High Performance and Low Power Monte Carlo Methods to Option Pricing Models via High Level Design and Synthesis

High Performance and Low Power Monte Carlo Methods to Option Pricing Models via High Level Design and Synthesis

Liang Ma, Fahad Bin Muslim, Luciano Lavagno
Department of Electronics and Telecommunication
Politecnico di Torino
Turin, Italy
Email: {liang-ma, fahad.muslim, luciano.lavagno}@polito.it

***Abstract*—This article compares the performance and energy consumption of GPUs and FPGAs via implementing financial market models. The case studies used in this comparison are the Black-Scholes model and the Heston model for option pricing problems, which are analyzed numerically by Monte Carlo method. The algorithms are computationally intensive but not memory-intensive and thus well suited for FPGA implementation. High-level synthesis was performed starting from parallel models written in OpenCL and then various micro-architectures were explored and optimized on FPGAs. The final implementations of both models to several options on FPGAs achieved the best parallel acceleration systems, in terms of both performance-per-operation and energy-per-operation, compared not only to the kernels on advanced GPUs but also to the RTL implementations found in the literatures.**

Keywords–Acceleration; High-level synthesis; GPU; FPGA; Parallel computation; Pipelining; Unrolling;

I. 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 [1]. Both the Black-Scholes model and Heston model are popular models for asset prices, which are also known as stock prices. The most significant feature of the Black-Scholes model is its simplicity, since all the other parameters are treated as constant. Instead, the Heston model describes both the stock price and its volatility by stochastic differential equations, therefore considering more parameters than the Black Scholes model. As a result, it is able to reflect the market characteristics more precisely [2].

The stochastic differential equations used by both the models do not have any feasible analytical solution that can be used for option pricing problems [3], especially when considering non-standard (also known as "exotic") options. 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 computationalresourcestobeappliedtothesestochastic 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.

The predominant design flow for FPGA and ASICs alike is based on Register Transfer Level (RTL) models written in a Hardware Description Language, which are then synthesized, placed and routed. However, this design flow is very time-consuming, because changes to the RTL in order to modify performance and cost have to be painstakingly coded manually and extensively verified for correctness. On the other hand, the standard software development flow, based on the principle of "write once, run anywhere" is very appealing, e.g. of many algorithms written in C, C++ and OpenCL for CPUs and GPUs alike. Recent advances in high-level synthesis (HLS) for software languages finally have dramatically reduced the design and verification costs, essentially eliminating the need to model the design at RTL. HLS, on the other hand, promises the best of both

worlds: the high performance and low energy consumption of FPGA hardware, and the flexibility and retargetability of software. In particular, both Xilinx and Altera FPGAs can be programmed by OpenCL models. The choice of GPU-oriented languages like OpenCL or CUDA as the input to HLS has two main motivations:

- They offer massive amounts of parallelism that can be exploited in the context of HLS by unrolling or pipelining the loops.
- They provide an explicit memory model with global memory, shared memory and private memory that matches performance and granularity of FPGA memory (off-chip DRAM, on-chip SRAM and registers).

This paper describes the implementation and architectural optimization of MC methods to accelerate the financial applications on GPU- and FPGA-based hardware accelerators. The performance and energy consumption on both platforms are compared and analyzed. The advantages of using HLS for the FPGA design are further analyzed, discussed and compared with manual RTL designs in the literatures.

II. Related Work

On account of the fact that the option pricing problem is one of the hot topics in financial market, it has been investigated by many researchers theoretically or practically.

Stephen et al. [4] reported that the algorithms for analyzing tranche credit derivatives could be executed on an FPGA over 30 times faster than those on a multi-core processor. In addition, they also showed that modeling the algorithms in a high-level language reduced the design effort to about one fourth. Christian et al. [2] implemented a MC method to analyze European barrier option pricing problems on a laptop CPU, a laptop GPU and an FPGA. They concluded that the acceleration by both the GPU and the FPGA saved time and energy. Furthermore, the FPGA offers a better balance between energy and performance than the GPU. Of course, the result of any comparison between GPU and FPGA highly depends on the algorithms and the technology node used to manufacture the devices. For instance, Anson et al. [5] demonstrated that an FPGA outperformed GPU in terms of both performance and energy, by implementing the MC method to Asian option using the Heston model.

Several HLS techniques have been implemented as both research and commercial tools in recent years. For example Handel-C was described in [6] while THDL++ was discussed in [2]. In [1] the authors took advantage of a high-level synthesis tool to generate one component of a system for multi-level MC simulation.

