Wim Vanderbauwhede · Khaled Benkrid
Editors

High-Performance Computing Using FPGAs

Springer

High-Performance Computing Using FPGAs

Wim Vanderbauwhede • Khaled Benkrid
Editors

High-Performance Computing Using FPGAs

Springer

Editors
Wim Vanderbauwhede
School of Computing Science
University of Glasgow
Glasgow, United Kingdom

Khaled Benkrid
School of Engineering and Electronics
The University of Edinburgh
Edinburgh, United Kingdom

ISBN 978-1-4614-1790-3 ISBN 978-1-4614-1791-0 (eBook)
DOI 10.1007/978-1-4614-1791-0
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013932707

Foreword

The field programmable gate array (FPGA) was developed in the middle of 1980s with the original intent to be a prototyping medium. The array of programmable logic blocks enabled it to be reconfigured to any of a variety of compute functions. As such it was an attractive vehicle for "in circuit hardware emulation" where designs could be prototyped and debugged before being committed to silicon. It was also an attractive teaching vehicle for students learning computer design. It was with this in mind that I was first introduced to the FPGA by one of the pioneers in the field, Ross Freeman, the founder of Xilinx (who tragically died just a few years after Xilinx's founding).

As the underlying silicon technology improved and the functional potential was better understood, the FPGA slowly permeated many aspects of computing. By the 1990s it was an accepted component in most communications technology, then consumer-electronics and automotive applications became apparent, and by the early 2000s the FPGA was well established in almost all areas of computing except high performance computing (HPC). It would seem that HPC is an unlikely target for FPGAs, as the FPGA with all of its flexibility in both routing and configuration has a clear disadvantage when compared to custom arithmetic design when measured in terms of an area time power product. Of course, even then it was understood that there were some small specialized compute applications for which the FPGA could offer some significant performance advantages mostly in areas such as cryptography and specialized arithmetic.

Around 2003 there was a seismic shift in the underlying silicon technology, and Moore's law of frequency scaling (processors double performance every 18–24 months) became inoperative because the power densities required to support higher frequencies could not be economically sustained. The future was parallel in one way or another. ln HPC the obvious approach was to use multicore implementations, but there is a problem with attempting to scale performance by simply increasing the number of cores or processors to execute an application. The programming models that we have developed over the past decade have been oriented toward sequential processing and not parallel processing. The introduction of paradigms such as layers of abstractions that hide the underlying hardware thus makes it difficult to find the right form of parallelism to best express the execution of an application.

Still the notion of using FPGAs as a fabric to realize HPC for large classes of applications is surprising to many. It surely was unforeseen a decade ago. So what enables the FPGA to make its mark in HPC? There are at least three reasons:

1. The aforementioned difficulty in achieving scalable speed up with multicore implementations.
2. While Moore's law for frequency scaling ceased in 2003, Moore's law on transistor density scaling is still very much active so that over the intervening decade transistor densities have scaled up on more than an order of magnitude. These densities enable very large FPGA configurations. An enormous number of cells are available to realize complex compute engines. And because FPGAs necessarily operate at lower frequencies, they have not hit the power density limits of the CPU.
3. The flexibility of the FPGA enables the designer to realize almost any computer configuration that can be imagined and use any form of parallelism to suit the application. This flexibility provides the opportunity to create ideal machines for specific applications and unlike a decade ago where these applications would necessarily small they now can be of significant scope—really large, important, and interesting applications.

In a sense we have come full circle: the designer is again using the FPGA to do emulation but now that emulation is not of some established CPU but an emulation of an ideal machine for a particular application using techniques, representations, and processor forms unavailable to conventional processor designs. In effect the designer is emulating the future of computing high-speed computing.

This extraordinary book brings together the work of the leading technologists in this important field and points to the direction not only for high speed computing but also for the very future of computing itself.

Stanford, CA, USA Michael J. Flynn
Palo Alto, CA, USA

Preface

The seamless exponential increase in computing power that scientists, engineers and computer users at large have enjoyed for decades has come to an end by the mid-2000s. Indeed, while until then, computer users could rely on computing power doubling every 18 months or so simply by means of increases in transistor integration levels and clock frequencies, with no major changes to software, physical limitations including voltage scaling and heat dissipation meant that this is no longer possible. Instead, the chip fabrication industry has turned to multicore chip technology to keep the "possibility" of doubling computer performance every 18 months alive. However, this is just a "potential" performance increase and not a seamless one as application software needs to be recoded to take full advantage of the performance potential of multicore technologies. Failing this, the computer industry would cease to become a growth industry as there would be no need for computer upgrades for performance sake. Instead, the industry would become a replacement industry where computers are only bought to replace faulty ones. This could have serious economic repercussions; hence the explosion of research activity in industry and academia in recent years aimed at bridging the semantic gap between applications, traditionally written in sequential code, and hardware, increasingly parallel in architecture.

The aforementioned semantic gap, however, is also opening a window of opportunity for niche parallel computer technologies such as field programmable gate array (FPGAs) and graphics processor units (GPUs) which have become more mainstream because the problem of parallel programming has to be tackled for general-purpose processors anyway. FPGAs in particular have the promise of custom-hardware performance and low power, with the software reprogrammability advantage of general purpose processors. This is precisely why this technology has attracted a great deal of attention within the high performance computing (HPC) community, giving rise to the new discipline of high performance reconfigurable computing (HPRC).

