Teraflops for Games and Derivatives Pricing

Christoph Bennemann
d-fine Ltd., Germany, e-mail: christoph.bennemann@d-fine.de
Mark W. Beinker
d-fine Ltd., Germany, e-mail: mark.beinker@d-fine.de
Daniel Egloff
QuantCatalyst Inc., Switzerland, e-mail: daniel.egloff@quantcatalyst.com
Michael Gauckler
QuantCatalyst Inc., Switzerland, e-mail: michael.gauckler@quantcatalyst.com

Abstract
Financial computing continuously demands higher computing performance, which can no longer be accomplished by simply increasing clock speed. Cluster and grid infrastructures grow, their cost of ownership explodes. On the other hand, the latest GPU (Graphics Processing Unit) boards show impressive performance metrics. This leads to the questions if and how one can harness this power to bring financial computing to the next level. We analyze the pricing of equity basket options with a local volatility model implemented on a GPU. Our performance gains are very impressive.

Keywords
Graphics processing units, high performance computing, cluster, grid, Monte-Carlo simulation, basket options, local volatility, derivatives pricing, risk analytics.

1 Introduction

Over the past twenty years, financial computing has emerged as a strategic discipline for the finance industry, and as a financial sciences domain in its own right. Today it is probably one of the fastest growing areas in scientific computing. This remarkable growth has been caused by a continuous development of innovative financial products, a dramatic increase in volumes, and the requirement to process transactions in ever shorter time frames.

Researchers and practitioners have developed a thorough understanding of the mechanics of the financial markets. The modern models they have developed to price and hedge complex financial products are computationally very demanding. Improved numerical schemes and substantial computing power are indispensable. To supply such huge amounts of processing power, the standard approach is to connect many servers and desktop machines to clusters and grids. This approach has been pioneered in academia and is now heavily used not only in the financial world but also in defence, engineering, life sciences, bio-technology and medicine, to name a few.

Because higher computing performance can no longer be accomplished with an increase in clock speed, clusters and grids must grow in size. The resulting infrastructures are likely to hit new barriers, such as bandwidth limitations, network latency, power consumption, cooling, floor space and maintenance. A robust operation, which is a key requirement in any financial services environment, cannot be achieved without paying attention to all of these issues. As a consequence, building and managing traditional clusters and grids is costly.

To pack more processing power onto a single chip, the CPU semiconductor industry has recently started moving to multi-core system designs with two, four and more processor cores. The graphics hardware manufacturing industry made this transition more than ten years ago. For instance, NVIDIA released its first dual core chip as early as 1998. Today, special processors on graphics cards bring realistic 3D real-time visualization and high-definition video processing to the desktop. They

have built-in remarkable general purpose computing capabilities for data-parallel problems. These new commodity computer graphics chips, also known as GPUs (Graphics Processing Units), are probably today's most powerful computational hardware on a per dollar scale.

In this article we investigate how we can harness this power for financial computing. Several typical financial applications can be significantly accelerated with specialized massively parallel algorithms running on GPUs. We illustrate the capabilities of GPUs with an implementation of a market standard local volatility model and apply it to the pricing of structured equity basket derivatives. Our pricing algorithms run 25 to 50 times faster on a GPU than a serial implementation on a high-end CPU. This enormous performance gain renders many of today's almost infeasible tasks possible, including real time pricing of path dependent multi-asset options. We therefore believe that using GPUs in financial computing will have a disruptive impact on the financial services industry: already a single GPU provides a cheap and simple alternative to a smaller cluster or grid, it brings high performance computing to traders' and risk managers' desks, and it excites financial product development.

2 The Free Lunch is Over

The past decades were dominated by an increase in clock frequencies. We all witnessed the transition from a few hundred MHz to the scale of GHz. The continuous increase in clock frequency caused the CPU performance to double every 18 to 24 months. Nowadays, the growth in frequency is hitting physical limits such as heat dissipation, signal propagation, radiation, and power consumption. To further increase computing performance, the industry is moving to aggregate systems. On the macroscopic level we see a dramatic rise of cluster and grid infrastructures. On the microscopic level, the semi-conductor industry develops multi-core chips with two and four cores, but also many-core processor units with even hundreds of cores.

2.1 Clusters and grids

Clusters and grids are a popular technology to provide high computing power. The aggregation of off-the-shelf commodity hardware to build a system with a computational capacity which was previously only achievable with a supercomputer is a compelling idea. It has proven of great use in many applications. Some business functions of financial institutions cannot be pursued without the help of hundreds of dedicated machines filling entire floors in data centres.

