Chapter 7
Pricing High-Dimensional American Options on Hybrid CPU/FPGA Systems

Javier Alejandro Varela, Christian Brugger, Songyin Tang, Norbert Wehn, and Ralf Korn

Abstract In today's markets, high-speed and energy-efficient computations are mandatory in the financial and insurance industry. As American options are amongst the most frequently traded products in the derivatives market, it becomes essential to place the focus on their pricing process. Calculating the price of an American option in particular is a challenging task due to the freedom the holder is given in terms of exercise date and the involved trading strategy. A well known algorithm that solves this task is the Longstaff-Schwartz (LS) algorithm, which applies least-squares linear regression on simulated Monte Carlo (MC) paths. This work presents a novel way to price high-dimensional American options, coined Reverse LS, using techniques of the embedded community. The proposed architecture targets hybrid Central Processing Unit (CPU)/Field Programmable Gate Array (FPGA) systems, and it exploits the FPGA reconfiguration to deliver high-throughput. With a bit-true algorithmic transformation based on recomputation, it is possible to eliminate the memory bottleneck and access costs present in a straightforward implementation. The result is a pricing system that is 16× faster and 268× more energy-efficient than an optimized Intel CPU implementation.

7.1 Introduction

In the financial world, Over-the-Counter (OTC) derivatives markets trade an average annual volume of approximately USD 700 trillion [12], which increases every year. Increasing competition and stringent regulations lead to a steady growth of computing requirements. Today, financial institutions operate huge clusters to

J.A. Varela (✉) • C. Brugger • N. Wehn
Microelectronic Systems Design Research Group, University of Kaiserslautern, Kaiserslautern, Germany
e-mail: varela@eit.uni-kl.de; brugger@eit.uni-kl.de; wehn@eit.uni-kl.de

S. Tang • R. Korn
Stochastic Control and Financial Mathematics Group, University of Kaiserslautern, Kaiserslautern, Germany
e-mail: tangs@mathematik.uni-kl.de; korn@mathematik.uni-kl.de

C. De Schryver (ed.), *FPGA Based Accelerators for Financial Applications*,
DOI 10.1007/978-3-319-15407-7_7

satisfy these computing needs. Due to their high costs, the financial industry has a high incentive to investigate efficient ways of performing the required computations, both in terms of speed and power consumption.

Not surprisingly, it has become a particular field of research among the engineering community in recent times, due to the challenges involved. In this regard, one approach is to build specialized computing architectures. While more effort is required to design them, they are able to perform computations much more efficiently compared to general-purpose architectures. In this regard, FPGAs have been demonstrated high performance and energy-efficiency when used to speed up financial simulations [5, 15].

While many numeric algorithms map nicely to FPGAs, there often remain parts that are best executed on CPUs. Hybrid devices combine CPUs and FPGA fabrics on a single device, delivering the best of both worlds. One recent example is the Xilinx Zynq All Programmable System on Chip (SoC) based on ARM cores. These devices are able to host fully featured operating systems like Linux and allow programs to reconfigure the FPGA fabric during runtime. A key challenge of such heterogeneous computing systems is to carefully balance all aspects of the hardware, including communication, reconfiguration times, memory bandwidth, FPGA area and CPU loads.

Among the products that are currently offered in the derivatives markets, options are particularly attractive to investors. In general terms, an option is a contract that gives the right, but not the obligation, to buy or sell the underlying asset at a fixed price and date. What makes it attractive is the potential gain associated with the contract, while presenting a limited risk to the buyer, which is equivalent to the premium paid at the moment of purchase. And it is precisely the computation of this premium (the option price) what concerns financial institutions.

American options present the additional challenge that the holder is allowed to exercise the option at any time from purchase until the expiry date, in contrast to the European option style, which can only be exercised at a fixed date. This freedom makes its pricing much more challenging, since now the estimation of an optimal exercise strategy comes into play.

The LS algorithm, which is implemented in this work, has been designed to address the problem of finding such a strategy and deriving from it the option price [9]. This is accomplished by working backwards, from maturity to the initial day, on simulated MC paths by means of the least-squares regression method. For multi-dimensional options, which derive their price from multiple underlying assets, MC is currently the only known method that can be efficiently used to price them.

The quality of the simulated paths also depends on the mathematical model used to describe the evolution of an underlying asset in the market. For high-dimensional options, the Black-Scholes (BS) model has been used extensively due to its relatively light-weight computation (only one parameter cannot be observed: volatility). Besides, its results are close enough to the observed market values, and it can be easily extended to make the model flexible enough for practical cases.

When it comes to implementation, several issues need to be addressed. At first sight, the LS algorithm does not present a clear way to perform hardware-software partitioning. It is also a computationally intensive algorithm. The choice of certain basis functions that work on the simulated paths influences the final price, and has to be matched to the option being priced. Besides, the method used to solve the least-squares process has an impact on the overall runtime. The chosen number of simulated MC paths, and the number of days in which the option can be exercised, define the amount of generated data. Storing this data temporarily in an external memory chip is a straightforward approach, but faces a certain bandwidth limitation and a considerable power consumption.

