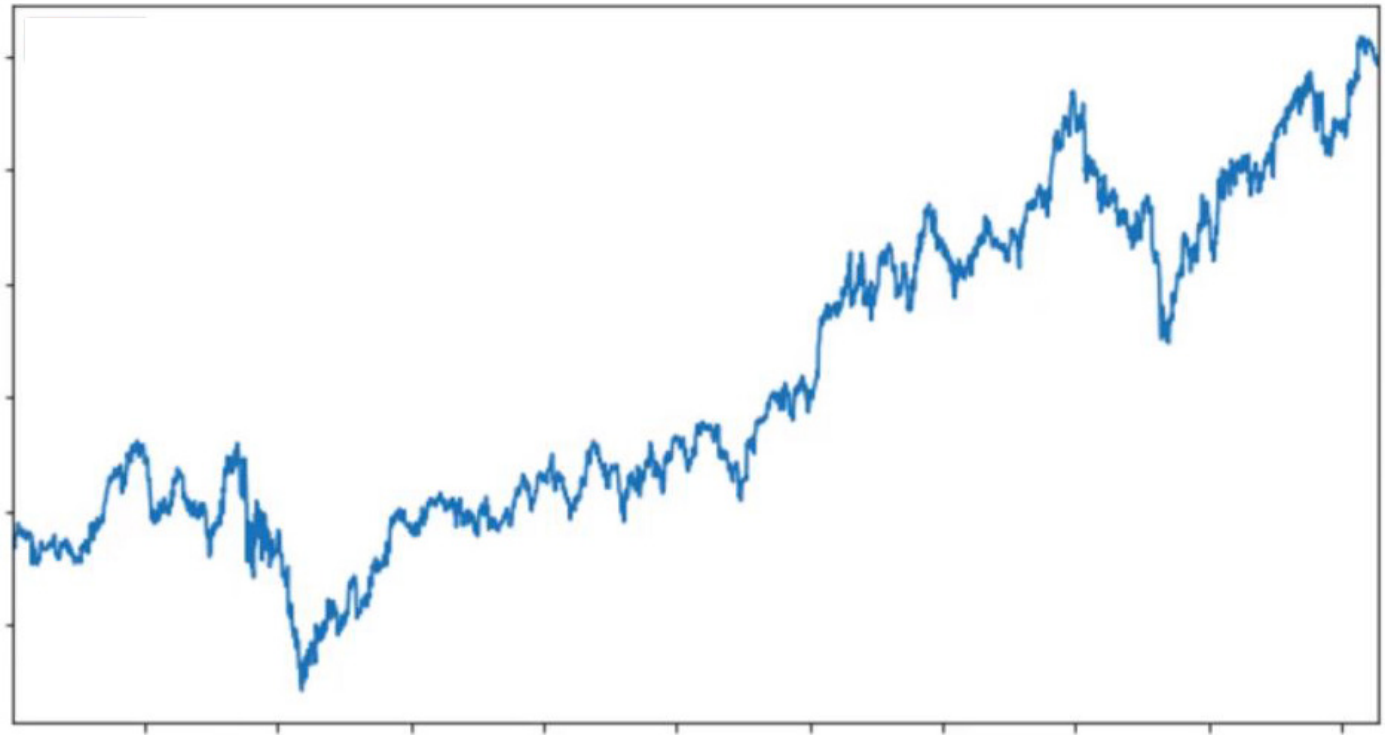
VIVIDSPARKS

Monte Carlo Methods to Option Pricing Models using SupersoniK



Copyright © 2021 VividSparks IT Solutions Pvt. Ltd.

Introduction

Complex financial models are commonly used to maximize returns on investments. They are particularly useful when more and more investors and institutions are involved and very different styles of options are introduced into the derivative markets. Fund managers and investors not only need proper models to estimate asset prices, but also to execute them quickly in order to obtain timely and reliable predictions. Both the Black-Scholes model and Heston model are popular models for asset prices, which are also known as stock prices.