With the paradigm shift to multi-core and many-core in view, we must see today's clusters as a machine of moderate parallelism. The most common computation model on a cluster is a master-slave setup implemented with a networking middleware, such as the Message Passing Interface (MPI). The master manages work packages, which are sent to the slaves for processing. A slave operates on a single machine, or even on a single core, and executes the instruction sequentially. Upon completion it sends the results back to the master, where all results are aggregated. The processors in a cluster can communicate only at relatively low speed. Therefore, the approach works very well if the number of sequentially executed instructions in a work package is large compared to the number of processors in the systems. Even if the cluster communicates over a high-speed network, I/O bandwidth limitations and communication latency require a large amount of processing to be done sequentially. Consequently, the so-called "embarrassingly parallel" problems are best suited for cluster computing.

When scaling up a cluster installation, we face new barriers. Communication between nodes, in particular many-to-one or many-to-many communication patterns, stress the network bandwidth. Expensive high-performance network technology is required. Other critical issues are power consumption, cooling, floor space and maintenance. The larger the cluster the more critical becomes the overall reliability of the system. The software must provide fault-tolerance mechanisms in order to account for failure of nodes during calculations. To quantify the total cost of ownership, all these issues must be taken into account. Isn't it time to shrink wrap clusters and make them small and energy efficient?

2.2 Graphics processing units

By now GPUs have matured from specialized graphics rendering devices to general purpose computing resources. A modern GPU can to some extent be seen as a small miniaturized cluster being placed on a single chip. Figure 1 depicts the components of GPU architectures. The GPU works closely in conjunction with the CPU of the host, which sends code and data to the GPU memory. A special hardware unit, which is comparable to the master in the cluster setup, is responsible for the scheduling of the threads on the thread processors. Threads executing in the same block of thread processors can communicate through dedicated on-chip shared memory. The global memory is used to communicate between all executing threads and to synchronize them. Aggregation and transfer of the result back to the host memory is also done through the global memory of the GPU board.

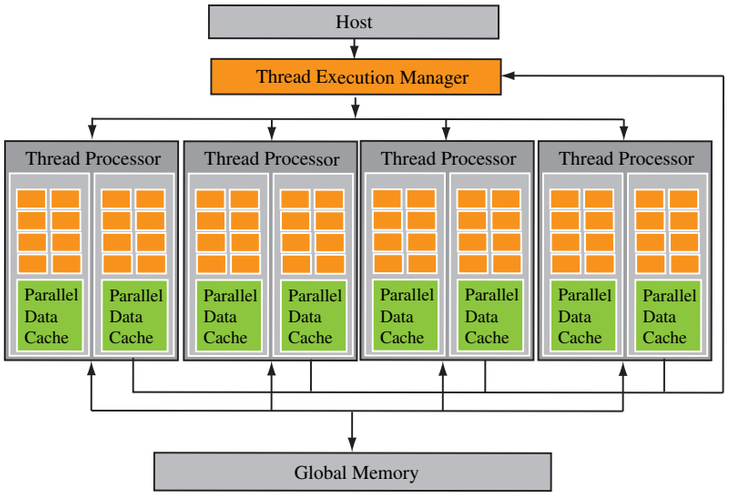

Figure 1: Schematics of the NVIDA GPU architecture.

The concept of threads is well-known in parallel programming. They are sections of a program which are executed concurrently with others. The difference between classical multi-threading programming on a CPU and parallel programming on a GPU is the number of threads which are spawned and the cost for context switching between threads. On a GPU, in order to provide work for hundreds of cores, we require tens of thousands of threads. Context switching between threads is implemented at the hardware level. It has virtually no overhead and can happen even between two instructions. The key advantage of such an execution model is its capability to hide the latency occurring from global memory access. Once a data transfer from memory to the arithmetic logic unit is initiated, another thread can continue executing immediately while the data is being moved and the former thread waits for its arrival. Such an execution model allows many transistors to be dedicated to arithmetic logical units, instead of using them for cache as in the classical CPU design. Assuming the same number of transistors, the GPU architecture features more arithmetic units than the corresponding CPU. The floor plans in Figure 2 illustrate this best.

2.3 CPUs and GPUs compared

The latest GPU boards have very impressive performance metrics. For instance, the NVIDIA Tesla C870 has a memory bandwidth of 76.8 GB per second and 512 GFLOPS peak performance of which 430 GFLOPS are achievable. This has to be compared with a 20 GB per second memory bandwidth and 81 GFLOPS achievable on a Quad-Core Intel Xeon E5472 processor (3.00 GHz, 2x6 MB L2 cache, 1600MHz FSB), see Intel (2008).