This work investigates custom computing solutions for the above mentioned LS method [9]. The proposed solution targets hybrid computing systems, like Xilinx Zynq, and is able to perform high-precision and energy-efficient computations. Besides the classical approach, a novel algorithmic improvement called Reverse LS is presented. This new approach does not require the storage of all intermediate steps for all paths, but recomputes them on the fly. Recomputation is a well known technique in embedded system to avoid energy-costly memory accesses [6, 7]. This allows us to reduce the energy consumption by trading-off memory bandwidth with FPGA resources, effectively moving less data across the board.

7.2 Background

This section covers the theoretical background and related work specifically relevant to the content of this chapter. For the general background of financial computations refer to Chap. 1 by Desmettre and Korn.

7.2.1 American Options

In simple terms, a financial derivative is a type of contract which derives its value from the performance of an underlying entity (e.g. an asset). There are many types of derivatives, being one of them the so called *options*. An option contract gives the buyer the right, but not the obligation, to buy or sell an underlying asset at a specified strike price and a specified date. In this regards, there are several exercising styles, being two of them:

- European options, which can only be exercised at the expiry date (also called maturity)
- American options, which could be exercised at any time before or at the expiry date

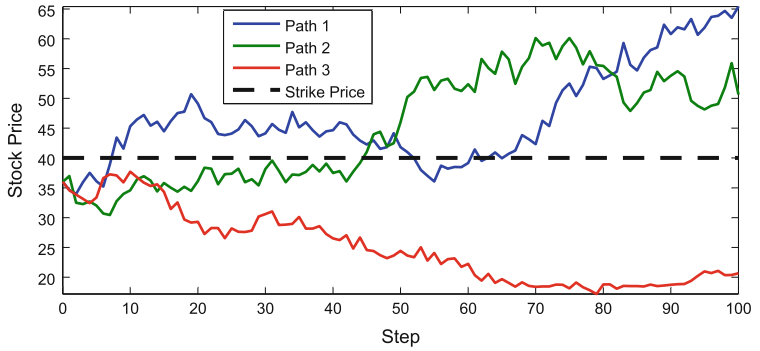

Fig. 7.1 Simulated paths using BS model, with initial price 36 [ad], strike price 40 [ad], American call option

The option gives the holder the right to either:

- Sell the underlying assets: *put option*
- Buy the underlying assets: *call option*

Consider the example presented in Fig. 7.1, where different simulated scenarios are presented for a given American call option, strike price and maturity. The holder of the option needs to decide at each time step on whether to exercise the option or hold it until a future date.

With $S(\tau)$ the value of the underlying asset at time τ, and K the strike price, the intrinsic value at the current time step τ is calculated as Eq. (7.1) for call options, and Eq. (7.2) for put options:

$$payoff(\tau) = max(S(\tau) - K, 0) \qquad \text{call} \tag{7.1}$$

$$payoff(\tau) = max(K - S(\tau), 0) \qquad \text{put} \tag{7.2}$$

The option is then said to be *In the Money (ITM)* if:

- $(S(\tau) > K)$ for a call option
- $(S(\tau) < K)$ for a put option

Following Fig. 7.1, whenever the option is ITM the holder has the choice of executing an early exercise of the option or holding it until further steps in an attempt to maximize its profit.

This right (to sell or buy) given by the option comes at a price, a premium that the buyer pays the seller at the moment of the purchase. The price of an American call/put option is given by Eqs. (7.3) and (7.4) respectively:

$$P = \sup_{\tau \in \mathcal{T}\{t_1,\ldots,t_m\}} \mathbb{E}(e^{-r\tau}(S(\tau) - K)^+) \qquad \text{call} \tag{7.3}$$

$$P = \sup_{\tau \in \mathcal{T}\{t_1,\ldots,t_m\}} \mathbb{E}(e^{-r\tau}(K - S(\tau))^+) \qquad \text{put} \tag{7.4}$$

where:

- $(x)^+$ means $\max(x, 0)$
- K is the strike price
- T is the maturity of the option
- $\{t_1, \ldots, t_m\} = \{\frac{T}{m} \times 1, \ldots, \frac{T}{m} \times m\}$ are potential exercise dates of the option
- $\mathcal{T}\{t_1, \ldots, t_m\}$ is the set of stopping times with values in $\{t_1, \ldots, t_m\}$
- r is the risk-free interest rate
- $S(\tau)$ can be simulated with an appropriate mathematical model, for example using BS, as it will be covered in later sections
- In the case of multi-dimensional options, their value is derived from several underlying assets (therefore dimensions)

A note is made on the fact that when the time interval is discretized as in Eqs. (7.3) and (7.4), the option is then called Bermudan options.

The main complexity associated to American options resides in their pricing. As mentioned before, this style of options can be executed not only at maturity (expire date), like in the case of the European style, but also at intermediate steps. This freedom that the option holder is given makes the estimation more complex. The seller of the option (normally a bank or financial institution) has to estimate its price expecting the worst case scenario where the holder would follow a sound strategy at each step that maximizes its return. And this is exactly where the LS algorithm comes into play [9].