The most widely used technique to solve such problems is numerical approach such as the Monte Carlo (MC) method. Due to the fact that MC method requires very significant computational resources to be applied to these stochastic process models, traditional CPU-based platforms are not able to execute it fast enough for practical use. Hence several kinds of hardware accelerators have been implemented in order to obtain a high performance with a low energy cost per computation.

Graphic processing units (GPUs) are commonly used as accelerators for parallel computations. Their architecture contains many Algorithm-Logic Units (ALUs) managed by a single control unit. MC methods, which performs plenty of independent simulations, can be executed on clusters of CPUs and GPUs with excellent performance. However, it has been shown that these platforms are not very efficient concerning about the energy consumption [2], for financial and for other kinds of applications. This issue has been addressed recently by using reconfigurable hardware platforms as accelerators.

Field-programmable gate array (FPGA) is advantageous with respect to GPUs because:

- It retains some SW-like runtime reconfigurability, which makes it suitable for usage e.g. in data centers

- It has dramatically lower power and energy consumption than both CPUs and GPUs, because it uses a customized hardware control unit, data-path and memory architecture instead of fetching, decoding and executing instructions.

In this work, we have shown MC methods to accelerate the financial applications on GPU and SupersoniK FPGA-based hardware accelerators. Applications are implemented in Python. The performance and energy consumption on both platforms are compared and analyzed.

Option Pricing Models

A. Black-Scholes Model

The Black-Scholes model considers one risk-free asset with a fixed interest rate and one risky asset, whose price is subject to geometric Brownian motion as shown in (1)

$$dS = rSdt + \sigma Sdz \qquad (1)$$

where S is the stock price, r is the fixed interest rate, σ is the constant volatility and z is a Wiener process.

According to Ito's lemma [1], the analytical solution for the stochastic differential equation (1) is shown in (2).

$$S_{t+\Delta t} = S_t e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma\epsilon\sqrt{\Delta t}} \qquad (2)$$

where $\epsilon \sim N(0,1)$, the standard normal distribution.

Apart from the analytical solution, a numerical solution (3) can be obtained by applying Euler discretization to (1)[1][2] for $\Delta t << 1$.

$$S_{t+\Delta t} = S_t(1 + r\Delta t + \sigma\epsilon\sqrt{\Delta t}) \qquad (3)$$

Copyright © 2021 VividSparks IT Solutions Pvt. Ltd.

(3) is commonly used in the literature for MC simulation to avoid the exponent in (2). On one hand, the exponential function requires more resources and time for the computation.

B. Heston Model

Volatility of a risky asset in the Heston model is no longer treated as a constant value, but a stochastic process. Thus (4) models the stock price and (5) models its volatility [2].

$$dV = \kappa(\theta - V)dt + \sigma_v\sqrt{V}dz_1 \quad (4)$$

$$dV = \kappa(\theta - V)dt + \sigma_v\sqrt{V}dz_1 \quad (5)$$

In (4), *z1, z2* are two Wiener processes, ρ is the correlation factor between them, and √V is the volatility of the stock price. In (5), θ is the long-run mean variance, κ is the speed of mean reversion (the rate at which V reverts to θ) and σ_v is the volatility (Standard deviation) of the volatility V.

For a short time Δt << 1, V can be assumed to be constant, so that Ito's Lemma can be applied to (4), which is then simplified as in (1). The numerical solution for (5) is also obtained by Euler discretization[2] with full truncation scheme avoiding negative values under the square root [3]. The final solutions are shown in (6) and (7).

$$S_{t+\Delta t} = S_t e^{(r-\frac{1}{2}V_t^+)\Delta t + \sqrt{V_t^+}(\rho\epsilon_1 + \sqrt{1-\rho^2}\epsilon_2)\sqrt{\Delta t}} \quad (6)$$

$$V_{t+\Delta t} = V_t^+ + \kappa(\theta - V_t^+)\Delta t + \sigma_v\sqrt{V_t^+}\epsilon_1\sqrt{\Delta t} \quad (7)$$

where ϵ_1, $\epsilon_2 \sim N(0,1)$, and $V_t^+ = \max(V_t, 0)$.

