An Energy Efficient FPGA Accelerator for Monte Carlo Option Pricing with the Heston Model

Christian de Schryver, Ivan Shcherbakov,
Frank Kienle, Norbert Wehn
Microelectronic Systems Design Research Group
University of Kaiserslautern
Erwin-Schroedinger-Str., 67663 Kaiserslautern, Germany
{schryver, shcherbakov, kienle, wehn}@eit.uni-kl.de

Henning Marxen, Anton Kostiuk, Ralf Korn

Stochastic Control and Financial Mathematics Group
University of Kaiserslautern
Erwin-Schroedinger-Str., 67663 Kaiserslautern, Germany
{marxen, kostiuk, korn}@mathematik.uni-kl.de

***Abstract*—Today, pricing of derivates (particularly options) in financial institutions is a challenge. Besides the increasing complexity of the products, obtaining fair prices requires more realistic (and therefore complex) models of the underlying asset behavior. Not only due to the increasing costs, energy efficient and accurate pricing of these models becomes more and more important. In this paper we present - to the best of our knowledge - the first FPGA based accelerator for option pricing with the state-of-the-art Heston model. It is based on advanced Monte Carlo simulations. Compared to an 8-core Intel Xeon Server running at 3.07GHz, our hybrid FPGA-CPU-system saves 89% of the energy and provides around twice the speed. The same system reduces the energy consumption per simulation to around 40% of a fully-loaded Nvidia Tesla C2050 GPU. For a three-Virtex-5 chip only accelerator, we expect to achieve the same simulation speed as a Nvidia Tesla C2050 GPU, by consuming less than 3% of the energy at the same time.**

***Index Terms*—Financial Mathematics, Option Pricing, Hardware Accelerator, Heston Model, Monte Carlo, FPGA**

I. Introduction

Nowadays, financial products have reached an impressive level of complexity and are expected to further increase their intricacy in the future. This has been made possible by the continuous advancements of the underlying mathematical models. As a consequence, the computational effort needed to accurately price modern products has grown significantly over time. This not only leads to higher computation times in general, but also to an immense increase in energy costs [1].

In many cases, product pricing relies on solving partial differential equations. In general, this is a non-trivial task that very often requires compute-intense stochastic simulation methods. These methods are mainly implemented on CPU and recently GPU clusters, and are usually examined in the context of the high performance computing (HPC) domain.

However, HPC is currently facing an immense energy problem. Therefore also financial institutes are forced towards alternative computing architectures like FPGAs, that provide high-end computational capabilities at a very low power consumption. At the same time, not only in automated trading systems tight timing constraints apply that require prices to be computed as fast as possible [2]. It is a challenge for the financial institutes to perform their HPC pricing under energy and time constraints at the same time. Efficient hardware acceleration is key to overcome this issue.

A very common (but usually non-trivial) pricing task is pricing derivatives such as options. Developing hardware accelerators for option pricing has been a very active research area for many years now. In the FPGA community, most of the publications up to now rely on the Black-Scholes model. However, the Black-Scholes model is known in the financial mathematics community to no longer reflect the real behavior of stock prices in most markets [3]. More accurate but also more complex is the Heston model that is widely accepted in the financial domain nowadays [4].

In this paper, we present an FPGA based hardware accelerator for option pricing with the Heston model. We focus on pricing European barrier options using the Monte Carlo method. We use a single precision FPGA implementation together with double precision statistics computation on a host PC (like Jin, Luk and Thomas [5]). This hybrid approach allows us to benefit most from both architectures: Compute intensive computation kernels with low-level arithmetic can be accelerated efficiently on the FPGA, complicated arithmetic only used in small amounts of the overall time stay on the host CPU.

We have set up a comprehensive benchmark set [6] that we apply to our implementation in order to ensure the correct functionality. Additionally, we give detailed numbers for speed and energy consumption. For a Xilinx Virtex-5 device, we provide synthesis results. We show that our accelerated system saves about 89% of energy compared to a fully loaded 8-core server, by achieving twice the speed at the same time. Compared to a state-of-the-art Nvidia Tesla C2050 graphics card, it provides only around 35% of the speed, but still saves 60% of energy. A chip only estimation shows that three FPGAs with our accelerator can achieve the same speed as the Tesla GPU, consuming less than 3% of the energy.

The contributions of our work in summary are:

- We present the first hardware accelerator for option pricing with the state-of-the-art Heston model.
- We give precise measured numbers for throughput and energy consumption and compare our implementation with CPU and GPU designs, based on a standardized

978-0-7695-4551-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ReConFig.2011.11

Heston benchmark set.
- We show our validation strategy that we use to ensure the quality of our design.