7.2.2 Black-Scholes Model

The BS model assumes, among other considerations, that the stock price follows a random walk, which implies that the stock price at any future time has a log-normal distribution (meaning its logarithm has a normal distribution) [4]. It describes the stock price $S(t)$ by means of the Stochastic Differential Equation (SDE):

$$dS(t) = S(t)(r - q)dt + S(t)\sigma dW(t), \tag{7.5}$$

where: r = risk-free interest rate, q = dividend yield, σ = constant volatility of stock's returns, and $W(t)$ is the associated Brownian motion.

The BS model is based on certain assumptions [4]. In particular, it assumes constant volatility, which might not be the case in the real market. However, it is still used nowadays due to its simplicity, ease of extension, and because it is a good approximation of how much profit the holder could expect.

7.2.3 Monte Carlo (MC) Methods

Simulating the BS model in Eq. (7.5) requires the application of an appropriate discretization scheme. In this work we have applied the *Euler discretization*. Discretizing into m steps with equal step sizes $\Delta t = \frac{T}{m}$ leads to:

$$\hat{S}_{t_{i+1}} = \hat{S}_{t_i} \exp\left(\left((r-q) - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t}\,\Delta W_i\right), \tag{7.6}$$

with ΔW_i being independent standard normal random variables.

The classic MC algorithm estimates the price P as the sample mean of simulated instances of the discounted payoff values $g(\hat{S})$. The complexity of MC methods depends only linearly on the number of dimensions, which makes them an excellent candidate for high-dimensional problems or a method of last resort for options with no other numerical scheme.

MC results depend heavily on the number of simulated paths, due to its slow convergence. This is based on the fact that the standard deviation of the error only decreases as the square root of the number of simulations [8]. Therefore, the higher the number of paths, the more accurate the result it yields. As an example, a showcase is designed to price an American maximum call option on two correlated stocks (correlation parameter $\rho \neq 0$) under the BS model Eq. (7.6) by means of the LS algorithm. The optimal expected discounted payoff is given by:

$$P = \sup_{\tau \in \mathcal{T}} \mathbb{E}\left[e^{-r\tau}(\max\{S_1(\tau), S_2(\tau)\} - K)^+\right], \tag{7.7}$$

with input parameters: $S_1(0) = S_2(0) = 100$, $K = 100$, $r = 0.05$, $q_1 = q_2 = 0.10$, $\sigma_1 = \sigma_2 = 0.2$, $\rho = 0.1$, $T = 1$, $m = 365$, $\mathcal{T} = \{\frac{T}{m} \times 1, \frac{T}{m} \times 2, \ldots, \frac{T}{m} \times m\}$, $N =$ 10,000.

The influence of the number of simulated paths N on the accuracy of the option price for the given example is displayed in Fig. 7.2, where the benchmark option value is found at 10.12 (unspecified currency) using the binomial tree method [4]. The boxplots show the distribution of the option values obtained for 100 runs of the LS algorithm. A comparison to the benchmark value of 10.12 clearly suggests a minimum number of paths at around 10K.

7.2.4 Paths Generation

The BS model requires a sequence of normally distributed random numbers to generate the paths. Furthermore, because the underlying assets (dimensions of the option) coexist in the same market, these random numbers need to be correlated to each other. In this work the following processes are executed in the given order:

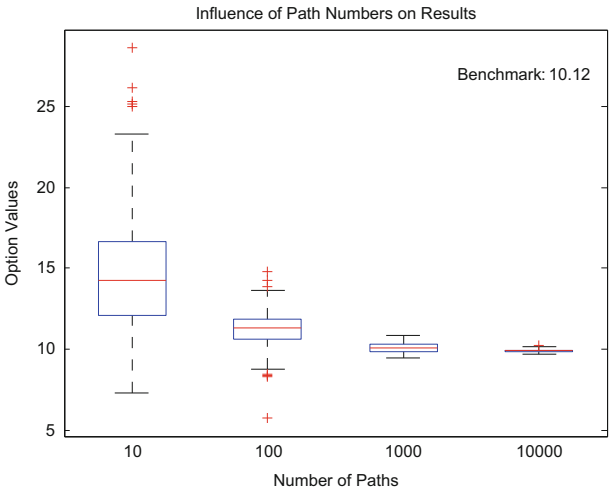

Fig. 7.2 Boxplots with the distribution of the results obtained by running the LS algorithm 100 times per number of paths: 10, 100, 1,000 and 10,000

1. *Mersenne Twister (MT)*: The MT is a widely-used pseudo-random number generator, whose MT19937 version is the one implemented in this work. It produces a sequence of 32-bit unsigned integer random numbers, and has a period of $2^{19937} - 1$. The algorithm code is explicitly shown in [10], and could be seen as split into two main parts [11]:

   - A set of 624 internal states used to generate the random numbers. This internal states are initialized through a seed that generates the initial values, and an actualization process modifies the states every 624 output numbers
   - A tempering function, a sequence of xor operations, that outputs the final number

