Journal of Theoretical and Applied Information Technology

www.jatit.org

TRENDS , PERFORMANCE AND GENERAL OPERATION OF THE GRAPHIC PROCESSING UNIT.

1 J. ANTONIO ÁLVAREZ, 1GUSTAVO MARTÍNEZ ROMERO, 2 J CORREA-BASURTO.

1 Centro de Investigación e Innovación Tecnológica IPN México, Cerrada de Cecati S/N. Col. Santa Catarina Azcapotzalco México D. F. CP:02250 Tel 57296000,
jaalvarez@ipn.mx , egustavo2000@yahoo.com.mx .

2 ESM IPN México,Sección de Estudios de Posgrado e Investigación y Departamento de Bioquímica de la Escuela Superior de Medicina del IPN, Plan de San Luis y Díaz Mirón s/n, Distrito Federal, México.
corrjose@gmail.com

ABSTRACT

A computer graphics card is an additional peripheral that enhances the performance of rendered graphics. Recently, there has been an increasing interest in general purpose computation on graphics hardware. The ability to work independently alongside the CPU as a coprocessor is interesting but not motivating enough to learn how to apply problems to the graphics domain. This paper analyzes the overall architecture of the GPU and its performance.

Keywords: *GPU architecture, Computer graphics card, GPU performance, CPU alternative.*

1. INTRODUCTION

Commodity graphics hardware has evolved tremendously over the last years – it started with basic polygon rendering via 3dfx's Voodoo Graphics in 1996, and continued with custom vertex manipulation four years later, the graphics processing unit (GPU) now has improved to a full-grown graphics-driven processing architecture with a speed-performance approx. 750 times higher than a decade before (1996: 50 b/s, 2006: 36,8 b/s).

This makes the GPU evolving much faster than the CPU, which became approx. 50 times faster in the same period (1996: 66 SPECfp2000, 2006: 3010 SPECfp2000) [1]. Figure 1.1 shows the GPU performance over the last ten years and how the gap to the CPU increases.

Experts believe that this evolution will continue for at least the next five years [2,3].

New graphics hardware architectures are being developed with technologies that allow more generic computation. GPGPU (general-purpose computation on GPUs) became very important and started a new area related to computer graphics research. This leads to a new paradigm, where the CPU (central processing unit) does not need to compute every non-graphics application issue.[5,6,7].

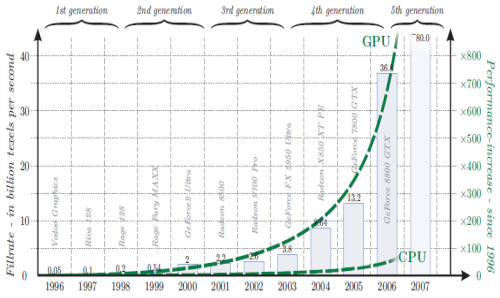


Fig. 1.1. The performance-increase of computer graphics hardware over the last decade . The green trend line shows that the GPU doubles its speed-performance every 13 months (i.e. GPU of 2006 are approx. 750 times faster than G Pus of 1996). In contrast, the performance of the CPU doubles only every 22 months [1].

However, not every kind of algorithm can be allocated for the GPU (graphics processing unit) but only those that can be reduced to a stream based process. Besides, even if a problem is adequate for GPU processing, there can be cases where using the GPU to solve such problems is not worthy, because the latency generated by memory

manipulation on the GPU can be too high, severely degrading application performance.

Many mathematics and physics simulation problems can be formulated as stream based processes, making it possible to distribute them naturally between the CPU and the GPU. This may be extremely useful when real time processing is required or when performance is critical. However, this approach is not always the most appropriate for a process that can be potentially solved using graphics hardware. There are many factors that must be considered before deciding if the process must allocate the CPU or the GPU.

Some of these factors may be fixed and some may depend on the process status.

A correct process distribution management is important for two reasons:

• It is desired that both the GPU and the CPU have similar process load, avoiding the cases where one is overloaded and the other is fully idle;

• It is convenient to distribute processes considering which architecture will be more efficient for that kind of problem.

2. THE COMPUTER GRAPHICS CARD

A computer graphics card is an additional peripheral that enhances the performance of rendered graphics. The card mainly consists of a graphical processing unit (GPU), memory, and a digital/analog converter and connections to and from the graphics card. Most applications on a computer requires some type of graphics to be displayed. The information is processed from the application to the central processing unit(CPU) which is then sent to the specialized graphics accelerator for quicker processing. After undergoing the transformation of computer code, the data is then sent to a monitor to be displayed. Figure 2.1 shows the general flow diagram of Graphics Card.