The aim of this book is to present a comprehensive view of the state of the art of HPRC to existing and aspiring researchers in the field. This book is split into three main parts: the first part deals with HPRC applications, the second with HPRC architectures, and the third with HPRC tools. Each part consists of a number of contributions from eminent researchers in the field. Throughout the book, emphasis is made on opportunities, challenges, and possible future developments, especially in relation to other technologies such as general-purpose multicore processors and GPUs. Overall, we hope that this book will serve as both a reference and a starting point for existing and future researchers in the field of HPRC.

Finally, we thank all contributors, reviewers, and Springer's staff for their efforts and perseverance in making this book project a reality.

Glasgow, UK Wim Vanderbauwhede
Edinburgh, UK Khaled Benkrid

Contents

Part I
Applications

The first part of the book covers research on applications in the emerging field of High-Performance Reconfigurable Computing. The first two chapters present work on FPGA-based financial computing, an application field which has grown considerably in the last decade in both research and industry. These are from de Schryver et al. of the University of Kaiserslautern, Germany, and Tian et al. from the University of Edinburgh, UK, respectively. These are followed by four chapter contributions on FPGA-based bioinformatics and computational biology (BCB), another application area which has attracted considerable attention in the last decade, mostly in academia but also industry. These are from Lars Wienbrandt of the Christian-Albrechts-University of Kiel, Germany, Herbordt et al. from Boston University, USA, Yamaguchi et al. from the Universities of Tsukuba, Ryukyus, Doshisha and Keio, in Japan, and Will Li et al. from the City University of Hong Kong, China. The following two contributions are on FPGA-based data search and processing, another interesting application in our information age characterised by an explosion of data. The two contributions are from Vanderbauwhede et al. of Glasgow University, UK, and the University of Massachussets, USA, and Sklyarov and Skliarova from the University of Aveiro, Portugal. The next two contributions are on FPGA-based stencil computations, a very important area with various applications in computational fluid dynamics, electromagnetic simulation based on the finite-difference time domain method, and iterative solvers e.g. for seismic modelling. The two contributions are from Kentaro Sano from Tohoku University, Japan, and Medeiros et al. from Universidad Federal de Pernambuco, Brazil. The following chapter from Gneysu et al. of Ruhr-University Bochum, Germany, Czech Technical University in Prague, Czech Republic, and the Christian-Albrechts-University of Kiel, Germany, presents dedicated FPGA-based cluster solutions for high performance efficient cryptanalysis. After this, Hamada and Shibata from Nagasaki University, Japan, present a contribution which deals with two floating point scientific applications, namely ocean model simulation with a particular emphasis on fast inter-task communications, and astronomical N-body simulations with a particular emphasis on performance per $ and performance per Watt measures of FPGAs compared to ASICs, GPUs and general purpose processors. Finally, Kapre and DeHon from Imperial College London, UK, and the University of Pennsylvania, USA, present an FPGA-accelerated solution for the SPICE simulator, a widely used open-source tool for the simulation and verification of analog circuits.

High-Performance Hardware Acceleration of Asset Simulations

Christian de Schryver, Henning Marxen, Stefan Weithoffer, and Norbert Wehn

Abstract State-of-the-art financial computations based on realistic market models like the Heston model require a high computational effort, since no closed-form solutions are available in general. Due to the fact that the underlying asset behavior predictions are mainly based on number crunching operations, FPGAs are promising target devices for this task. In this chapter, we give an overview about current problems and solutions in the finance and insurance domain and show how state-of-the-art market models and solution methods have increased the necessary computational power over time. For the reason of universality and robustness, we focus on Monte Carlo methods that require a huge amount of normally distributed random numbers. We summarize the state-of-the-art and present efficient hardware architectures to obtain these numbers, together with comprehensive quality investigations. Build on these high-quality random number generators, we present an efficient FPGA architecture for option pricing in the Heston model, tailored to FPGAs. For the problem *pricing European barrier options in the Heston model* we show that a Xilinx Virtex-5 device can save up to 97% of energy, providing the same simulation throughput as a Nvidia Tesla 2050 GPU.

1 The Need for High Performance Computing in Secure Economies

The happenings on the financial markets all around the world in the last years have clearly demonstrated the inherent risks prevailing in our current economic system.

C. de Schryver (✉) • S. Weithoffer • N. Wehn
Microelectronic Systems Design Research Group, University of Kaiserslautern, Germany
e-mail: schryver@eit.uni-kl.de; weithoffer@eit.uni-kl.de; wehn@eit.uni-kl.de

H. Marxen
Stochastic Control and Financial Mathematics Group, University of Kaiserslautern, Germany
e-mail: marxen@mathematik.uni-kl.de

Due to the permanent news, nowadays every citizen is sensitized to these problems, even if not everybody (not to say nearly nobody) understands what is really going on in the financial system right now.

One main reason for the financial crisis was the wrong assessment of financial products with respect to their values and risks. For example, *collateralized debt obligations* (CDOs) considered to be one of the major causes for the crisis [8] are challenging to evaluate. CDOs bundle other products together and split the resulting pool again into new tranches with different ratings. Determining realistic risks and values for these tranches is a highly compute-intensive task.

However, CDOs are just one example of demanding products. Financial institutes have to price complex product portfolios containing many different ingredients regularly. In addition to that, countermeasures taken by the governments after the crisis in 2007 have further increased the demand for a fast simulation environment. In the European Union, for example the *Basel III* and *Solvency II* regulations for the financial and insurance sector require frequent monitoring and analysis of the institutions' financial situation, in particular of the equity risks.