II. Related Work

To the best of our knowledge, no hardware accelerator for option pricing with the Heston model has been published until now. However, a lot of FPGA architectures that rely on the Black-Scholes model have been presented over the last years. Luk, Thomas et al. have carried out comprehensive research activities on a large variety of suitable architectures for different algorithms like Monte Carlo methods, explicit finite difference methods or quadrature methods. They recently have introduced convenient metrics to evaluate their work [5], where they give numbers for single precision floating or fixed point implementations. They suggest to use Monte Carlo methods only if no other solvers are available. However, we are targeting to price more complex option types in the future (see Section III), therefore we implement the Monte Carlo method in our design. Again for the Black-Scholes model, by using a hybrid FPGA-CPU cluster Weston et al. have shown in 2010 that they can achieve a speedup of more than $31x$ compared to a CPU-only implementation [7]. Their loaded hybrid system consumes 6% less power than the CPU-only system under consideration, that means it saves 97% of energy per simulation.

A more generally applicable methodology for automatic generation of financial Monte Carlo simulations has been presented by Thomas et al. in 2007 [8]. In their work they have already considered a non-constant volatility, in particular for the GARCH model where the current volatility depends on the simulation history [9]. We have manually implemented our presented hardware architecture in a similar manner, adding additional hardware to cope with the correlated stochastic processes for price and volatility in the Heston model.

However, we could not find any publications that show hardware accelerators for pricing the Heston model. Recently, the Heston model has been ported to GPUs [10], and available papers show the massive potentials of GPU acceleration compared to CPU only simulations. Zhang and Oosterlee have investigated several workload splits between CPU and GPU in 2010 [11] to acclerate Heston pricing. For the practical showcase of European options they have shown that they achieve the highest speedup if most of the basic arithmetic operations on the high numbers of paths are performed directly on the GPU. This avoids the bottleneck of limited bandwidth between CPU and GPU, and is directly applicable to Monte Carlo simulations. Bernemann et al. from the German bank WestLB have recently investigated that for Monte Carlo based Heston pricers they achieve up to 340 GFlops on a Nvidia Tesla 1060 GPU card, compared to the maximum of about 11 GFlops on an Intel Xeon E5620@2.4GHz CPU [12].

Option pricing with the Heston model is more complex and therefore requires much more computational effort than with the Black-Scholes model. However, none of the GPU papers has investigated energy aspects that are becoming more and more important in the HPC domain. By considering the status quo for FPGA based Black-Scholes and GPU based Heston acceleration, we expect a very high benefit for energy efficiency and speed from using FPGA based accelerators for pricing the Heston model. We show that this holds in Section VI.

The next section briefly summarizes the mathematical background of option pricing and the Heston model.

III. Option Pricing with the Heston Model

An option is a contract between two parties for a future transaction on an asset on a reference price, the *strike* price. The buyer of the option gains the right, but not the obligation, to exercise the transaction. For a European option the buyer can only exercise the transaction at *maturity* time T. Depending on whether he would gain or loose money at maturity, he will exercise the option or let it expire. This transaction can be reflected in a monetary *payoff*.

In this paper we focus on the pricing of European barrier options, that are popular exotic options and intensively traded "over the counter". *Exotic options* is a term describing all kinds of options but the simplest onces.

The payoff of barrier options depends on whether an underlying asset has hit one or two prespecified *barrier(s)* until the maturity date. For example, a single barrier up-and-out option is active only if the price of the underlying asset remains below the barrier during the option's lifetime. In the case of a single barrier up-and-out *call* option, the payoff will be the exceeding of the strike price by the asset price at the maturity. But this only holds if the barrier has not been hit, otherwise the payoff will be zero. It will also be zero if the asset price at maturity is below the strike. The strike and the barrier are fixed in the option contract when the option is bought.

As the payoff can be derived from the price path of an asset, the fair price for the option is deduced from that asset as well. This shows the necessity for an underlying model to describe the characteristics of the future asset price development.

The Black-Scholes model has been presented in 1973 and led to a boom in option trading. However, in the model a constant volatility is assumed that can not be observed in real market behaviors. The Heston model generalizes the Black-Scholes formula and has a stochastic volatility [3]. It consists of two stochastic differential equations, that describe the dynamics of the option's underlying *asset price* S and its *volatility* V:

$$dS(t) = \quad rS(t)dt + \sqrt{V(t)}S(t)dW_1(t) \tag{1}$$

$$dV(t) = \quad \kappa(\theta - V(t))dt + \sigma\sqrt{V(t)}dW_2(t) \tag{2}$$

Here, W_1 and W_2 are two Brownian motions with the correlation ρ that model the randomness of the market. t is the time, and the other parameters further specify the specific behavior of the financial market [3].

