this was done with ANTIC, an alpha-numeric television interface circuit [2]. The GPU was not restricted to only game systems; GPUs were also incorporated into Apple laser printers and Texas Instruments graphing calculators. Microprocessors had great success during the late ‘80s, that success inspired developers to create high-end GPUs using microprocessors. This new trend of developing GPUs further enabled their evolution; soon GPUs were able to render 3D images. The movement of Microsoft® Operating System from DOS command line to its new Graphical User Interface (GUI) based operating system had a major impact on the development of the GPU. Prior to Windows™, companies had no standard for developing GPUs. Windows™ allowed for

the development of a standard graphical interface. The implementation of a single programming interface allowed developers to focus on a single interface [4]. This enabled GPU to grow at an even faster rate than before. While advanced 3D rendering was not achieved till 2000, the GPU still enjoyed success in being a part of the development of Nintendo® N64™ and Sony® Play Station™ video game consoles [5].

NVIDIA® Corp. has established itself as a leader in the GPU industry, they have paved the road for other companies such as ATI® in developing the first GPU that is able to fully support 3D rendering without the need of a 3D accelerator. NVIDIA® NV10™ and NV20™ were the first GPUs to support 3D image rendering and shading [9]. ATI® followed in producing a GPU that is able to fully support looping and lengthy floating point operations [1]. ATI's R300™ model proved to be a big reason on the next big jump in the evolution of the GPU. Scientist and developers realized that with the new added capabilities, the GPU would be able to handle a wider array of data.

Architectural design has allowed for GPUs to evolve rapidly [9]. High-end GPUs were designed using microprocessors to incorporate a large amount of Arithmetic Logic Units (ALU.) While the ALUs have progressively decreased in size, their performance has progressively increased. With the smaller size ALUs, developers have been able to increase the number of ALUs available to the GPU. With more ALUs and the ability to process information in parallel data streams, GPUs are able to process data faster than its predecessors. With the added math precision power that NVIDIA® and ATI® have introduced coupled with the ability of the GPU to process data in parallel, the GPU has become the perfect tool to perform general scientific calculations in what is known as General-Purpose Computation using GPUs.

GENERAL-PURPOSE COMPUTATIONS USING GPU

General-Purpose Computation using GPUs (GPGPU) is a relatively new term in the computer science field. The idea behind GPGPU is to perform computations using the paralleled architecture of the GPU [6]. While the GPU mainly deals with graphic rendering, GPGPU aims to use it for non graphical applications. Prior to the development of NV20™ by NVIDIA®, using the GPU for something other than image rendering was though as farfetched; GPGPU was something you could only dream about. NV20™ provided a spark of hope, but it was not until recently that GPGPU finally became a reality.

The idea behind the development of GPGPU is that while the GPU processors are dedicated to performing image rendering, intense image rendering are not always being done. Unless we are always working with intense graphic applications, we have a piece of equipment that is performing minimal duties. It is during this time that a programmer should be able to use the resources allocated to the GPU to perform general computations. In this manner, a developer would be able to process an algorithm in parallel, greatly improving the performance of the application. While writing code to be executed on the GPU is quite complicated and done through a special API, new platforms have been developed to facilitate the transition from typical OpenGL programming to a more traditional way of writing C and C++.

NVIDIA CUDA

In February of 2007, NVIDIA® Corp. released its Computer Unified Device Architecture (CUDA) platform [3]. CUDA™ is brand new revolutionary architecture that allows developers to develop and run complex applications on the GPU [8]. CUDA™ was initially the first platform of its kind [8]. It is made to work with the GeForce 80™ Plus ™ graphics card as well as all of the next generation GPUs to be developed by NVIDIA®. Previously, writing to the GPU was a very daunting task. Code had to be written trough a special Application Programming Interface (API.) As a result debugging and optimization was very difficult to do. However, CUDA™ is implemented to where we are able to use 'C' to write the application [7]. CUDA™ requires for a relatively new way of implementing code, one would not be able to copy and paste code and expect it to run on the GPU.