Besides that, the increasing mathematical complexity of the underlying stock market models and their calibration has already led to a tremendous increase of simulation effort in the past. For example, the Heston and jump-diffusion stochastic differential equations (SDEs) lacking closed-form solutions in general are currently state-of-the-art [11]. The construction of more and more complicated financial products has further contributed to this, and since those products are available right now, there is no perspective that the complexity will decrease again in the future.

The energy needed for portfolio pricing is immense and lies in the range of several megawatts for a single bigger institute nowadays. Already in 2008 the available power for the financial center of London had to be clipped to assure a reliable supply for the Olympic games in 2012 [35]. Therefore, there is an urgent need for bringing down the energy consumption on the one hand, and to allow even higher simulation speed in the future on the other hand. This gap can only be bridged by using optimized hardware accelerators for the simulations.

Most institutes are currently running their simulations on standard CPU clusters, exploiting the highest flexibility by using pure software models. We will see in Sect. 2 that a lot of simulation methods are based on basic number crunching operations. So, a standard CPU is certainly not the most efficient architecture for this task with respect to throughput and energy efficiency. GPUs are currently emerging in the financial business and are more and more used in productive environments, for example by JP Morgan Chase, Bloomberg, or BNP Paribas [21]. Optimized architectures based on FPGAs have a huge potential for saving energy and speed up the simulations at the same time. However, FPGAs have just been used for experimental studies in financial business [36, 37], and we are not aware of these devices being used in productive risk assessment environments today.

In this chapter we cover the following topics:

- We introduce the state-of-the-art Heston model and the Multi-Level Monte Carlo method to solve derivative pricing in this context in Sect. 2.

- For the application "pricing European double barrier options in the Heston model" we shortly present a comprehensive benchmark set that allows to compare implementations on different target architectures transparently.
- We present a hardware accelerator for European barrier option pricing with the Heston model in Sect. 3, together with throughput and energy measurement results. We show that hybrid FPGA-CPU systems can already today save far more than 60% of the energy consumed by a state-of-the-art Nvidia Tesla C2050 GPU.
- In Sect. 4 we show efficient hardware architectures to generate normally distributed high-quality random numbers. These random numbers are key for efficient Monte Carlo simulations.

2 Pricing Options: Model, Algorithm and Comparison

One *problem* in financial mathematics is the pricing of derivatives. In this chapter we focus on the valuation of barrier options in particular. In order to solve this real-world problem, we need a specific *model* to reflect the behavior of the underlying asset. In our case we employ the Heston model that is widely used nowadays and is a further development of the famous Black–Scholes model.

For the *solution* of the problem we need an *algorithm* and an *implementation* thereof. A detailed systematic methodology to clearly distinguish between these terms has been given by de Schryver et al. in 2011 [27].

In this section we give an overview about different Monte Carlo methods and why they fit well to the problem that we target. Section 3 shows the details of our hardware implementation.

Besides the implementation itself, evaluating and comparing it to different algorithms and architectures is a challenge. We suggest to rely on standardized application benchmarks for this task. In Sect. 2.3 we propose a meaningful benchmark set for European barrier option pricing in the Heston model.

2.1 The Heston Model

In 1973 Fisher Black and Myron Scholes have introduced the famous Black–Scholes model [4]. In the same year Robert C. Merton [19] expanded the mathematical understanding of the model. Therefore, the model is sometimes called Black–Scholes–Merton model.

Since prices for European vanilla options were easily calculated and for more complicated ones one could model the behavior of the asset prices, the Black–Scholes model has fundamentally changed the way how the financial industry works. In 1997, Merton and Scholes received the Nobel price for their work.

The Black–Scholes model consists of certain assumptions on the market behavior. The most important ones are the absence of arbitrage—which is needed to fairly evaluate prices—and the log normal characteristic of the asset price. The price of an asset under the risk-neutral measure follows the SDE

$$\mathrm{d}S(t) = S(t)r\mathrm{d}t + S(t)\sigma\mathrm{d}W(t). \tag{1}$$

S denotes the price process of the asset, r the risk-less interest rate, W a Brownian motion and σ the volatility. Furthermore, the process has some starting condition $S(0) = s_0$.

This SDE can be solved. Its solution is

$$S(t) = S(0)\exp\left(\left(r - \frac{\sigma^2}{2}\right)t + \sigma W(t)\right).$$

In order to price a derivative of an asset following the SDE above, the fundamental theorem of asset pricing states that the price is just the discounted expected payoff under the risk-neutral measure.

Even though the SDE of the Black–Scholes model can be solved, various derivatives can only be priced numerically in this setting.

Nevertheless, besides the huge impact on the financial world, the Black–Scholes model has some drawbacks. The main is that it assumes a constant volatility. From real market data of asset prices and options it is, however, known that the volatility is generally not constant.

The Heston model [9] tackles this problem by using a second SDE to describe the behavior of the volatility process. Under the risk-neutral measure the SDEs of the Heston model are as follows:

$$\mathrm{d}S(t) = S(t)r\mathrm{d}t + S(t)\sqrt{V(t)}\mathrm{d}W^S(t),$$
$$\mathrm{d}V(t) = \kappa(\theta - V(t))\,\mathrm{d}t + \sigma\sqrt{V(t)}\mathrm{d}W^V(t).$$

The asset price process is denoted by S, and V denotes the volatility process. The latter process has the important property that it is always non-negative. Under a certain condition, called the Feller condition, the origin cannot be obtained. This is important for several mathematical results. However, this condition is seldom satisfied in real-world applications. The Brownian motions W^S and W^V are correlated, typically in a negative way. This implies that if the stock price falls, the variance tends to increase and the market becomes more volatile.