C. Options

The derivative market offers plenty of different mechanisms (called "options") to calculate the payoff of a contract. The options are classified into different styles, such as vanilla and exotic option, according to the payoff calculation.

1) European Vanilla Option: European vanilla option is one of the simplest option. It can only be exercised at the expiration date and thus its payoff price only depends on the stock price at the expiration date and is computed by (8).

$$P_{Call} = max\{S_T - K, 0\} \quad (8)$$

where T is the pre-set time of the option, ST is the stock price at the expiration date and K is the strike price.

2) European Barrier Option: The European barrier option is exercised only if the stock price over the pre-set time period remains within the pre-set barrier level(s). There could be only one barrier (upper bound or lower bound) or two barriers (bot upper and lower bounds) in a given contract. For example, S_u and S_d are the upper bound and lower bound respectively. The option can be exercised only if the stock price does not go beyond any of the two barriers. So the call price is calculated as (9).

$$P_{Call} = \begin{cases} max\{S_T - K, 0\} & \forall t \in (0,T) \Rightarrow S_d \leq S_t \leq S_u \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Copyright © 2021 VividSparks IT Solutions Pvt. Ltd.

Implementation

A. Simulation Algorithm

1) Time Partitioning: Time partitioning is a necessary approach to deal with a time-dependent stochastic differential equation. Taking Black-Scholes model as an example, there are mainly three steps to do the computation.

- The time interval *(0, T)* is uniformly partitioned into *M* steps and denoted as t_0, t_1,.... t_M.

- *M* standard normally distributed independent random numbers *1, 2,....M* are generated.

- Calculate $S_T = S_{tM}$ starting from S_{t0} by (2) or (3).

This simulation that computes S_T from S_{t0} is called one "path" in option pricing problems. The total number of paths is denoted by *N* in the following.

2) Random Number Generator: A key aspect of the quality of the results of the MC method is the quality of the random numbers that it uses. The normally distributed random numbers in the simulation are generated by the Mersenne-Twister (MT) algorithm followed by the Box- Muller transformation. These algorithms have been broadly implemented in the literature, e.g. in [1]. The overall algorithms for the Black-Scholes and the Heston models are listed in Algorithm 1 and 2 respectively.

Algorithm 1 Black-Scholes model

Input: parameters for the stock and option
Output: payoff price
Initialization: Random number generators
1: **for** $i = 1$ to N **do**
2: **for** $k = 1$ to M **do**
3: $U_1, U_2 \leftarrow MersenneTwist()$
4: $\epsilon_1, \epsilon_2 \leftarrow BoxMuller(U_1, U_2)$
5: $S_{t_{k+2}} \leftarrow Price(S_{t_k}, \epsilon_1, \epsilon_2)$
6: $k+ = 2$
7: **end for**
8: $P_{option}[i] \leftarrow Option(S_t[], K)$
9: i++
10: **end for**
11: **return** $P_{Call} = ave(P_{option})$

Algorithm 2 Heston model

Input: parameters for the stock, volatility and option
Output: payoff price
Initialization: Random number generators
1: **for** $i = 1$ to N **do**
2: **for** $k = 1$ to M **do**
3: $U_1, U_2 \leftarrow MersenneTwist()$
4: $\epsilon_1, \epsilon_2 \leftarrow BoxMuller(U_1, U_2)$
5: $S_{t_{k+1}} \leftarrow Price(S_{t_k}, V_{t_k}, \epsilon_1)$
6: $V_{t_{k+1}} \leftarrow Volatility(V_{t_k}, \epsilon_2)$
7: k++
8: **end for**
9: $P_{option}[i] \leftarrow Option(S_t[], K)$
10: i++
11: **end for**
12: **return** $P_{Call} = ave(P_{option})$

3) Performance Metrics: One of the important metrics for performance is the simulation time used to execute a given computation. The entire simulation contains *N* paths and each path is partitioned into *M* steps, hence the total simulation time Ts is proportional to $C = M \cdot N$. The factor *C* is called computational cost which is a key factor that affects the performance of the simulation and the quality of the results. In the following, performance will be reported with normalization respect to C. As defined in (11), t_c is the time for computing one simulation step.

$$t_c = \frac{T_s}{C} \tag{11}$$

$$E_c = t_c \cdot P_d \tag{12}$$

Another key characteristic of an implementation platform is its energy consumption, or to be more precise the energy consumed to perform a given computation (e.g. a simulation step as shown in (12), where Pd is device power).

