

Parallelized Trinomial Option Pricing Model on GPU with Cuda.

Grégoire Jauvion, Ecole Centrale Paris*
Tuan Nguyen, Arbitragis Trading, Paris, France.†

August 7, 2008

Abstract

The need for speed has always been an endless quest in the field of derivatives pricing. Real-time risk analysis of a book of derivatives is extremely demanding in terms of computing power. In the past, one of the trick-of-the trade was to pre-compute vectors of greeks for each derivatives product and use interpolations in real-time in order to compute the generated risk. This is now a thing of the past: we are now able to compute prices for derivatives with an increase of speed of up to two orders of magnitude, allowing real-time risk analyses.

This article describes the trinomial option pricing model (or Cox-Ross-Rubinstein model) parallelized on GPU (Graphical Processing Unit) which is a simplified version of the model used by Arbitragis Trading to price options and to compute real-time scenario risk analyses. The method described in this article is an extension of the parallel pattern of the binomial method implemented by Nvidia [1].

1 Definitions

A derivative is a financial instrument whose value depends on the value of other variables [3]. Derivatives have become very important in the world of finance: forward contracts and many different types of options are traded by financial institutions. We will later focus on options.

An option gives the right, but not the obligation, to buy (for a *call* option) or sell (for a *put* option) a specific amount of a given stock, commodity, currency, index, or debt, at a specified price during a specified period of time. The price in the contract is known as the *exercise price* or *strike price* and the date in the contract is known as the *maturity*. For a call option, the profit made at the exercise date is $S - X$, where S is the underlying price at exercise date and X the strike price. For a put option, the profit made at the exercise date is $X - S$. There are two basic types of options: *american options* can be exercised at any time up to the expiration date, and *european options* can only be exercised at maturity.

An option is of course not free: its price (called the *option premium*) is the rewards expectancy of an optimal option's exercise. We will see next how to compute this price using the trinomial pricing model.

2 Trinomial pricing model

The binomial pricing model is a useful and popular technique for pricing a stock option. We will describe an improvement of this model : the trinomial model or the Cox-Ross-Rubinstein model.

The trinomial model is a model where the price of the option's underlying asset is monitored over successive short periods of time from today to maturity. At each time step, the price can either move

*Email: gregoire.jauvion@student.ecp.fr

†Email: tuan.nguyen@arbitragis.com

up, remain unchanged or move down with probabilities P_u , P_e and P_d . All possible underlying's prices eventually form a trinomial tree. Each node of this tree has three child nodes of values $u.S$, S and $d.S$ (see Figure 1). Notice that the root node's value is S_0 , which is the value of the underlying at time 0.

Let us see now how to choose the probabilities and the factors of upward and downward movements. The trinomial pricing model is based upon the Black Scholes model, which is a model of price variation in continuous time of financial instruments. We will not see this model in details, but it is important to understand that in this model, the stock price depends on two variables :

- the riskless rate of return r , which is the annual interest rate of *risk-free* investment.
- volatility σ , which is a variable quantifying the expected variation of the underlying returns between now and the option's maturity.

The hypotheses in the Black Scholes model allow the computation of P_u , P_e , P_d , u and d . We find $u = e^{\sigma\sqrt{3\Delta T}}$ and $d = e^{-\sigma\sqrt{3\Delta T}}$, where ΔT is the time difference between two consecutive time steps. Notice that $u.d = 1$, which involves that after N time steps, the vector of prices contains $(2N + 1)$ nodes (and not 3^N).

