# Binomial option pricing model

Victor Podlozhnyuk
vpodlozhnyuk@nvidia.com

September 2013

# Document Change History

| Version | Date | Responsible | Reason for Change |
|---|---|---|---|
| 0.9 | 2007/03/19 | vpodlozhnyuk | Initial release |
| 1.0 | 2007/04/05 | Mharris | Grammar and clarity fixes. |
| | | | |
| | | | |

# Abstract

The pricing of options is a very important problem encountered in financial engineering since the creation of organized option trading in 1973. As more computation has been applied to finance-related problems, finding efficient ways to implement option pricing models on modern architectures has become more important. This sample shows an implementation of the binomial model in CUDA.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com

# Introduction

The most common definition of an *option* is an agreement between two parties, the *option seller* and the *option buyer*, whereby the option buyer is granted a right (but not an obligation), secured by the option seller, to carry out some operation (or *exercise* the option) at some moment in the future. The predetermined price is referred to as *strike price*, and future date is called *expiration date*. (See Kolb & Pharr. [1])

Options come in several varieties:

A *call option* grants its holder the right to *buy* the *underlying asset* at a *strike price* at some moment in the future.

A *put option* gives its holder the right to *sell* the *underlying asset* at a *strike price* at some moment in the future.

There are several types of options, mostly depending on when the option can be exercised. European options can be exercised only on the expiration date. American-style options are more flexible as they may be exercised at any time up to and including expiration date and as such, they are generally priced at least as high as corresponding European options. Other types of options are path-dependent or have multiple exercise dates (Asian, Bermudian).

For a call option, the profit made at exercise date is the difference between the price of the asset on that date and the strike price, minus the option price paid. For a put option, the profit made at exercise date is the difference between the strike price and the price of the asset on that date, minus the option price paid.

The price of the asset at expiration date and the strike price therefore strongly influence how much one would be willing to pay for an option.

Other important factors in the price of an option are:

- **The time to the expiration date, T**: Longer periods imply wider range of possible values for the underlying asset on the expiration date, and thus more uncertainty about the value of the option.
- **The riskless rate of return, r,** which is the annual interest rate of bonds or other "risk-free" investment: Any amount P of dollars is guaranteed to be worth $P \cdot e^{rT}$ dollars T years from now if placed today in one of theses investments or in other words, if an asset is worth P dollars T years from now, it is worth $P \cdot e^{-rT}$ today.

This example demonstrates a CUDA implementation of the binomial model for European options.

# Binomial option model

The binomial option pricing model is an iterative solution that models the price evolution over the whole option validity period. For some types of options, such as the American options, using an iterative model is the only choice since there is no known closed-form solution that predicts price over time.

More precisely, the binomial model represents the price evolution of the option's underlying asset as the binomial tree of all possible prices at equally-spaced time steps from today under the assumption that at each step, the price can only move up and down at fixed rates and with respective pseudo-probabilities $p_u$ and $p_d$. In other words, the root node is today's price, each column of the tree represents all the possible prices at a given time, and each node of value $S$ has two child nodes of values $u \cdot S$ and $d \cdot S$, where $u$ and $d$ are the factors of upward and downward movements for a single time-step $dT$.

$u$ and $d$ are derived from volatility $v$: $u = e^{v \cdot \sqrt{dT}}, d = e^{-v \cdot \sqrt{dT}}; (u \cdot d = 1)$.

$p_d$ is simply equal to $1 - p_u$ and $p_u$ is derived from the assumption that over a period of $dT$ the underlying asset yields the same profit as a riskless investment on average, so that if it is worth $S$ at time $t$, then it is worth $S \cdot e^{r \cdot dT}$ at time $t + dT$. This leads to the following equation: $S \cdot e^{r \cdot dT} = (p_u \cdot u \cdot S + (1 - p_u) \cdot d \cdot S)$, from which we deduce $p_u$:

$$p_u = \frac{e^{r \cdot dT} - d}{u - d}$$

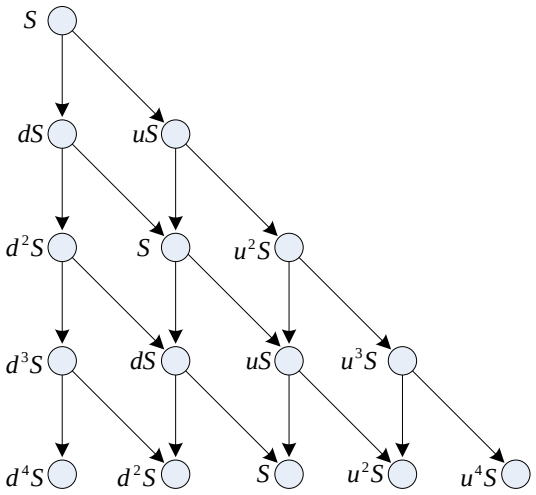


Figure 1. Binomial tree for 4 time steps. $u \cdot d = 1$

From the binomial tree representation, we can then iteratively derive the option price for each node of the tree, starting at the leaves. At each leaf of the tree (i.e. at option expiry) deriving call and put option price is simple:

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

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

Having calculated all possible option prices at expiry date, we start moving back to the root, using the following formula: $V_t = (p_u \cdot V_{u,t+1} + p_d \cdot V_{d,t+1}) \cdot e^{-r \cdot dT}$, where $V_t$ is the option price for one of the nodes at time $t$ and $V_{u,t+1}$ and $V_{d,t+1}$ are the prices of its two child nodes. This formula is derived from the observation that an option that is worth $V_t$ at time $t$, is worth at time $t + dT$, $V_t \cdot e^{r \cdot dT}$ on one hand, and its expected value on the other hand, which is $p_u \cdot V_{u,t+1} + p_d \cdot V_{d,t+1}$, by definition.

# Implementation details

## CPU implementation

Having calculated all the quotients by the formulas given above, implementation of binomial tree tracking on the CPU is fairly easy. At first, all leaf values have to be generated as shown in Listing 1.

```
for(int i = 0; i <= NUM_STEPS; i++){
    data_t price = S * EXP(vDt * (2.0f * i - NUM_STEPS));
    Call[i]  = MAX(price - X, 0);
}
```

Listing 1. CPU price array generation.

We begin at the leaf nodes of the tree, with indices in the range [0, NUM_STEPS], corresponding to time step NUM_STEPS. We reduce the price array step-by-step, moving back in time to the root of the tree (time step 0) as shown in Listing 2.

```
for(int i = NUM_STEPS; i > 0; i--)
    for(int j = 0; j <= i - 1; j++)
        Call[j] = puByDf * Call[j + 1] + _pdByDf * Call[j];
```

Listing 2. Successive reduction of the prices array.