In this setting, a fair price of an option can be calculated as its discounted expected payoff. In general there are no (semi-)closed form solutions (except when the the riskless

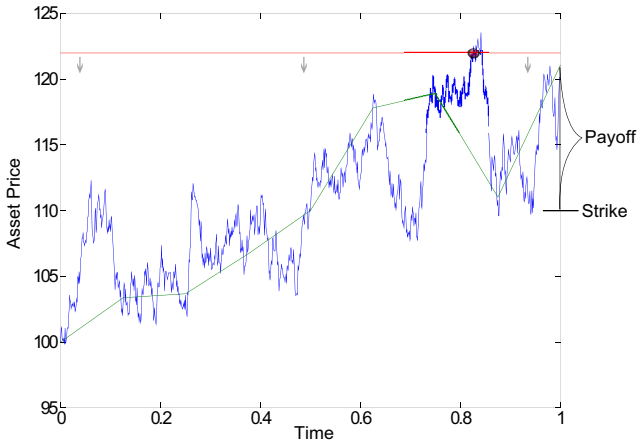


Fig. 1. Asset Price Process and its Discretized Version

interest rate r and the correlation ρ are zero) and therefore numerical methods are needed to approximate the fair price of a barrier option.

Our design is based on the Monte Carlo algorithm, that is known to be very robust and applicable for a wide range of problems [13]. It can be applied to price nearly all available European exotic option types, for example barrier options and path-dependent multi-asset options. On top of that, Monte Carlo algorithms possess natural parallelism and flexibility properties. Furthermore, we are going to enhance our accelerator to multi level Monte Carlo processing in the future, that allow significant speedups and better convergence behavior than single level methods implemented so far [14].

For the Monte Carlo algorithm we simulate a large amount of discrete approximations of stock prices and volatility paths by using a specific discretization scheme, in our implementation the Euler-Maruyama scheme [13]. One asset price process simulation and the discretized version can be seen in Figure 1. The discretized process is simulated by iteratively computing the discretization steps for both the asset price process and the volatility process. At the maturity time the payoff of the option for the simulated underlying asset price is calculated. The discounted mean value of the payoffs is the approximation for the option price. As the volatility process V is always non-negative, its simulations is a challenging algorithmic issue. Therefore in our implementation we combine the Euler-Maruyama scheme with the full truncation technique [4] to simulate the volatility. In addition, in the case of barrier option pricing we check the "barrier hit" event for each simulated path. But as we simulate only discrete approximations of the price paths, we use the barrier correction technique [15] to speedup the convergence of the Monte Carlo algorithm. These account for the underrepresentation of hitting the barrier by shifting the barrier on the discrete times. This is implied by the arrows in Figure 1.

One can see that even within the Monte Carlo methods there are many different algorithmic varieties that influence the speed of the algorithm. Together with all possible implementation decisions, a large design space exists for the accelerator in total. An efficient implementation requires to take algorithmic and hardware choices into account at the same time.

Option Type	At-The-Money Double Barrier Knock-Out Option						
Asset Price Parameters	S_0	V_0	r	κ	θ	σ	ρ
	100	0.04	0	0.5	0.04	1	0
Option Parameters	Lower Barrier		Upper Barrier		Strike		
	90		110		100		
Option Price	0.7487						

TABLE I
BENCHMARK SET

IV. Validation and Quality Assurance

Besides thorough bit-by-bit testing of each component in a hardware-in-the-loop setup, we have performed excessive validation on the application level. We have recently developed a standardized benchmark set for option pricing with the Heston model [6], that summarizes several practically relevant stress test scenarios provided from the financial mathematics literature.

The benchmark consists of the parameters for the option and the underlying asset price process. Furthermore the price for the option is provided. Table I shows all numbers for an exemplary option from the benchmark set.

Our benchmark set is freely available for download[1], and we strongly encourage authors to apply it to their future designs.

In order to ensure the high quality of our implementation, we have simulated all specified parameter sets from this benchmark and observed that they converge to the given results.