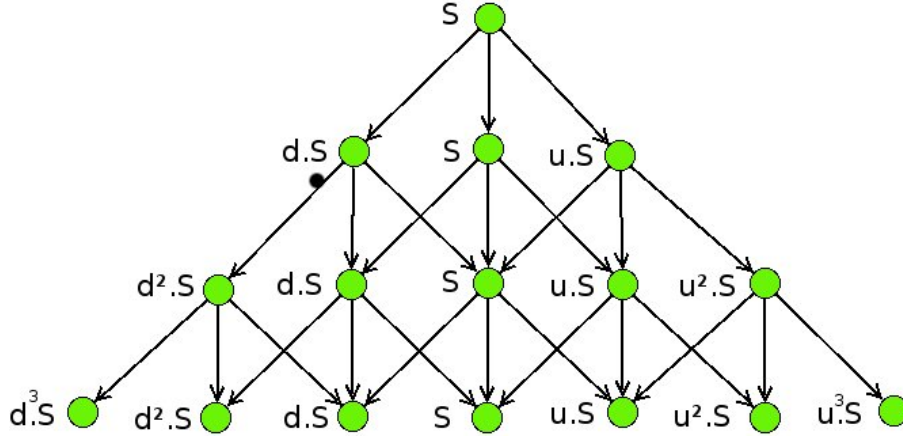


Figure 1 : the trinomial tree

Likewise, we compute probabilities which respect to Black Scholes Model's hypothesis. We will use later :

$$\begin{cases} P_u = \frac{\Delta T(\sigma^2 + \mu(\mu\Delta T + \sigma\sqrt{3\Delta T}))}{6\sigma^2\Delta T} \\ P_d = \frac{\Delta T(\sigma^2 + \mu(\mu\Delta T - \sigma\sqrt{3\Delta T}))}{6\sigma^2\Delta T} \\ P_e = 1 - P_u - P_d \end{cases} \quad \text{with } \mu = \left(r - \frac{\sigma^2}{2}\right) \quad (1)$$

From this trinomial representation, we will see how to compute the option's value at time 0. We will compute the option's value at each tree node, and the option premium will be the option's value at the root tree node.

At each leaf of the tree, deriving the option price is simple: for a call, $V_{call} = \max(S - X, 0)$ and for a put, $V_{put} = \max(X - S, 0)$, where X is the strike price and S the stock price at the leaf. Indeed, the option is exercised only if it is *in the money* (if $S > X$ for a call, and $S < X$ for a put). Otherwise, the option is not exercised.

Having calculated all possible option prices at maturity, we start looping the tree backwards (see Figure 2) by using the following formula: $V_n = (P_d.V_{d,n+1} + P_e.V_{e,n+1} + P_u.V_{u,n+1}).e^{-r.\Delta T}$, which gives the price at a given node, where $V_{d,n+1}$, $V_{e,n+1}$, and $V_{u,n+1}$ are the prices of its three child nodes.

If we want to price an american option, we use the following formulas:

$$\begin{cases} V_{call,n} = \max((P_d \cdot V_{d,n+1} + P_e \cdot V_{e,n+1} + P_u \cdot V_{u,n+1}) \cdot e^{-r \cdot \Delta T}, S - X) \\ V_{put,n} = \max((P_d \cdot V_{d,n+1} + P_e \cdot V_{e,n+1} + P_u \cdot V_{u,n+1}) \cdot e^{-r \cdot \Delta T}, X - S) \end{cases} \quad (2)$$

Indeed, $(P_d \cdot V_{d,n+1} + P_e \cdot V_{e,n+1} + P_u \cdot V_{u,n+1}) \cdot e^{-r \cdot \Delta T}$ is the expected value of the option, which is compared at each time step to its current value. In fact, technically, we are not pricing an *american option* but a *bermudean option* which is an option that can be exercised at several discrete times (traders have a geekish humor and an option which has both an european feature because it can only be exercised at maturity and an american feature because it can also be exercised at anytime until maturity is somewhere in between... Guess where ? In the Bermudes islands, of course !). The more time steps, the better the approximation. However, we will see later that using too many time steps can be a drawback.

Now that we know the option values at the leaves as well as the recurrence formula, we can compute the value corresponding to the root node: the option premium is the option's value at time 0.

