Improving Performance for Pricing Options by Simulating Trinomial Trees

Marcelo L. Rocha and David N. Prata
Post-Graduate in Computational Modeling Systems, Universidade Federal do Tocantins, Palmas, Brazil
Email: {marcelolisboarocha, ddnprata}@gmail.com

***Abstract*—Financial market models, like scenarios sequences analysis, deals with exponentially growing data volumes, e.g., stock prices and interest rates. Methods to solve these problems usually require simulations to cover a time span for multiple future periods. Trinomial tree is a technique that is based on the projection of different values to the price a share could achieve during its lifetime. This work aims to demonstrate the benefits of using Graphics Processing Units (GPU) to implement a software prototype for trinomial tree technique to price options. For this purpose, we compare the performance between GPU and Central Processing Unit (CPU) implementation's methods.**

***Index Terms*—trinomial trees, pricing options, parallel computing, GPU, financial market**

I. Introduction

High performance computing is extremely useful in finance to solve problems model defined in financial market like sequences of scenarios, and the feasibility to perform them, e.g., the evolution of stock prices and interest rates. From known initial state and covering a time span for multiple future periods, these models take the form of a tree scenario. For example, to find a dynamic evaluation for the probability of three financial shares, during the period of 10 years (each share described in degrees). The odds could go up or down the prices within a year. For this case, a resulting tree scenario is arranged over one billion (23^{10}) of terminal nodes. The computation time required for solving these problems may extend for a period of hours or even days. Hence, the use of parallel computing to accelerate the turn to a solution in a reasonable time is needed [1].

Methods to perform simulations for financial markets application are extremely intensive alternatives for the computational point of view, since it requires creating a set of simulations for each sample price of the day underlying share [2]. Among these methods, there is a numerical evaluation algorithm known as "Trinomial Trees". The trinomial tree is a technique that is based on the projection of different values to the price a share could achieve during its lifetime [3]. The volume of calculations used in processing trinomial trees is great, demanding efficient implementations.

In order to demonstrate the feasibility of using GPU to solve financial problems with high computational power, a software prototype was implemented for pricing options using trinomial trees.

II. Background

A. Financial Marketing

Finance professionals have devoted enormous effort to understand the process of price formation from securities traded in capital markets. In this case, some models were developed having a good degree of reliability [4].

Contract options are fundamentally different from other types of futures contracts traded on stock exchanges. There are two basic types of options, call options (calls) and put options (puts). A call option gives the option holder the right to buy a share at a certain price on a certain date. In put option, the holder obtains the right to sell the share for a specified price at a future date. The price agreed between the parties to a contract is known as the strike price, and the agreed date for achieving this is called the due date [5].

There are six factors to influence the prices of options, namely: the cash price of the share ($\boldsymbol{S_0}$), the strike price ($\boldsymbol{X}$) time for payment until the due date ($\boldsymbol{T}$), volatility of the stock price ($\boldsymbol{\sigma}$) free interest rate risk ($\boldsymbol{r}$), and dividends and bonuses expected during the life of the option (in some cases). These factors are the basis for various models to price options, where the most famous is the Black and Scholes.

B. Trinomial Trees

The trinomial tree is a useful technique for pricing stock options. This technique creates a diagram representing the different paths that can be followed by a share price during the life of an option [6].

The trinomial model is a model where the price of options on the stock is monitored by successive short periods of time, from the date the contract until settlement is made [7]. At each step, the price may take three new values by performing the upward move, maintain the value, or downward movement, with their probabilities of occurrence $\boldsymbol{P_u}$, $\boldsymbol{P_m}$ and $\boldsymbol{P_d}$. All the possibilities a stock price can eventually reach form the structure of a trinomial tree.

Each node of the trinomial tree has three nodes whose values are $\boldsymbol{u.S}$, $\boldsymbol{S}$ and $\boldsymbol{d.S}$,, and $\boldsymbol{S_0}$ is the value at the root

Manuscript received August 29, 2014; revised December 27, 2014.

corresponding to the value of the share at time 0. Accordingly, ***u*** e ***s*** represents respectively the volatility of the value on the rise (up) and descent (down) in time. The structure of a trinomial tree is shown in Fig. 1 [8].