Originally, computers were once all text based and did not come with a Graphical User Interface (GUI) that consumers have grown accustomed to. As computer technology has advanced, graphics has become an important part in the way humans interact with computers and the demand for graphic oriented computing has called for the need of specialized graphical processing. This is when the age of graphical accelerated cards began.

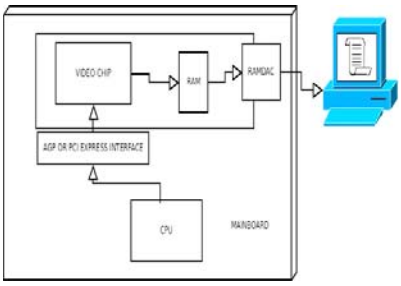


Fig. 2.1. General Flow Diagram of Graphics Card.

Most computer chips are produced as silicon chips. A silicon chip is composed of thousands of transistors, which are in turn composed of three layers of conducting material, mainly silicon, forming a “sandwich” design. The layer might contain either a positive or negative type of silicon. With a difference in electrical charge, and allowance to accept and receive electrons, transistors are used as switches. Current cannot pass through a transistor because of the diode effect. Diodes are devices that block current going in one direction while allowing the opposite direction of the current to flow. This is very important as it can keep sensitive electronics safe from a reverse charge.

3. GPU DESING

The GPU , The processor accelerates graphical processing while taking the CPU’s workload, which is used to process every instruction code for the computer. Since graphical processing units are devoted to processing graphics, the processing of advanced graphical algorithms and coding are accelerated compared to the regular CPU, that is illustrated in the figure 3.1.

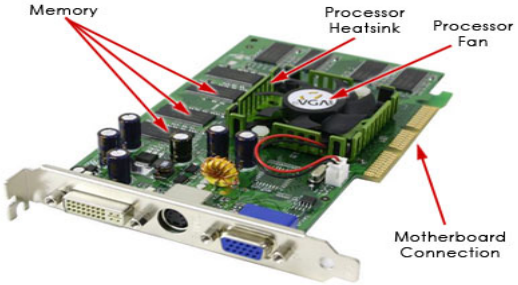


Fig. 3.1. Basic Layout of Graphics Card

Originally, computers were once all text based and did not come with a Graphical User Interface

2.1 Pipelined Architecture

A pipelined architecture is the standard procedure for processors as it breaks down a large task into smaller individual grouped tasks.[4] When a set of instructions are transferred to the GPU the GPU then breaks up the instructions and sends the broken up instructions to other areas of the graphics card specifically designed for decoding and completing a set of instructions. These pathways are called stages. The more stages the graphics card has, the faster it can process information as the information can be broken down into smaller pieces while many stages work on a difficult instruction, that is illustrated in the figure 3.1.1.

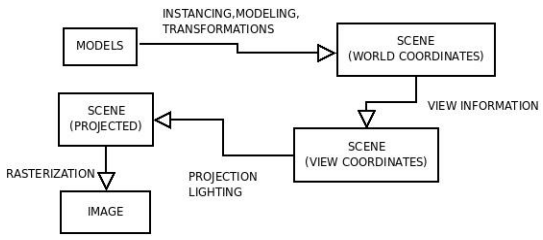


Fig. 3.1.1 The Graphic Pipeline

Pipeline Process : Stage 1

The pipeline process starts with an "Application/Scene" stage, also known as the workload-reduction trick. This stage is devoted to deciding which particular object will be rendered in the three dimensional(3D) environment. The way that 3D environments are created is through a Cartesian Coordinate Systems (an x, y, and z axis) in which objects are placed to create a scene. Scenes can have multiple angles to view them, in which they are created by reference points (cameras). The view space is determined by objects and angles depending on how the creator programmed the scene. The first process of the graphics pipeline is to only render and produce the images of the view space and to skip over unnecessary instructions that will not be displayed even if it was processed. This system allows the graphics card to produce scenes and graphics efficiently.

There are also factors that involve in processing the image efficiently, mainly the Level of Detail (LOD). An objects distance to the reference point of the view camera has an effect on the object's LOD. Some objects are assigned multiple resolution settings (the quality of image that is determined by the number of triangles that composes the object) in which the closer object might receive a higher resolution, while the same object farther away might be at a lower resolution to reduce the workload of the graphics card.