The Heston model fits much better to the data observed in real markets and provides more realistic results compared to the Black–Scholes model. On the other hand analytic pricing formulas are known for simple European options. This is especially important to calibrate the model and one of the reasons why the model is so popular.

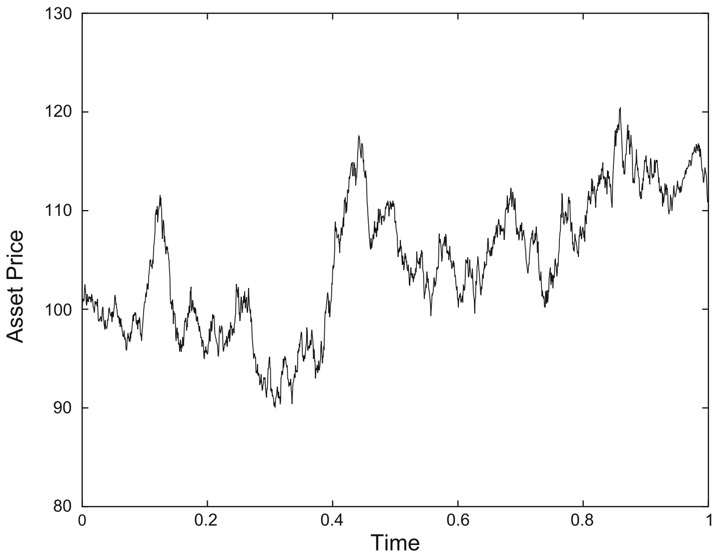


Fig. 1 A modeled asset price path in the Heston model

Figure 1 shows a realization of an asset price path following the Heston SDE. The erratic behavior is typical for most models and can be seen on the market.

2.2 *The Multi-Level Monte Carlo Method*

The price of an option is the discounted expected payoff of the option under the risk-neutral measure. One can analytically calculate the price of a plain European call or put option in the Heston model, i.e., $\mathbb{E}(e^{-rT} \cdot \max((S(T)-K),0))$, where $\mathbb{E}$ means the expectation value, T is the maturity time, and K the strike price. However, for other options such as barrier options this is not the case. In these situations numerical methods have to be used to estimate the expectation. There are several methods available that fit best to different situations. To name the most popular ones, these are finite difference method, the quadrature scheme, tree-based methods such as binomial or trinomial trees, and the Monte Carlo method.

We will concentrate on the Monte Carlo method in this chapter. It is not always the fastest method but very flexible and applicable to a wide range of applications. The basic idea of the Monte Carlo method comes from the Law of Large Numbers. To calculate $\mathbb{E}X$ for some random variable X, one has to simulate independent realizations X^i of random variables with the same distribution as X. The mean value of all results is an estimator for the expectation. The variance of the estimator is depending on the variance of X and the number of simulations. The error introduced by this is called the *statistical error*.

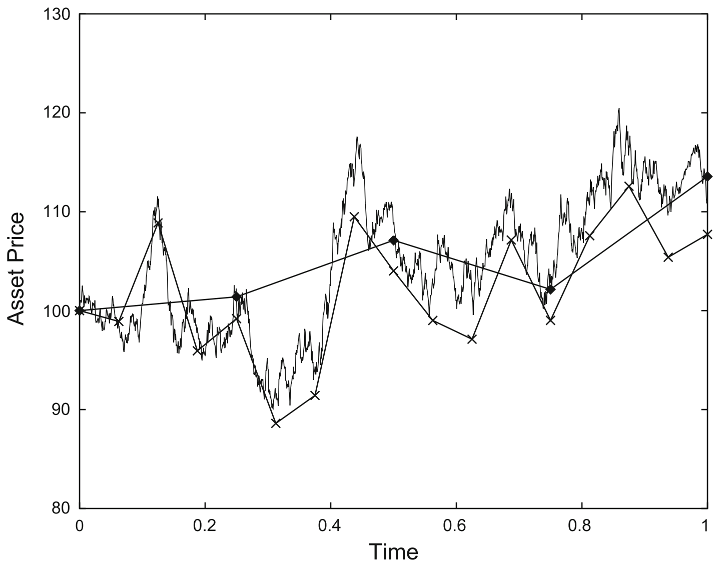


Fig. 2 A simulated Heston path and its discretizations on two different levels

Using Monte Carlo methods for asset simulations in the Heston model, however, leads to a problem: We cannot simulate $S(T)$ directly in the Heston setting. Therefore, the two SDEs are discretized and simulated. This introduces a second type of error called the *bias*. The bias is a systematic error and can be decreased by using more discretization steps. The plain Monte Carlo method now fixes the number of time steps and simulates many paths with these number of time steps. The chosen discretization has to be carefully selected, since it directly determines the bias.

It arises a second difficulty in the discretization of the volatility process in the Heston model. As we have seen, the variance process is always non-negative. The discretized version thereof, however, can become negative, if it is not adjusted. The obvious adjustment of setting a negative value to zero has turned out to be ineffective in general. More advanced schemes like the *full truncation* scheme that only set the volatility to zero when it is used as an argument of $sqrt()$ perform better [16].

Besides the discretization, the algorithm can be modified as well. The *Multi-Level Monte Carlo method*, for example, uses a slightly different approach than the plain Monte Carlo method. First, one simulates on a very coarse scale, that means with only a few time steps. These coarse scale simulations can be computed very fast. Then, iteratively, only the difference to the next finer *level* is simulated. Level in this context means a finer discretization (see Fig. 2). The variance of the difference is smaller and therefore less simulations on the finest level are needed compared to the plain Monte Carlo method. This gain can be a lot bigger than the cost of the additional simulations on the coarser levels. The benefits of the Multi-Level method increase with the required precision.

