SOLUTIONS FOR OPTIMIZING THE MONTE CARLO OPTION PRICING METHOD'S IMPLEMENTATION USING THE COMPUTE UNIFIED DEVICE ARCHITECTURE

Ion LUNGU[1], Dana-Mihaela PETROŞANU[2], Alexandru PÎRJAN[3]

Finance-related problems require more and more computations; therefore, the problem of finding efficient implementations for option pricing models on modern architectures has become an important challenge. Although there are numerous implementations of the Monte Carlo method on central processing units, many of them face limitations arising from the necessary increased computational power. In this paper, we have implemented the Monte Carlo approach to option pricing using the Compute Unified Device Architecture and its optimization solutions.

Keywords: financial options, Monte Carlo method, optimization solutions, CUDA, Kepler architecture.

1. Introduction

Option pricing, a very important problem encountered in financial engineering, has been based on a theoretically consistent framework ever since the Black-Scholes option pricing formula has been published [1]. Monte Carlo methods are useful in many theoretical and practical problems, covering a wide range of applications from different fields, such as: finance, business, computational mathematics, telecommunications, applied statistics, computational biology, medicine, engineering and physical sciences.

Although there are numerous implementations of the Monte Carlo method on central processing units, many of them face limitations arising from the necessary computational power. The ability to implement the Monte Carlo method on parallel processing architectures from the latest generation (such as those implemented in graphics processors) offers a viable solution to overcome the limitations of computational architectures based on central processing units. A particular interest in our research was to develop solutions for optimizing the implementation of the Monte Carlo option pricing method using the latest

[1]Professor, Economic Informatics Department, Academy of Economic Studies, Bucharest, Romania, e-mail: ion.lungu@ie.ase.ro
[2]Lecturer, Department of Mathematics-Informatics I, University POLITEHNICA of Bucharest, Romania, e-mail: danap@mathem.pub.ro
[3]Teaching Assistant, IT, Statistics and Mathematics Department, Romanian-American University, Bucharest, Romania, e-mail: alex@pirjan.com

graphics processing unit (GPU) architecture, Kepler, implemented in the GeForce GTX 680 processing unit. In the following, we depict the main theoretical notions of the Monte Carlo method.

2. The Monte Carlo method

Generally, the term "option" refers to an agreement between two parties, the option seller and the option buyer. Options are derivatives that give the option buyer the right and the option seller the obligation to carry out some operation (or exercise the option, or conclude a transaction) at some moment in the future, under previously established conditions [2]. Options can be classified according to the moment when they can be exercised. An option's type can be either European or American. The American option may be exercised at any time before the expiry date, while the European option may be exercised only at the option's expiry date, i.e. at a single pre-defined point in time.

According to the Black-Scholes model, noting S_t the price of the underlying asset at the moment t, S_t follows a Brownian motion characterized by the constants μ (drift), ν (volatility) and satisfies the stochastic differential equation:

$$dS_t = \mu S_t dt + \nu S_t dW_t \tag{1}$$

where W_t is a Wiener random process so that

$$X = W_t - W_0 \tag{2}$$

follows a normal distribution with average 0 and standard deviation T.

The equation (1) can be written in its equivalent form:

$$\frac{dS_t}{S_t} = \mu dt + \nu dW_t \tag{3}$$

that can be interpreted as modelling the percentage variations $\frac{dS_t}{S_t}$ of the asset's price as increments of the Brownian motion. The parameter ν in this equation represents the standard variation of the asset's prices or the asset's price volatility and the coefficient μ is the average profit rate. Choosing the interest rate equal to the average profit rate μ, we obtain the description of the risk-neutral dynamics of the asset's prices. The solution of the equation (3) is:

$$S_t = S_0 e^{\mu T + \nu (W_T - W_0)} \tag{4}$$

or, using the Wiener's process definition, the solution may be written in its equivalent form:

$$S_t = S_0 e^{\mu T + \nu N(0,T)} = S_0 e^{\mu T + \nu\sqrt{T} N(0,1)} \tag{5}$$