Pipeline Process : Stage 2

The second stage involves the scene's geometry. Objects mainly get moved from frame to frame to give an illusion that the object is moving in a real time setting. Objects can both be moved and manipulated in a scene depending on the application in which it is running. This manipulation of objects is generally called transformation. The objects can be stretched, skewed, moved or moved about an axis, or scaled differently. It is in the second stage of processing that the objects in the environment are altered.

Geometric lightning also occurs in the second stage after the objects are in their proper place, and once the figures receive their shape through the geometric transform process. Different types of lightning, depending on the application being run, will be processed to give the graphics a realistic appearance.

After the lighting is calculated the scene needs to get rid of unnecessary triangles that are only partially shown through the view space. This process is similar to the process which occurs in the first stage and also includes the process of "clipping." "Clipping is the operation to discard only the parts of triangles that in some way partially or fully fall outside the view volume".

Pipeline Process : Stage 3

The third stage implies an algorithm called the digital differential analyzer (DDA) which calculates the position of each part of all the triangles, and determines if the triangles are connected to other triangles. This process is done by computing the slope of each triangle's edge in hope to improve the quality of the image being produced and by allowing more detailed information to be assigned to the triangles

Sometimes when two triangles are touching, or even overlapping each other, a rough pixel "stair-step" occurs in which the edges between the two triangles create non realistic images of edges extruded surfaces that would not normally occur.

Another thing that occurs in the triangle setup phase is the assignment of color and depth values for each pixel. Since the edges of the triangles were calculated, the color and depth values may be interpolated using each triangle's vertex vales of color and depth. Along with the color and depth, the texture coordinates of each pixel is also interpolated in which they will be processed in the fourth stage.

Pipeline Process : Stage 4

The last stage of the pipeline is considered the rendering / rasterization stage. "To fill the frame buffer the drawing primitives are subdivided into pixels, a process known as scan-conversion or rasterization" (Schneider, Benyt-Olaf, pg 245). After all the processes of computing location, color, geometric values, etc., this last stage puts it all together and produces the 3D environment onto a 2D screen.

After the triangle setup is completed in the third stage, the next step is to provide shading values. Shading values are similar to the color and depth values contained in stage 3 and add the finishing touches on the scene. There are three common shading methods: Flat, Gouraud, and Phong shading.

Flat Shading - operates per triangle and provides a quick render of the scene that does not involve extensive computations. This type of shading does not produce a high quality image.

Gouraud Shading - operates per vertex of the triangles. Compared to flat shading, Gouraud shading produces a higher quality image while sacrificing render speed. Because of Gouraud shading takes the lighting values of each vertex of the triangle and interpolates the values across the surface of the triangle, the object will appear smother and not as rigid as flat shading.

Phong Shading - operates per pixel and is the most computation demanding shading process compared to flat and Gouraud shading. Phong shading incorporates the Gouraud Shading idea of taking the average shading of the vertices and also implies its own process that includes other triangle's shading as well. This makes the object blend together easier for more complex designs and results in a higher quality image making it more realistic.

3.2 Memory

Random Access Memory (RAM) assists the graphics card process information. RAM is composed of transistors and memory cells that are arranged in a row and column grid that allows date to be stored and accessed quickly. New technology yielded the Double Date-Rate Synchronous Dynamic Random Access Memory (DDR SDRAM, commonly referred to as DDR), and the DDR2 RAM modules. These types of RAM modules are economically beneficial as they have higher efficiency, cost less, and have a higher potential for improvement.

The memory cells of the RAM can read either the number 1 or 0, which changes due to the capacitor change in gain or loss of electrons. Dynamic memory has a slight flaw, the capacitors have a natural electron leak and are drained of electrons every few milliseconds. To solve this problem, the CPU or the memory controller has to recharge all of the capacitors holding a 1 before they discharge. That is why RAM refreshes thousands of times per second. Memory cells have their own special support infrastructure of circuits that enables it to identify each row and column in the memory cell, keeping track of the refresh sequence, reading and restoring signals from a cell, and telling a cell if it should take a charge or not. Figure 3.2.1 shows the The General flow chart of Gouraud/Phong shadingull .