V. Architecture

The main goal for FPGA based Heston model accelerators is to minimize the energy consumption and to maximize the performance at the same time. Thus, we have chosen the following partitioning between the software part running on the PC and the hardware part implemented in FPGA:

- The hardware accelerator generates random numbers, simulates the paths including the barrier checking and computes the final price for each path. These are the kernels of the whole simulation algorithm that can be executed in parallel.
- The final price for each path is transmitted to the PC over an USB interface (based on FT2232H module; top average measured throughput is 6 MB/s).
- The PC analyzes the statistics and computes the final option price and variance.

For the Brownian motion inputs, the Heston model accelerator utilizes our non-uniform random number generator presented at ReConFig 2010 [16]. It allows an arbitrary output precision at low hardware costs.

To provide the maximum flexibility, we have implemented a configuration protocol that allows to dynamically reconfigure

[1] http://www.uni-kl.de/benchmarking

the accelerator parameters for the Monte Carlo simulation, the asset, and the option at runtime. These functions are transparently available as a software communication framework used in the host program.

We have implemented our hardware part on a Xilinx ML-507 development board (based on XC5VFX70T FPGA). Our accelerator can easily be ported to any other Xilinx FPGA supported by the Xilinx floating point IPCore library.

The proposed architecture is strongly related to the automatically generated simulator design from Thomas et al. [8]. The platform interface for the host connection is realized over an FT2232H mini-module from FTDI right now, but can be exchanged to different interfaces easily. Since this interface is register mapped, instead of using a bus we directly connect each accelerator to the related registers in the interface module.

The accelerator implementation consists of two main parts: the data path and the control logic. The data path is maximally pipelined. In contrast to the work from Thomas et al. [8] we do not use strict C-slow retiming as described by Weaver et al. [17], but also exploit the pipelined architecture by simulating paths in parallel.

For simplicity reasons, we have used a *packet concept* throughout the design:

- Each packet contains the current state of a path (price, volatility, step number, etc.) and a validity flag. Instead of having complex early termination strategies for paths that have hit a barrier, we introduce *dummy packets* with a cleared validity flag. This decreases the throughput to some extent, but at the same time reduces the hardware complexity.
- The data path is a pipeline that computes price and volatility for the next step and performs the barrier checking (see Section III). In every clock cycle, it consumes one packet and produces another one.
- The pipeline latency with 32-bit single precision floating point numbers is 60. This means that at every clock cycle the pipeline outputs a packet that was sent to it 60 cycles earlier.
- When a packet goes through the pipeline, its contents are updated according to the chosen algorithm for solving the Heston model from Section III.

The output of the data path is connected to a queue (FIFO unit) with a size greater than the pipeline depth. This allows building the data path from simple pipelined floating point cores and does not require support for stall signals. In our case we have exploited the maximum depth of a BRAM36 slice from the target Virtex-5 device for the queue. Figure 2 illustrates the relation between the data path, queue and the control logic.

The random number generator provides one random number in nearly every clock cycle. However, simulating the asset price and its volatility for each path in parallel requires two correlated random numbers in the Heston model. Therefore we use antithetic paths [13], that employ the same number pair for two different paths, with the inverted values in the second one. So in average only one random number per clock cycle is needed (that means we can start / continue one packet per clock cycle).

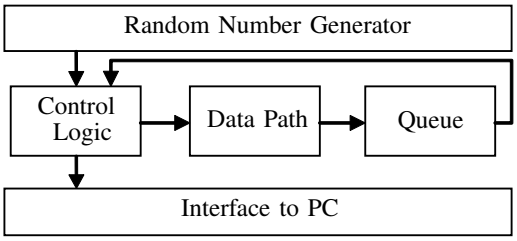


Fig. 2. High-Level Architecture

Component	Adders	Multipliers	Subtractors	Sqrt
Heston Step Generator	4	6	2	1
Barrier Checker	1	1	1	0

TABLE II
FLOATING POINT COMPONENTS USED IN THE DATA PATH

The control logic decides whether to consume a pair of random numbers, whether to send a packet to or to pop a packet from the queue and whether to send a price value to the PC. Both, the random number generator and the PC, can stall the transaction using a push-pull handshaking system, so they do not need to consider the state flow inside the control logic.

A simple set of rules defines the behavior of the control logic:

- If the amount of created packets is less than the queue size, a new path is created.
- If enough packets are active, the control logic checks if a packet is available at the queue.
- If the queue contains a packet, its step number is checked. If this was the last step, the final price is sent to PC and a new packet is created. If not, the packet is resent to the pipeline along with a new pair of random numbers.