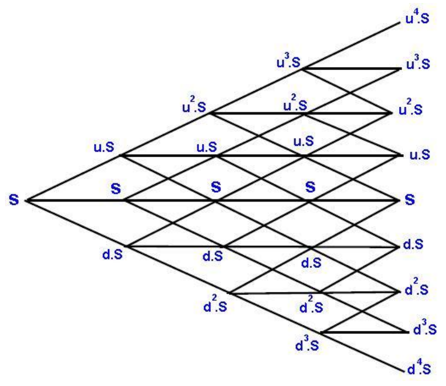


Figure 1. Trinomial tree with four steps.

The trinomial pricing method is based on the Black & Scholes model as a continuous variation in prices over time. The *Black Scholes* model allows calculating *Pu*, *Pm*, *Pd*, *u* and *d*, as specified in equations 1-5.

ΔT is the time difference between two consecutive steps in the tree. Note that ***u.d* = 1**, and after N steps the leaves of the trinomial tree will have (2N + 1) nodes. Thus, it is possible to calculate the probabilities using the *Black and Scholes* model.

$$u = e^{\sigma\sqrt{3\Delta T}} \tag{1}$$

$$d = e^{-\sigma\sqrt{3\Delta T}} \tag{2}$$

$$Pu = \frac{\Delta T(\sigma^2 + \mu(\mu.\Delta T + \sigma\sqrt{3.\Delta T}))}{6.\sigma^2.\Delta T} \tag{3}$$

where, $\mu = r - \frac{\sigma^2}{2}$

$$Pd = \frac{\Delta T(\sigma^2 + \mu(\mu.\Delta T - \sigma\sqrt{3.\Delta T}))}{6.\sigma^2.\Delta T} \tag{4}$$

where, $\mu = r - \frac{\sigma^2}{2}$

$$Pm = 1 - Pu - Pd \tag{5}$$

To calculate the value at node V_n we use the formula specified in equation 6. The prices we found in $V_{d,\ n+1}$, $V_{m,\ n+1}$ e $V_{u,\ n+1}$ are the prices of the three nodes of the new value. Accordingly to Black and Scholes model, all values of the leaves can be generated using the formula shown in Equation 7.

$$V_n = (P_d.V_{d,n+1} + P_m.V_{m,n+1} + P_u.V_{u,n+1}).e^{-r.\Delta T} \tag{6}$$

$$S_n = S_0.e^{i.\sigma\sqrt{3.\Delta T}} \tag{7}$$

where *i* varies from –*n* to *n*

Computer algorithms to solve securities pricing problems require massive computational power [9]. The use of parallel computing is necessary for large volumes of calculations allowing quickly responses.

C. GPU Architecture

Several tools have been developed in order to assist in the processes of parallelization of computations required for solve problems like securities pricing. CUDA™ (Compute Unified Device Architecture) library aims to provide the processing power of GPUs for generic processing to solve many complex computational problems in a fraction of execution time compared to a CPU.

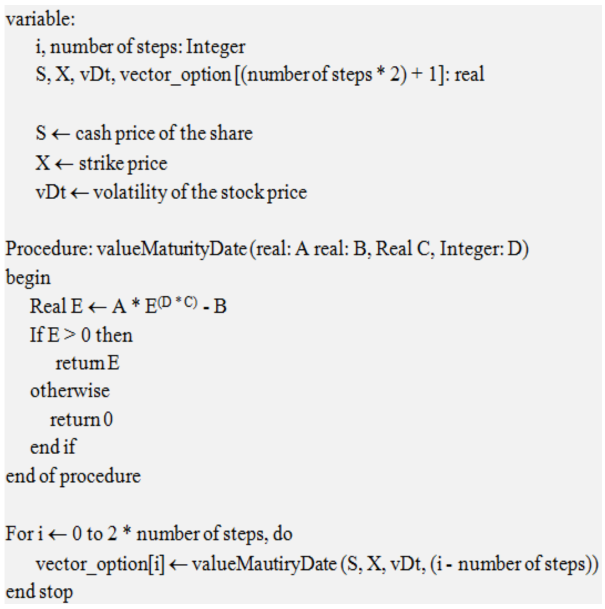
GPU is composed of several multiprocessors (currently ranging in range 4-30). A multiprocessor consists of eight scalar core processors with the function of creating, managing and executing operations of concurrent threads. To manage hundreds of threads in the execution of various programs, the multiprocessor employs a new architecture known as *SIMT* (Single Instruction Multiple Thread) based on *SIMD* (Single Instruction Multiple Data) architecture. The multiprocessor maps each thread to one scalar core processor, and each thread independently executes its instructions. The SIMT multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps.

The GPU using CUDA™ can be defining a SIMD processor with great power of parallelism, being limited only by the amount of available memory on the graphics hardware [10].