The memory of the graphics card (VRAM) is controlled by the GPU that stores data in specialized video memory storage space. This storage space operates by the use of common storage space but is especially reserved for graphical processing. The memory operates along with the GPU to produce quick instructions and processing that the graphics card can accomplish. VRAM is necessary to keep the entire screen image in memory. The CPU sends instructions to the video card which undergoes the graphical process and eventually is able to be displayed on a screen.

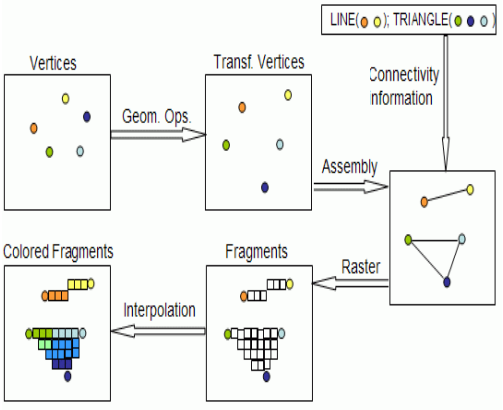


Fig. 3.2.1. General flow chart of Gouraud/Phong shading

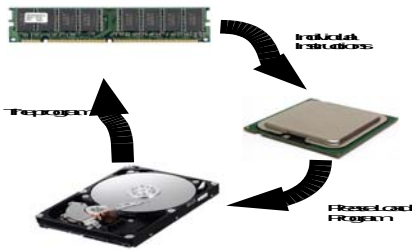


Fig.3.2.2: General flow of information exchanges by RAM.

3.3 Connections

Computer Bus Connector - the graphics card itself is connected directly to the motherboard either through an Accelerated Graphics Port (AGP) or a Peripheral Component Interconnect Express (PCIe) slot.

Recently PCIe has replaced the AGP in becoming the quickest method to transfer information between a additional device and the computer. "A connection between a PCIe device and the system is known as a 'link' and this link is built around a dedicated, bi-directional, serial (1-bit), point-to-point connection known as a 'lane'.

This allows the GPU and CPU to interact with one another at high speeds to process different instructions. Because of the PCIe high bandwidth there can be up to 32 "lanes.". The links and lanes that is illustrated in the figure 3.3.1.

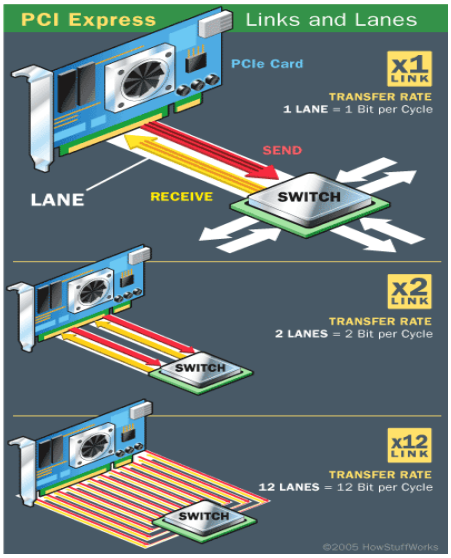


Fig 3.3.1: Overview of PCIe capabilities .

4 Performance

How much faster can one make applications run by using GPU? , the answer is: it depends. It depends on the nature of the application itself, and what you using as the basis for comparison, for example, code that is single-core or multi-core, or optimized or not. Given some approximation of expected performance, one can evaluate other considerations such as the effort to port, price/performance, performance/watt, and performance/space. [1]

4.1 Configuration

The configuration tested is:

- OPTIPLEX GX-360 – 1U Intel server with one of the two processors installed. This server contains one 2.66GHz E5430 quad-core processor and 16GB of memory.
- Nvidia Geforce 8600 GTS
- The Nvidia Geforce 8600 GTS connects to the OPTIPLEX GX-360 via 16x PCIe slot.
- The OPTIPLEX GX-360 is running Ubuntu 9.04. CUDA 1.1 is installed.

4.2 Benchmarks

Four benchmarks have been ported to the Nvidia Geforce 8600 GTS so far:

1.matmatmul – matrix matrix multiply

2.FFT – One and two dimensional FFT benchmark

3.bandwidthTest – tests bandwidth for writing to and reading from the card

4.Monte Carlo Black-Scholes

4.2.1 Matmatmul (matrix matrix multiply) Benchmark.

This benchmark computes C = A*B + C, where A, B, and C are matrices. It is run for a range of dimensions from 100 to 10000. In all cases the matrices are dimensioned as square matrices. Thus, it is testing a subset of the functionality of the BLAS SGEMM and DGEMM routines.[8].