   It is possible to pipeline this algorithm in order to achieve one output per clock cycle. In fact, the work is done on the actualization process itself, so that each state is actualized as soon as it has been used for the last time in the current cycle.
2. *Inverse Cumulative Distribution Function (ICDF)*: The MT module presented before generates uniformly distributed random numbers, whereas the BS model requires normally distributed ones. Previous work on this field has provided with an efficient implementation of the ICDF to obtain the desired standard normal distribution [13]. Furthermore, the mentioned implementation generates single-precision floating-point random numbers, which will match later with the setup for this work. A note is made, however, on the fact that the method does not precisely guarantee a valid output at every single clock cycle, but nevertheless presents a good tradeoff between hardware utilization and performance, as compared to more expensive approaches like the Box-Muller method [13].

3. *Antithetic Variates*: As mentioned before, the MC method suffers from slow convergence (high simulation runtime), which is overcome by attempting a faster reduction of its variance. In this regard, the easiest one is the method of antithetic variates, which works by introducing symmetry [8]. In this work, the antithetic method is implemented after the ICDF module, meaning that it works on normally-distributed random numbers. Under this condition, it can be proven that for a single random number z, then $-z$ is also a valid number, which reduces the overall number of generations by half. Furthermore, when using models based on Brownian motion to generate the paths, the payoffs of high-dimensional options can be typically written as:

$$P = h(Z_1, \ldots, Z_k). \tag{7.8}$$

Under the assumption that h is monotonic on each variable, then it is possible to prove that Eqs. (7.9) and (7.10) are negatively correlated, which means that it can be used as a variance reduction technique. A similar approach on uniform random numbers is presented in [8].

$$P_1 = h(Z_1, \ldots, Z_k) \tag{7.9}$$

$$P_2 = h(-Z_1, \ldots, -Z_k) \tag{7.10}$$

4. *Correlation*: In the case of a two-dimensional option, the correlation process mentioned before is obtained in practice through the correlation of two independent random numbers, $y \sim N(0,1)$ and $z \sim N(0,1)$, and coefficient ρ, as in Eq. (7.11), delivering two correlated random numbers z and w as outputs [8].

$$w = \rho z + \sqrt{(1-\rho^2)}\, y \qquad \text{(correlation)} \tag{7.11}$$

The generated random number following the previous sequence are then fed into the BS model Eq. (7.6) in order to obtain the required paths.

7.2.5 LS Algorithm to Price American Options

The LS algorithm approximates the value of an American option by means of simulation [9]. The simulated MC paths represent the behaviour over time of the underlying assets (e.g. stocks), which compose the option to be priced. These paths could be obtained by different mathematical models with different degrees of complexity, for example BS. Once the paths have been generated, the option price is estimated by assessing which would be the best strategy the holder would follow that maximizes its profit. This strategy becomes, in turn, the worst-case scenario for the seller.

At the expiry date (maturity) the holder has only one choice, and that is to exercise the option only if it is ITM. However, at any other time, the holder can decide between:

- Exercising the option immediately
- Holding the option (called continuation)

The option should be exercised if the payoff of immediate exercise is higher than the continuation value. However, this continuation value is defined as the conditional expected value of continuing the option, assuming that the option is not exercised at or before the current time step. In general terms, the LS algorithm estimates this conditional expectation based on all generated paths at the current step, in order to derive the optimal excercise strategy.

In more detail, the LS algorithm uses least-squares linear regression to find the optimal exercise boundary. The basic steps are:

1. Generate N independent paths per underlying (stock) at all possible exercise dates, using a chosen Random Number Generator (RNG) and a chosen mathematical model (in our case with Eq. (7.6)). For multi-dimensional options, the Random Numbers (RNs) need to be correlated.
2. Initialize the cash-flow with the discounted payoffs at maturity.
3. Moving backwards one step in time, proceed as follows:
   - Linear regression: the goal is to find out whether to exercise the option or to hold it. For this purpose, the current discounted payoff (when exercised) is compared to the future expected return (for holding the option), approximated by regression. As an example, Fig. 7.3 plots the future return (cash-flow) over the current stock price for each path. Least-squares linear regression with user-defined basis functions is applied to obtain the expected future value, as shown in Fig. 7.3.
   - Cash-Flow update: For every path at the current time step compare the expected return in Fig. 7.3 with the current discounted payoff, take the larger one and update the corresponding value of the cash-flow.

   Repeat this process step by step until the initial day.
4. At the initial day, average all values in the cash-flow to obtain the option value.

The challenging part for LS is the choice of basis functions for regression. They highly depend on the exact form of the option being priced and need to be matched to the characteristics of the payoff function.

The flow described previously for the LS algorithm has been explained in simple terms aimed at giving a quick background on the topic. For the study case discussed in later sections pricing two-dimensional American maximum call options, a formal algorithm is given in the Appendix.

