



# Using GPU to Compute Options and Derivatives

## Introduction

Algorithmic Trading has created an increasing demand for high performance computing solutions within financial organizations. The actors of portfolio management and risk assessment have the obligation to increase their computing resources in order to provide competitive models for financial management and pricing financial instruments.

GPU Stands for "Graphic Processing Unit". GPU processing (or Stream Processing) defines a class of algorithms that uses Massively Parallel Architectures as a starting point for Software Design. Computational performance is no longer coming from increase in clock speed, but in leveraging the astonishing performances of those massively parallel architectures.

This document describes how GPU technologies can improve efficiency and throughput in the computation of Option Prices. It presents a number of benchmarks specific to financial analysis in order to demonstrate the tremendous advantage of porting trading algorithms to the GPU platform.

The benefits of porting computing algorithms to the GPU are:

- Improve application performance from 10 to 700 fold.
- Quickly build and deliver massively parallel applications.
- Leverage existing grid infrastructure by simply adding low cost graphic hardware to the units.
- Applications built to run on GPU hardware have virtually no limits in their scalability.

## GPU Implementation of Option Pricing Algorithms

### *Option Models and Definitions*

Used terminology and concepts:

- Options derive their value from an *underlying asset* (or security).
-

- A *Call Option* contract gives the *holder* the right, but *not the obligation*, to buy the underlying asset for a *strike price*, on the *expiration date*.
- A *Put Option* gives the *holder* the right to sell the underlying asset at a *strike price*, on the *expiration date*.

Both the strike price ( $X$ ), the price of the underlying asset at the expiration date strongly influence the future value of the option, and thus its value now.

Under certain assumptions, one can statistically model the asset's future price fluctuation using its current price ( $S$ ), and its volatility ( $v$ ), which describes how widely the price changes over time. Additional factors influence the price of the option:

- The time to expiration date ( $T$ ). Longer terms imply a wider range of possible values for the underlying asset on the expiration date, therefore more uncertainty about the value of the asset when it expires.
- The riskless rate of return ( $r$ ). The value of an asset at some future time ( $t$ ),  $0 \leq t \leq T$ , must be discounted by multiplying it by  $e^{-rt}$ .
- Exercise restriction.
  - European options may be exercised only on the day the option expires.
  - American options may be exercised at any time up and including the expiration date. This flexibility means that an American option will be priced at least as high as the corresponding European option.
- Other factors, such as dividend payouts, can also be incorporated. We however limit our model to the preceding considerations.

### Experiment Methodology

- Hardware and Software setup
  - Hardware
    - CPU: AMD Athlon 64 X2 Dual Core Processor 4200+, 2.21 GHz, 2Gb of RAM
    - GPU: NVIDIA Ge Force 8800 GT, 1.5 GHz, 1Gb of RAM
  - Software
    - Microsoft Windows NT 5.1.2600 Service Pack 3
    - Microsoft Visual Studio 2005
    - NVIDIA CUDA 2.0 Beta 2 Driver v177.35
    - NVIDIA CUDA 2.0 Beta 2 SDK
- Software for Option Pricing Experiment:
  - The same pricing modules have been coded in C++, for the CPU version and in CUDA language (NVIDIA's "Compute Unified Device Architecture") for the GPU version.
  - The C++ version implements double precision floating numbers, the CUDA version implements single precision floating numbers. The L1 Norm of the difference between CPU and GPU results is monitored for every option calculation, in order to control the precision.

## Black-Scholes Model

### Description

The Black-Scholes model computes the value ( $V$ ), on an European call option as (the formula for a put option is similar):