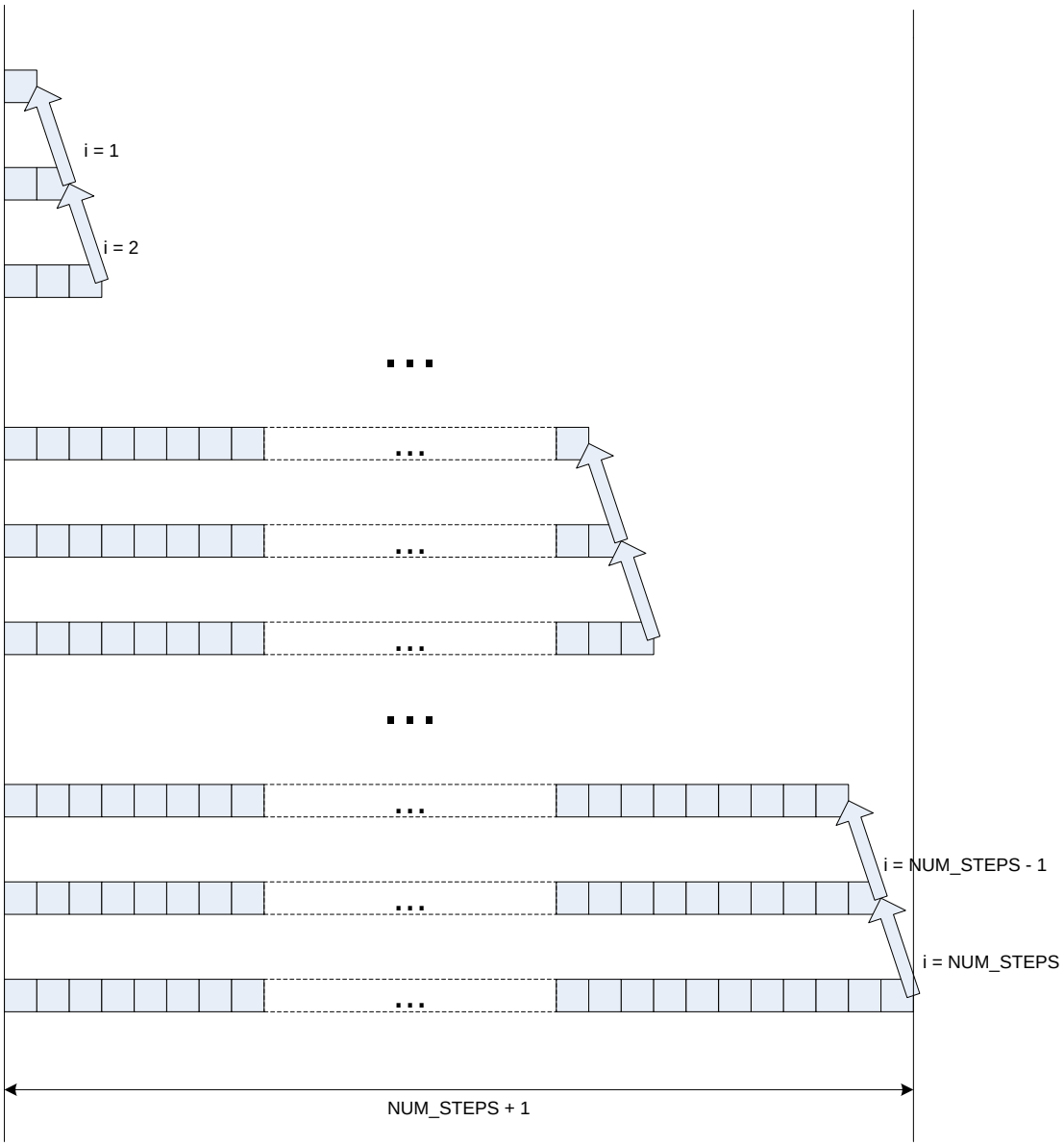


Figure 2. Successive reduction of the prices array.

# GPU implementation

On parallel architectures such as the NVIDIA G80 GPU, a more robust computation scheme has to be developed, since everything is different on GPUs: there are many physical devices executing instructions in parallel as well as constraints on memory access patterns.

A straightforward approach would be to load all leaf values into a high-speed shared memory buffer and perform calculations in shared memory. But the size of shared memory on the G80 GPU is limited. So, we take as basis our intention to store as many data as possible in shared memory, but taking into account that working data sizes can be much larger than the available shared memory, forcing us to spill to global memory at some steps of the computation.

The solution is to process nodes of the tree in portions that fit into shared memory. Looking at the reference CPU implementation, we can see that only the values at nodes [A .. B] at time step T are needed to derive the values at nodes [A .. B – 1] at time step T – 1 and on down the tree to nodes [A .. B – D] at time step T – D.

This primitive, reducing nodes from [A .. B] to [A .. B – CACHE_DELTA] (B – A <= CACHE_SIZE - 1) is the core of our CUDA kernel, as shown in Figure 3.