The decomposition between the control logic and the data path significantly reduces the validation effort:

- The pipeline can be tested separately from the control logic in a separate testbench.
- The control logic can be tested separately by using a dummy pipeline that only counts the steps and has no floating point logic.

Our pipeline has a structure that is similar to the GARCH example presented by Thomas et al. [8], but includes the Heston specific modifications. Like their data path, our implementation only requires a very low number of floating point components. Table II shows the usage of floating point units separately for the Heston step generator (that generates successive values for price and volatility) and the subsequent barrier checking.

The pipeline consists of pipelined floating point units (adders, multipliers, subtractors and sqrt()) provided by Xilinx as a part of the ISE suite. Due to space limitations, we can not present all the details here. For the implementation, we have used our VisualHDL methodology [18]. A THDL++ to VHDL

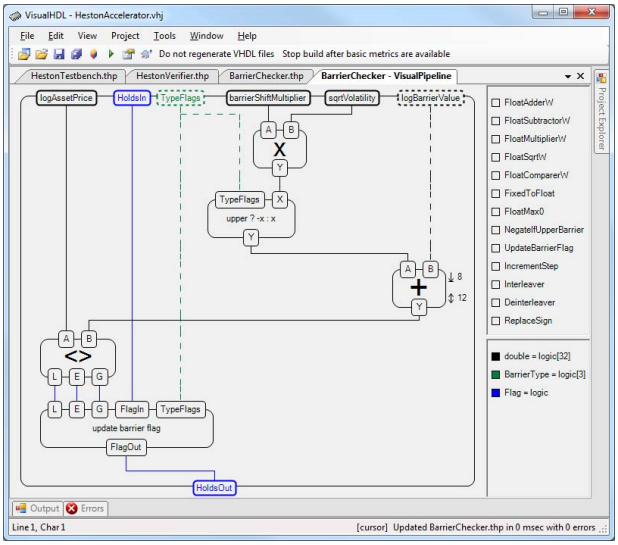


Fig. 3. VisualPipeline Plugin Editing the Heston Barrier Checker

compiler and a powerful IDE supporting code completion and design visualization (VisualHDL) are available online[2]. In this work, we have developed a special plugin that allows to visually create and modify the pipeline, using a drag-and-drop approach.

Figure 3 shows the Heston barrier checker in the visual pipeline editor plugin. The inputs are visible at the very upper part of the screenshot, the single output flag is located at the bottom. The visual representation shows the structure of the data flow at one glance and makes reviewing and modifying the pipeline intuitively easy.

VI. Results

We have synthesized our design for a Xilinx Virtex-5 XC5VFX70T device (as on the ML-507 evaluation board) with the Xilinx ISE Design Suite 13.1. The results have been optimized for speed. Table III shows the number and percentage of resources used for two different corner scenarios: Using no DSP slices in the dataflow at all (the one remaining is occupied by the random number generator), and using the maximum amount of DSP slices. These parameters can be set for the floating point cores when being generated with the Xilinx CoreGen tool. All given numbers are post place & route and include the interface logic needed to communicate with the host PC. Both configurations can run with clock frequencies up to 100 MHz.

In total, three instances of our accelerator can be mapped into a single XC5VFX70T device. This mapping is the reference for the following speed and energy results. The Virtex-5 is no longer state-of-the-art, and Xilinx is currently releasing the Virtex-7 generation. However, for the Virtex-7 series no evaluation kits are available at the moment. For this reason we use the ML-507 kit in order to provide system level results for speed and energy. On a Virtex-7 device that provides up to

[2]http://visualhdl.sysprogs.org

	Minimum DSP Usage		Maximum DSP Usage	
	Number	Percentage	Number	Percentage
Slices	4,862	43%	2,497	22%
LUTs	11,382	25%	5,481	12%
Flip-Flops	13,530	30%	6,950	15%
LUT-FF pairs	15,041	33%	8,176	18%
DSP48E slices	1	1%	43	33%
BRAM36 slices	5	3%	5	3%
Max. frequency	102 MHz		100 MHz	

TABLE III
Synthesis Results for one Instance on a Virtex-5

two millions of logic slices and over 5,000 DSP slices, several hundred accelerators could be mapped. We therefore expect a tremendous increase in speed and energy efficiency for the Virtex-7 series.

The FPGA accelerated setup does not require high computation capabilities of the host CPU, because only the final pricing is calculated there. Therefore we have decided to use a low-power laptop as host: a Fujitsu Siemens Lifebook E8410 with an Intel Core 2 Duo T7250@2.0 GHz and 2 GB RAM, running Windows 7 Professional SP1 64 Bit. In the idle state, the laptop itself then consumes around 20 W.