In this case, the expected future value is given by:

$$E(S_T) = S_0 e^{\mu T} \cdot E(e^{N(0,\nu^2 T)}) = S_0 e^{\mu T} e^{0.5\nu^2 T} = S_0 e^{(\mu + 0.5\nu^2)T} \tag{6}$$

However, using the definition,

$$E(S_T) = S_0 e^{rT} \tag{7}$$

where r is the risk-free rate of return. Consequently, using relations (6) and (7) we obtain:

$$\mu = r - 0.5\nu^2 \tag{8}$$

Introducing the relation (8) in (5) we obtain:

$$S_t = S_0 e^{(r - 0.5\nu^2)T + \nu\sqrt{T} N(0,1)} \tag{9}$$

The relation (9) represents the possible end stock price based on a random sample with distribution $N(0,1)$ that describes the changing of the stock prices. At the end of the studied period, the possible prices of derivatives are derived from the underlying asset.

In the case of a call option, the profit is given by:

$$V_{call}(S,T) = \max(S_T - X, 0) \tag{10}$$

If, at the exercise date, the market stock price is greater than the strike price, the call option makes its holder a $S_T - X$ profit, otherwise the profit is zero.

In the case of a put option, the profit is given by:

$$V_{put}(S,T) = \max(X - S_T, 0) \tag{11}$$

Based on the relation (11), if the strike price at the exercise date is greater than the market stock price, the put option makes its holder a $X - S_T$ profit and zero otherwise.

In order to mathematically estimate the expectations for $V_{call}(S,T)$ and $V_{put}(S,T)$, one can use the Monte Carlo numeric integration. There are generated N numeric samples with the normal distribution $N(0,1)$ that corresponds to the underlying Wiener process, and consequently the average of the possible end-period stock profits $V_i(S,T)$, corresponding to each of the sample values, is:

$$V_{mediu}(S,T) = \frac{\sum_{i=1}^{N} V_i(S,T)}{N} \tag{12}$$

This is the essence of the Monte Carlo approach to option pricing. Introducing a discount factor e^{-rT} for the approximate future price, one can obtain an approximation of the actual value of the derivative price:

$$V_{actual}(S,0) = V_{mediu}(S,T)e^{-rT} \tag{13}$$

In the case of the above considered example (European option), one can obtain analytical solutions (closed forms) to calculate $E(V_{call}(S,T))$ and $E(V_{put}(S,T))$ using the Black-Scholes formula [1], [3]. These closed-form solutions will be used to compute the reference values for comparison with results obtained by using the Monte Carlo integration.

3. The normally distributed sample generation

The first stage in applying the Monte Carlo method is the generation of a random, normally distributed number sequence with parameters 0 and 1. The Monte Carlo method is based on pseudorandom number sequences, for which most of the probability theory laws hold. However, to apply the method to numeric integration we need uniformly distributed samples over the integration space.

In the following, we briefly present some theoretical elements of probability theory that are needed in order to study the generation of a normally distributed sample [3]. In the probability theory, the normal (or Gaussian) distribution is a continuous probability distribution, characterized by two constants, the mean μ (location of the peak) and the variance σ^2 (the measure of the width of the distribution). If X is the normal distributed random variable, this is denoted by $X \in N(0,\sigma^2)$ and its probability density function is given by:

$$f_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{14}$$

The distribution with the mean $\mu = 0$ and the variance $\sigma^2 = 1$ is called standard normal, the variable is denoted by $X \in N(0,1)$ and the corresponding probability density function is given by:

$$f_{0,1}(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{15}$$

The cumulative distribution function corresponding to the probability density function given by (15) is called the cumulative normal distribution function and is given by:

$$F_{0,1}(x) = P(X < x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt \tag{16}$$

Since the function $y = F_{0,1}(x)$ is strictly ascending, exists its inverse $x = F_{0,1}^{-1}(y)$, with $y \in (0,1)$.