CUDA™ is able to execute applications on the GPU trough the use of kernels and threading. The programmer will write their code in blocks, called kernels, which can be processed in parallel. CUDA™ will then thread the kernel; creating many instances of the kernel [8]. While the kernel is being executed simultaneously, each kernel is independent of each other; each has its own data that it will be working with. Shared memory is a crucial part in the implementation of CUDA™, without the shared memory threads would have to communicate with the DRAM on the hosts CPU in order to communicate. DRAM is typically slow, hindering the processing speed that it would take to write and fetch data to CPU memory [8].

NVIDIA® has adopted a couple of methods to alleviate the slow process of having to read and write to the CPUs memory. It uses a technique called "Gather and Scatter" technique for reading and writing to memory. The Gather and Scatter technique allows for the GPUs processor to be able to read and write to any place in memory. This technique allows for the GPU processors to write to any open location in memory without having to search for blocks of open memory locations to store the data sequentially. When gathering, the processor will read the data from the scattered locations as if it were reading it sequentially. In addition to the "Gather and Scatter" technique, developers have also introduced shared memory. Each of the GPUs processors contains memory that is local to each processor. In this manner, the GPU becomes less dependent on the slower DRAM [8]. Not having to make as many roundtrips to memory allows for a faster performance. The shared memory is not only accessible by the local processor, but it is also accessible to other processors. They are able to read and write to these memory locations instead of having to go to the host memory.

Not all CUDA™ written programs consist of just the kernels. CUDA™ requires for the programmer to be able to identify the methods that will need to be implemented in parallel. The programmer is responsible for allocating the space required for each item being introduced to the GPU. Programmers are also responsible for providing specific time that the GPU will be called. While this might sound difficult, it is not. Using an extension of the C language, programmers are provided with the new methods to achieve this. At runtime, the application will be broken down and the Runtime Library will determine which parts of the application need to be run in the GPU and which ones need to be executed in the CPU [8]. When writing an application, programmers need to be

aware that while the majority of the code is done in C, certain functions and variables will have to be written differently than what one is used to. As an example, variable and type qualifiers will be different as opposed to the norm. Variable and type qualifiers will now be declared with the type and as to where they will be used be it on the GPU or on the host. The number of grids and threads that the GPU will be using is also declared by the programmer. As can be expected, all new code will be compiled using an NVIDIA® CUDA™ compiler (NVCC.) This new compiler has been deemed as "industry's first C-compiler development environment for the GPU." [8] Currently NVIDIA® provides the option of downloading the CUDA™ Software Development Kit, SDK, free of charge. Programmers are also given the option of using an emulator if the GPU is unavailable to them, however the speed will not reflect the actual performance of the GPU.

RAPID MIND DEVELOPMENT PLATFORM™

While CUDA™ was the first development platform of its kind to be available, it is certainly not the only one. Rapid Mind® Inc. has also released its own version of using GPUs for general purpose computing, but with quite a few dissimilarities. The main difference between them is that Rapid Mind is not GPU specific. In fact, Rapid Mind™ was developed so that it can interfere with not only a GPU but also a multi core processor [10]. Rapid Mind Development Platform™ allows for the programmer to use standard C++ code. Unlike CUDA, Rapid Mind does not allow the programmer to directly call the GPU or the Multi-Core Processor. This has proven to be a positive thing; it eliminates the possibility of a deadlock. Instead of writing the code directly to the GPU, the programmer will write the code in a kernel that will be executed in parallel. Rapid Mind™, being a middle ware, will provide an API to which the coder can write to using pre-existing software such as Microsoft® Visual C++. At runtime, Rapid Mind™ will break up the data and distribute the operations to either the GPU or the processor. In this manner, like CUDA™, Rapid Mind allows for the application to be processed in parallel. Rapid Mind™ does not require a new compiler; it relies on previous Windows Visual C++ compilers as well as GCCv4 for Linux based systems [10].

The feasibility that Rapid Mind provides has propelled Rapid Mind to become a popular option for GPGPU. It is very simple to write code for Rapid Mind rather than to write code for NVIDIA CUDA™. In addition, Rapid Mind is not written with a particular GPU in mind. It is developed for transportability [10]. Currently, Rapid Mind supports NVIDIA GeForce® 6000, 7000 and 8000 cards as well as various ATI® graphics cards [10]. Like early GPUs that had success in the video game consoles, Rapid Mind has also had success in the industry. Sony® Play Station 3™ uses the Rapid Mind Platform™ under their Yellow Dog Linux based system [10].