Table IV shows detailed runtimes and energy consumptions for a simulation of 10 million paths, both without and with FPGA acceleration. For the software only solution on the laptop, we have observed that our test system constantly consumes 44 W with the CPU fully loaded. Real time and energy consumption for each run are therefore linearly related to the number of simulated steps.

For the FPGA accelerated solution, we have added the FPGA board with an idle power consumption of 9 W to the laptop and included it into our energy gauging. Here we have observed that for 32 to 128 steps, the consumed power was around 40 W during the simulations. For 256 and more steps, it drastically fell to constant 35 W. The reason for this is that for up to 128 steps, the interface bandwidth is the limiting factor. Since the host CPU has to handle the transmission tasks for the USB interface that is used to connect the FPGA board, it is therefore producing the maximum possible load for this scenario. For more steps, the average CPU load over one simulation run decreases.

From Table IV we see that the FPGA accelerators speeds up the computation 21 times in average, compared to the software only simulation on the host laptop. At the same time, the FPGA accelerated scenario only consumes 4% of the energy per simulation.

Today, state-of-the-art financial product pricing is performed at high-end CPU and GPU clusters. For a fair comparison to real-world competitive architectures, we have therefore implemented our model on a recent Nvidia Tesla C2050 graphics card.

The Tesla GPU is hosted by a FluiDyna TWS 1xC2050-1xIQ-8 server workstation with an Intel Xeon CPU W3550@3.07 GHz and 8 GB RAM running OpenSuSE Linux 11.4 64 bit with Kernel 2.6.37.6-0.5-default. The CPU provides four physical cores with hyperthreading. We refer to this system as *server*. The idle power consumption for the server

Number of	Laptop: Software Only on 2 Cores			Laptop + FPGA with Three Instances			Factor (Laptop / FPGA)	
Time Steps	Real Time	Energy	Energy/Step	Real Time	Energy	Energy/Step	Real Time	Energy
32	56 s	2,442 J	76.31 J	4 s	172 J	5.38 J	13.88	14.20
64	116 s	5,104 J	79.75 J	8 s	344 J	5.38 J	14.50	14.84
128	230 s	10,120 J	79.06 J	9 s	401 J	3.14 J	24.64	25.22
256	465 s	20,438 J	79.84 J	18 s	630 J	2.46 J	25.81	32.44
1,024	1,852 s	81,466 J	79.56 J	72 s	2,532 J	2.47 J	25.60	32.18
4,096	7,344 s	323,114 J	78.89 J	287 s	10,057 J	2.46 J	25.56	32.13
average			78,90 J			3.55 J	21.66	25.17

TABLE IV
SPEED AND ENERGY RESULTS FOR LAPTOP-FPGA SETUP

Number of	Server: Software Only on 8 Cores			GPU Accelerated			Factor (Server / GPU)	
Time Steps	Real Time	Energy	Energy/Step	Real Time	Energy	Energy/Step	Real Time	Energy
32	5 s	930 J	29.06 J	0.95 s	295 J	9.22 J	5.25	3.15
64	10 s	1,860 J	29.06 J	1.88 s	582 J	9.09 J	5.33	3.20
128	21 s	3,953 J	30.88 J	3.74 s	1,158 J	9.05 J	5.69	3.41
256	41 s	7,673 J	29.97 J	7.43 s	2,305 J	9.00 J	5.55	3.33
1,024	166 s	30,923 J	30.20 J	29.68 s	9,201 J	8.99 J	5.60	3.36
4,096	660 s	122,760 J	29.97 J	118.46 s	36,722 J	8.97 J	5.57	3.34
average			29.86 J			9.05 J	5.50	3.30

TABLE V
SPEED AND ENERGY RESULTS FOR SERVER-GPU SETUP

has been measured to 87 W without the GPU, and to 148 W on average with the Tesla card plugged in. Again, the GPU has been removed for the software only measurements, where the fully loaded system consumes 186 W in average. With the fully loaded GPU and nearly no CPU load in our simulations, the power consumption was around 310 W.

Table V shows the measurement results for two scenarios: a software only run on the virtual eight cores of the server, and the fully loaded GPU. We can see that the GPU achieves a speedup of 5.5 in average, compared to the software only simulation on the server. At the same time, the energy consumption is reduced to around one third.