Only the single precision version of matmatmul was run, since Nvidia Geforce does not support double precision. The Nvidia CUDA SDK includes a cublas library that provides a subset of the blas library functions. The cublasSgemm function was used to implement this benchmark on Nvidia Geforce. The cublas library includes functions to copy data from the host to the card and copy results back from the card to the host, and these were also used for all tests. Thus, no board-side code needs to be written or compiled to use these cublas functions, but C code needs to be written to make the CUDA and cublas calls to:

1. Select which Nvidia Geforce card to use
2. Copy input data from the host to the card
3. Calculate the result by calling cublasSgemm
4. Copy the results back to the host

This sequence is straightforward to code in C and does not require any knowledge of optimizing board-side code. The benchmark is illustrated in the figure 4.2.1.

In addition to the cublasSgemm included in the CUDA 1.1 SDK, we tested a tuned version of a sgemm sent to us by Nvidia. Comments in the code say that it was written at UC Berkeley. Note that sources for the cublas library are available from the Nvidia web-site. This version runs faster than the cublasSgemm function. It implements a subset of sgemm options and only accepts array-size parameters that are certain multiples of powers of 2. While all multiples of 64 can be used, some array dimensions can be multiples of 4 or 16. Some results for this function are included.

The bottom two curves show the performance measured using SGEMM in the Intel MKL library with one or four cores of the processor. The MKL library implementation of SGEMM should be considered highly optimized code that has been tuned carefully by experts. This implementation also allows SGEMM to use multiple threads under the control of the environment variable OMP_NUM_THREADS, so it is code that has already been modified to take advantage of multiple cores.

The next line shows the performance using cublasSgemm from the CUDA 1.1 library. This measurement of Gflops/second includes the time needed to copy data to and from the GPU card. Note that there are some significant peaks in the results at certain array sizes. The next line (cublasSgemm N*64) uses the same code but uses only array sizes that are multiple of 64. The top line shows the performance of the CUDA SGEMM that has been tuned for a subset of SGEMM arguments.

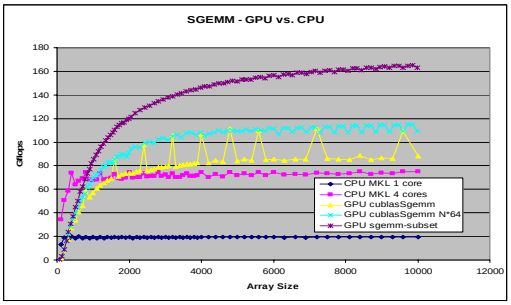


Fig 4.2.1 Shows the Gigaflops/second measured for a single-precision matrix-matrix-multiply

4.2.2 Bandwidth Tests

This benchmark is an HP test tool used to evaluate the data transfer characteristics of multiple GPGPUs on different system platforms. It is based on the original "bandwithTest" example contained in the NVIDIA CUDA SDK. There are four basic transfer types provided, device to host,

host to device, device to device, and read after write. The transfer type, length of transfer, host memory buffer type (pinned or paged) can be selected on a per GPGPU basis, and any of the available GPGPUs may be selected in a test, providing a wide variety of test cases that can be performed.

The bandwidth tests whose results are shown below were conducted using the OPTIPLEX GX-360 system platform configured with an NVIDIA Geforce 8600 GTS. The 8600 GTS is connected to the OPTIPLEX GX-360 via PCIe x16 interface.

Bandwidth tests were conducted using data transfer sizes from 1,000,000 to 100,000,000 bytes in increments of 1,000,000 bytes. The transfer rate of a 50,000,000 byte transfer was selected to display in the graphs. Transfer types of host to device and device to host, using both pinned and paged host memory buffers are presented.

The software used to conduct these tests consists of NVIDIA CUDA SDK and TOOLKIT version 1.1, NVIDIA Driver for Linux with CUDA (171.05) and Linux Ubuntu 9.04.The benchmarks are illustrated in the figure 4.2.2.1 and 4.2.2.2

Overall, these are impressive I/O rates, with transfers from device to host reaching a maximum of about 3GB/sec, showing that these GPU devices and host OPTIPLEX GX-360 are utilizing the available PCIe 16x bandwidth.