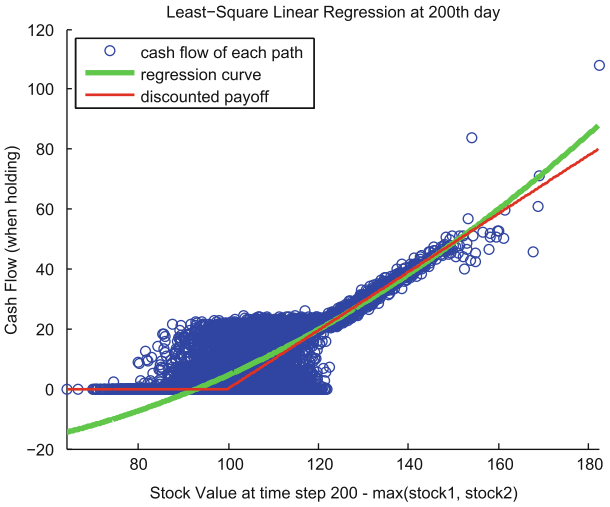

Fig. 7.3 Regression process example for a two-dimensional American max call option at time step $i = 200$. For each path the cash flow (holding value) is drawn over the current stock values (*circles*). The discounted payoff (exercising value) is shown, as well as the computed regression curve based on the drawn circles (expected mean future holding value)

7.2.6 Related Work

The use of FPGAs for accelerating financial simulations has become attractive with the first available devices. Many papers are available that propose efficient random number generation methods and paths generations. Most of the research focuses on the BS market model. For MC methods De Schryver et al. have shown that FPGAs are 33× more energy-efficient in the Heston market model [14]. For the GARCH model Thomas et al. could show speedups of 80× [16], for the Black Scholes model they showed speedups of 313× [19]. Sridharan et al. have extended this work to multi-asset options in the Black Scholes model [15], presenting speedups of up to 350× for one FPGA device. All four implementations are not able to price American options.

At the time this work was being carried out, there was only one publication of an architecture able to price American options by means of MC methods [18]. Their work is based on the LS algorithm and it has presented speedups of 20× in FPGA compared to CPU. It makes use of an efficient fully parallel architecture and an external memory chip to store the simulated MC paths. Some of the ideas presented in their work have been used as the basis of our new architecture. Nevertheless, their design makes use of 26/32-bit fixed-point arithmetic with a target resolution of 10^{-4} [17, 18].

However, their design presents several opportunities for improvement:

- Only 4K paths in MC simulations (compared to the minimum 10K paths suggested in the preceding sections)
- The use of an external memory chip, with its related power consumption and bandwidth limitation (imposed by technology)

The latter can be overcome by means of recomputation. A new approach, coined Reverse LS [2, 20], is based on this technique and is the subject of the following sections.

7.3 Reverse Longstaff-Schwartz

In the formulation of the LS algorithm in Sect. 7.2.5, first all paths are generated in step 1 and then traversed in reverse order in step 3. That means the value of each stock price at each time step for all paths has to be stored and communicated between these steps. A total of d.m.N values are generated, d being the dimension of our derivative, m the number of steps, and N the number of MC paths. We call this standard approach the *path storage solution*.

For FPGAs, with only limited internal storage of a few MB, this poses a huge design challenge and in general requires to use several external high-speed memory devices, making the design much more complex. We will now present a novel idea based on recomputation to avoid this massive storage of data.

Instead of storing the paths at each time step, we only store the final stock prices at maturity $\hat{S}_{t_m}$ and then recompute all the other alongside step 3 of the LS algorithm. For that to work we need to find a way to compute the stock price $\hat{S}_{t_i}$ based on the future price $\hat{S}_{t_{i+1}}$:

$$\hat{S}_{t_m} \to \hat{S}_{t_{m-1}} \ldots \to \hat{S}_{t_1} \to \hat{S}_{t_0}.$$

The discretized BS equation in Eq. (7.6) is reversible provided we supply the same RNs, such that:

$$\hat{S}_{t_i} = \hat{S}_{t_{i+1}} \exp\left(\left(\frac{\sigma^2}{2} - r + q\right)\Delta t - \sigma\sqrt{\Delta t}\,\Delta W_i\right).$$

In this work, we are using the MT 19937 algorithm to generate a sequence of RNs. Instead of storing the RNs the idea is to build a RNG that generates exactly the opposite sequence, starting from the last one. Fortunately, the MT is a linear RNG, meaning that its state transition function is reversible. In fact, while the tempering function is kept unaltered, only the internal states are to be recomputed [3]. In general this works for all linear RNGs. Based on this a reversed MT can be built. As a result, the Reverse LS method only needs to store and communicate d.N values.

7.4 Architecture

In this section an overview of the whole operation is given, beginning with the paths generation, going through the LS algorithm and computing the final option value. The two proposed solutions are described and compared: *Paths Storage* in an external memory chip versus the novel approach coined *Reverse LS*. A more detail description of the main blocks is covered in the subsequent sections, concluding with notes on how the architecture achieves high-throughput operation.