However, even though the Multi-Level Monte Carlo method is asymptotically better, the benefit is not always present in practical situations. Therefore, one has to be careful when to choose the method. In the Heston setting, a start level optimization that determines whether to use plain or Multi-Level Monte Carlo is mandatory. For more details about the Multi-Level Monte Carlo method and also the different discretization schemes in the Heston model, refer to Marxen et al. [16].

2.3 *The Need for Fair Metrics: A Benchmark Proposal for Option Pricing with the Heston Model*

Even though the Heston model is state-of-the-art and widely used in the financial industry, hardware accelerator publications are rare in that field (see Sect. 3.1). However, for the Black–Scholes model a lot of papers presenting sophisticated hardware architectures based on different methods exist.

The presented speedups look very impressive and the designs are likely well done. However, comparing the different implementations is a challenging task. A variety of attributes like speed, accuracy, and energy consumption can be considered. Furthermore, many different *solutions* are available in literature: not only the implementation and the architecture vary but also the algorithm. It is in many cases not clear by itself to which extent a speedup results from the employed algorithm and from the implementation. In addition to that, it is not possible to differentiate whether the presented algorithm or the implementation has the desired properties only for a special set of parameters, or if it performs well in a more general framework.

This challenge can only be bridged by using a unified benchmark set on application level, that means for a specific *problem* solved with a certain *model*. This application benchmark itself has to be independent of the algorithm and the implementation used. Morris and Aubury [20] already claimed the need for a benchmark for option pricing in 2007. By giving performance results for a benchmark set, authors allow their work to be compared fairly with respect to certain metrics without looking into details of the algorithm or the implementation.

In this section, we will describe our benchmark set for the application "pricing European double barrier options with the Heston model" presented in 2011 [26]. The benchmark was developed in a joint work with the financial mathematics group at the University of Kaiserslautern. It is freely available for download,[1] and we strongly encourage authors of future publications dealing with this problem to use it and provide application-specific metrics and therefore to make their work transparently comparable.

Twelve different settings for the Heston model, including parameter sets that have to be considered to be important in literature already, are used for the benchmark.

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

Table 1 One example of the benchmark parameters

Parameters for the Heston model	κ	θ	σ	r	S_0	V_0	ρ
	2.75	0.035	0.425	0	100	0.0384	-0.4644
Option specific parameters	Option type	Strike	Lower barrier	Upper barrier	Time to maturity (in years)		
	Double barrier call	90	80	120	1		
Price of the option	5.7538	Precision	0.0001				

They span a wide range of parameters observable on the markets. Our benchmark consists of three different components:

- The parameter sets defining the current *market situation*, such as the current volatility or the correlation between price and volatility
- The *option parameters* such as the type of option and the strike price
- The correct *reference price* or a good approximation thereof, together with a reference precision

To allow a comparison on application level, we recommend to provide the following metrics for all presented solutions:

- The consumed energy for pricing one option in *joule/option*
- The number of priced options per real time in *options/second*
- The numerical accuracy that is achieved by the proposed design, compared to the presented benchmark results
- The consumed area on chip for hardware architectures (slices, LUTs or mm^2 on silicon)

Table 1 exemplarily shows one of the twelve cases from the benchmark set [26]. The focus is not only on double barrier calls, but also on other types of options such as puts and digital calls are included.

In this section, we have briefly introduced our terminology, the Heston model, and the Multi-Level Monte Carlo method that we use in our hardware implementation described in Sect. 3, together with a benchmark set that allows to fairly compare different implementation on application level.

The key for Monte Carlo methods is a huge amount of high-quality random numbers. For hardware architectures, we therefore require efficient architectures for in our case non-uniform random numbers. We present suitable architectures for this task in Sect. 4. The next sections shows our proposed design for FPGA-based acceleration of option pricing in the Heston model.

3 Hardware Architectures for Asset Simulations

This section gives a short overview of the available FPGA implementations for option pricing. In the second part, we present an energy efficient FPGA architecture for this problem, together with detailed measured numbers for energy and throughput.

3.1 Related Work

Although the Heston model including its varieties (for example, the Heston–Hull–White model or the Heston model with additional jumps) is currently state-of-the-art in the financial industry [2, 11], the first GPU accelerators for solving this model have been presented just in 2010.

Zhang and Oosterlee have used the Fourier-Cosine Series Expansions (COS) method for multiple strike European and Bermudan option pricing in the Heston model on a NVIDIA GeForce 9800 GX2 GPU [40]. Compared to an Intel Core2Duo E6550@2.33 GHz CPU, they could achieve speedups between 10 and 100 for multiple strike European options, depending on the form of the characteristic function and on the number of strikes computed simultaneously.

Bernemann et al. have put the random number and path generation for Monte Carlo simulations on a Nvidia GPU, using a hybrid CPU-GPU option pricing system on top of the C++ *QuantLib* [23]. They could achieve up to 340 Gflops on a Nvidia Tesla C1060 GPU, compared to the maximum of about 11 Gflops given by a multi-threaded C++ implementation with SSE2 running on an Intel Xeon E5620@2.4 GHz [2]. Energy measurements are not provided in this work.