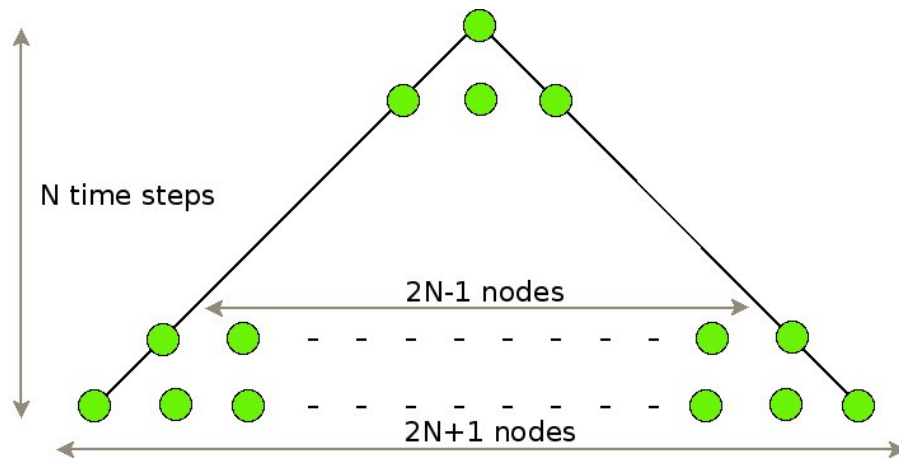


Figure 2: looping the tree backwards

3 Implementation on GPU with Cuda

The CPU implementation of the trinomial method is easy. All values at leaves are generated using the Black Scholes model: $S_n = S_0 \cdot e^{(n-N)\sigma\sqrt{3}\Delta T}$ where N is the number of time steps.

We will see now how to implement the trinomial pricing method on a GPU.

3.1 A GPU: a brief description

A GPU is composed of several multiprocessors (at the time of print, this number typically ranges between 4 and 16 multiprocessors), themselves containing several physical devices. The G80-chip on a Nvidia 8800 Ultra graphics card has 16 multiprocessors with 8 processors each, for a total of 128 processors. Each multiprocessor can process parallel groups of threads, called warps. These are generalized floating-point processors capable of operating on 8,16 and 32-bit integer types and 16 and 32-bit oating point types. Each multiprocessor has a memory of 16 KB size that is shared by the processors within the multiprocessor. Access to a location in this shared memory has a latency of only 2 clock cycles allowing fast nonlocal operations. The processors are clocked at 1.6GHz, giving the GeForce 8800 GTX a tremendous amount of oating-point processing power. Assuming 2 oating point operations per cycle (one addition and multiplication) we obtain $2 \times 1.6 \times 128 \text{ GFLOPS} = 410 \text{ GFLOPS}$. Each multiprocessor has a Single Instruction, Multiple Data architecture (SIMD).

Programming on the hardware is done by using Cuda, a toolkit developed by Nvidia that allows a massively parallel programming of any kind of algorithm. However, the hardware structure does dictate

a certain way to parallelize the problem: it is easy to synchronize each multiprocessor (to synchronize the threads) but it is harder to synchronize the multiprocessors (it takes much more time).

We can make an analogy with parallel computing on MPI: bottlenecks in terms of computational speed are set by the interconnection delay between cluster nodes on the local ethernet network. The bottleneck for GPU computing lies in the transfer of information into the memory of the GPU and at the synchronization between multiprocessors. Additional latency is added by the synchronization of threads required between different multiprocessors.¹