III. Methods

A. Resolution of Trinomial Trees on CPU

The calculations on CPUs using trinomial trees can be easily achieved through algorithms operating in serialized arrangement. The values of tree leaves using the method of Black and Scholes are generated, seeing Fig. 2.



```
variable:
    i, number of steps: Integer
    S, X, vDt, vector_option [(number of steps * 2) + 1]: real

    S ← cash price of the share
    X ← strike price
    vDt ← volatility of the stock price

Procedure: valueMaturityDate (real: A real: B, Real C, Integer: D)
begin
   Real E ← A * E^(D * C) - B
   If E > 0 then
      return E
   otherwise
      return 0
   end if
end of procedure

For i ← 0 to 2 * number of steps, do
    vector_option[i] ← valueMautiryDate (S, X, vDt, (i - number of steps))
end stop
```

Figure 2. Calculation for maturity values on CPU.

The calculations of the upward movement of the tree were executed by successive reductions of the tree vector, until the value of the root is found, Fig. 3.

```
variable:
    i, j, number of steps: Integer
    vector_option, probUpR, probMaintainR, probDownR: real

probUpR ← upward likelihood
probMaintainR ← maintain likelihood
probDownR ← downward likelihood

For i ← 0 to number of steps, do
    For j ← 0 to (2 * number of steps) - 2 do
        vector_option [i] ← probUpR * vector_option [j +2] + probMaintainR *
vector_option [j + 1] + probDownR * vector_option [j]
        end stop
end stop
```

Figure 3. Reduction of trinomial tree in CPU.

B. *Solving Trinomial Trees on GPU*

The computation of option pricing using trinomial trees in GPUs was conducted in parallel with threads controlling to efficient manipulation of memory areas.

In a first step, the necessary to compute the tree is generated in a CPU, and transferred from the host to the software device. In GPU, the calculations are parallelized for all possible values of options, where each processor (in multiprocessor) is in charge for the calculation of each respectively option.

Maturity prices are stored in vectors, and declared so that their sizes are always multiples of 16, This software device is used to maintain data consistency in memory providing performance gains in reading and writing operations. The schema is shown in Fig. 4.

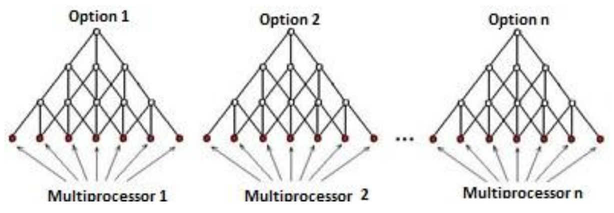


Figure 4. Calculation of maturity values on GPU.

In Fig. 5, the algorithm to resolution of trinomial trees in parallel is described.

In order to parallelize the pricing algorithm running over the whole tree structure, it was necessary to perform a geometric decomposition of the tree, in a recursive method. The shared memory of the GPU has a maximum storage capacity of approximately 4000 floating point values. This size is easily exceeded by many steps executions of trinomial trees. For example, a tree whose evaluation involves a number of 2000 steps will have a vector containing 4001 floating point values, exceeding memory.

The data structure of trinomial tree has a original recursive pattern and copies geometric decomposition "pieces" to the same shared memory. The copied data is stored continuously in the global GPU memory. For the implementation of the algorithm many sizes were tested for geometric "chunks" with the intention to find the best configuration to increase in performance. Nevertheless, no significant differences were observed. The shared memory access provides a time saving factor of 100x to 150x higher than the access time of the global memory.