Based on this setup, investigations for exotic option pricing and Heston model calibration have been presented in 2011 [3]. Here Bernemann et al. have achieved a speedup between 10 and 50 for option pricing in the Heston model and 4–25 for simulations in the Heston–Hull–White model using a Hybrid Taus random number generator (RNG). The results are similar for a Mersenne Twister. For the Heston model calibration, they achieve a speedup between 15 and 50 with pseudo random numbers and 15–35 with quasi-random Sobol sequences, depending on the number of underlyings.

For option pricing in the Black–Scholes model, several FPGA architectures have been published in the last years [1,5,10,33,34,38]. These works show the wide range of potential speedups for FPGA-based accelerators, from 10 to more than 100.

In the last years, commercial FPGA systems have emerged for financial domain specific acceleration. Maxeler Technologies[2] offers hardware and software solution bundles for financial computing. They provide Xilinx Virtex-6 based platforms for professional server environments and desktop workstations. Their *MaxCompiler* for

[2]www.maxeler.com.

general purpose applications takes Java code and splits it into parts that remain on the host CPU and accelerated kernels executed on the FPGAs. The FPGA programming, including all the glue and interface logic, is done automatically.

Based on this system, the Maxeler CEO Oscar Mencer et al. presented speedup results for a single-asset Monte Carlo option pricer based on the Heston model with additional price jumps at the IEEE Workshop on High Performance Computational Finance (WHPCF) in November 2011. They have used a professional Maxeler MaxNode system with four MAX3 FPGA cards and could achieve a speedup of more than 100× over a 12 thread CPU version running on two Intel Xeon X5650@2.67 GHz CPUs [18]. Energy aspects have not been considered in this work.

Further available commercial systems are Wall Street FPGA,[3] Compaan Design,[4] and Impulse Accelerated Technologies.[5]

Wall Street FPGA uses National Instruments' LabView to bring a Monte Carlo-based European call option pricer on a Xilinx Virtex-5 FPGA [29, 30]. They state that their FPGA accelerated implementation is 131 times faster than the reference software running on an Alienware Area-51 7500 Dual Core CPU@3.0 GHz. Another application field for Maxeler is oil & gas exploration.

Impulse Accelerated Technologies and Compaan Design do not provide finance specific tools or benchmarks and cover a much wider application range.

3.2 A Multi-Level Monte Carlo Accelerator for Option Pricing with the Heston Model

In this section, we describe our dedicated FPGA accelerator architecture for pricing European double barrier options in the Heston model presented at ReConFig 2011 [25]. We give an overview about the architecture and provide detailed synthesis, performance, and energy results for a hybrid CPU-FPGA setup.

3.2.1 Architecture

By designing our FPGA-based accelerator, we wanted to achieve the maximum performance together with a minimal energy consumption. On the other hand, not all parts of the pricing process described in Sect. 2.1 are suitable for being implemented in hardware. For example, mathematical operations like $exp()$ or $/$ that are only needed for the final payoff computation would use up a lot of hardware resources, but could not contribute very much to increase the overall simulation speed.

[3]www.wallstreetfpga.com.

[4]www.compaandesign.com.

[5]www.impulseaccelerated.com.

Therefore we have decided to chose a hardware–software partitioning scheme that only brings those parts of the computation to hardware that are mainly data-flow oriented and use up most of the simulation time. We call these parts compute-intensive *kernels*. Complex mathematical operations or control driven parts remain on the host-CPU. This partitioning approach is also used by a number of authors proposing related accelerator designs [2, 3].

In particular, we have decided to chose the following partition for our implementation:

- The random number generation, the path simulation, and the barrier checking are ported to the FPGA. These kernels can be conveniently executed in parallel for different paths and return the final price for each path.
- The final path prices are transmitted to the host over USB. We have used the FTDI FT2232H interface module with a top average throughput of measured 6 MB/s.
- The reduction of all path results and the payoff computation remain on the host CPU.

For the random number generation, we have used a Tausworthe 88 uniform RNG together with our conversion unit described in Sect. 4.2.2. Since it provides a stream interface with handshaking, it can stall the rest of the design easily if no random number is present in the current clock cycle. However, any kind of uniform RNG may be used together with this converter. For example, interleaved parallel Mersenne Twisters as described in Sect. 4.1 that independent streams of random numbers from a single generator unit seem to be especially beneficial for high-quality multi-accelerator setups. Nevertheless, by simulating our benchmark introduced in Sect. 2.3 we have ensured that three Tausworthe 88 instances with independent seeds provide sufficient randomness for our application (see results in Sect. 3.2.2).

Our hardware has been implemented on a Xilinx ML-507 evaluation kit with a Virtex-5 XC5VFX70T FPGA. It uses single precision floating point units generated with the Xilinx CoreGen tool.

We have decided to use a similar approach to the automatically generated designs proposed by Thomas et al. [31]. However, we use our own host interface framework on top of the USB connection that allows to transparently read and write registers and data streams from a software application. Therefore we do not require a bus, but directly use a handshake-driven stream interface for the output prices and registers for the parametrization. Our protocol allows to dynamically reconfigure the accelerator parameters for the Monte Carlo simulation, the market and the option at runtime.