Furthermore, Table V reveals that, in contrast to the laptop-FPGA setup, no interface limitations are observable. The speedup and energy factors stay nearly constant over all step sizes.

In order to compare our FPGA accelerator with the GPU and server system, we have normalized the speedup and energy factors to the fully loaded 8-core server. The ML-507 FPGA board with its power supply already consumes 9 W in contrast to the 10 W with the FPGA loaded. The FPGA itself needs significantly less power than the 10 W, so the overall accelerator energy consumption can be significantly reduced by using optimized boards and power supplies.

We therefore have estimated a *FPGA chip only* scenario, that only considers the FPGA itself. To obtain a power estimation, we used the Xilinx XPower Estimator [19] that gave an upper bound of less than 3 W for our design. With the pricing that is currently performed on the host CPU implemented on the Virtex-5's PowerPC core, the FPGA chip only scenario highlights the enormous potential of energy saving.

The speedup factors are shown in Figure 4(a). We clearly see that the employed Virtex-5 device can not outperform the state-of-the-art Tesla C2050 with respect to speed. On the other hand, even a Virtex-5 FPGA with three accelerator instances achieves around 35% of the simulation speed of the Tesla C2050.

However, with respect to energy, the FPGA clearly outperforms all other architectures. Although Figure 4(b) shows that the laptop used in our FPGA accelerated setup consumes more than 2.5 times of energy than the fully loaded 8-core server, the combination of laptop and FPGA only consumes 12% of the energy normalized to the server. Compared to the GPU, the laptop-FPGA setup consumes only around 40% of the energy.

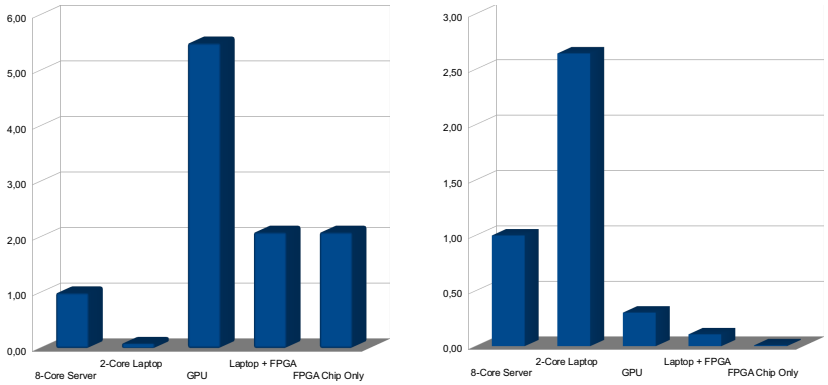
For both, speed and energy comparison, we have taken the average factors from Table IV and Table V. With numbers of steps higher than 128, the FPGA accelerated setup even profits more.

The FPGA chip only estimation forecasts nearly incredible 0.8% of energy consumed per simulation, compared to the server. At the same time, the speed is doubled. Extrapolated to three FPGAs with three accelerator instances on each, we achieve the same throughput as the Tesla 2050 GPU, but only consume less than 3% of the energy. This clearly highlights the enormous potential of FPGAs for energy efficient option pricing.

VII. CONCLUSION AND FUTURE WORK

In this paper, we present the first FPGA based implementation of an accelerator for option pricing based on the Heston model. It is based on Monte Carlo simulations and single precision floating point computations. We provide the mathematical background of our work, give detailed insight into our architecture and show our exhausting validation strategy used to ensure the high quality of our design.

Based on detailed speed and energy measurements, we clearly show that FPGAs can outperform state-of-the-art CPUs and GPUs for this task with respect to energy. For this purpose, we compare a combined setup of a dual-core laptop together with a Xilinx ML-507 Virtex-5 board, an 8-core Intel Xeon



(a) Average Speedup Factors

(b) Average Energy Factors

Fig. 4. Speedup and Energy Factors Compared to Fully Loaded 8-Core Server

server at full load and a Nvidia Tesla C2050 graphics card. We show that the laptop-FPGA setup only consumes 12% of the energy of the server, and about 40% of the energy of the GPU. At the same time, the FPGA accelerated system provides twice the simulation speed of the server, and around one third of the simulation speed of the GPU.

Future work will include the enhancement to multi level Monte Carlo methods that provide better asymptotical behavior. This is especially useful when fine precisions are needed. In this context, we will move to double precision computations. Besides improving the algorithm, we will go for a multi asset option accelerator. In this case, the payoff of the option – and therefore also the price – does not only depend on the path of one asset but on two or more.