In the following, we consider a uniform distribution $\{X : x \in (0,1)\}$ obtained through a quasirandom generation. In this case, the probability density function is given by:

$$f(x) = \begin{cases} 1, \text{ for } x \in (0,1) \\ 0, \text{ for } x \notin (0,1) \end{cases} \tag{17}$$

and the cumulative normal distribution function is:

$$F(x) = P(X < x) = \int_{-\infty}^{x} f(t)dt = \int_{0}^{x} dt = x, x \in (0,1) \tag{18}$$

In the following we study the existence of an application Ψ so that $\Psi(x) = y$, the image of the uniformly distributed random variable X being the variable $Y \in N(0,1)$. The application $F_{0,1}^{-1}$ introduced above is strictly ascending and one can obtain:

$$\{X < x\} \Leftrightarrow \{F_{0,1}^{-1}(X) < F_{0,1}^{-1}(x)\} \tag{19}$$

Taking into account the relations (18) and (19) we obtain:

$$P(F_{0,1}^{-1}(X) < F_{0,1}^{-1}(x)) = x \tag{20}$$

By using the $x = F_{0,1}(y)$ substitution, equivalent with $y = F_{0,1}^{-1}(x)$, the last expression becomes:

$$P(Y < y) = F_{0,1}(y) \tag{21}$$

The $F_{0,1}^{-1}$ application transforms the random uniformly distributed variable X in the normal standard distributed variable $Y \in N(0,1)$. Taking into account that, in the previous reasoning there have not been used any of the specific properties of the standard normal distribution, the involved technique can be used to obtain any other type of distribution from a sequence of uniformly distributed numbers on the interval $(0,1)$. Even if there are no known closed-form expressions for the inverse cumulative normal distribution function, several accurate polynomial approximations exist. Two of these polynomial approximations are used for obtaining our sample: Moro and Acklam [4].

4. Experimental results

Once we have generated the desired number of $N(0,1)$ samples, we have used them to compute an expected value for the underlying option. Typically, the number of options is of a few hundred or lower and if one option per thread is assigned, there is not generated an enough computational load to efficiently utilize the tremendous power of the GPU. Therefore, it is preferable to use multiple threads per option. In order to achieve this, we have two choices: we can use one thread block per option or multiple thread blocks per option. For each option, there are computed hundreds or even thousands of paths. In order to diminish the latency arising from reading the random input values, the computations of each option are divided and allocated to multiple thread blocks. Depending on the number of underlying options and samples, different methods may be chosen to obtain the highest performance. One must take into account that pricing single European option using Monte Carlo integration is obviously a one-dimensional problem. If more options are assessed, the problem can be considered two-dimensional [4].

After experimental determinations, we have chosen a number of 512 threads, as this setting provided the best performance on the Kepler GK104 architecture. In order to compute the expected price and confidence width for each option, we have to compute the sum of all the stored partial sums per option. In this purpose, a second kernel that parallel reduces the partial sums to their sum must be launched. This parallel reduction is a summation based on a tree of n values through $\log(n)$ parallel steps and this is an efficient way to combine values on a GPU, as a data-parallel processor [4].

In the following, we depict a series of experimental tests. In the benchmarking, we have used the following configuration: Intel i7-2600K clocked at 4.6 GHz with 8 GB (2x4GB) of 1333 MHz DDR3, dual channel. We have used the GeForce GTX 680 (from the Kepler architecture) graphics processor. For programming and access to the GPU we have used the CUDA toolkit 4.1 with the Nvidia driver version 301.10. We have developed the algorithm as to accept a variable number of options, no matter if the specified number is a power of 2 or not. In the experimental tests we have chosen a wide range of values for the number of options. We have decided to represent in this article the obtained results when choosing 128, 256, 512 respectively 1024 options, as these values are relevant for proving the algorithm's efficiency in a variety of scenarios. We assessed for each case the execution time and the number of paths per second obtained after running the benchmark suite. The results represent the average of 10,000 iterations.

In order to compute the average execution time that the GPU spends for executing the tests, we have used the CUDA application programming interface

(API). We have preferred this option instead of those based on the CPU's or on the operating system's timers, because those methods would have included latency and variations from different sources. In this way, we get a reliable measurement of the execution time for computing the tests.