Our hardware design mainly consists of two parts: the control logic and the actual data path. In order to bring up the clock frequency to the maximum, our data path implementation is maximally pipelined. To get rid of additional control logic and to provide maximum scalability, we have decided to use a *packet-based* concept in our design:

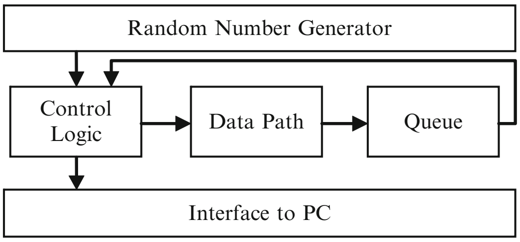


Fig. 3 High-level architecture of our hardware implementation

- Each packet describes the current state of a single path, including the price, volatility, step number, and a validity flag. Instead of having complex early termination strategies for paths that have hit a barrier, we change the status of those packets to *dummy packets* by clearing the validity flag. These packets remain in the processing pipeline, which decreases the throughput to some extent, but at the same time drastically 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 Sect. 2.1). It consumes one packet and produces another one in every clock cycle.
- 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 selected algorithm for solving the Heston model, that is full truncation with antithetic variance reduction in our case (see Sect. 2.1).

Figure 3 shows the structure of our design and the interaction between the data path, a queue and the control logic. The queue buffers all packets coming out of the data path for future processing or final transmission to the host. This decision is made by the control logic. The depth of the queue has to be greater than the pipeline length of the data path, which is 60 in our case. We therefore have exploited the maximum depth of a BRAM36 slice from the target Virtex-5 device for the queue. It is important to note that the data path block is only made up of simple pipelined floating point cores, uses handshake-driven stream interfaces, and does not require support for any stall signals.

The role of the control logic is to act as a broker between the RNG, the data path and the host system. It follows the following set of rules:

- 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 from the queue.
- If the queue contains a packet, its step number is checked. If the control logic sees that it was the last step, the final price is sent to the host, and a new packet is created. If not, the packet is resent to the pipeline along with a new pair of random numbers.

Table 2 Single precision floating point components in the data path

Component	Adders	Multipliers	Subtractors	sqrt()
Heston step generator	4	6	2	1
Barrier checker	1	1	1	0

The control logic has been implemented equivalently in a bit-true software model to allow easy testing of the design. Together with a bit-true model of the hardware RNG, each hardware component can be validated against the software reference independently. As the processing order of the packets does not depend on interface delays, this ensured bit-by-bit equivalence between software and hardware results.

The decomposition between the control logic and the data path further contributes to the reduction of the validation effort:

- The pipeline can be tested separately from the control logic, only considering the floating point operations.
- The control logic can be checked on its own by using a dummy pipeline that only counts the steps and has no floating point logic inside at all.

The internal structure of our pipeline is similar to the GARCH example presented by Thomas et al. [31], but includes the Heston specific modifications. Table 2 shows the number of floating point units in the Heston step generator part of the pipeline (that generates successive values for price and volatility) and the subsequent barrier checking.

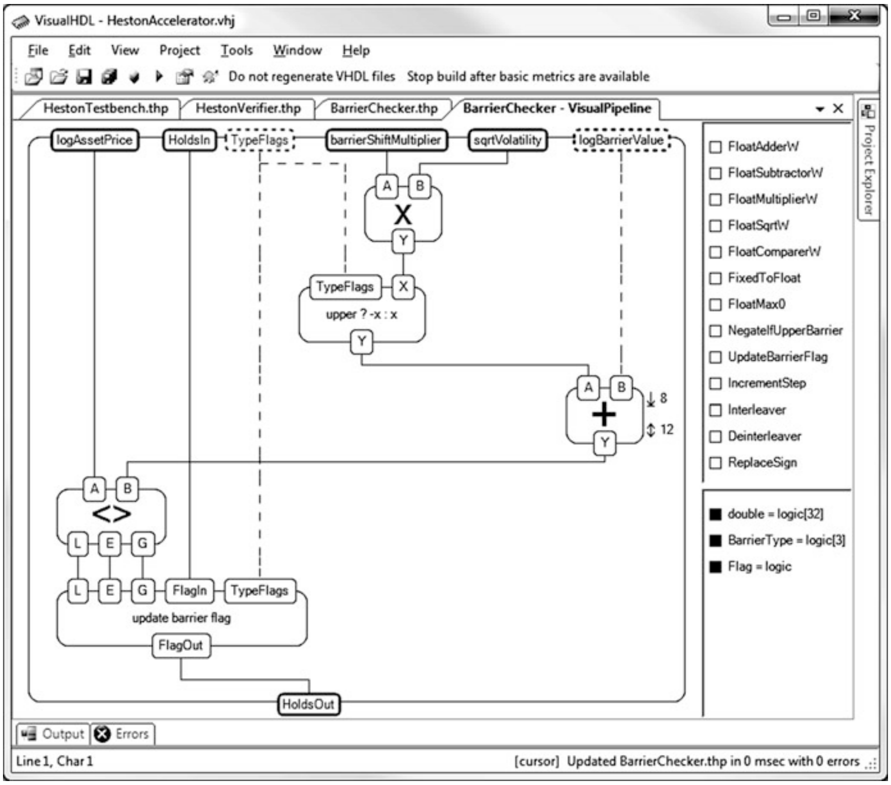
We have used THDL++, a high-level approach for HDL design together with the free VisualHDL tool for the development.[6] For this task, the VisualHDL tool has been enhanced by a data path pipeline designer plugin that is shown in Fig. 4. It allows creating a data path by just dragging-and-dropping operations and connecting them from the inputs in the upper part of the screenshot to the output at the bottom.

3.2.2 Results

All synthesis results have been generated 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, are post place & route, and include the host interface logic. Although Xilinx is currently releasing the Virtex-7 family, no evaluation kits are available at the moment. Therefore we use the ML-507 kit in order to provide system level results for speed and energy (for all details refer to de Schryver et al. [25]).

Table 3 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

[6] visualhdl.sysprogs.org.