$$V = S \times \text{CND}(d_1) - Xe^{-rT} \times \text{CND}(d_2)$$

where,

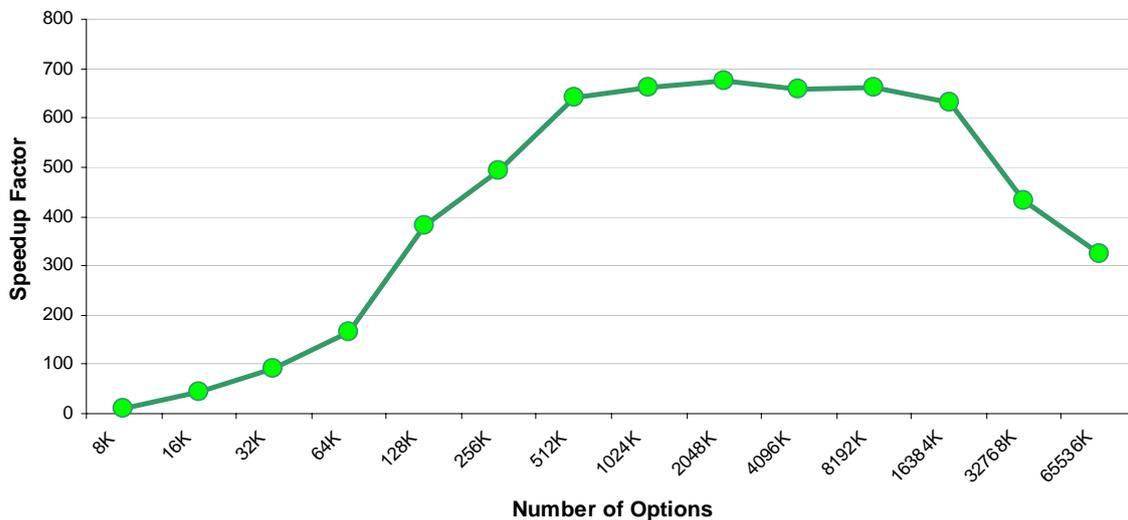
$$d_1 = \frac{\log\left(\frac{S}{X}\right) + T\left(r + \frac{v^2}{2}\right)}{v\sqrt{T}}$$

$$d_2 = d_1 - v\sqrt{T}$$

The Cumulative Normal Distribution function,  $\text{CND}(x)$ , gives the probability that a normally distributed random variable will have a value less than  $x$ . There is no closed-form expression for this function, and it must be evaluated numerically. A polynomial function is used to approximate it.

### Results

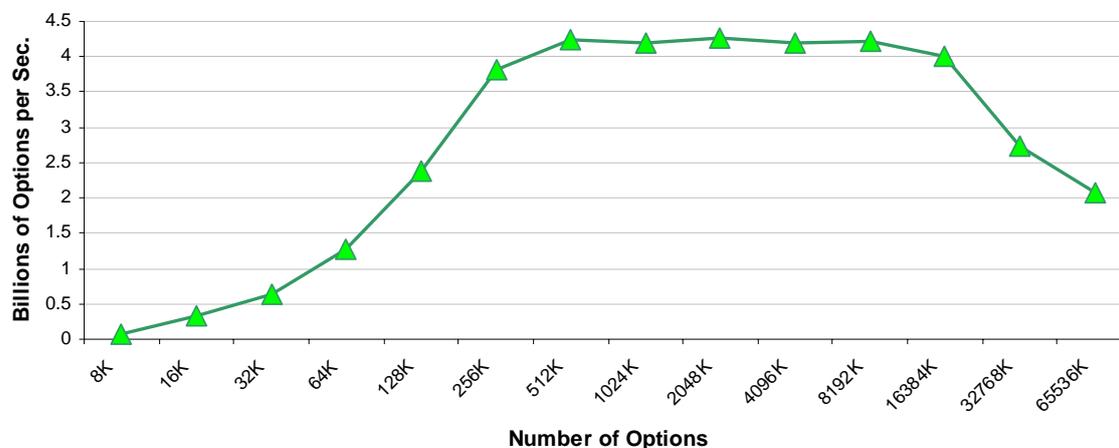
#### Speedup, function of the total number of Options



**Figure x-xx.** CPU vs. GPU Performance of Black-Scholes Option Pricing.

The horizontal axis gives the total number of pricing problems to be solved. The vertical axis represents the speedup factor between CPU and GPU computation times. The GPU significantly outperforms the CPU, when enough options are priced to allow for high resource utilisation, and to allow the constant costs of GPU buffer management to be amortized. The slight decrease in performances of the GPU software after around 10 Mega options, is due to the fact that the whole data set is loaded at once to the GPU, reaching its memory bandwidth limits. The production version of this software should consider those limits and streamline the memory transfers to maintain the performances at the optimal level.

### Options Computed per second, function of the total number of Options



**Figure x-xx.** CPU vs GPU Performance of Black-Scholes Option Pricing. The horizontal axis gives the total number of pricing problems to be solved. The vertical axis represents the number of options computed per second. This graph presents a curve that has a similar shape of the preceding.

## Lattice Models

The Black-Scholes equation provides a convenient analytical method to compute the price of European options. However, it is not suitable for pricing American options, which can be exercised at any time prior to the date of expiration, or the Bermudan option which can be exercised at various points. In fact, there is no known closed-form expression that gives the fair price of an American option under the same set of assumptions used by the Black-Scholes equation.