4.2.3 1D FFT and 2D FFT

These benchmarks compute the discrete Fourier transform (DFT) using the fast Fourier transform (FFT) algorithm in one and two dimensions. The 1D FFT is performed for sizes varying in size from 2^1 to 2^20. The 2D FFT is performed for sizes varying from 2^2 to 2^12 in each dimension.

Only the single precision version of FFT benchmarks was run, since 8600 GTS Nvidia Geforce does not support double precision floating point. The NVIDIA CUDA SDK includes a cufft library that provides APIs for 1D, 2D, and 3D real

and complex FFTs. The cufft library provides a programming interface that is similar to the well known FFTW open source software package.

Fig. 4.2.2 .1 Hows the measured transfer rate, Host to device

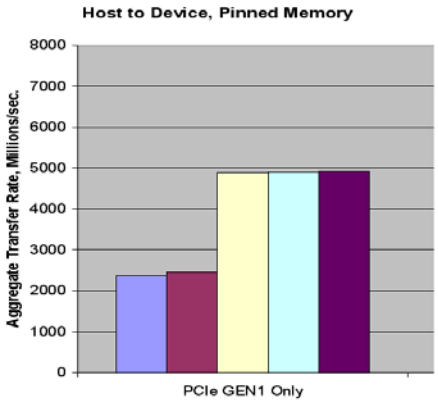
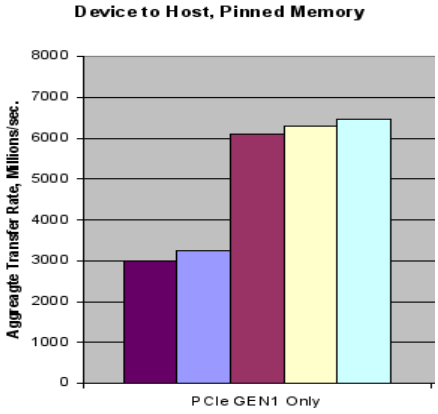


Fig. 4.2.2.2 Hows the measured transfer rate, Device to Host



To use the cufft library, one first creates a plan that defines the size, number of dimensions, and type of FFT to be performed, real to real, complex to complex, etc.

As cufft library does not include functions to copy data to/from the GPGPU, the cudaMemcpy function is used for this purpose. Once the data has been moved to the GPGPU, the FFT is performed by calling the cufftExecC2C function. This then performs a complex to complex FFT.

The benchmark consists of a timed loop that performs two FFTs, one forward and one inverse. The results of the inverse transform are compared to the input data to the forward transform and tested for accuracy as they should be same. Unlike Intel's MKL library, the cufft library does not offer the ability to scale the result of an FFT.

In order to do the comparison the results of the inverse transform are scaled by the reciprocal of

the transform size using a small CUDA kernel written as part of this benchmark. Thus the timed portion of the benchmark looks like this:

1. Create the FFT plan
2. Copy input data from the host to the Nvidia Geforce GPGPU
3. Calculate the forward FFT
4. Calculate the inverse FFT
5. Scale the results
6. Copy the results from the Nvidia Geforce GPGPU to the host
7. Destroy the FFT plan

In order to provide accurate timing, smaller FFTs may be performed up to a 100 or a 1000 times before the execution time is calculated. Larger transform sizes may be performed a few as 1 or 10 times. he benchmarks are illustrated in the figure 4.2.3.1and 4.2.3.2.

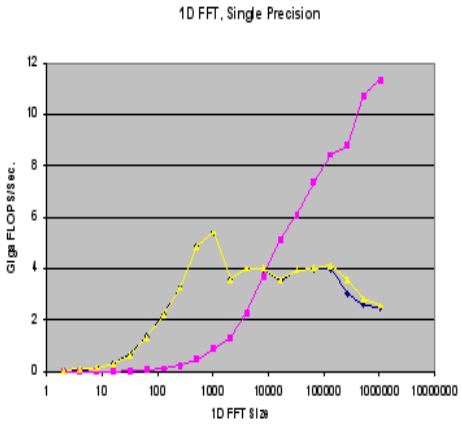


Fig. 4.2.3.1 Shows the Gigaflops/sec. measured for 1D complex-to-complex forward and inverse FFT. The lines on the graph represent results for a single core OPTIPLEX GX-360

These results only show an advantage to FFT on the GPU vs. CPU for large sizes. However, this does not take into consideration that a real application would perform some computations on the GPU between the forward and inverse FFT. It should also be noted that forward transforms can be considerably faster than inverse transforms on the GPGPU. We have measured speeds that are up to 3 times as fast for forward transforms when compared to inverse transforms for larger problem dataset sizes.

