Parallel Computation on Graphical Processor Units with an Eye on Gravitational-wave Data Analysis Needs

Győző Egri
Gyozo.Egri@gusgus.hu

February 22, 2008

Abstract

In this report we investigate the possibilities of using a Graphical Processor Unit (GPU) as a mathematical coprocessor for non-graphical computational tasks with some attention on common tasks in gravitational-wave data analysis. We measure the raw floating point performance of the Nvidia Geforce 8800 GTX card in itself and in comparison to INTEL CORE 2DUO 2.4GHz CPU. We describe previous successful applications of this technology. We also compare different floating point acceleration techniques. We conclude that the technology is promising but development specific to gravitational wave data analysis must take place to ensure the full potential of this emerging parallel computation platform is reached.

LIGO-T070308-00-D

Contents

1 Introduction

Modern Graphical Processor Units (GPUs) may be seen as small parallel supercomputers. Their hardware is optimized for rendering 3D images via rasterization, but it is also possible to use them for General Purpose GPU (GPGPU) computational tasks. The GPU technology brought the fastest evolution in the history of computation: their performance is doubled in every 6 months (performance doubling is around 18 months for CPUs). This process began around the mid '90s, it continues today and according to the predictions it will continue further to at least 2010. During these years videocards evolved from a very specific parametrizable graphical accelerator to a fully programable SIMD (Single Instruction Multiple Data) data parallel computational device. One can usually achieve from 3 to 100 speed-up by using GPUs as a floating point coprocessor, depending on the nature of the computational problem and the level of competence of the programming team.

There are two companies, whose competition drives the market of video cards: Nvidia and ATI. Both companies recognized the possibility of using

GPUs in high performance computing, and they offer their own solutions to help programmers using their products. Since in the last three years Nvidia leads the market, we did not try using ATI video cards for our purposes. Nvidia supports the proliferation of the GPGPU technology in several ways. They introduced a new product line, *Tesla*, specifically for GPGPU users. The other two product line is GeForce (for gamers) and Quadro (for workstation solutions). They also offer a special programming language for GPGPU programmers, the CUDA (Compute Unified Device Architecture) language. They support the GPGPU community by maintaining forums [1], where their engineers answer the submitted questions, and by publishing books on GPGPU [2].

1.1 Nvidia GeForce 8800 GTX, Tesla

The best graphics card available (at the start of our work) is the Nvidia GeForce 8800 GTX card, which brought revolutionary breakthrough in many respects to GPGPU computations. This GPU consists of 16 (675 MHz) multiprocessors and each multiprocessor contains 8 (double frequency) processors which work together in SIMD fashion. The size of the global RAM on the card is 768 MB. The RAM and the GPU communicates through a 384 bit wide bus, that has approximately 80 GB/s bandwidth. The 8 processors in a multiprocessor have a shared memory area (16 KB), which allows communication among individual processors in a multiprocessor. The multiprocessors also have a constant memory (64 KB together) and texture memory cache (8 KB per multiprocessor). We used this card for all of the following measurements. Since then Nvidia's new Geforce 8800 GT card appeared, which has around 10% less performance, but a 1.5 better price/performance ratio altogether. Arrival of new generation of GeForce cards is expected around May 2008.

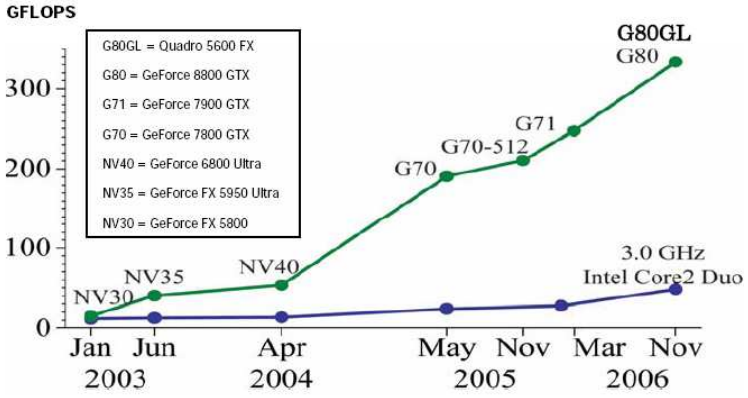

Figure 1: Comparison of floating-point performance of Nvidia GPUs and CPUs (from CUDA programming guide).