GPGPU APPLICATIONS

Being able to use the GPU for general purpose computation allows for applications to improve their run-time speed. This speed improvement might not matter when dealing with small applications that only work with a limited amount of data, however when dealing with large applications performance matters. GPGPU is best suited for large sorting and searching algorithms. When dealing with large amount of data, both CUDA™ and Rapid Mind™ are able to break down the data and perform the operations in parallel according to the amount of processors available. This is done while

allowing the processors to communicate and share data with each other. Speed will be dependent upon the amount of processors that are available to the application. As a result, an application with a Big O notation of $nlogn^2$ will now be processed at $nlogn^2/p$ where p represents the amount of processors [11]. Not all parts of an application can be performed in parallel; nonetheless the improvement in speed can be greatly noticeable. Other types of applications that benefit from using a GPU to increase performance time include, digital image processing, analog and signal processing as well as ray tracing and computer vision.

Computer vision is the science behind allowing computers to see images and identify items. A good example of computer vision is that of counting the number of items that appear in a given image. Taking a Portable Grayscale Picture (PGM) and assigning each pixel a value that corresponds to a given range depending on the color, one is able to count the number a certain item appears, as an example a red blood cell. A computer vision application would store the picture in a two dimensional array and perform multiple operations when trying to count the number of cells. Through the use of GPGPU, we would be able to perform the same operations in parallel with a minimum amount of time. This is extremely important, especially if we are performing these operations on large images, or if we have a limited amount of time to analyze them.

CONCLUSION

Graphical Processing Units have evolved tremendously from its early days of serving the monolithic Atari™ game console. While GPUs are still widely used in consoles, they are no longer restricted to only performing graphical rendering. They are currently able to support operations that surpass their initial development purpose. GPUs are now able to process data that would otherwise be done in the CPU. This advancement has lead to the development of NVIDIA® CUDA™ and Rapid Mind Inc. Development Platform™ for the intention of being used in general purpose computation using GPUs. These two platforms allow programmers to transform C/C++ code into kernels that can be executed in parallel within the many processors available to the GPU. While not all tasks can be performed in parallel, those that can benefit greatly from the increase in performance time. GPGPU is great for applications that handle large amount of data such as computer vision and databases. Through the use of GPGPU, we are not only improving the performance but we are also opening the door for the next generation of software development.

AKNOWLEDGEMENTS

The author would like to acknowledge Dr. Ahmed Mahdy, professor at Texas A&M-Corpus Christi Computer Science department. Dr. Mahdy has had a positive influence in not only the direction of this research but also on the direction of my career. Dr. Mahdy has served as both a mentor and a friend through this process.

In addition, the author would also like to thank Dr. Michael Scherger, professor at Texas A&M-Corpus Christi. Dr. Scherger is responsible for sparking my interest in this particular topic.

REFERENCES

[1] ATI Radeon 9700 Pro PCSTATS Review, www.pcstats.com, November 2007.

[2] Chadwick, I. Mapping the Atari-Revised Edition, US: COMPUTE! Publications, 1985.

[3] Gruener W., NVIDIA™ Activates supercomputer in your PC, February 2007, www.tgdaily.com, November 2007.

[4] Graphics Processing Unit, www.wikipedia.com, November 2007

[5] Liedholme M., The N64 Long Way to Completion, 1997, www.nintendoland.com, November 2007.

[6] GPGPU, www.gpgpu.org, November 2007.

[7] NVIDIA™, NVIDIA CUDA Programming Guide 1.0, www.nvidia.com/cuda, November 2007

[8] NVIDIA™ , NVIDIA CUDA Homepage, www.nvidia.com/cuda, November 2007

[9] Papakipos, M., Covering Design Features in CPUs and GPUs. www.hpcwire.com, November 2007.

[10] RapidMind Architecture, www.rapidmind.net/technology, November 2007.

[11] Sintorn E., Assarsson, U., Fast Parallel GPU-Sorting Using a Hybrid Algorithm, 2007

[12] Wissman, R., Nvidia NV10, www.Speedy3D.com, November 2007.