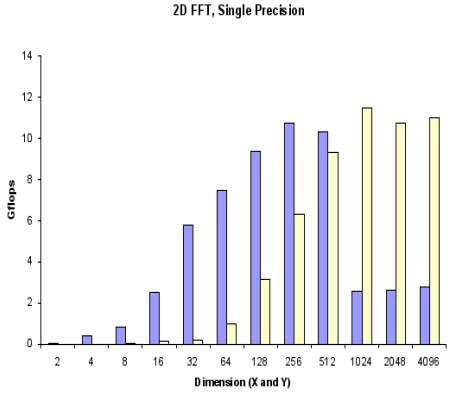


Fig. 4.2.3.2 Show the Gigaflops/sec measured results for 2D complex-to-complex forward and inverse FFT. The bars on the graph represent results for a single core. The results shown are only for square transform sizes, this was done for sake of simplicity of the graph.

4.3.3 Monte Carlo Black-Scholes

This benchmark is an evaluation of European Stock Option Pricing using a Monte Carlo simulation of the Black-Scholes equation. The benchmark evaluates only a single stock option, from 1 to 256 million times, each evaluation is referred to as an experiment. An experiment consists of a draw from the random number generator, an evaluation of the equation, and a sum of the option price and sum of the square of the option price. This benchmark was chosen for its very high computational density and minimal amount of input/output data.[9].

The benchmark uses a Hammersley sequence to generate a uniform random number sequence. This is then transformed to a normal distribution using a polar form Box-Muller transform. This method of generating random numbers was chosen for its ability to be implemented in a highly parallel form. In both implementations of the benchmark, for multi-core CPUs and the GPGPU, the same random number sequence is generated.

In the multi-core CPU version of the benchmark parallelism is achieved through the use of OpenMP pragmas and setting the

OMP_NUM_THREADS environment variable to specify the number of threads to be used in the simulation. In the GPGPU version, a CUDA kernel was developed to execute on the GPGPU. The GPGPU operates as a coprocessor to the host system and is capable of executing a very high number of threads in parallel. The number of threads used in the simulation is determined at run-time, when the kernel is launched. For this benchmark, 4096 threads where used.

The benchmark as implemented on the NVDIA GPGPU generates an array of partial sums of the option price and the square of the option price. Each partial sum is generated by a single thread running on the GPGPU. At the end of the simulation, the array is transferred to the host and the CPU generates the final sums and option pricing.

Only a single precision version of the benchmark was run, as the 8600 GTS only supports single precision floating point format.

The Black-Scholes NVIDIA GPU results shown below were conducted using the OPTIPLEX GX-360 system platform configured with an NVIDIA 8600 GTS. The CPU results are from running the benchmark on a BL460c server blade. The BL460c is a dual-socket quad-core 3.0GHz Intel Xenon based platform. The software used to conduct NVIDIA 8600 GTS test consists of NVIDIA CUDA SDK and TOOLKIT version 1.1, NVIDIA Driver for Linux with CUDA (171.05) and Linux Ubuntu 9.04.

The software used to conduct the CPU based test consists of the Intel C++ compiler for Linux, Intel Math Kernel Library for Linux and Linux Ubuntu 9.04. The benchmark is illustrated in the figure 4.2.3.1

The results for the NVDIA GPGPU demonstrate the computational capability of the GPGPU, providing a 23X increase when compared to a single CPU core and a 3X increase for an 8-core platform. In addition to the execution time of the benchmark on the GPGPU, results include all of the necessary overhead operations to allocate memory on the GPGPU and host, transfer of parameters and the execution kernel to the GPGPU, transfer of the results from the GPGPU to the host, and final calculations on the host. [2] In this benchmark, the overhead items are essentially fixed for each simulation and have an effect on the results that can be achieved for a given simulation size. For example, in a simulation with 4 million experiments the GPGPU results are 12X that of a single-core CPU result; with 16 million experiments one sees 19X that of a single-core CPU result. Simulations with larger numbers of experiments rapidly approach the maximum observed.

