ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 8(36), DOI: 10.17485/ijst/2015/v8i36/90318, December 2015* ISSN (Online) : 0974-5645

Monte-Carlo Black-Scholes Implementation using OpenCL Standard

Chirag Patel, M. Srikanth, Kalyani Chetan Kumar and S. Sivanantham*

School of Electronics Engineering, VIT University, Vellore - 632014, Tamil Nadu, India; ssivanantham@vit.ac.in

Abstract

The OpenCL is a standard parallel language which is based on 'C' language. It offers users to take full advantage and also provide the flexibility of high level language. In this paper, we explore the use of OpenCL language to implement the complex design on FPGAs by describing the design with high level abstraction language. To demonstrate, we consider the most important benchmarks in financial markets known as Monte-Carlo Black-Scholes implementation to estimate the stock price option

Keywords: Embedded Unit, FPGA Implementation, Kernel, Monte-Carlo Simulation, OpenCL Standard

1. Introduction

The Field Programmable Gate Arrays (FPGAs) consist of millions of programmable elements and interfaces to implement any complex designs. We can configure these FPGAs using low-level hardware description languages like Verilog Hardware Description Language (HDL) and Very high speed integrated circuits HDL (VHDL). These HDLs used to implement highly efficient logic circuits. However, there is no portability to convert this in to low-level assembly language for advanced embedded processors implementation. This limits the productivity of new design cycle and difficulty in adopting FPGAs for larger scale circuits' implementation.

Now-a-days in financial market, computation of option prices is going on through Monte Carlo method. Monte-Carlo Black-Scholes is popularly used technique in stock markets to compute the option price and it is also considered as a benchmark. This method is especially useful when solving problems where closed form solutions are not possible for certain applications. In this method, the underlying stock prices are evaluated with random simulation and the expected payoff is computed by taking the average over millions of different paths. Monte-Carlo pricing gives the present value based on random experiments and statistical analysis[1]. So, we attempt to implement Monte-Carlo Black-Scholes algorithms as an application in which design is described using OpenCL. FPGA's have offered greater performance than Central Processing Units (CPUs) which are useful for computational tasks like Monte-Carlo simulation[2].

In this paper, we explore the feature of OpenCL language to program on FPGAs for real-time applications. This is first kind of work using the OpenCL feature.

2. OpenCL Language

An OpenCL description based FPGA implementation provides many advantages cover conventional HDL based implementations. In OpenCL based implementation, the specification for the proposed application is coded using conventional programming languages like C of C++, where as HDL based implementation need complex architectural information to describe the design. OpenCL Programmable technologies consist of different programmability options. Being programming on processors, it contains list of instructions executed in the sequential order. Sometimes at the run time, some of the advance processors convert sequential instructions

* *Author for correspondence*

into parallel instructions. The other one being completely parallel execution of program it creates hardware circuit on FPGA. Recently, many technologies came into existence which allows parallel execution of program that is faster than sequential execution of a program[3]. Host program which is written in standard C/C++ language can communicate to FPGA through the kernel program which is programmed on the FPGA. Host program has access to Application Programming Interface (API) of OpenCL. It allows data transfer between the FPGA board and OpenCL with the help of kernel program. These kernel parallel threads to inputs which are provided by the host program. The parallel thread operates on each element of the input and computes the operations more quickly which is accelerated by parallelism such as FPGA.

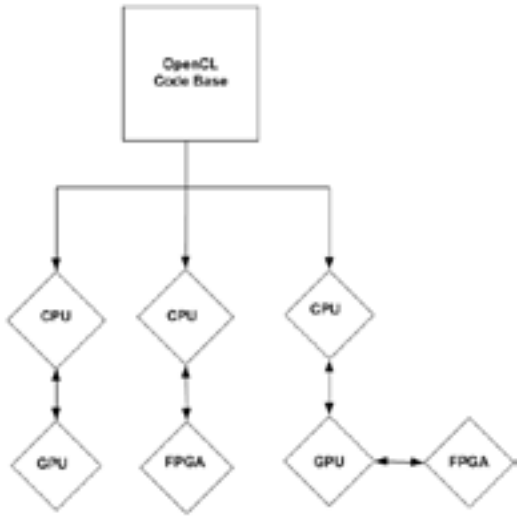

Figure 1. Portability of OpenCL.