Figure 2: Tesla card (left), server (middle), Deskside Computing System (right).

The Tesla product line offers three different modes of using GPUs. One can plug a Tesla card into a normal PC, or use the Deskside Computing System, or install 1 unit high Tesla servers. From the point of view of the price/performance ratio only the first solution is viable. The Tesla cards come with more reliable, tested RAMs, offer 1.5 GB global GPU RAM, and will support double precision computation in the near future.

1.2 CUDA

Nvidia's CUDA is an extension to the C programming language that allows the programmer to use the graphics card in a transparent manner. One can use function type qualifiers to determine, where the different parts of the code will run, and variable type qualifiers to determine where the variables should be located in the GPU memory hierarchy. By using CUDA, the GPU becomes a highly multi-threaded parallel computer, which is able to run thousands of threads concurrently. Threads are grouped into thread blocks and blocks are grouped into a grid. A GPU call (kernel program) handles a whole grid of threads. One block runs on one multiprocessor, which makes communication and synchronization possible among threads in a block. Threads running in different blocks can not be synchronized and can not communicate (easily) among each other.

The introduction of CUDA and Tesla is an effort of NVIDIA, which makes it clear that GPGPU is not only a spin-off of their work, but it is one of their main development directions. In the near future the GPUs may be serious competitors on the HPC market.

1.3 Using multiple GPUs

It is possible to insert more than one GPU into a PC. Currently the best solution is the MSI P7N Diamond motherboard, which has three double size and one single size PCI-Express slots. (High-end video cards usually need more space, because of their large cooling system.) With this motherboard and a quad core CPU it is possible to use four GPUs in an effective way. Four (CPU) threads

can use different cores and different GPUs, so they can run with the same speed as one thread that uses one GPU. It is clear that it worths to use the most dense system possible. Presently the best solution is plugging four 8800 GTX or 8800 GT, or Tesla cards in a PC, the type of card is depending on the computational problem.

2 Floating-point performance

We wrote a test programm to measure the raw floating-point performance of the 8800 GTX card. This code is as simple as possible: we make one N component result vector from two N component input vectors componentwise. We measured the floating-point performance as a function of arithmetic intensity (a). Arithmetic intensity of a computational problem tells how many floating-point operations are made per moved bytes. For example to perform a simple addition, 2 floats are read from memory and the result is written out, and 1 Flop is performed, so in this case the arithmetic intensity is 1/12 .

The measurement was optimistic in the sense that every operation of the GPU is of the type $z = z \cdot x + y$ (multiply-add, MADD), which takes the same time as a multiplication or an addition. The time is the same, but a MADD operation is counted as 2 Flops. Results are shown in Figure 3. To fully understand the behavior of the GPU, one have to know lots of details about the GPU internals.