A comparison between GPU and FPGA has also been performed for several other algorithms, such as the k-Nearest Neighbor in [7].

III. 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)[8].

$$dS = rSdt + \sigma Sdz \tag{1}$$

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 Itô's lemma [9], the analytical solution for the stochastic differential equation (1) is shown in (2).

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

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][10] for $\Delta t << 1$.

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

(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. On the other hand, some majority commercial HLS tools don't even support the synthesis of exponential function. However (2) gives more accurate result in case of small amount partitions over time. Since the HLS tool (VIVADO_HLS) used in this work can deal with the exponential function, (2) has been implemented.

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[10].

$$dS = rSdt + \sqrt{V}Sd(\rho z_1 + \sqrt{1-\rho^2}z_2) \tag{4}$$

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

In (4), z_1, z_2 are two Wiener processes, ρ is the correlation factor between them, and $\sqrt{V}$ 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 $\Delta t << 1$, V can be assumed to be constant, so that Itô'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[10] with full truncation scheme avoiding negative values under the square root [8]. The final solutions are shown in (6) and (7).

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

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

where $\epsilon_1, \epsilon_2 \sim N(0,1)$ 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).

$$P_{Call} = max\{S_T - K, 0\} \tag{8}$$

where T is the pre-set time of the option, S_T 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).

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

3) Asian Option: The Asian option, which is also called average value option, is an exotic option. The payoff price depends on the average of the stock price over the time period. (10) defines the call price by using the arithmetic mean, while (11) defines the call price by using the geometric mean. (10) is implemented in this paper.

$$P_{Call} = max\{\frac{1}{T}\int_0^T Sdt - K, 0\} \tag{10}$$

$$P_{Call} = max\{e^{\frac{1}{T}\int_0^T ln(S)dt} - K, 0\} \tag{11}$$

IV. Hardware 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 computatioin.

- The time interval $(0, T)$ is uniformly partitioned into M steps and denoted as $t_0, t_1, \ldots, t_M$.
- M standard normally distributed independent random numbers $\epsilon_1, \epsilon_2 \ldots, \epsilon_M$ are generated.
- Calculate $S_T = S_{t_M}$ starting from S_{t_0} by (2) or (3).

This simulation that computes S_T from S_{t_0} 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
for $i = 1$ to N **do**
  for $k = 1$ to M **do**
    $U_1, U_2 \leftarrow MersenneTwist()$
    $\epsilon_1, \epsilon_2 \leftarrow BoxMuller(U_1, U_2)$
    $S_{t_{k+2}}, S_{t_{k+1}} \leftarrow Price(S_{t_k}, \epsilon_1, \epsilon_2)$
    $k+ = 2$
  end for
  $P_{option}[i] \leftarrow Option(S_t[], K)$
  i++
end for
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
for $i = 1$ to N **do**
  for $k = 1$ to M **do**
    $U_1, U_2 \leftarrow MersenneTwist()$
    $\epsilon_1, \epsilon_2 \leftarrow BoxMuller(U_1, U_2)$
    $S_{t_{k+1}} \leftarrow Price(S_{t_k}, V_{t_k}, \epsilon_1)$
    $V_{t_{k+1}} \leftarrow Volatility(V_{t_k}, \epsilon_2)$
    k++
  end for
  $P_{option}[i] \leftarrow Option(S_t[], K)$
  i++
end for
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 T_s 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

TABLE I
GPU PLATFORMS

Model Name	f[MHz]	Cores	Power[W]
GeForce GTX 960	1178	1024	120
Quadro K4200	784	1344	108

TABLE II
FPGA PLATFORMS

Models and Parts	BRAMs	DSPs	FFs	LUTs
Virtex-7(xc7vx690t)	2940	3600	866400	433200
Virtex-5(xc5vfx70t)	296	128	44800	44800
Virtex-5(xc5vlx330t)	648	192	207360	207360

with normalization respect to C. As defined in (12), t_c is the time for computing one simulation step.

$$t_c = \frac{T_s}{C} \tag{12}$$

$$E_c = t_c \cdot P_d \tag{13}$$

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 (13), where P_d is device power). Compared to a GPU, an FPGA may not be as fast, due to e.g. the use of an older process or a narrower and slower DRAM interface. However, it is typically more energy-efficient, because it has an application-specific control and data-path

In this paper, 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 synthesis tool (e.g. Vivado from Xilinx).

B. Heterogeneous Platforms

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 [7].

In a GPU, there are plenty of independent cores in order to execute kernels in parallel and each core contains several computing elements (ALUs) for SIMD (or SIMT) approach. Apart from these, frequency and the memory (including cache) sizes also affect the GPU performance. The GPU devices used in this paper are listed in Table I.