Advances in fabrication technology increase at the same rate for both platforms. The above disparity in performance originates in the fundamental architectural differences. The latest GPUs and CPUs have comparable numbers of transistors; the Intel Xeon series 5400 has 820 million, the upcoming GPU of NVIDIA has over 1 billion. CPUs are optimized for sequential code. Many transistors are dedicated for branch prediction, out-of-order execution, and a significant amount to the L2 cache. The data-parallel design of a GPU uses only small caches and devotes the majority of transistors to computation, see Figure 2. We conclude that the two architectures are suited to execute different kinds of code.

2.4 How serious is GPU computing?

Is serious financial computing on gaming hardware possible at all, or is this hardware solely there for entertainment? All the leading GPU manufacturers, including ATI and NVIDIA, have realized the need for serious GPU computing and launched specialized product lines for these market segments. The product lines are built for serious computing and feature enterprise level support, guarantees and availability, rack mountable cases and are compatible with all major operating systems. The GPU itself is not over-clocked and their boards do not have – somewhat counter intuitively, but in view of the above consistently – connectors for displays.

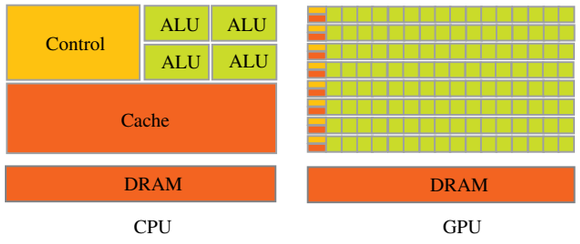

Figure 2: CPU and GPU architectures compared.

One point which is often criticized is lacking support for double-precision arithmetic. Even though not all algorithms require double precision, chip makers cannot discount this industry request. In order to address it, NVIDIA will manufacture a double-precision GPU, which, at the time of writing this article, has already been made available as a developer edition. This strategy ensures that current software libraries will support the new hardware by the time it becomes available publicly. The finance industry can also inherit new features from the gaming industry, which are unavailable in current CPU and memory architectures. One such feature is the texture memory which we used intensively for our local volatility model implementation in section 4. Finally, being linked to the entertainment industry is no bad thing at all: with about 15 millions units sold monthly by NVIDIA alone, a steady flow of investment and economies of scales are guaranteed.

2.5 Future-high performance infrastructures

For small to medium sized problems, a single GPU can replace a cluster of dozens of CPUs and therefore make a cluster obsolete. For larger problems one will most likely see a symbiosis. Clusters of CPUs will first partition the problem on a coarse grained level and let GPUs attached to compute nodes process the highly parallel sub-problems. Hardware vendors already offer special rack versions to support hybrid clusters. For instance the NVIDIA's Tesla 1U computing system with four GPUs (512 cores in total) can sustain more than 1.2 TFLOPS, packed in a standard 19" 1U rack-mount chassis. It can be expected that GPU installations will become ubiquitous in any larger data center. GPUs will also find their niche in ecological high-performance computing systems. For instance the power consumption of a Tesla 1U computing system is as low as 550W to 800W.

3 Efficiently Programming Modern GPUs

In the early days of GPU programming one had to trick the GPU into a general-purpose computing device by casting the problem as a graphics operation and program the various elements in the graphics rendering pipeline. The highly constrained memory layout and access model hindered a broad application of these devices and made the programming for non-graphics experts hard and time consuming.

Fortunately, the new GPU generation is based on a fully general data-parallel architecture. Moreover, NVIDIA released a new hardware and software architecture, named CUDA. It is specifically designed for issuing and managing computations on the GPU as a data-parallel computing device without the need of mapping them to a graphics API. CUDA is a programming model. The GPU is viewed as a computing device capable of executing a very high number of threads in parallel. It operates as a co-processor to the main CPU and any data-parallel, computing-intensive part of an algorithm can be delegated to the device.

The best performance on a GPU is achieved if many threads execute the same code on different data, in a single instruction multiple data (SIMD) fashion. Memory access to local shared memory, through the local parallel data cache (constant and texture memory) and well aligned access to global memory are particularly fast. Taking care of proper memory access results in programs that can achieve a large fraction of the theoretical peak memory bandwidth which is in the order of 100 GB per second for today's GPU boards. The GPU will achieve best performance for algorithms that contain a high degree of data parallelism and problem sizes that fit into the memory of the GPU board which is currently between one and four GB.