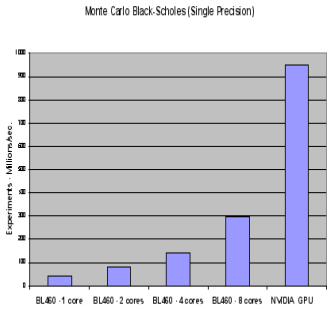


Fig. 4.2.3. Shows the results of the Monte Carlo Black-Scholes simulation in millions of experiments per second. The data points shown are for the maximum results achieved over the course of simulations containing 1 to 256 million experiments. For the CPU results, one sees a near linear scaling over 1 to 8 CPU cores. This is an expected result given the highly parallel nature of the benchmark.

5 Conclusion

A applications demand more processing power, graphic cards will continue to advance in technology to meet these demands. The world of graphics, either through movies, games, or regular software applications, is a huge industry that is pushing graphical processing to it's limits. Graphical pipelines are annual being added to newer models of graphic cards to increase processing speeds. Also, with the added potential of processing graphics, displays such as monitors, projectors, or even televisions will be affected by the graphic cards to meet the demand of displaying higher quality images. Graphical technology has increased exponentially since the first computers were invented and will continue to do so meeting the demands for more life-like graphics in applications.

A number of things have been learned thus far in our investigations:

- A GPU can deliver 10x the single-precision Gflops of CPU core, but a wide

range of speedups can be stated for a given problem. It is important to describe the conditions of both the GPU and CPU execution of the computation.

- In making comparisons to CPU performance, it is important to note if the GPU performance includes the time to transfer data to and from the GPU board.
- CUDA BLAS and FFT libraries provide optimized GPU implementations of these functions, and do not require expertise in optimizing code for the GPU. However, applications will likely require some user-written GPU code to be used in combination with calls to these libraries.
- FFT on the GPU outperforms the CPU, but only if the transform size is sufficiently large. Smaller sizes may be a win depending on the other computations to be carried out on the data prior transferring the data back to host memory. Using pinned verses paged memory buffers may also be a win depending on transfer sizes. Also, batching of 1D transforms needs to be considered, these are likely to effective over many transforms. A key to achieving good acceleration is to have a high ratio of computation to data movement.
- Applications that require little data transfer, have long computation times, and are readily adapted to use parallelism such as Monte Carlo Black-Scholes show impressive speed-ups compared to optimized multi-core implementations.
- The addition of additional interfaces to the overall benchmark framework to accommodate the features of accelerators is clearly a beneficial endeavor. For instance, allowing a particular accelerator to allocate memory using a mechanism optimal for that accelerator is likely to provide an improved result.

Future investigations will include an analysis of the effect of competition between GPU's for I/O bandwidth. Comparisons to CPU performance using up to 8 cores will also be done. The results will be extended to include AMD FireStream 9170, which we have just begun to test.

REFERENCES:

[1] J. L. SPEC CPU2000: measuring CPU performance in the new millennium. IEEE Computer, 33(7):28-35, 2000.

[2] D. Kirk. The future: programmable GPUs & cinematic computing. Presentation at WinHEC'03, 2003. On line available at http://developer.nvidia.com/object/cg_tutorial_teaching.html.

[3] W. R. Mark. Future visualization platform. Panel Presentation at IEEE Visualization (VIS'04), 2004. On line available at http://wwwcsl. csres.utexas.edu/users/billmark/talks.

[4] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. In Computer Graphics (SIGGRAPH'90 Proceedings), volume 24, pages 327-335, August 1990.

[5] G. Kedem and Y. Ishihara. Brute force attack on UNIX passwords with SIMD computer. In USENIX Security Symposium (SECURITY'99 Proceedings), pages 93-98, August 1999.

[6] K. E. Hof III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In Computer Graphics (SIGGRAPH'99 Proceedings), pages 277-286, July 1999.

[7] P. Kipfer, M. Segal, and R.Westermann. UberFlow: A GPU-based particle engine. In ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware (EGGH'04 Pro-ceedings), pages 115-122, 2004.

[8] E. S. Larsen and D. McAllister. Fast matrix multiplies using graphics hardware. In High Performance Networking and Computing (SC'01 Proceedings), November 2001.

[9] T. Jansen, B. von Rymon-Lipinski, N. Hanssen, and E. Keeve. Fourier Volume Rendering on the GPU using a Split-Stream-FFT. In Vision, Modeling, and Visualization (VMV'04 Proceedings), November 2004.