By choosing 512 threads and a variable number of options, after having evaluated the performance (assessed as number of paths per second), we found that it has increased with the number of paths per option up to a maximum value. One can observe a performance cap and a horizontal plateau of values from a specific number of paths per option. This is shown in Fig. 1. One can also remark that if there are sufficient paths per option, the graph is almost horizontal. In an ideal implementation, the entire graph should be horizontal, because the GPU should be able to sustain a constant computation rate (if we can reduce the overhead for small path counts).

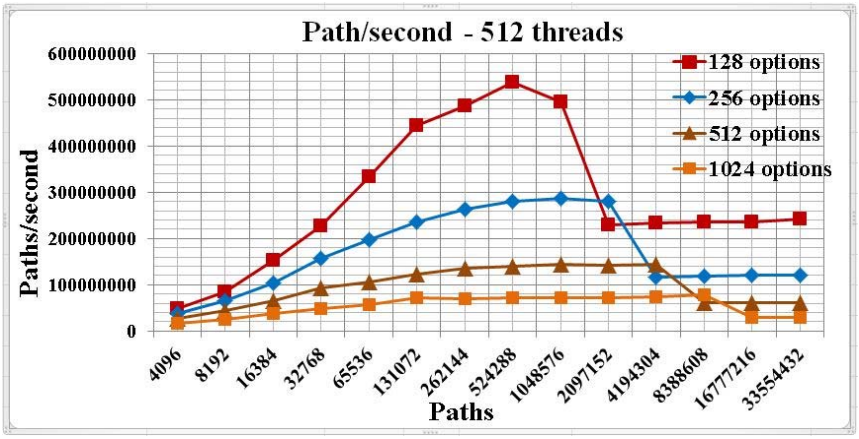

Fig. 1. Experimental results for a variable number of options and 512 threads

Analysing the obtained experimental results, one can notice the efficiency of optimizing the Monte Carlo approach to option pricing, using the latest CUDA graphics processor, Kepler. Our implementation provides optimal results in various situations and represents a viable solution to overcome the limitations of CPU-based architectures. The CUDA architecture has proved to be a very useful tool for designing scalable parallel applications and for developing solutions that optimize data processing.

5. Conclusions

The study in this article demonstrates that the GTX 680, the latest CUDA-enabled GPU from the Kepler architecture is capable of efficient and accurate Monte Carlo option pricing even for small path counts. Analysing the literature,

we noticed that none of the works so far (to our best knowledge) has studied how well Kepler, the latest generation of GPU architectures, scales in implementing the Monte Carlo option pricing method. In implementing the Monte Carlo option pricing method in the CUDA parallel processing architecture, we have identified and developed a series of solutions for optimizing the performance. The experimental results have shown that the Monte Carlo implementation based on the CUDA architecture must dynamically adapt the type and level of parallelism to the computational load, in order to obtain the best performance. We have shown how using performance analysis across a wide variety of problem sizes can point the way to important code optimizations.

Lately, there has been a lot of interest in the literature for optimizing the Monte Carlo option pricing method but none of the works so far (to our best knowledge) tried to validate if the huge computational processing power of a GPU from the latest Kepler architecture could provide an improved solution. The optimized Monte Carlo approach to option pricing, using the latest CUDA graphics processor, Kepler, proves to be a very efficient method. We have obtained optimal results in a variety of scenarios and we have surpassed the computational limitations of CPU-based architectures. Analysing the solutions for optimizing the Monte Carlo option pricing method's implementation using the Compute Unified Device Architecture, we conclude that this architecture is a powerful, efficient and useful tool in developing applications that require a huge parallel computational processing power.

REFERENCES

[1] *F. Black, M. Scholes*, "The Pricing of Options and Corporate Liabilities", in Journal of Political Economy, **vol. 81**, no. 3, May - Jun. 1973, pp. 637-654
[2] *P. Glasserman*, Monte Carlo Methods in Financial Engineering, Springer, New York, 2003
[3] *G. E. P. Box, M.E. Müller*, "A Note on the Generation of Random Normal Deviates", in The Annals of Mathematical Statistics, **vol. 29**, no. 2, Mar. 2008, pp. 610-611
[4] *V. Podlozhnyuk, M. Harris*, Monte Carlo Option Pricing, Nvidia Corporation Tutorial, 2008