7.4.1 General Architecture

In general terms, CPUs can be considered as a general purpose device with a fixed hardware structure, which run a program based on a set of predefined hardwired instructions. On the contrary, FPGAs provide with a flexible hardware that can be configured according to the application, enabling dedicated blocks to run more efficiently. There are, however, recent hybrid CPU/FPGA systems, like the Xilinx Zynq, which combine both worlds and provide enough resources to attempt an efficient hardware-software partitioning with low communication latency between both parts. By pipelining the design and fully exploiting the available FPGA resources through multiple parallel instances, the architecture is able to achieve high throughput. The efficiency in terms of energy consumption is the result of carefully implemented modules with minimum resources utilization.

As mentioned before, one particular characteristic of the LS algorithm is that it can only start working (backwards from maturity towards the initial day) once all MC paths have been generated. At this point, the modules in charge of generating this data return to idle, unnecessarily consuming valuable resources on the FPGA. This situation is overcome by exploiting a powerful feature available in Xilinx Zynq devices: the FPGA can be dynamically reprogrammed, either totally or partially.

The preceding explanation leads to an architecture divided into three steps:

- STEP 1: Forward paths generation until maturity
- STEP 2: FPGA reprogramming
- STEP 3: LS operation

Reprogramming in step 2 implies that the preceding and succeeding steps have access to the total amount of resources on FPGA. The time it takes to reprogram the FPGA could be amortized depending on the setup, as it will be explained in later chapters.

The general architecture is presented in Fig. 7.4. In step 1, multiple instances of the forward paths generation block increase bandwidth. In step 3, after reprogramming, the LS algorithm starts working in a pipelined fashion, in order to compute one value per clock cycle. Again, multiple parallel instances are possible in order to

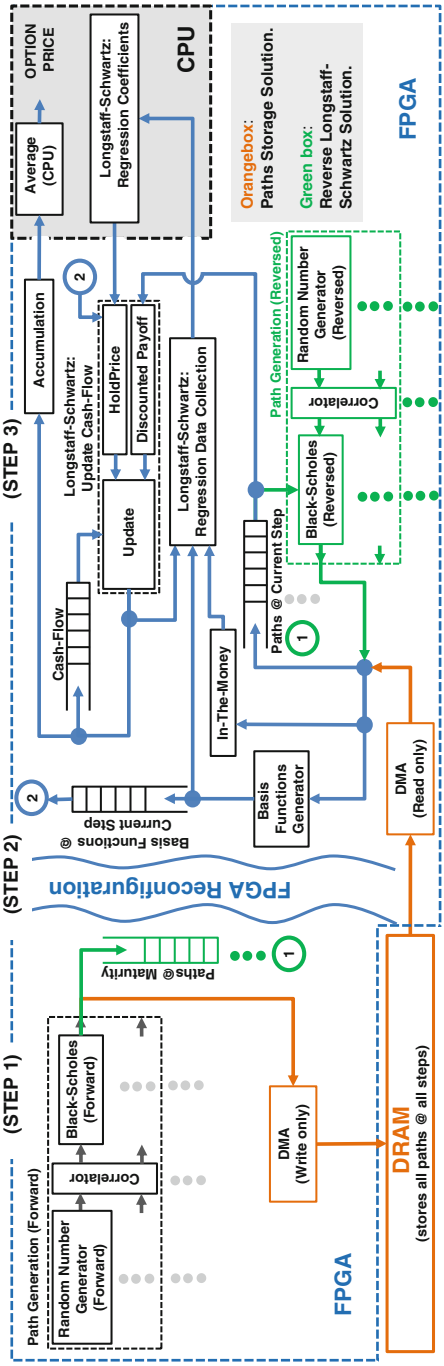

Fig. 7.4 Design architecture including both solutions: *Paths Storage* vs *Reverse LS*

increase bandwidth. Once the initial day is reached, the values from the cash-flow are averaged, which yields the option price.

The architecture in Fig. 7.4 is suited for high-dimensional options, where each instance of the path generation blocks (either forward of reversed) is capable of generating paths for each of the underlying assets (dimensions) simultaneously. Besides, the building blocks of the LS can also be adjusted accordingly.

7.4.2 Paths Storage vs Reverse LS

A straightforward approach is to store all generated paths in an external Dynamic Random-Access Memory (DRAM), as depicted in Fig. 7.4. First, there is a full write process to DRAM that takes place alongside the paths generation. Once the DRAM has been populated and the FPGA reprogrammed, the LS requests all paths, step by step, in a reverse sequence (from maturity towards the initial day). However, this approach presents three disadvantages:

- Data size: a large number of paths, steps, or dimensions, might be enough to exceed the available memory capacity
- Bandwidth: limited by technology based on the memory type (e.g. DDR3), data-bus width, and clock frequency
- Dynamic power consumption: while writing and reading data

Alternatively, the proposed Reverse LS solution overcomes the mentioned disadvantages by recomputing the paths backwards, from maturity, in parallel to the LS algorithm, as shown in Fig. 7.4. The forward paths generation process still computes all MC paths, but only needs to store the paths at maturity. A partial reconfiguration of the FPGA keeps this data on the FPGA, in order for the LS algorithm to start operation immediately in step 3.