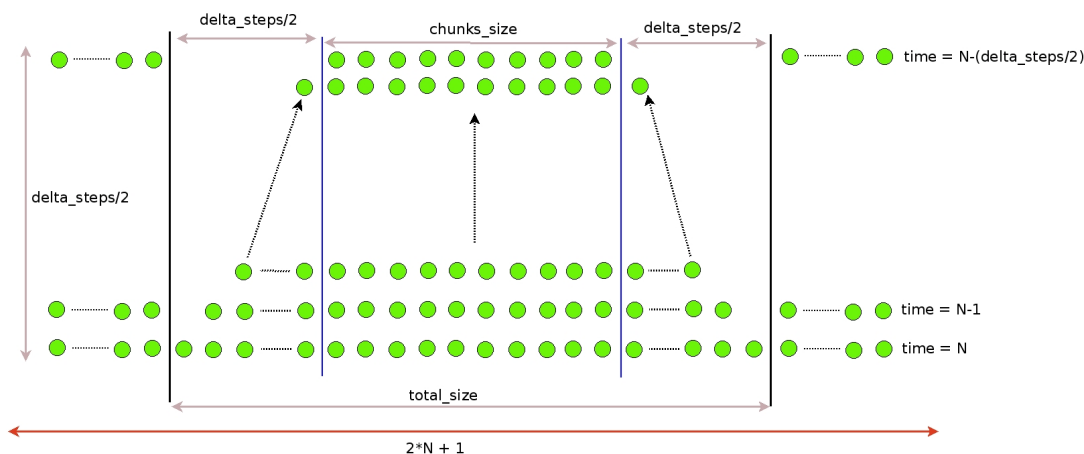
3.2 Description of the *Geometric Decomposition Parallel Pattern*

In order to make a full use of the computation power of the GPU, we decided to use a mix of *task* and *data parallelism*. First, we parallelize the task by pricing vectors of options: each multiprocessor of the GPU prices one option. In order to parallelize the data structure, we decomposed the tree of prices on each processor by using the *Geometric Decomposition Parallel Pattern* [2]. This pattern involves the decomposition of the global data structure into substructures or "chunks".

The idea is to exploit the recursive pattern of the data structure and decompose the data structure into contiguous substructures. The problem then becomes embarrassingly parallel. The optimal number of chunks remains an intricate issue to solve. Even though it is probably possible to mathematically derive the optimum granularity for the data decomposition, we tested with a range of chunk sizes in order to determine empirically the best size for a given GPU. One of the consequences of the hardware constraint is that it is optimal, in terms of computation time, to price short-dated options and long-dated options with the same number of steps. However, in reality, long-dated options should require more time steps in order to maintain a good level of precision.

Our approach is to load the leaves' values in the shared memory (high-speed memory) and then to parallelize the looping of the tree to compute the option's value at time 0. However, we have to take into account the limited size of the shared memory: the average size of the shared memory of a GPU is about 4000 floating points. If we use 1000 time steps, the price vector contains 2001 prices at maturity. We will see that we need to store other data to loop through the tree, and 4000 floating points is not enough to compute the price. Consequently, it is necessary to process nodes of the tree in chunks (of size *chunks_size*) that fit into the shared memory.

To loop backwards one chunk of the tree of size *chunks_size* of $\frac{\text{delta_steps}}{2}$ time steps, we have to consider $\text{chunks_size} + \text{delta_steps} = \text{total_size}$ nodes. After having looped backwards each part of the tree of $\frac{\text{delta_steps}}{2}$ time steps, we bring them together and continue the computation (see Figure 3 for more details).



¹The well-known Amdahl law of parallelism can obviously be applied for a GPU

Figure 3

The drawback of this computation method is that there is a redundancy in computations at each time step (see Figure 4). We can easily see that the number of redundant computations is proportional to $\frac{\text{delta_steps}}{\text{chunks_size}}$.