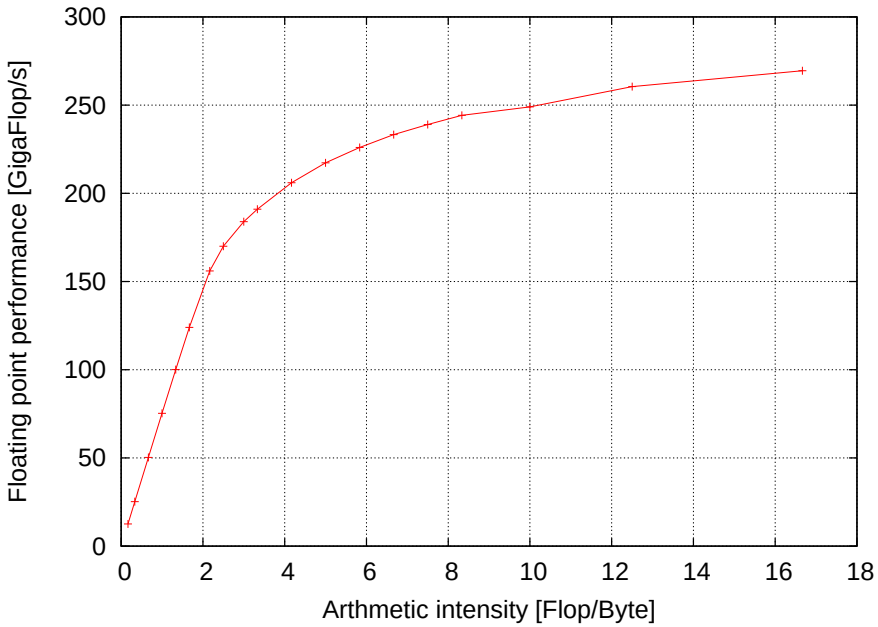

Figure 3: Floating-point performance as a function of arithmetic intensity

2.1 Floating-point standard

Today's graphics cards support single precision floating-point numbers only. Floats on the GPU follow the IEEE-754 standard with the following deviations:

- Addition and multiplication are often combined into a single multiply-add instruction, which has slightly worse error bound.
- Division is implemented via the reciprocal in a non-standard-compliant way.
- Square root is implemented via the reciprocal square root in a non-standard-compliant way.
- Directed rounding towards +/- infinity is not supported.
- Denormalized numbers are not supported
- Underflowed results are flushed to zero.
- There is no mechanism for detecting that a floating-point exception has occurred.
- Operation involving one or more input NaNs is not one of the Nans, but a canonical NaN.
- Conversion from float to integer clamps to the end of the supported range. This is unlike the x86 architecture behaves.

Tesla cards are promised to support double precision floating-point numbers as well.

3 Applications of GPGPU

In this section I describe two completed GPGPU projects, a lattice QCD and a stock exchange application. In these projects I was responsible for the GPU programming part. Then possible acceleration of FFT is given.

3.1 Contribution to Lattice QCD

In the fall of 2005 the Author worked at the Bergische Universitat, Wuppertal, Germany. His supervisor was Prof. Zoltan Fodor, and his task was to accelerate the Lattice Quantum Chromodynamics (LQCD) computations by using GPUs. In LQCD, special forms of the conjugate gradient method are used to invert large complex sparse matrices. The basic building blocks of the computation is sparse matrix - vector multiplication, linear combination of vectors, and computing scalar products. In 2005 there was no CUDA, and GPGPU was all about hacking OpenGL and graphical shader languages (Cg, HLSL, etc). The overall result was 30 GFlop/card, a significant breakthrough, which was about 10-15

times faster than the CPU solutions of the time. Our work is documented in [3]. Results based on the GPU acceleration are given in [4].

Based on this work a 100 node PC (3 TFLOP/s altogeter) cluster was built at the University of Wuppertal, and another, smaller (30 node) cluster was built at Eötvös University, Budapest, Hungary. Eötvös University has won an European Research Council grant (a so called Starting Independent Research grant), so this project will go on, and another 600 000 euro will be spent on a GPU cluster.

3.2 Machine Learning at the Stock Exchange

From the spring of 2007 the Author worked with GusGus, a very innovative Hungarian company active at the stock exchange. The author's task was to accelerate their machine learning algorithm by using CUDA and GPUs. The results of this work are not public, so they are not published either. However, a very remarkable acceleration was achieved in this case, the GPU code runs at least 50 times faster than the previous CPU implementation. In December 2007 a 10 PC cluster was built, which performs at 10 TFLOP/s speed for this specific machine learning task.

3.3 FFT

Nvidia made the CUFFT library, which provides a simple interface for computing parallel FFTs on an Nvidia GPU. It allows users to leverage the floating-point power and parallelism of the GPU without having to develop a custom, GPU-based FFT implementation. It's design is similar to the design of the fftw3 library. The user first copies the data to the device, then makes a plan of the FFT, and then executes it on the device. The plan provides an algorithm that is optimized for the GPU. The plan takes into account the special structure of the GPU, the size of the shared memory in the multiprocessors etc. One main difference compared to the fftw3 library is, that one has to define a batch size when the plan is made. If this is one, everything works like in fftw3. If batch size is not one, many FFTs are performed in parallel, and a whole spectrogram is made in one step. Since the plan takes into account this, the performance of the FFTs depends on the batch size. In the following, we compare the FFT (on 1D real data) performance on the Intel Core2 CPU and the GeForce 8800 GTX GPU.