7.4.3 Paths Generation: Forward and Reversed

The summarized forward paths generation block from Fig. 7.4 is presented in its full version in Fig. 7.5, and follows the same steps detailed in Sect. 7.2.4. Since paths belonging to each underlying are independent of each other, multiple parallel instances of the complete block are possible, as shown in Fig. 7.4 with dots. The block shown in Fig. 7.5 is configured for two-dimensional options, where each of the BS modules generates paths for one of the underlyings (dimensions). Therefore, this block can be easily extended to higher dimensions by adjusting the number of parallel internal modules, as shown in Fig. 7.4 with dots.

The reversed paths generation block is presented in Fig. 7.6, configured for two-dimensional American options, and following the explanation in Sect. 7.3. It is similar to its forward counterpart, with the exception that now the paths

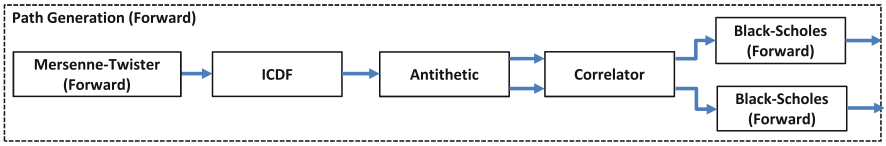

Fig. 7.5 Paths generation forward in detail, configured for two-dimensional American options

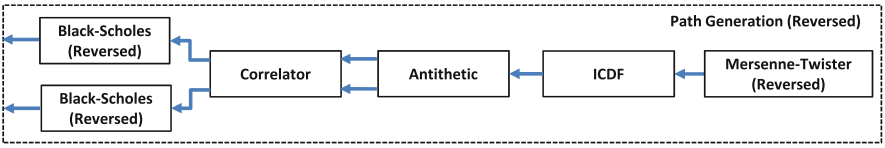

Fig. 7.6 Reversed paths generation in detail, configured for two-dimensional American options

are regenerated from maturity until the initial day (backwards), step by step. As presented before, the BS module is easily reversible. The backward operation of the MT module only needs to reverse the update process that modifies its internal states (the tempering function is kept unaltered). To obtain the same sequence of random numbers in reverse order, it only requires a copy of the last states and final index of its forward counterpart.

7.4.4 LS Implementation

The blocks required to update the cash-flow are fairly straightforward to implement following Sect. 7.2.5, and can be easily parallelized. However, the regression step presents a higher complexity in terms of implementation.

The core of the regression process consists of finding the regression coefficients required to generate the conditional expectation function at every step. These coefficients **b** are obtained by solving the system of linear equations:

$$X\,\mathbf{b} = \mathbf{y}, \tag{7.12}$$

where each row of X contains the values of the basis functions for every path that is *ITM*, and **y** contains the corresponding value in the cash-flow. The number of coefficients in **b** equals the number of basis functions.

Solving the regression process in hardware becomes either too expensive in terms of resources (fully parallel implementation) or requires a large latency (serialized version). It also becomes inflexible in terms of the method used and the number of coefficients to be calculated. To lift these restrictions, an intelligent hardware-software partitioning is introduced by calculating the coefficients on CPU. In order

to reduce the communication overhead between FPGA and CPU, the size of the matrices is reduced, following [18], by rewriting Eq. (7.12) as:

$$(X^T X)\,\mathbf{b} = (X^T \mathbf{y}), \tag{7.13}$$

where for k basis functions, the size of $(X^T X)$ and $(X^T \mathbf{y})$ is $k \times k$ and $k \times 1$ respectively. It has already been proven that this process can be pipelined by means of accumulators [18]. For monomial-type basis functions x^0, x^1 and x^2, these accumulators become:

$$X^T X = \begin{pmatrix} \sum_n x_n^0 & \sum_n x_n^1 & \sum_n x_n^2 \\ \sum_n x_n^1 & \sum_n x_n^2 & \sum_n x_n^3 \\ \sum_n x_n^2 & \sum_n x_n^3 & \sum_n x_n^4 \end{pmatrix} \quad ; \quad X^T \mathbf{y} = \begin{pmatrix} \sum_n y_n \\ \sum_n y_n x_n \\ \sum_n y_n x_n^2 \end{pmatrix} \tag{7.14}$$

Different methods can be used to solve Eq. (7.13), such as Cholesky decomposition, or the direct method via matrix inverse Eq. (7.15). Although the latter is the one implemented in this work, the Cholesky decomposition is more efficient and can be also easily implemented in the proposed architecture since these operations are executed in software.

$$(X^T X)^{-1} = \frac{1}{det(X^T X)} (Adjoint(X^T X)) \tag{7.15}$$

7.4.5 High-Throughput Operation

It is possible to achieve high-throughput operation along the entire architecture presented in Fig. 7.4. In fact, every module is designed in a pipelined fashion in order to process one new value every clock cycle. Furthermore, several blocks work in parallel, with minimum latency between each other:

- Paths Generation Forward and Direct Memory Access (DMA) (full write): data is sent to DRAM as soon as it is available, with a minimum latency enough to prepare the first DMA burst
- LS and Paths Generation Reversed / DMA (full read): regression coefficients are computed in CPU and sent to the Update Cash-Flow module. As soon as the first path in the cash-flow is updated, two extra events happen:
  - This new value is available for the next Regression Data Collection
  - The value of the stock (path) already used at the current step is no longer required and is immediately updated by either the Paths Generation Reversed module or the DMA (full RD), depending on the implemented solution. In either case, at this point in time the new value of the path has been waiting

to be delivered. It is then not only sent to the corresponding vector, but it is also sent simultaneously to determine if it is ITM and to generate the basis functions

By means of the previous explanations, high-throughput operation for the overall architecture is possible.