Copyright © 2021 VividSparks IT Solutions Pvt. Ltd.

In this work, the power consumption of a GPU is estimated by using both its data sheet and power profiling tools, while for the FPGA it relies on the analysis capabilities of the Vivado synthesis tool.

B. Algorithm optimization on Hardwares

Modern high-performance computing platforms are normally heterogeneous, i.e. they contain CPUs and accelerators (e.g. GPUs or FPGAs). The basic architecture of such a heterogeneous framework has also been depicted and implemented in several other literatures such as in [4].

1) GPU: In the algorithms of MC method, there are N independent paths along time with identical inputs. The N independent paths can be unrolled partially on GPUs. This is realized by N_u (unroll factor) independent works of a kernel and their parallel execution on GPU. Nu depends on the characteristic of a GPU and is relatively small compared to $N = N_u N_s$, where N_s is the number of paths executed in sequential by each independent work item. The values of global size and local work-group size have to be carefully chosen in order to take the full advantages of a GPU. The simulation time T_s is then proportional to $N_s M$.

2) SupersoniK : The optimization of the algorithm on FPGA is more tricky than that on GPU due to the flexible architecture on an FPGA.

Partial unrolling of the outer-most loop is implemented both platforms. The unroll factor N_u depends on the percentage of resources utilization of the rolled iterations (N_s paths). Pipelining of the inner-most loop is another efficient way to accelerate algorithms on an FPGA. This technique is able to increase the throughput of an algorithm. The task is to reduce the initiation interval (II) between two successive iterations. Hence each iteration can be finished in few clock cycles (II cycles) on average. As can be seen in Algorithm 1 and 2, every iteration contains two parts of computation, one is the random number generation and the other one is to update the stock price (and volatility in Heston model) over time partitions.

The optimization of the first part concerns the algorithm of random number generation. The critical problems in the original MT algorithm are the memory accesses and mathematical computation such as modulo. *II* = 7 on Alveo U200 FPGA card at 300 MHz for unoptimized algorithm to generate one random number. In the optimization, the critical computations are replaced by simple operations such as +/- and the array used to store state values is partitioned into two according to the memory access pattern in order to double the throughput. The optimized algorithm achieves *II* = 2 and generates two random numbers in parallel in each iteration. It indicates that each Gaussian random number is obtained in single clock cycle on average (*II* = 1 instead of 7).

Once the random number generation is optimized, the next step is to deal with the second part. In each iteration of the inner most loop, the stock price (and volatility) depends on its value calculated in the previous iteration. Without any modification to the architecture of the nested loops, the II may go up to 30 clock cycles for Heston model at the frequency 300MHz on Alveo U200 FPGA card due to the complicated mathematical computation. This bottleneck is removed by merging a portion of the outer loop (N_i iterations out of N_s) into the inner loop since each iteration in the outer loop is independent. By this technique, the stock price (and the volatility) does not depend on the values of previous N_i *-1* iterations any more because they are on different paths. Of course, it increases the utilization of BRAMs. In the end, the inner-most loop is pipelined with *II* = 2 for both algorithms. So t_c can be roughly estimated by (13) and (14) for the two algorithms respectively.

$$t_c^{B.Scholes} = \frac{t_{clock}}{N_u} \qquad (13)$$

$$t_c^{Heston} = \frac{2t_{clock}}{N_u} \qquad (14)$$

where t_{clock} is the clock period applied to an Alveo U200 FPGA card.

Copyright © 2021 VividSparks IT Solutions Pvt. Ltd.

The algorithms can be further optimized by controlling the IP cores in the synthesis in order to balance the resource utilization and then increase the value of N_u. It supports the implementation of an operations such as multiplier by specific resources. For example, the multiplication of two floating-point variables can be realized fully by DSP or by Flip-flops (FFs) and Look-up tables (LUTs). Especially it is essential for the low-end FPGA chips with limited DSPs.