The tree computing is performed to instant zero date by loading values leaves in a loop (algorithm, Fig. 3).

```
Kernel: ExecuteTrinomialTreeGPU ( )
    Referencing memory address values at maturity for the current block: early
        Option ← option address of currently block [ (2 * Number of steps ) + 16 ]
    end referring

    variable

        OptionX [ Cash Size ]: real (SHARED MEMORY OF GPU)
        OptionY [ Cash Size ]: real (SHARED MEMORY OF GPU)
        IDTHREAD ← THREADIDX.X
        A ← Current value of the asset to the current option
        B ← strike price for the current option
        C ← Volatility maturity to variation in time for the current option
        PROBS ← Probabilidade de Subida para a opção atual
        PROBM ← Probabilidade de Manutenção para a opção atual
        PROBD ← Probabilidade de Descida para a opção atual
        i, j, m, CASH_A, CASH_B, CASH_C: integer

        For i ← (Number of steps * 2) until greater than 0 steps Variation CASH, do
            For CASH_A ← 0 until greater than i steps CASH jump, do
              CASH_B ← minimum (Cash Size – 1 , i – CASH_A)
                 CASH_C ← CASH_B – Variation CASH

                 Synchronize THREADS
                 If (IDTHREAD <= CASH_B)
                     OptionX [ IDTHREADS ] ← Option [ CASH_A + IDTHREAD ]

                 For m ← (CASH_B – 1) until greater than CASH_C, do
                     Synchronize THREADS
                     If (IDTHREAD < m)
                        OptionY [ IDTHREAD ] ← PROBS*OptionX [ IDTHREAD+2 ] +
                                                                  PROBM*OptionX [ IDTHREAD*1 ] +
                                                                  PROBD*OptionX [ IDHREAD]
                     m ← m – 1

                     Synchronize THREADS
                     If (IDHREAD < m-1)
                           OptionX [ IDTHREAD ] ← PROBS*OptionY[ IDTHREAD+2 ] +
                                                                  PROBM*OptionY [ IDTHREAD+1 ] +
                                                                  PROBD*OptionY [ IDTHREAD ]
                     m ← m – 3
                  end stop
              end stop
              OPTION VALUE ← OptionX [ 0 ] ( TO CURRENTLY BLOCK )
end Kernel
ExecuteTrinomialTreeGPU <<< Number of options, Cash Size >>> ( )
Copy of option data for DEVICE HOST: begin
    Option HOST ← Option DEVICE
end copy
```

Figure 5. Algorithm for solving trinomial trees in parallel on GPU.

IV. RESULTS

The hardware used for the computational testing was comprised of an Intel ® CoreTM 2 Duo E8400 3 GHz processor with 3GB of RAM, graphics card accelerator model GeForce 8800 GTS 512's NVidia® manufacturing. The video card was used due to the fact that CUDA^{TM} API support only card generation 8 or higher. The P5K Premium motherboard from Asus model was used in the tests. This board was chosen because it has 2 PCI Express slots with 16x speed. These slots were necessary due to the need of installing a second video card to use with the monitor, freeing up the resources of the GeForce 8800 GTS for the execution of tests.

The platform used for the development of option pricing program was composed by Visual C++ 2005 Express Edition SP1 compiler chosen to be distributed free of charge supporting multi languages. The installation was done on Windows XP Professional 32bit operating system. CUDA™ the Toolkit and SDK CUDA™ both version 2.0 for Windows 32 bit were also installed.

For test, 512 trees with 1024, 2048, 4096, 8192 and 16384 steps each were used for performance comparison. The data of 512 assets required to measure the fair value of options at date 0 were generated randomly.

The time (in seconds) was measured during the execution of the calculations in the CPU trinomial trees compared to the performance of parallelized algorithm on the GPU, Fig. 6.

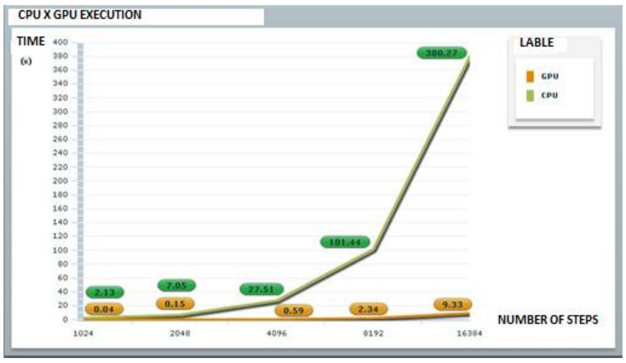


Figure 6. Execution times of the GPU and CPU.

In Table I, the measured values of time (in seconds) and the time gain achieved (number of times faster) by algorithm execution on the GPU.

TABLE I. EXECUTION TIMES OF THE GPU AND CPU.

OPTIONS	STEPS	GPU TIME	CPU TIME	TIME SAVING
512	1024	0,038	2,130	56,05
	2048	0,149	7,050	47,32
	4096	0,589	27,510	46,71
	8192	2,340	101,440	43,35
	16384	9,330	380,270	40,76

During testing CUDA™ on GPU we measured the percentage of CPU usage. We could observe approximately 50% of E8400 CPU being used throughout the program execution of GPU. Consequently, one of the cores was 100% used in this process. The CPU remains all the time running, performing checks, and awaiting the GPU complete kernel executions, Fig. 7.

