# Implementation of Parallel Processing Techniques on Graphical Processing Units.

Brad Baker bradley.baker@gd-ais.com, Wayne Haney wayne.haney@gd-ais.com, Charles Choi charles.choi@gd-ais.com.

## Abstract

For more than 40 years, computer technology has followed Moore's Law, which states that the number of transistors on an Integrated Circuit (IC) will double every two years. This doubling, as applied to Central Processing Units (CPUs), has been used to great effect by software developers, allowing them to add functionality without needing a larger hardware footprint. However, as far as traditional CPUs are concerned the era of Moore's Law will soon come to a close: technology is advancing to the point where making the transistors on a CPU any smaller would defy the laws of physics. This begs an obvious question: moving forward, how can one add processing power without increasing the required hardware footprint? This question suggests a follow-up: do any of these methods of increasing processing power have the potential to keep pace with the theoretical curve of processing power versus time predicted by Moore's Law? One potential solution to maintain the pace of Moore's Law is to utilize a Graphical Processing Unit (GPU) to perform the processing. An existing algorithm that was initially designed for use on a CPU was ported over for use on a GPU with impressive results. This paper summarizes a research effort that addresses the questions and problems that arose for both the planning and the porting of the application.

## Introduction

The computer chip industry has been challenged to find alternative ways of achieving increases in processing power without hardware footprint growth. Parallel processing has emerged as their way forward. One of the promising new parallel computing platforms comes from the video game community. The GPU is a relatively inexpensive graphics accelerator and is a primary platform for a new emerging field of stream processing. The term stream processing refers to the parallel execution of a software application to take advantage of the performance offered by GPUs [1]. A "stream" is defined as a collection of data, such as an array, that can be operated on in parallel. Stream processing relies on a collection of mathematical functions called a "kernel" that can be applied to each stream. Since the data is operated on in a stream instead of in memory, there is a decrease in memory access and therefore an increase in performance. GPUs are a highly parallel processing unit and in the past year, several companies and research groups have been looking into ways to harness them as a mathematics co-processor. The general consensus in industry is that if a mathematical algorithm can be written in a parallel manner, then it is a solid potential candidate for processing on the GPU. The research team decided to attempt porting an existing signal processing application to a GPU in an effort to determine how viable a solution this technology could be.

**Figure 1: Example dual GPU solution.**

## Methodology

There are two main ideas that presented in this paper, a technology overview, and an application port. The GPU was chosen to be investigated and the first task was to assess the differences between the various commercially available GPUs. To develop this list, a survey was conducted by visiting websites that highlight the latest GPU technology. One primary site of interest was GPGPU.org [2] which focuses on furthering the utilization of the GPU for use in non-graphics related problems. There were two major criteria used in selecting the hardware platforms: 1) they must be COTS products; 2) they must utilize a unified shader architecture. The first criterion was used in an effort to keep the platforms low cost and ensure that they would be easily accessible in mass quantities. The second criterion limited the choices of the GPUs to units that do not utilize a unified architecture as this would add too many layers of complexity to effectively use it for general purpose processing.

The application that was chosen to be ported was one that leveraged the Department of Defense sponsored math API (VSIPL++) framework. This application was ported over using Nvidia's CUDA framework [3]. Concurrently, the application was altered to use the new CUDA routines instead of existing VSIPL++ classes and FFT calls and integrated.

During the first phase, the team discovered that the CUDA FFT routines could support batch processing. Batch processing is the ability of the stream processor to perform parallel math operations on multiple sets of data in one memory space – similar to threading on CPUs. The application was modified to make use of this batch processing by taking the FFT calls out of a for loop.

## Results

Overall, approximately 500 lines of code were changed or added in the application in order to use the FFT library on the GPU, however most of these changes were to accommodate breaking the FFT out of a for loop. The performance boost observed between the original CPU implementation and the GPU implementation can be seen in Table 1. It is important to note, however, that time only allowed implementation of the FFT routines and instead of both the FFT and vector operations.

**Table 1: Test Algorithm Performance Increase**

| Algorithm | Average Time Needed |
|---|---|
| VSIPL++ Algorithm on Intel QX6700 CPU | 735.78 msec |
| CUDA on a Nvidia g80 GPU | 367.23 msec |

The performance of several systems were considered in the report, and an emphasis on providing a common benchmark was chosen for comparing multi-core CPUs to GPUs. The gflop numbers used are all theoretically based; real-world applications will realize smaller performance. However, power consumption was measured in a laboratory environment and represents the overall system draw including all subcomponents at a full simulated load. Chart 1 illustrates one of the key benefits over performance over traditional multi-core CPUs.

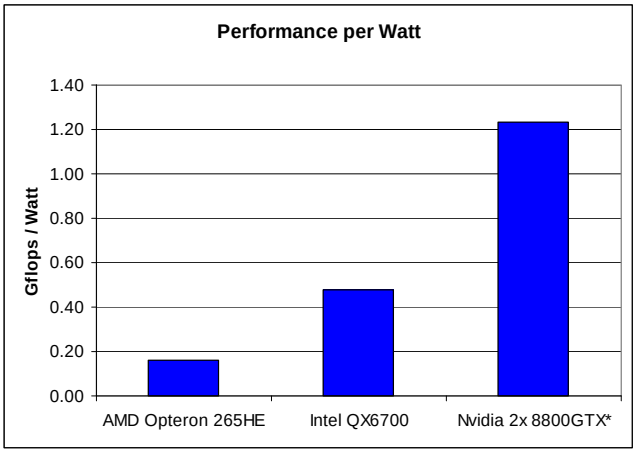


**Chart 1: Performance per Watt analysis**

## Conclusion

In summary, companies which continue to use serial programming practices will fall behind agile companies which choose to embrace the new paradigm of parallel programming. Parallel processing is the only viable method to take advantage of the higher processing power that GPUs and multi-core CPUs bring to the table. Once this is understood, the suitability of the two hardware solutions comes into greater focus: multi-core CPUs require the least amount of legacy code modification and are probably the easiest of the technologies to port to; GPUs are still in their infancy as a math co-processor and still leave a lot to be desired in terms of ease of use. However the GPU is maturing rapidly and its ability to perform large numbers of floating point operations per second (flops) far exceeds any other COTS solution.

As the research progressed, the team came to the conclusion that in general, the GPUs are best suited for algorithms which involve large amounts of matrix manipulation, or algorithms where the entire algorithmic string can be converted into one kernel. Based on this conclusion, the chosen application was not an ideal candidate for testing out the GPU, since there was limited matrix manipulation is performed in this algorithm. However, the gains achieved by performing minor changes were notable, which signifies the potential for larger gains in other more well suited applications.

## References

[1] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, Timothy J. Purcell, *A Survey of General-Purpose Computation on Graphics Hardware*, Vol. 26, No. 1, 2007.

[2] " General-Purpose Computation Using Graphics Hardware." GPGPU.org. 24 May 2007. 25 Nov. 2006 <http://www.gpgpu.org>.

[3] "NVIDIA CUDA Homepage." Nvidia.com. 27 Apr. 2007. 11 Dec. 2006 <http://developer.nvidia.com/object/cuda.html>.