Results

The execution time and energy consumption of the models described above are compared for the various considered platforms by providing all of them with the same input data (e.g. initial stock price) and the same simulation parameters (e.g. *N* and *M).*

1) Black-Scholes Model: For the Black-Scholes model, the simulation parameters and simulated results such as time and energy per step are shown in Table 1, where "B" denotes Billion. Clearly, the RTX3090 is better than the RTX3060 for this application, in terms of both performance and energy per time step. Compared to the RTX3060, the SupersoniK has 3X speed and only consumes 9.5% of the energy per step. t_{clock} = 6:08*ns* and P_d = 21.2*W* for the Alveo U200 FPGA card. The resource utilization is 40% of the DSPs, 27% of BRAMs, 34% of FFs and 65% of LUTs.

2) Heston Model: The results for the Heston model are shown in Table 2, where "M" denotes Million. The RTX3090 in this case has better performance than the K4200 again. However, the RTX3060 consumes slightly less energy per step than the RTX3090. The SupersoniK is also faster than both GPUs in each step computation again, by about 3.21X and consumes only 21.1% of the GPU energy. t_{clock} = 7:53*ns* and P_d = 17:58*W* for the SupersoniK. The utilization is 23% of the DSPs, 22% of BRAMs, 31% of FFs and 46% of the LUTs.

Device	N	M	T_s[s]	t_c[ns]	E_c[nJ]
RTX3090	16.4B	1	1.32	0.093	12.3
RTX3060	32.8B	1	1.18	0.081	15.4
SupersoniK	1.12B	1	0.114	0.031	1.17

Table 1: Time and Energy Consumption per Step Black-Scholes Model

Device	N	M	T_s[s]	t_c[ns]	E_c[nJ]
RTX3090	16.78M	1024	6.14	0.412	52.7
RTX3060	16.78M	1024	7.61	0.407	48.3
SupersoniK	21.12M	1024	4.03	0.128	11.14

Table 2: Time and Energy Consumption per Step Heston Model

Conclusion

The Black-Scholes and Heston models of financial products are described and implemented in this article. The results for several options are presented and compared for a number of GPU and SupersoniK, by analyzing the time and energy consumption by each MC simulation step. All models in this work are coded in Pyhton, to allow direct comparison between GPU and FPGA platforms, and in order to exploit a high-level model which still provides good control over the quality of the implementation.

Thsi work shows that energy per computation by using an FPGA can be from 9.5% to 21.1% (depending on the algorithm) as much as that by using an GPU as an accelerator for financial models, while performance can be from 3X to 3.21X as fast as the GPUs. One can conclude that the SupersoniK has a better overall performance than the advanced GPUs in option pricing problems, which is computation-bounded, rather than memory-bounded. On the energy efficiency aspect, the FPGA is 10X more frugal than the GPUs.

Copyright © 2021 VividSparks IT Solutions Pvt. Ltd.

References

[1] X. Tian and K. Benkrid, "Design and implementation of a high performance financial monte-carlo simulation engine on an fpga supercomputer," in ICECE Technology, 2008. FPT 2008. International Conference on, Dec 2008, pp. 81–88.

[2] M. Broadie and O¨ . Kaya, "Exact simulation of stochastic volatility and other affine jump diffusion processes," Operations Research, vol. 54, no. 2, pp. 217–231, 2006.

[3] T. Odelman, Efficient Mont Carlo Simulation with Stochastic Volatility. Skolan f¨or datavetenskap och kommunikation, Kungliga Tekniskah¨ogskolan, 2009.

[4] F. B. Muslim, A. Demian, L. Ma, L. Lavagno, and A. Qamar, Energy-efficient fpga implementation of the k-nearest neighbors algorithm using opencl," ANNALS OF COMPUTER SCIENCE AND INFORMATION SYSTEMS, vol. 9, pp. 141–145, 2016.

Copyright © 2021 VividSparks IT Solutions Pvt. Ltd.