The FPGA devices that are supported by the SDAccel OpenCL synthesis tool that was used for this paper are the Virtex7-series and Kintex7-series. To compare the performance of such kernels designed via HLS with the implementations from the literatures, the kernels are also synthesized on the FPGAs, e.g. Virtex-5-series that were used in those publications. The FPGA devices are presented in Table II.

C. Algorithm optimization on Hardwares

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

Firstly, partial unrolling of the outer-most loop is implemented as on GPU. The unroll factor N_u depends on the percentage of resources utilization of the rolled iterations (N_s paths). However, the HLS tool does not support the unrolling of out-most loop currently. So some techniques have been taken to adapt the architecture of the algorithms such as modifying the orders of the nested loops.

Secondly, 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 Virtex-7-series FPGA at 100MHz 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 100MHz on Virtex-7-series FPGA 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 (14) and (15) for the two algorithms respectively.

$$t_c^{B.Scholes} = \frac{t_{clock}}{N_u} \tag{14}$$

$$t_c^{Heston} = \frac{2t_{clock}}{N_u} \tag{15}$$

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

Finally, 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. This optimization is realized by the directives provided by the HLS tool. 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.

V. 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).

A. European Vanilla Option

This section presents the results of implementing the European vanilla option using both two models, and then comparing them across platforms.

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 III, where "**B**" denotes **B**illion. Clearly, the GTX960 is better than the K4200 for this application, in terms of both performance and energy per time step. Compared to the GTX960, the Virtex-7 has 1.71X speed and only consumes 9.8% of the energy per step.

$t_{clock} = 6.08ns$ and $P_d = 21.2W$ for the FPGA Virtex-7. The resource utilization is 86% of the DSPs, 26% of BRAMs, 34% of FFs and 70% of LUTs.

2) Heston Model: The results for the Heston model are shown in Table IV, where "**M**" denotes **M**illion. The GTX960 in this case has better performance than the K4200 again. However, the K4200 consumes slightly less energy per step than the GTX960. The Virtex-7 is also faster than both GPUs in each step computation again, by about 2.56X and consumes only 5.9% of the GPU energy.

$t_{clock} = 7.53ns$ and $P_d = 17.58W$ for the FPGA Virtex-7. The utilization is 66% of the DSPs, 26% of BRAMs, 34% of FFs and 50% of the LUTs.

TABLE III
TIME AND ENERGY CONSUMPTION PER STEP, BLACK SCHOLES MODEL

Device	N	M	T_s**[s]**	t_c**[ns]**	E_c**[nJ]**
GTX960	16.4**B**	1	2.69	0.164	19.7
K4200	32.8**B**	1	6.66	0.203	21.3
Virtex-7	2.15**B**	1	0.205	0.0958	1.94

TABLE IV
TIME AND ENERGY CONSUMPTION PER STEP, HESTON MODEL

Device	N	M	T_s**[s]**	t_c**[ns]**	E_c**[nJ]**
GTX960	16.78**M**	1024	10.37	0.604	72.4
K4200	16.78**M**	1024	11.40	0.663	69.7
Virtex-7	33.55**M**	1024	8.11	0.236	4.14

TABLE V
EXECUTION TIME, ASIAN OPTION

Result	**FPGA**	T_s**[s]**	**BRAM**	**DSP**	**FF**	**LUT**
Article [5]	V-5-330	18.3	3%	93%	62%	38%
This study	V-5-330	7.64	8%	97%	12%	16%

B. Exotic Option

This section presents the results for two exotic options, including a comparison with previously published FPGA implementations.

1) Asian Option, Black-Scholes Model: For this model evaluation, the same parameter values and FPGA platform as [5] have been implemented in order to meaningfully compare performance. Table V shows T_s and the resource utilization. Even though the resource balancing has not been applied to this system in this study, the HLS-based implementation is still about 2.4X as fast as the one presented in [5], which modeled the algorithms in RTL.

The clock frequency $f_{clock} = 125MHz$ implemented in this work and $f_{clock} = 200MHz$ in [5]. According to the resource utilization and the clock frequency, one can conclude that the FPGA power from this work is not more than that from the literature [5]. Even if the device powers of both works are assumed to be identical, the energy consumption can be considered roughly proportional to the execution time T_s. It means the parallel computation system designed in this paper (even though not fully optimized by IP core controlling) via HLS saves at least 58.3% energy of the one designed in RTL.