4 Local Volatility Model on GPU

Based on the PricingCatalyst™ framework, QuantCatalyst Inc (2008), which allows an efficient implementation of pricing algorithms on GPU without going through the steep learning curve of GPU programming, we implemented the pricing of complex equity basket options with different barrier and payoff types. We chose a local volatility model for every underlying asset and constant correlations between them. Local volatility models as introduced by Dupire (1994) and Derman and Kani (1994) are very popular among practitioners. They provide a relatively straightforward approach to price exotic options consistently with the volatility smile. Because local volatility models can fit the entire implied volatility surface, traders use it also to hedge exotic options with the underlying stocks and the vanilla options markets, even though the model does not properly capture the true market behaviour. As analyzed in Engelmann, Fengler and Schwendner (2006) the hedging performance can be considerably improved if one hedges against the movements of the implied volatility surface.

The computationally challenging parts of the valuation are the Monte Carlo simulation of the local volatility process and the payoff evaluation along the paths. The calibration of the local volatility surfaces, for instance along the lines of Andersen and Brotherton-Ratcliffe (1997), also merits GPU acceleration. It has to be executed once for every underlying asset of the basket. If Greeks have to be calculated as well, we need to recompute the local volatility surfaces several times to implement a sticky strike or sticky moneyness calibration.

Our implementation takes advantage of the texture memory provided by the GPU. Texture memory is accessible as a one, two or three-dimensional lookup table, with interpolation between the nodes realized in hardware. In Figure 3 the bilinear interpolation is depicted. Assuming the texture map to be defined on an integer lattice, the expression

$$f(u,v) = (1-\alpha)(1-\beta)f[floor(u), floor(v)] + \alpha(1-\beta)f[ceil(u), floor(v)] \\ +(1-\alpha)\beta f[floor(u), ceil(v)] + \alpha\beta f[ceil(u), ceil(v)]$$

interpolates for arbitrary real coordinates u, v, where α and β are the fractional parts of the coordinates. This formula is implemented on the hardware level within the texture addressing unit. What would correspond to four separate memory fetches and several integer and floating point operations on a normal architecture happens on the GPU at the speed of a single memory access.

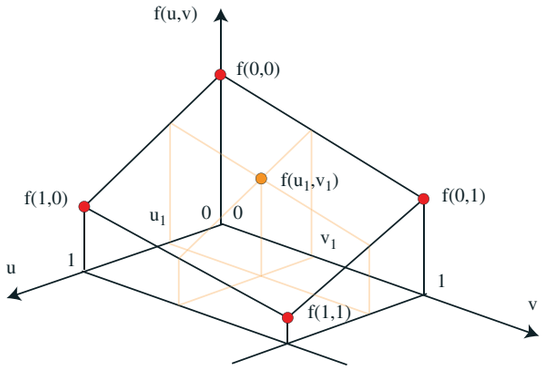

Figure 3: Bilinear Texture Interpolation.

When simulating from a local volatility model, the local volatility is a function of time to maturity and stock value. The function is constant during the sampling of the paths and an interpolation in both dimensions is sensible. Using the texture memory of the GPU for the local volatility mapping speeds up the simulation by orders of magnitude compared to an explicit interpolation procedure. When it comes to the implementation of interest rate models, note that texture memory can also handle three-dimensional structures and is therefore a perfect tool to represent the swaption volatility cube.

The timings in Figure 4 show the price calculation with Monte-Carlo simulations of 10'000 samples on a single GPU. The paths are modelled as correlated Brownian motions which have local volatility. These are simulated on a time grid with daily resolution over a period of one year. The local volatility function is stored in texture memory with a resolution of 100 values in each dimension. Dividends are possible at each time step without affecting the performance. All our examined payoffs are path-dependent. The worst-of instrument pays the nominal in case none of the equities ever hits its barrier or all of the equities quote above the initial value at maturity. If any of the barriers is hit, the stock with the worst relative performance is delivered instead of the nominal. A coupon is paid in any case. The worst-of with the auto-callable feature has additional upper barriers for each of the equities. If all equities of the basket quote at 105% of their initial value, the nominal and the coupon are paid back immediately. This requires additional checks along the paths and daily monitoring.