ACKNOWLEDGMENT

We gratefully acknowledge the partial financial support from the Center of Mathematical and Computational Modeling $(CM)^2$ of the University of Kaiserslautern.

REFERENCES

[1] P. Warren, "City business races the Games for power," *The Guardian*, May 2008. [Online]. Available: http://www.guardian.co.uk/technology/2008/may/29/energy.olympics2012

[2] I. Schmerken. (2011, Mar.) Deutsche Bank Shaves Trade Latency Down to 1.25 Microseconds. www.advancedtrading.com. [Online]. Available: http://www.advancedtrading.com/infrastructure/229300997

[3] S. L. Heston, "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options," *Review of Financial Studies*, vol. 6, no. 2, p. 327, 1993.

[4] R. Lord, R. Koekkoek, and D. van Dijk, "A comparison of biased simulation schemes for stochastic volatility models," *Quantitative Finance*, vol. 10, no. 2, pp. 177–194, 2010.

[5] Q. Jin, W. Luk, and D. B. Thomas, "On Comparing Financial Option Price Solvers on FPGA," in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, May 2011, pp. 89 –92.

[6] C. de Schryver, M. Jung, N. Wehn, H. Marxen, A. Kostiuk, and R. Korn, "Energy Efficient Acceleration and Evaluation of Financial Computations Towards Real-Time Pricing," in *Knowledge-Based and Intelligent Information and Engineering Systems*, ser. Lecture Notes in Computer Science, A. König, A. Dengel, K. Hinkelmann, K. Kise, R. J. Howlett, and L. C. Jain, Eds., vol. 6884. Springer, Sep. 2011, pp. 177–186, proceedings of 15th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES). [Online]. Available: http://www.uni-kl.de/benchmarking

[7] 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.

[8] D. B. Thomas, J. A. Bower, and W. Luk, "Automatic Generation and Optimisation of Reconfigurable Financial Monte-Carlo Simulations," in *Application -specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on*, Jul. 2007, pp. 168 –173.

[9] X. Tian, K. Benkrid, and X. Gu, "High Performance Monte-Carlo Based Option Pricing on FPGAs," *Engineering Letters*, vol. 16, no. 3, pp. 434–442, 2008.

[10] A. Bernemann, R. Schreyer, and K. Spanderen, "Pricing Structured Equity Products on GPUs," in *High Performance Computational Finance (WHPCF), 2010 IEEE Workshop on*, Nov. 2010, pp. 1 –7.

[11] B. Zhang and C. W. Oosterlee, "Acceleration of Option Pricing Technique on Graphics Processing Units," Delft University of Technology, Tech. Rep. 10-03, Feb. 2010.

[12] A. Bernemann, R. Schreyer, and K. Spanderen. (2011, Feb.) Accelerating Exotic Option Pricing and Model Calibration Using GPUs. WestLB et al. Herzogstrasse 17 Düsseldorf 40217 Germany. [Online]. Available: http://ssrn.com/abstract=1753596

[13] R. Korn, E. Korn, and G. Kroisandt, *Monte Carlo Methods and Models in Finance and Insurance*. Boca Raton, FL: CRC Press., 2010.

[14] M. B. Giles, "Multilevel Monte Carlo path simulation," *Operations Research-Baltimore*, vol. 56, no. 3, pp. 607–617, 2008.

[15] M. Broadie, P. Glasserman, and S. Kou, "A continuity correction for discrete barrier options." *Math. Finance*, vol. 7, no. 4, pp. 325–349, 1997.

[16] C. de Schryver, D. Schmidt, N. Wehn, E. Korn, H. Marxen, and R. Korn, "A New Hardware Efficient Inversion Based Random Number Generator for Non-Uniform Distributions," in *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, Dec. 2010, pp. 190–195.

[17] N. Weaver, Y. Markovskiy, Y. Patel, and J. Wawrzynek, "Post-Placement C-slow Retiming for the Xilinx Virtex FPGAs," in *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, ser. FPGA '03. New York, NY, USA: ACM, 2003, pp. 185–194. [Online]. Available: http://doi.acm.org/10.1145/611817.611845

[18] I. Shcherbakov, C. Weis, and N. Wehn, "Bringing C++ Productivity to VHDL World: from Language Definition to a Case Study," in *Specification Design Languages, 2011. IC 2011. Forum on*, Sep. 2011, pp. 76 –82.

[19] Xilinx, "XPower Estimator (XPE)," Jul. 2011. [Online]. Available: http://www.xilinx.com/products/technology/power/index.htm