Another family of option pricing models is Lattice models. These models use a dynamic programming approach to derive the value of an option at time 0 (now) by starting at time  $T$  (expiration date) and iteratively stepping "backward" toward  $t = 0$  in a discrete number of time steps ( $N$ ). This approach is versatile and simple to implement, but it can be computationally expensive due to its iterative nature. The binomial lattice model presented here can be used to compute both European and American options.

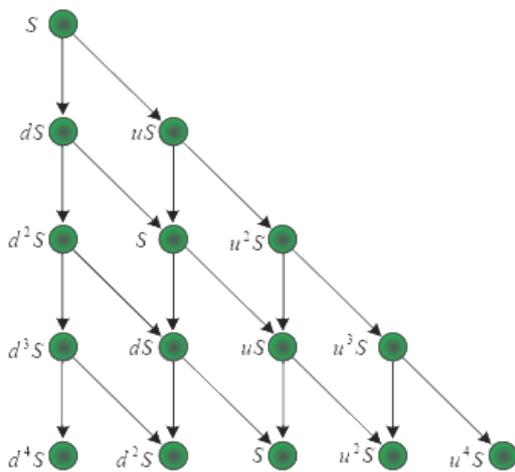
### The Binomial Model

The binomial model is so named because it assumes that if the underlying asset has a price  $S_k$  at a time step  $k$ , its price at step  $k+1$ , can take only two possible values  $uS_k$  and  $dS_k$ , corresponding to an "up" or "down" movement in the price of the stock. Typically,  $u$  is calculated by assuming that during a small period  $dt$ , the change of the asset value is normally distributed, with a standard deviation that is equal to  $S_k v \sqrt{dt}$ .

If  $d$  is chosen such that  $u \times d = 1$  ( $u = e^{rdt}$ ,  $v = e^{-rdt} = \frac{1}{u}$ ), the possible asset prices during the lifetime of the option form a tree. The set of nodes at depth  $k$  from the root represent

the range of possible asset prices at time,  $t = k \times dt$ , where  $dt = \frac{T}{N}$ . The nodes at  $k = N$  ( $t = T$ ), represent the range of prices the asset might have at the expiration date. The pseudo-probability  $P_u$  that the asset price will move "up" at any given time, is computed assuming a "risk-neutral" world, which implies that the expected return on the underlying asset is equal to  $r$ . The pseudo-probability of the "down" movement is given by  $P_d = 1 - P_u$ . Even if  $P_u$  is not a true probability, one will treat it as such.

$$(uP_u + d(1 - P_u))S_k = e^{rdt} S_k \Rightarrow P_u = \frac{e^{rdt} - d}{u - d}$$



**Figure x-xx.** Binomial Tree for 4 time steps,  $u \times d = 1$

From the binomial tree representation, one can iteratively derive the option price for each node of the tree, starting at the leaves. At each leaf of the tree, call and put option price is derived:

$V_{call} = \max(S - X, 0)$  : if the market price  $S$  at expiry date is greater than strike price  $X$ , a call option returns  $S-X$  profit, for the same day sale transaction, or zero otherwise.

$V_{put} = \max(X - S, 0)$  : if the market price  $S$  at expiry date is lower than strike price  $X$ , a put option returns  $X-S$  profit or zero otherwise.

After computation of all the possible options prices at expiry date, the algorithm moves back to the root, using the formula:

$$V_t = (P_u \times V_{u,t+1} + P_d \times V_{d,t+1}) \times e^{-rdt}$$

Where,  $V_t$  is the option price for one node at time  $t$ , and  $V_{u,t+1}$  and  $V_{d,t+1}$  are the prices of its two child nodes.