Pricing of baskets with three or four equities takes less than 20ms. The runtime increases with increasing basket size. Due to memory alignment the runtime increases step-wise in buckets of four. Pricing baskets of up to eight equities takes between 32ms and 37ms. Baskets of 9 and 10 equities can be priced in 45ms. Comparing the execution times of the two payoff types suggests that path generation is more costly in terms of computing power than path evaluation and that therefore even more complicated payoff structures can be priced with only a minor increase in runtime. The pricing algorithms run 25 to 50 times faster on a GPU than on a high-end CPU, depending on the complexity of the payoff, the basket size, and the number of simulations. To provide a fair comparison we used Intel's Math Kernel Library to generate the random numbers. By just comparing

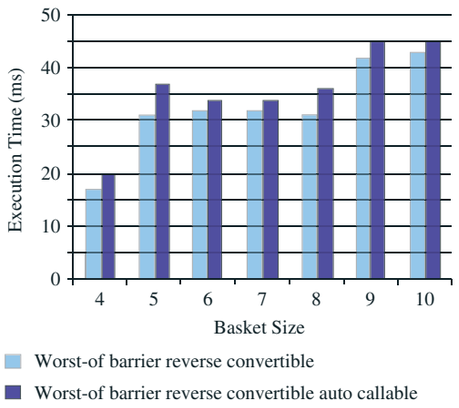

Figure 4: Timings.

the peak performance of a GPU and a high-end CPU we would not anticipate such an enormous performance gain. It can only be justified by the distinct architectures which allow fundamentally different programming models. The GPU design is so much better capable to cope with data-parallel problems. Additional features like texture memory and hardware interpolation increase the GPU efficiency further.

5 Conclusions

We may ask the question if GPUs are just another trendy fashion or if they will cause a real breakthrough in financial computing. Let us assume that you want to price complex derivatives on the basis of a sophisticated model. A speedup of a factor of two to five is an incremental change. It can be achieved with a relatively minor effort. For instance changing the compiler might already do the job. One waits a little less or increases accuracy slightly. To make your pricing requests five to ten times faster already requires a substantial effort. You need to optimize your code, switch to a higher performing numerical scheme, or in the worst case you might need to install ten times more computing nodes. Pricing routines that took a few seconds are now done at real time, or you can process significantly larger volumes. However, it does not fundamentally change the way you do business. Things change fundamentally if the calculation speed becomes 50 to 100 times faster. In this case a disruptive change is likely to happen. For such a change you might be willing to buy new infrastructure, rewrite entire applications and think about new work flows, products and services. You spin new innovations, for instance:

- Prices and Greeks of complex structured products are available in real-time, instead of delayed by minutes. Agile movements and market making of complex structured products would be possible when the markets play rough.
- Interactive structuring of multi-asset barrier options, with barrier and strike levels set to meet the investor's risk profile can attract new clients.
- On a hybrid CPU/GPU cluster you can refine your risk analysis. Nested Monte-Carlo for a full revaluation risk calculation is in reach if performance demanding pricing algorithms are implemented to run on GPU acceleration boards.

It is now up to traders, practitioners, and researchers to exploit the capabilities and advantages of the new GPU acceleration boards. We believe they have the potential to significantly push the limits of computational finance and to create interesting new business opportunities for the financial services industry.

FOOTNOTES & REFERENCES

Christoph Bennemann and Mark Beinker are senior manager and partner, respectively, at d-fine, the consulting firm. Daniel Egloff and Michael Gauckler are managing directors and founding partners of QuantCatalyst, a software company specialising in cluster and GPU solutions for the finance industry.

Inquiries regarding this work should be directed to Daniel Egloff (daniel.egloff@quantcatalyst.com)

Andersen, L. B. G. and Brotherton-Ratcliffe, R.: 1997, The equity option volatility smile: An implicit finite-difference approach, *Journal of Computational Finance* 1(2), 5–37.
Derman, E. and Kani, I.: 1994, Riding on a smile, *Risk* 7(2), 32–39.
Dupire, B.: 1994, Pricing with a smile, *Risk* 7(1), 18–20.
Engelmann, B., Fengler, M. and Schwendner, P.: 2006, Better than its reputation: An empirical hedging analysis of the local volatility model for barrier options, *Technical report*, Sal. Oppenheim jr. & Cie., Frankfurt, Germany.
Intel: 2008, Intel® Xeon® E5472 processor, *Specification*, http://www.intel.com/performance/server/xeon/hpcapp.htm.
NVIDIA: 2007, CUDA™ Compute Unified Device Architecture Programming Guide, *Technical Report* 1.1, NVIDIA® Corporation http://www.nvidia.com/object/cuda_home.html.
NVIDIA: 2008, NVIDIA® Tesla™ C870, *Specification*, http://www.nvidia.com/object/tesla_c870.html.
QuantCatalyst Inc.: 2008, PricingCatalyst™, *Software documentation*, http://www.QuantCatalyst.com/.