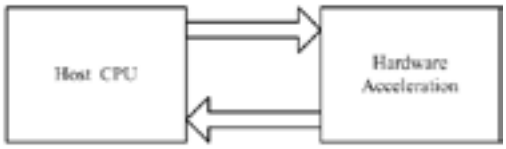

Figure 2. Hardware acceleration.

Open Computing Language (OpenCL) is a platform for writing programs and are executed in parallel manner that are independent of the platform, which means it will be executed on Central Processing Unit (CPU), FPGA, Graphic Processing Unit (GPU) as shown in Figure 1. The performance optimization can be achieved in FPGA with integrated CPU. This will enable even more efficient implementation of acceleration of FPGA's in the future. OpenCL programming consists of two parts, 1. Kernel, and 2. Host program as shown in Figure 2. Host program can be a standard C or C++ in Visual studio environment or GCC compiler that runs on any microprocessor like DSP, CPU, GPU etc. At some point during execution of this host program, there are some functions which are complex in computations and which get benefit from the parallel acceleration on parallel devices like FPGA's[4]. The function which is used to accelerate is OpenCL kernel. However, this OpenCL kernel program is written in the standard C language.

OpenCL host CPU and FPGA interfacing can be done two ways.

- External Host - Accelerators which are implemented on FPGA using kernel program can interact with Host CPU with the external environment through OpenCL host program.
- Embedded Host - in this Host, the CPU and FPGA are embedded on the SoC (System on Chip) which is a single integrated chip. Communication between them becomes easier without loss of data. No extra hardware is required for FPGA in the embedded host, which decreases the area.

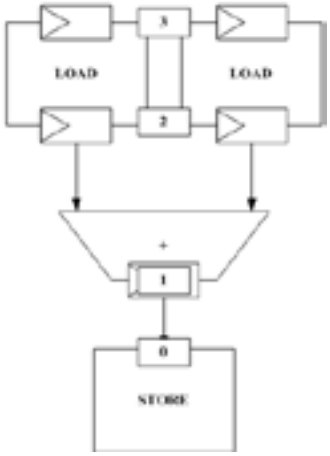

Figure 3. Pipeline of vector addition.

For example, Let A and B are considered as two arrays which perform vector addition. This addition operation can be performed on general microprocessor and FPGA. In microprocessor, the parallel threads which are created and executed on the different cores are bit slower than the FPGA. But the FPGA offer different strategies. Kernel function which is implemented on the FPGA is transformed into pipelined hardware circuits that use multithreading i.e., parallelism. In the first clock, the thread 0 is executed which means fetching of data is done

from the array A and B as shown in Figure 3. The thread 1 is clocked during second clock cycles, and at the same time, thread 0 has completed its addition operation as well as the result is stored in the register. The thread 2 is clocked during the third clock cycle, thread 1 completes its addition operations as well as the thread 0 returns its value. This is how the pipeline is done on FPGA using kernel program.

3. Monte Carlo Simulation for the European Stock Option

Underlying algorithm is an OpenCL (kernel) that combines three algorithms:

3.1 Mersenne Twister - Generation of Uniformly Distributed Pseudorandom Numbers

The Mersenne twister is used to generate a pseudo random number[5]. It generates very high- quality pseudorandom integers, compared to other algorithms. The variants of the algorithm differ only in the size of the Mersenne primes used. The newer and more commonly used is 32-bit word width. There is also a variant with 64-bit word width, which generates a different sequence. For k bit word width, the Mersenne Twister generates integers with an almost uniform distribution in the range. The Mersenne Twister has been optimized for use with Monte Carlo simulations in a number of fields using Matlab, Python, C++ and PHP. MT is the default random number generator.