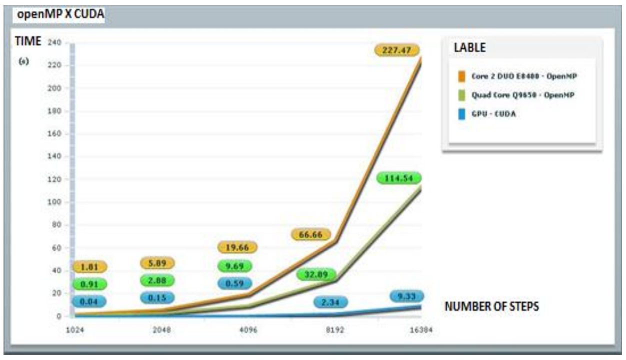


Figure 7. CPU utilization while executing GPU.

Analyzing the results obtained, we can find that the program execution on the GPU showed far superior performance compared to the same algorithm running on CPU. We could observe gains of 40 times in time performance over 17.000 steps execution.

V. CONCLUSIONS

The programming of generic applications on GPU can allow the use of several techniques to assist the process of optimization. The use of graphics processors for highly parallelizable applications as a means of maximizing existing resources proved to be efficient for algorithm performance.

We concluded the use of graphics processing units (GPU) as a viable alternative for working with high performance computing in financial market models, such as pricing options.

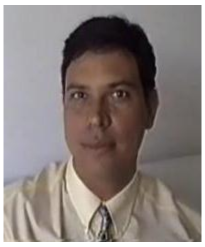
ACKNOWLEDGMENT

We would like to thank professor Patrick Letouze for his support and invaluable contributions.

REFERENCES

[1] H. Moritsch, "High performance computing in finance—On the parallel implementation of pricing and optimization models," 2006. PhD Dissertation—Institut fur Softwaretechnik und Interaktive Systeme, 2006.
[2] L. A. R. Souza, "Value risk in crisis age," Master dissertation, Universidade de São Paulo, FEA/USP, 1999.
[3] P. Clifford and O. Zaboronsk, "Pricing options using trinomial trees," University of Warwick, 2010.
[4] E. F. Lembruber, "Contract assessment of options," BM&F. Bolsa de Mercados e Futuros, São Paulo, 1995.
[5] J. C. Hull, *Foundations of Future Markets and of Options*, 4th ed. BM&F, Bolsa de Mercados e Futuros, 2005.
[6] L. P. Grosso, "Option charging about future of interfinancial deposits from one day," Master dissertation, PUC-Rio, 2006.
[7] P. Boyle, "Option valuation using a three jump process," *International Options Journal*, vol. 3, pp. 7–12, 1986.
[8] J. C. Cox, S. A. Ross, and M. Rubinstein, "Option pricing: A simplified approach," *Journal of Financial Economics*, vol. 7, no. 3, pp. 229–263, 1979.
[9] P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA," Center for Visual Information Technology International Institute of Information Technology Hyderabad, India, 2007.
[10] H. Johnson. "Pricing complex options," Mimeo, Dept. of Finance, College of Business Administration, Louisiana State Univ., Baton Rouge, LA (1981).

Marcelo Lisboa holds a degree in Computer Science from the Catholic University of Petrópolis (1994), Masters in Computer from Fluminense Federal University (1997), Master Science in Electrical Engineering from the Federal University of Rio de Janeiro (1999) and Ph.D. in Electrical Engineering from the Federal University of Rio January (2008). Dr. Marcelo Lisboa is currently a reviewer of the journal INFOCOMP, Journal of Computer Science, and associate professor 3 from Federal University of Tocantins. He has experience in the area of Computer Science, mainly in the following topics: metaheuristics, combinatorial optimization, mathematical programming, computer networking and high performance computing



David Prata was born in Goiânia, Brazil on 18th September, 1965. He completed his Bachelor of Computer Science in 1992. Then on, he went to complete his specializing in Academician. He worked as system analyst to Tocantins Government, being in charge for the accountability and financial systems. Later, he successfully completed his Master Degree in Computer Science from Campina Grande Federal University, with application research in education in 2000 year. He coordinates graduate and undergraduate courses in computer science at Alagoas Faculty in Maceio, Brazil. He was allotted to Federal University of Alagoas in 2006. Then, he moved to Federal University of Tocantins. His doctoral was developed in part at Carnegie Mellon University, USA, completed in 2008. Dr. David Prata is currently coordinating a Master Degree in Computational Model. His research interests are education and ecosystems.