On Figure 4 the acceleration compared to an Intel Core2 processor is shown for different sample and batch sizes. In this plot only the time of the calculation is taken into account, without the time of data movement from the CPU RAM to the GPU RAM. If additional computationally intensive calculations are performed on the data on the GPU, that takes more time, than the FFT, then this plot is the relevant one, since the time of data movement will be inflated among the different tasks. To fully understand the picture one has to take into account the memory access pattern of FFT and the size of the shared memory on the GPU. For example $N \geq 16384$ behaves differently than smaller sample

sizes. The reason for this is, that the algorithm used for smaller sample sizes would need more shared memory than given on the GPU, and a different, less efficient algorithm is used for $N \geq 16384$.

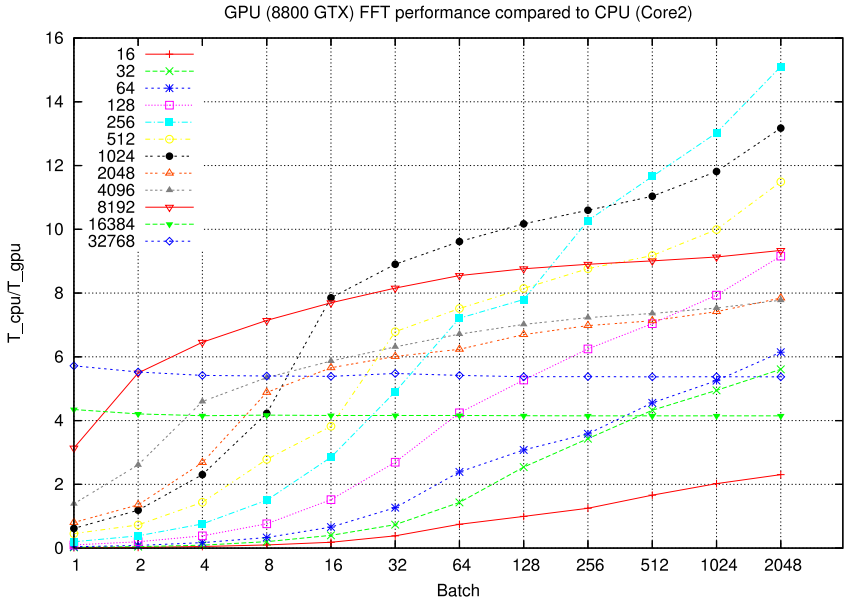

Figure 4: FFT performance for different sample sizes, and different "batch" sizes.

3.4 Other Applications

The last three years brought the accelerating proliferation of GPGPU. As always, scientists and universities are the first, who use this new technology. GPUs are used to solve problems in DNS sequencing, N-body simulations, hydrodynamic simulations, Monte Carlo simulations and many other physical, chemical and biological problems. Instead of describing some of these GPGPU applications, I refer to [1], where a comprehensive list can be found. A workshop on possible application of GPGPU in astrophysics was held in Princeton, Institute for Advanced Study [5].

4 Comparison of acceleration techniques

A new trend in HPC is the usage of special computing elements. These include ASIC-s, FPGA-s, general purpose floating point accelerators (ClearSpeed for example), graphics cards, game consoles etc. Here I summarize the differnet

floating point acceleration techniques, especially those that I met on the SC'07 conference. These platforms have very different energy consumption values, which should be taken into account in price/performance calculations. I calculated with a 3 year lifetime. The calculations on performance and prices here are only approximate values.

Calculation of price and performance of a cluster: let a be the performance of one coprocessor in Flop/s, b the energy consumption of one coprocessor in Watt, c the price of 1 coprocessor in \$, d the lifetime of the cluster in years, e the price of the host machine in \$, f the energy consumption of the host machine in Watt, g the price of energy in \$/kWh and h the Flop/s of host machine.

$$PetaFlop/\$ = \frac{(4 \cdot a + h) \cdot d \cdot 365 \cdot 24 \cdot 3600 \cdot 10^{-15}}{e + 4 \cdot c + (f + 4 \cdot b) \cdot 24 \cdot 365 \cdot d/1000 \cdot g}$$

The 4 multiplier at some places is there because it is possible to use 4 accelerator cards for all of the discussed platforms. I calculated with $d = 3$ years and $g = 0.18$ \$/kWh (price of electricity in Hungary) . In the final column the "performance" means 3 year aggregated performance. The unit of the last column is PetaFLOP/\$.