3.2 Box-Muller Transform

Monte Carlo option algorithm requires normally distributed pseudorandom numbers for possible price paths generation. Mersenne twister provides uniformly distributed numbers, and then the Box-Muller transform converts this uniformly distributed numbers to normally distributed numbers which is required for Monte-carol option algorithm. It is commonly expressed as *z1* and *z2* and are defined as a normally distribution shown in Equations (1) and (2), with Mean = 0, Variance = 1.

$$Z1 = \sqrt{-2\times\ln(x1)}\times\cos(\prod x2) \quad (1)$$

$$Z2 = \sqrt{-2x\ln(x1)}x\sin(\prod x2) \quad (2)$$

3.3 Monte-Carlo Method and Black-Scholes Model

European options pricing has the exact closed form given by the Black-Scholes equation. This uses exact Black-Scholes formula implemented in C code on the host i.e., on PC to determine correctness of the Monte Carlo method. Black-Scholes equation computes the European option price shown in the Equation (3). The prices of call option are as follows:

$$S(t) = N(x1)S - N(x2)Ke^{-r(T-t)} \quad (3)$$

$$x1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right)+\left(r+\frac{\sigma^2}{2}\right)(T-t)\right] \quad (4)$$

$$x2 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S}{K}\right)+\left(r-\frac{\sigma^2}{2}\right)(T-t)\right] \quad (5)$$

where N( ) is cumulative distributive function, S is the Price, K is Strike Price, T-t is time to expiry, r is risk free rate. Monte-Carlo method is a computational algorithm to obtain numerical results, shown in Equation (6). Consider that price S,

$$dS = \alpha Sdt + \sigma S\varepsilon\sqrt{dt} \quad (6)$$

where εis standard normal deviation, αis capital gain rate, and σis volatility of stock price S. For, $\mu = \alpha + \gamma$ $dS = (\mu-\gamma)Sdt + \sigma S\varepsilon\sqrt{dt}$ and For , $\mu = r$ $dS = (r-\gamma)Sdt + \sigma S\varepsilon\sqrt{dt}$. The solution for the above equation,

$$S_t = S_0\left[\left(\alpha - 0.5\sigma^2\right)\Delta N(0,1)\Delta t\right] \quad (7)$$

4. Implementation Results and Discussions

Figure 4 shows the overall implementation flow for the proposed work. Host program which is written in standard C language needs to link the OpenCL host library file. This host library will map all its files to host program which converts to executable file. Kernel program is written in the OpenCL environment which has .cl extension. When kernel program is compiled it is converted to intermediate project, this step will takes

couple of minutes[6]. Now intermediate project which has created in the last step is compiled, this converts to hardware programming file which takes few hours on a fast workstation. Generated hardware programming file is programmed on the DE4 board. Launching the host program will enable the communication between them as shown in Figure 4.

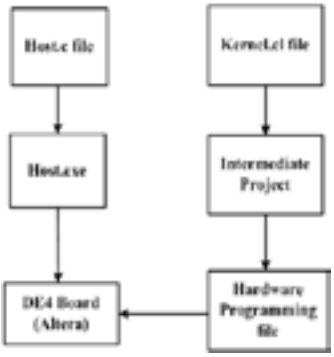

Figure 4. Implementation flow-chart.

Figure 5 shows an overall OpenCL system implementation architecture. It consists of multiple kernel pipelines and their peripherals[7]. Apart from kernel pipeline, OpenCL compiler creates interface between the external and internal memory. During each pipeline stage, global interconnection establishes the connections to external memory from both load and store units.

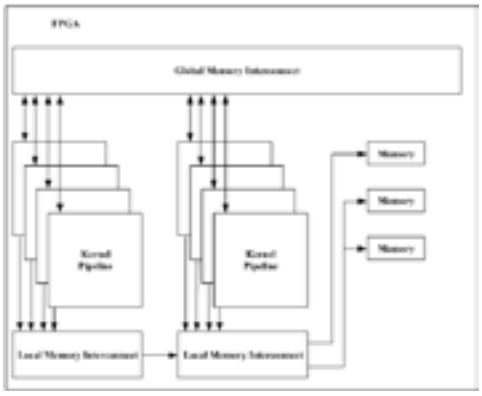

Figure 5. OpenCL system implementation.