[cursor] Updated BarrierChecker.thp in 0 msec with 0 errors

Fig. 4 VisualPipeline plugin editing the Heston barrier checker

Table 3 Synthesis results for one instance on a Virtex-5

	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	

occupied by the RNG from [24]) and using the maximum amount of DSP slices, depending on the Xilinx CoreGen settings.

From Table 3 we see that (without the triple interface logic) in total three instances can be put on a single XC5VFX70T device. We assume a three-instance FPGA accelerator for the following considerations. A Virtex-7 device would provide enough space for several hundreds of accelerator instances.

Table 4 Speed and energy results for the laptop-FPGA setup

	Laptop only		Laptop + FPGA		Factor (laptop/FPGA)	
Time steps	Real time (s)	Energy/step (J)	Real time (s)	Energy/step (J)	Real time	Energy
32	56	76.31	4	5.38	13.88	14.20
64	116	79.75	8	5.38	14.50	14.84
128	230	79.06	9	3.14	24.64	25.22
256	465	79.84	18	2.46	25.81	32.44
1,024	1,852	79.56	72	2.47	25.60	32.18
4,096	7,344	78.89	287	2.46	25.56	32.13
Average		78.90		3.55	21.66	25.17

Since the host CPU in the hybrid CPU-FPGA setup only computes the final payoff and performs the communication with the ML-507 board, we have chosen 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 consumes around 20 W.

Detailed measured numbers for runtimes and energy consumptions in this setting are given in Table 4, with and without FPGA acceleration. In each case, ten millions of paths have been computed.

For the software-only simulations, it can be seen that the measured real time and consumed energy are linearly related to the number of time steps in the simulation. This is not surprising, since the power consumption of the laptop with the CPU fully loaded remains constantly 44 W. In this case, the idle power consumption of the FPGA board has not been included in the measurements.

Measuring the hybrid setup, the FPGA board with an idle power consumption of 9 W has been added to the 20 W of the laptop, so that we are talking about a 29 W idle load in total. In this setting Table 4 shows that the energy per step is much higher for small numbers of time steps (32–128). For 32 and 64 time steps, we have measured a power consumption of 40 W during the simulations for the whole system. For 256 and more steps, it remained constant 35 W.

The explanation for this observation is that the host-to-board interface provides a limited bandwidth. The host CPU also runs the tasks for communicating with the FPGA board, so that the amount of energy used for communication is very high for small step sizes, compared to the total energy consumed for one simulation. For more than 256 step sizes, the computations on the FPGA take enough time, so that the interface is no longer the limiting factor.

Table 4 also clearly shows that the average speedup of the hybrid system compared the the CPU-only scenario is 21 times in average, by only consuming 4% of energy per simulation.

Nowadays, financial simulations are performed on high-end CPU and GPU clusters. To give a fair comparison of our design to the state-of-the-art, we have implemented our Monte Carlo algorithm on a Nvidia Tesla C2050 graphics card. Preliminary work in our group has shown that the performance loss of using OpenCL is insignificant compared to CUDA. Thus, for the reason of higher flexibility, we have coded our accelerator in OpenCL.

Table 5 Speed and energy results for the server-GPU setup

	Server only		GPU accelerated		Factor (server/GPU)	
Time steps	Real time (s)	Energy/step (J)	Real time (s)	Energy/step (J)	Real time	Energy
32	5	29.06	0.95	9.22	5.25	3.15
64	10	29.06	1.88	9.09	5.33	3.20
128	21	30.88	3.74	9.05	5.69	3.41
256	41	29.97	7.43	9.00	5.55	3.33
1,024	166	30.20	29.68	8.99	5.60	3.36
4,096	660	29.97	118.46	8.97	5.57	3.34
Average		29.86		9.05	5.50	3.30

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 (referred to as *server* in the following). The CPU provides four physical cores with hyperthreading, so that we can count them as eight cores. The idle power consumption for the server is 87 W without the GPU, and 148 W on average with the Tesla card plugged in. As in the laptop-FPGA setting, we have removed the GPU for all software-only measurements.

With the CPU fully loaded (but without the GPU), the server system consumes 186 W in average. If we run the simulations with full load on the GPU, the CPU still has to compute the payoff at the end of all Monte Carlo simulations. In this case, the overall power consumption of system is 310 W.

In Table 5 we see all measured runtime and energy results for the server-GPU setting. Again, we provide the numbers for a software only run on the virtual eight cores of the server and for the fully loaded GPU setup. We see that on average the simulations on the GPU run 5.5 times faster than the CPU-only simulations, by only requiring one third of the energy per simulation. Furthermore, Table 5 shows that the speedup and energy factors remain constant over different time steps. Therefore we conclude that in this setting with the fast PCIe connection of the GPU the interface is not a bottleneck, in contrast to the laptop-FPGA setup.

To provide a unified comparison of our four simulation setups including the GPU and FPGA accelerators, we have normalized the speedup and energy factors to the fully loaded 8-core server.

As mentioned above, the Virtex-5 device that we use is no longer state of the art, and the overhead of the ML-507 board for the idle energy consumption is immense. The complete board consumes 9 W in the idle mode, and not more than 10 W with the FPGA running. To obtain a power estimation for the FPGA itself, we used the Xilinx XPower Estimator [39] that gave an upper bound of less than 3 W for our design. The energy efficiency of the system could therefore be drastically increased by using optimized boards without peripherals, hosting several FPGAs with a tailored power supply.

To provide an estimation of potential energy savings, we have constructed the *FPGA chip only* scenario that assumes the 3 W from the XPower Estimator, with