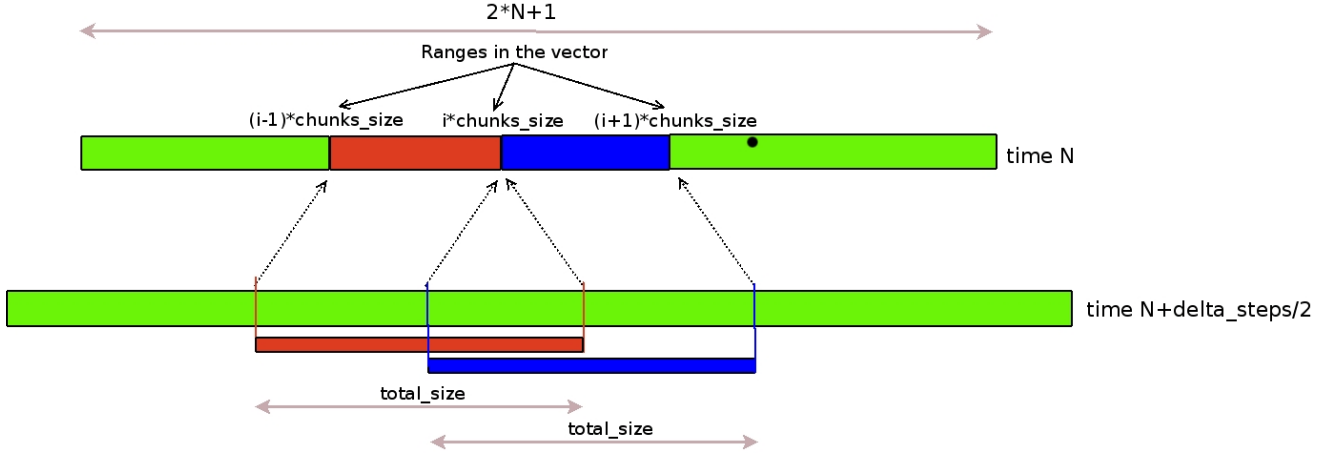


Figure 4: Description of the redundant computations: the common nodes of the blue vector and the red one are computed twice

4 Benchmark: CPU vs GPU

After finding a parallel pattern to implement the trinomial method on GPU, let us describe the results of the algorithm. Two aspects of the algorithm will be seen: pricing accuracy and execution time. The GPU used in the following tests is a Nvidia Geforce 8800 GTX, and the CPU is a Quad Core Duo 2 3.2 Ghz.

The precision of the GPU is sufficient in this kind of applications. Indeed, for one option's pricing with 1000 time steps, the relative precision between the price given by our algorithm and the price given by the same algorithm implemented on a CPU is better than 10^{-4} .

The most interesting is to compare execution times between CPU and GPU (see Table 1).

Number of options	Number of time steps	GPU time (ms)	CPU time (ms)	Speed up factor
1	256	1.17	2.4	2.1
	1024	17.1	37	2.2
4	256	1.27	9.6	7.6
	1024	18.7	147	7.9
16	256	1.31	38.3	29.2
	1024	18.8	591	31.4
64	256	5.17	153	29.6
	1024	75.0	2370	31.6

Table 1: CPU and GPU execution times. The CPU was used with only one non-threaded pricing.

Table 1 gives CPU and GPU execution times for different values of the number of time steps and the number of options priced. Notice that the speed-up of the GPU increases with the number of time steps and the number of options priced. This is a general characteristic of the GPU: the more complex

the computation, the more efficient the GPU.

Moreover, notice that the GPU execution time is almost the same to price 1 option or 16 options. Indeed, one option's pricing is done on one multiprocessor, and the GPU used has 16 multiprocessors. Execution time within the GPU is consequently the same to price 1 option or to price 16 options. However, the time used to copy the data from the memory of the CPU to the memory of the GPU increases with the number of options priced.

5 Conclusions and future developments

We managed to parallelize the Cox-Ross-Rubinstein pricing model on a GPU, with a speed-up factor of up to 30. This has to be considered as a toy project. Within Arbitragis Trading, we have developed for ourselves and for our clients a more advanced version of this pricing model that allows us to compute risk analyses in real-time on a portfolio of equity derivatives with dedicated hardware.

This model remains to be improved as it is slightly suboptimal because of the redundancy of computations which becomes a larger issue as the number of time steps increases.

References

- [1] Victor Podlozhnyuk: Binomial option pricing model
Nvidia, 2007
- [2] Timothy G. Mattson, Beverly A. Sanders, Berna L, Massingill: Patterns for Parallel Programming
Addison-Wesley, 2005
- [3] John Hull: Options, Futures and other Derivatives (Fourth edition)
Prentice Hall, 2000