We have implemented Monte-Carlo and Black-Scholes algorithm on Matlab environment and results are as shown Figure 6(a). Figure 6(b) shows the Monte Carlo results, where the output matches to the Black-Scholes. As the numbers of underlying paths are increased, Monte-Carlo almost matches to the Black-Scholes result.

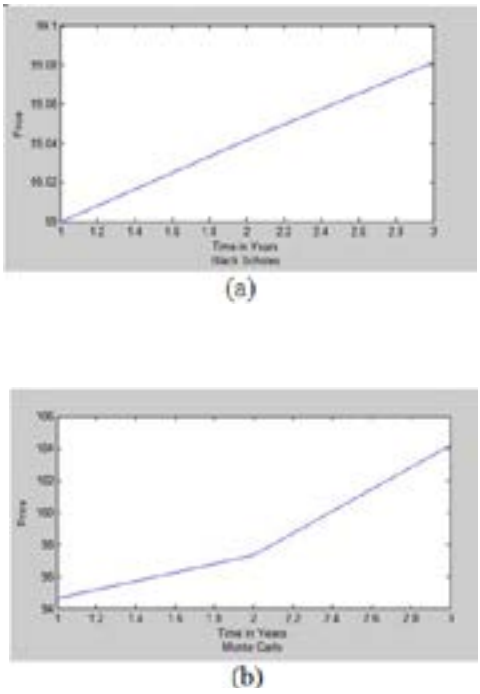

Figure 6. (a) Black-Scholes simulation. (b) Monte-Carlo simulation.

In our implementation, we have used the Mersenne twister random number generator to obtain uniformly distributed values. Then normally distributed sequences are produced by taking inverse normal cumulative density function. With the help of these random number, a Geometric Brownian motion based varying stock prices are estimated by performing simulation. The final call option payoff is recorded and averaged from the simulation results to produce an expected value for the payoff.

5. Conclusion

We have implemented the Monte-Carlo Black-Scholes algorithm using OpenCL standard. We also compared the obtained results from the Matlab and OpenCL, the output is approximately same. But the algorithm which is developed on Matlab takes more computational time in order of few seconds to compute the price where as the algorithm written on OpenCL platform computes in less time in order of milliseconds. The host code calls the OpenCL kernel which is programmed in FPGA and the OpenCL accelerates the FPGA makes computation faster. An OpenCL based FPGA implementation provided

significantly better performance than other hardware architectures implementations like CPU, GPUs, etc. Furthermore, an FPGA-based heterogeneous system (CPU + FPGA) described with OpenCL standard also lead to reduce the time-to-market pressure to the design implementation teak which is a common problem in design industries.

6. References

1. Thomas D. Acceleration of financial monte-carlo simulations using FPGAs. IEEE Workshop on in High Performance Computational Finance (WHPCF); New Orleans, LA, USA. 2010 Nov 14. p. 1–6.
2. Shagrithaya K, Kepa K, Athanas P. Enabling development of OpenCL applications on FPGA platforms. IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP); Washington, DC. 2013. p. 26–30.
3. Fraire J, Ferreyra P, Marques C. OpenCL overview, implementation, and performance comparison. IEEE Latin America Transactions, (Revista IEEE America Latina). 2013; 11(1):274–80.
4. Tian X, Benkrid K. Design and implementation of a high performance financial monte-carlo simulation engine on an FPGA supercomputer. International Conference on ICECE Technology, FPT 2008; Taipei. 2008. p. 81–8.
5. Chen J, Feng L. Using lower and upper bounds to increase the computing accuracy of monte carlo method. International Conference on Computational and Information Sciences (ICCIS); 2010. p. 630–3.
6. Sakamoto R, Sato M, Koizumi Y, Amano H, Namiki M. An OpenCL runtime library for embedded multi-core accelerator. IEEE 18th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA); Seoul. 2012. p. 419–22.
7. Grewe D, Wang Z, O'Boyle M. Portable mapping of data parallel programs to OpenCL for heterogeneous systems. IEEE/ACM International Symposium on Code Generation and Optimization (CGO); 2013. p. 1–10.