## Results

### Average Speedup, function of the number of time steps

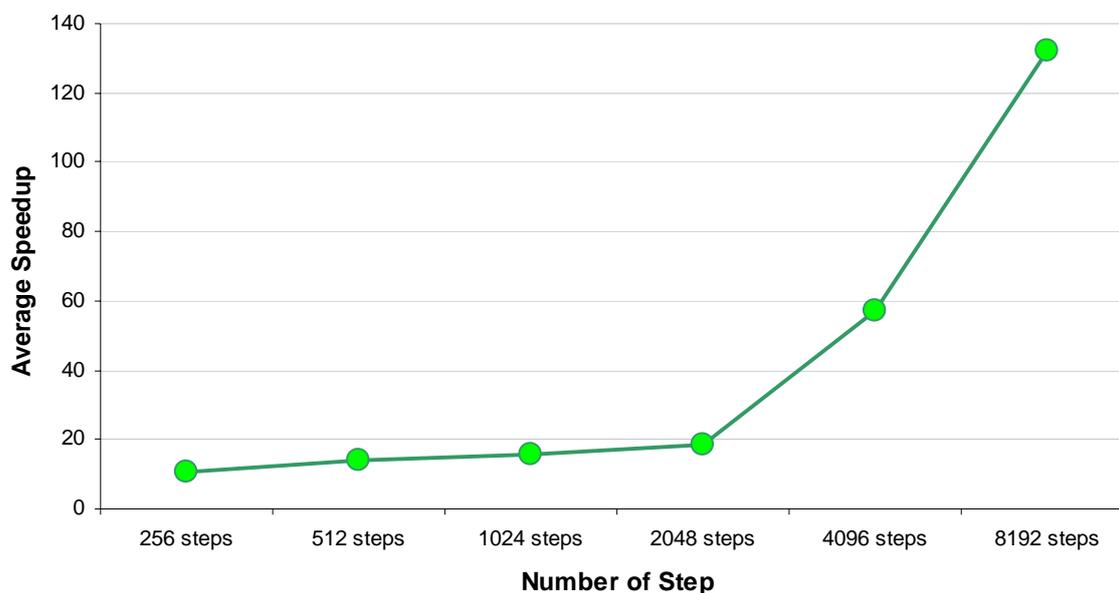


Figure x-xx. Speedup

The horizontal axis gives the number of steps computed between "now" and the option expiry date. The vertical axis gives the speedup between the CPU and GPU code. In this analysis, one can measure the real impact of the GPU acceleration, when the code is written to handle recursive iterative computations.

## Conclusion

Both Black-Scholes model and lattice model for option pricing are basic building blocks of computational finance. This article shows how the GPU can be used to price these models more efficiently than the CPU. As the GPUs continue to become faster more quickly than CPUs, the difference in performance between the two is likely to grow for this type of application.

### Financial Applications demanding great computing performances

- **Monte Carlo Simulation**

Monte Carlo and Quasi-Monte simulations analysis are also widely used for implementing financial models in the industry. These simulations have many advantages, including the ease of implementation and applicability to multi-dimensional problems commonly encountered in finance. However, calculations using Monte Carlo techniques are very time consuming due to the need for simulating many trajectories with multiple parameters. These algorithms are therefore well suited for the GPU, because they rely on running a large number of independent trials and computing the overall estimates based on all the trials

together. The independent estimates are computed in parallel, and the final result is computed by reductions.

- **Portfolio Optimization**

Using GPUs, analysts run their models on parallel servers, clusters, and grids to optimize thousands of individual portfolios overnight based on the previous day's trading results.

- **Valuation of Financial Derivatives**

Valuing financial derivatives is computationally intensive, and requires large amounts of computer time. A firm may need to value and compute hedge strategies for hundreds of thousands of policy holders in its portfolio on a regular and timely basis. Software written for GPU enables analysts to explore new valuation methodologies using high performance computing techniques to run billions of complex scenarios.

- **Detection of Credit Card Fraud**

The rise of identity theft together with the popularity of online shopping has resulted in a huge increase in credit card fraud. As thieves become increasingly shrewd in exploiting security weaknesses, banks and credit card companies need to be extremely agile to stay ahead of them. GPUs enable a bank to run more sophisticated fraud detection algorithms against tens of millions of credit card accounts.

- **Hedge Fund Trading**

In balancing a large portfolio of stocks, analysts need to search for short- and long-term patterns, identify correlations between securities, and develop forecasts. Intense computations are required against terabyte-sized "tick data" databases – potentially a decade or more of trading data for thousands of securities. Using GPU software allows faster reaction time to market conditions, enabling analysts to evaluate more sophisticated algorithms that take into account larger data sets.

- **Risk Analysis**

GPU is a natural fit for creating and running the multiple simulations (each with numerous scenarios and variables) needed to accurately assess the degree of risk in stock portfolios, in a set of financial contracts, or other investment vehicles. In particular, "outlier" cases require a great many simulations to capture the level of risk.

## References

1. Black, Fisher, and Myron Scholes. 1973. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy* 81(3), pp. 63-654.
2. Cox, John C., A. Ross and Mark Rubinstein. 1979. "Option Pricing: A Simplified Approach." *Journal of Financial Economics* 7: 229-263.
3. General Purpose Computation on Graphic Processing Units reference website, <http://www.gpgpu.org>.