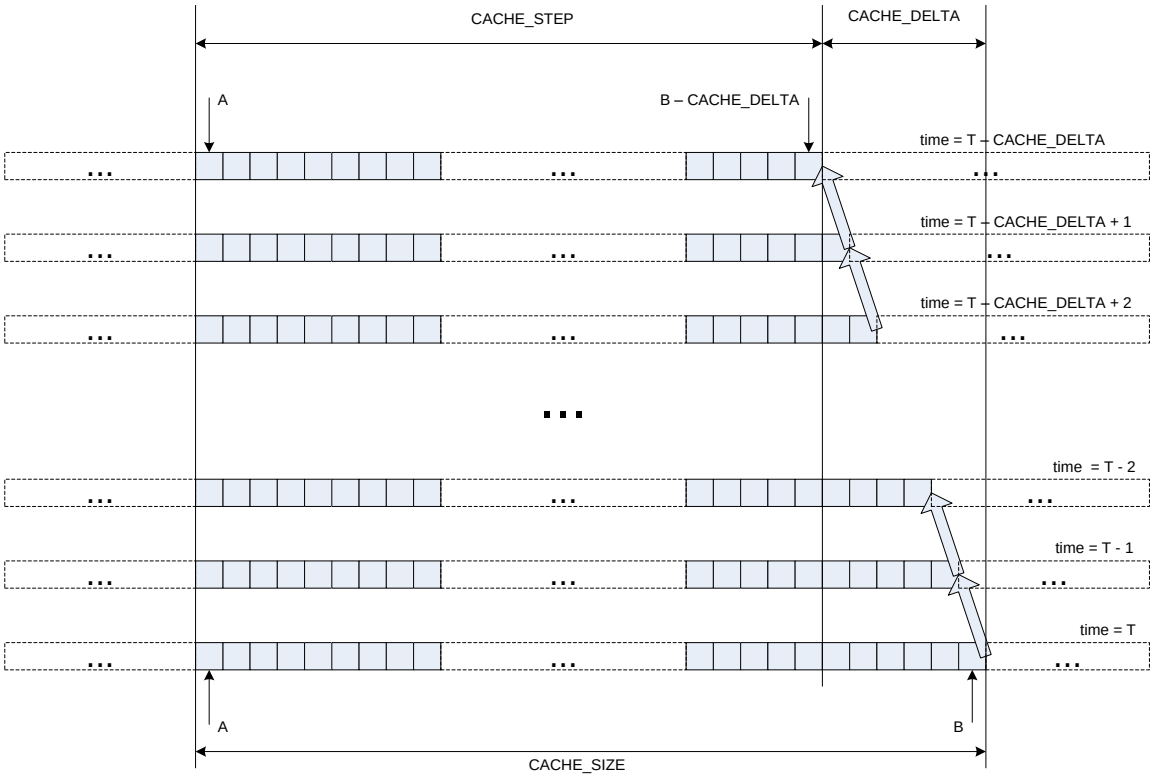


Figure 3. Reduction primitive.

## Figure 3: The binomial options tree reduction primitive

In terms of the code in the `binomialOptionsKernel()` function of the binomialOptions_kernel.cuh file, A = `c_base`, B = `min(c_base + CACHE_SIZE - 1, i);` `i` contains the highest index of the entire prices array for the current group of CACHE_DELTA steps. The [A .. B] range contains exactly CACHE_SIZE elements if index B = `c_base + CACHE_SIZE - 1` is "valid" (in other words not greater than `i`), or fewer otherwise. On one hand, it's good to have a large CACHE_DELTA to perform memory spills as rarely as possible, but on the other hand, each reduction primitive loads data with an apron of size CACHE_DELTA — to produce *N* output elements *N* + CACHE_DELTA elements are always loaded at each invocation of the reduction primitive. Therefore, memory read overhead is inevitably increased.

Having applied the primitive to the entire vector, we reduce it by CACHE_DELTA nodes, stepping back CACHE_DELTA time steps, as shown in Figure 4.

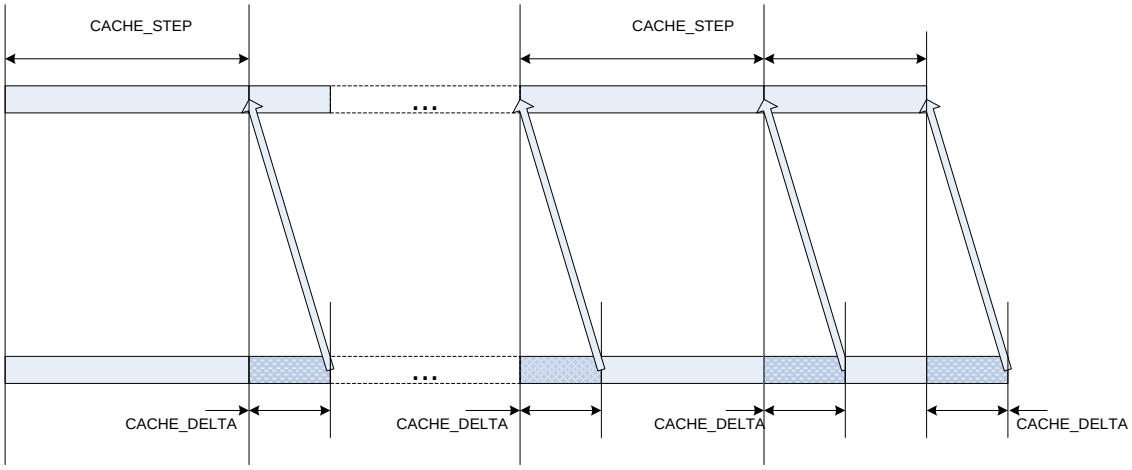


Figure 4: Applying reduction primitive to the entire prices array.

In addition, since the order of thread scheduling is complex and best viewed as simply undefined, the reduction primitive is double-buffered, ensuring by means of __syncthreads() that results from the previous stage are ready before they are used in the next, as shown in Listing 3.

```
//Calculations within shared memory
for(int k = c_start - 1; k >= c_end;){
    //Compute discounted expected value
    __syncthreads();
    if(tid <= k)
    callB[tid] = puByDf * callA[tid + 1] + pdByDf * callA[tid];
    k--;

    //Compute discounted expected value
    __syncthreads();
    if(tid <= k)
    callA[tid] = puByDf * callB[tid + 1] + pdByDf * callB[tid];
    k--;
}
```

Listing 3. Double buffering in the reduction primitive.

# Bibliography


1. Craig Kolb, Matt Pharr (2005). "Option pricing on the GPU". *GPU Gems 2*. Chapter 45.
2. Fischer Black, Myron Scholes (1973). "The Pricing of Options and Corporate Liabilities". *Journal of Political Economy* **81** (3): 637-654.