Platform	a	b	c	e	f	h	$\frac{Perf.}{Price}$
GeForce cluster	300e9	165	600	1000	250	0	15.1
Tesla cluster (in PCs)	300e9	165	1200	1000	250	0	11.1
Tesla cluster (1U rack)	300e9	165	3000	1000	250	0	6.6
Cell Broadband engine	0	0	0	500	380	230e9	9.8
ClearSpeed accelerator	80e9	30	1000	1000	250	0	4.5

5 Conclusions

GPUs are very fast floating-point computational devices. They work in SIMD fashion, which narrows their possible usage, but if one has a problem that can be efficiently described for a data parallel machine, usage of GPUs can considerably (3-100 times) accelerate computations. The theoretical computational power of the 8800 GTX is around 300 GFlop/s. It's bandwidth between the GPU and the global video RAM is 80 GB/s. It is possible to use multiple GPUs in a PC (up to 4 currently), so for a problem that fits nicely to the GPU one can achieve around 1.2 TFlop with a single PC. The price of this is around 5\$/GFlop (including the price of the host machine). From the results of Section 4, it is clear that this is the best solution for floating point acceleration.

If GPUs are used for a bandwith limited problem, where streaming of the data is possible, one expects 10-15 times faster code than normal CPU implementations. For computationally more intensive tasks 30-100 times faster code is possible. Experience shows that the final performance and achieved acceleration is strongly dependent on the expertise and experience of programming

team in GPU computation, as the hardware technology only enables but does not ensure the breaktrough performance.

However, there are tasks where the fascinating capabilities of GPUs can not be or mistakenly are not harnessed. If the memory access pattern of the problem is such, that streaming is not possible, or the shared memory of the GPU can not be used, then it is possible that no acceleration will be achieved at all. Care must be exercised when formulating the problem and the algorithms for GPU environments.

Parallel Computing is between art and science in the sense, that there are no recipes which algorithm should be used in case of a new hardware setup. Sometimes the most trivial parallelization works, but sometimes even new algorithms have to be developed. But it is incontrovertible, that the GPUs brought one of the fastest evolution in hardware technology, they are faster and faster, and they are applicable to more and more computational problems.

It seems clear that at this time, gravitational wave data analysis can profit from the emerging GPU technology. The possibility to speed-up present applications and enabling presently prohibited paths exist. However, the quantification and prototyping of these solutions require significant amount of focused research of a dedicated expert team.

Acknowledgment

The Author is grateful to the scientific and commercial organizations who allowed and promoted the accumulation of experience in this field. The Author is grateful for the access to the hardware test platform access provided by GECo of Columbia University in the City of New York. This report was assigned a LIGO Document Number: LIGO-LIGO-T070308-00-D.

References

[1] www.gpgpu.org, forums.nvidia.com

[2] Books on GPGPU: M. Pharr, R. Fernando, "GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation", H. Nguyen "GPU Gems 3"

[3] G. I. Egri, Z. Fodor, C. Hoelbling, S. D. Katz, D. Nogradi and K. K. Szabo, "Lattice QCD as a video game," Comput. Phys. Commun. **177** (2007) 631 [arXiv:hep-lat/0611022].

[4] Z. Fodor, K. Holland, J. Kuti, D. Nogradi and C. Schroeder, "New Higgs physics from the lattice," PoS **LAT2007** (2007) 056 [arXiv:0710.3151 [hep-lat]]. Y. Aoki, Z. Fodor, S. D. Katz and K. K. Szabo, "The QCD transition temperature: Results with physical masses in the continuum limit," Phys. Lett. B **643** (2006) 46 [arXiv:hep-lat/0609068].

[5] www.astrogpu.org