2) European Barrier Option, Heston Model: The parameters for European barrier option by Heston model and the

TABLE VI
EXECUTION TIME, EUROPEAN BARRIER OPTION

Result	FPGA	T_s[s]	BRAM	DSP	FF	LUT
Article [2]	V-5-70	4	9%	99%	36%	54%
Unopt.	V-5-70	3.13	8%	99%	29%	44%
Opt.	V-5-70	1.56	16%	95%	47%	90%

part of FPGA were also chosen identical to those from [2] for the sake of comparison. The unoptimized implementation in this work achieved 20% better performance than the one reported in [2], which is also based on HLS. By using a more aggressive IP cores control scheme, which reduces the resource utilization for an acceleration unit, it is able to increase the value of N_u, and thus improves the overall speedup to about 2.56X with respect to that in the literature. These results are shown in Table VI.

$f_{clock} = 100MHz$ for both designs in this work and the literature. The resource usage of unoptimized system is comparable to that in the literature. Since the utilization of optimized system is no more than twice of the unoptimized one, the power is assumed to be twice of the system in the literature in the worst case. In total, the energy consumption is roughly 78.1% of the system from literature.

VI. 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 FPGA platforms, by analyzing the time and energy consumption by each MC simulation step. All models in this paper are coded in OpenCL/C++, to allow direct comparison between GPU and FPGA implementations, and in order to exploit a high-level model which still provides good control over the quality of the implementation.

The comparison of performance and energy consumption between GPU and FPGA corroborates previous results from the literature. In particular, this paper shows that energy per computation by using an FPGA can be from 5.9% to 9.8% (depending on the algorithm) as much as that by using an GPU as an accelerator for financial models, while performance can be from 1.71X to 2.56X as fast as the GPUs. One can conclude that the Virtex-7 FPGA 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.

The implementations in this work are also significantly better than those in previous works in the domain of FPGA acceleration of financial models. For the Black-Scholes model of the Asian option problem, 2.4X of the performance and 41.7% of energy consumption are obtained compared to a previous manual RTL design. For the Heston model of the European barrier option, this paper has achieved 2.56X of the performance and 78.1% of energy consumption of a previous implementation designed via HLS. Overall, it shows that HLS not only reduces the effort in system design, but also achieves higher performance and lower energy consumption than the traditional RTL design approach.

The random number generation algorithm implements floating-point numbers and some complicated mathematical functions such as sin/cos. It can be optimized in future to reduce the resource utilization on FPGA. In addition, the authors are also planning to explore other efficient numerical methods, such as the multi-level MC method reported in [1].

ACKNOWLEDGMENT

The authors woud like to give thanks to Xilinx Inc. for their support. This work is also partially supported by the European Commission through the ECOSCALE project (H2020-ICT-671632).

REFERENCES

[1] C. de Schryver, P. Torruella, and N. Wehn, "A multi-level monte carlo fpga accelerator for option pricing in the heston model," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 248–253.

[2] C. d. Schryver, I. Shcherbakov, F. Kienle, N. Wehn, H. Marxen, A. Kostiuk, and R. Korn, "An energy efficient fpga accelerator for monte carlo option pricing with the heston model," in *2011 International Conference on Reconfigurable Computing and FPGAs*, Nov 2011, pp. 468–474.

[3] R. Sridharan, G. Cooke, K. Hill, H. Lam, and A. George, "Fpga-based reconfigurable computing for pricing multi-asset barrier options," in *Application Accelerators in High Performance Computing (SAAHPC), 2012 Symposium on*, July 2012, pp. 34–43.

[4] S. Weston, J. T. Marin, J. Spooner, O. Pell, and O. Mencer, "Accelerating the computation of portfolios of tranched credit derivatives," in *High Performance Computational Finance (WHPCF), 2010 IEEE Workshop on*, Nov 2010, pp. 1–8.

[5] A. H. T. Tse, D. B. Thomas, K. H. Tsoi, and W. Luk, "Dynamic scheduling monte-carlo framework for multi-accelerator heterogeneous clusters," in *Field-Programmable Technology (FPT), 2010 International Conference on*, Dec 2010, pp. 233–240.

[6] G. W. Morris and M. Aubury, "Design space exploration of the european option benchmark using hyperstreams," in *2007 International Conference on Field Programmable Logic and Applications*, Aug 2007, pp. 5–10.

[7] 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.

[8] T. Odelman, *Efficient Mont Carlo Simulation with Stochastic Volatility*. Skolan för datavetenskap och kommunikation, Kungliga Tekniska högskolan, 2009.

[9] 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.

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