7.5 Amortization of FPGA Reconfiguration

Reconfiguring the FPGA implies certain time and energy consumption which can easily exceed the runtime and energy consumption required when pricing a single option. However, when pricing a large set of options, the combination of the Paths Storage approach and the novel Reverse LS allows for easy amortization of the mentioned reconfiguration. In this case, all paths are generated for every option, but only the ones at maturity are stored in an external memory chip. Once the process is finished, the FPGA is reconfigured only once and the options are priced one by one, initializing the paths from the external memory and recomputing the paths backwards by means of the Reverse LS.

7.6 Setup

A comparison between the paths storage approach against their recomputation is only possible in a common setup. In this regard, there is a key observation to make: whereas DRAM chips have an upper limit on bandwidth (defined by the memory type, the clock frequency and the width of its data bus), the bandwidth in an FPGA is only dependent on the number of available resources (hence the number of parallel instances). However, FPGA resources vary considerably among devices and vendors. As a result, both implementations are set to run at the maximum DRAM bandwidth and compared in terms of the energy consumption. Up to the mentioned bandwidth, the lowest energy consumption determines the most profitable approach. Above it, the DRAM itself will not suffice the required bandwidth.

The complete setup, as well as the hardware resources, are detailed in Table 7.1. The FPGA clock is a submultiple of the one used in DRAM, and enough instances of all blocks are used in order to achieve the target bandwidth of 4,266 MB/s.

Although the chosen setup targets two-dimensional options as a testcase, the architecture proposed in Sect. 7.4.1 can be easily adapted for high-dimensional American options.

Our implementation has also been cross-verified with a binomial tree implementation: *Reverse LS*: $P = 9.92 \pm 0.24$; *Binomial Tree (Benchmark)*: $P = 10.12$; Setup:

Table 7.1 Setup table

Detail	Description
Option style	American
Option type	Call
Option characteristics	maximum
Dimensions	2
Basis functions type	Monomial
Basis functions detail	$1, max(S_1, S_2), max(S_1, S_2)^2$
Paths per dimension	10K
Steps	365
Data type	Single-precision floating point
Total data	27.85 MB
Platform	ZC702 evaluation kit
Operating system	Linux (Linaro)
DRAM type	DDR3
DRAM data-bus	32 bits
DRAM clock	533.33 MHz
DRAM bandwidth	4266.64 MB/s
FPGA clock	133.33 MHz
FPGA bandwidth	4266.64 MB/s

$S_{1,2}(0) = 100$, $K = 100$, $r = 0.05$, $q_{1,2} = 0.10$, $\sigma_{1,2} = 0.2$, $\rho = 0.1$. The chosen basis functions generally deliver good results for general options, however not the best result. This depends on the type and the number of basis functions, which need to be tried and tested.

7.7 Tools and Estimation Methodology

The different modules have been implemented in Vivado High-Level Synthesis (HLS) using C, and optimized for high-performance at a clock period of 7.5 ns (133.33 MHz) with a minimum number of FPGA resources. The place-and-route (P&R) report on resources utilization was then fed into the Xilinx Power Estimator [21] in order to obtain power estimations of individual blocks. The estimated values have been checked by means of testbenches on the Xilinx Zynq ZC702 Evaluation Kit. In a similar way, DRAM DDR3 power consumption is based on measured values at different bandwidths on the same board. The testbench followed the same access pattern used in the full architecture and achieved a maximum of 83 % and 87 % of the peak theoretical bandwidth for writing and reading respectively [20]. Then these values were extrapolated to the maximum theoretical bandwidth available (4,266 MB/s at 533.33 MHz and 32-bit data bus).

In terms of energy consumption, all values are derived from the obtained average power consumption and the required runtime.

7.8 Results

For the given setup, the resources utilization on the FPGA is detailed in Table 7.2, grouped by major blocks. A note is made on the fact that the minimum Zynq device on which the given configuration fits, with the required parallel instances, is the Z-7030 device.

As mentioned before, every single building block in the proposed architecture has been fully pipelined with an initiation interval of one clock cycle (II = 1). This means that every block starts processing a new data value in every clock cycle. At 4,266 MB/s, the total amount of data (27.85 MB) is processed in approximately 6.53 ms, as presented in Table 7.3. The total runtime in this case, including the communication overhead between CPU and FPGA and excluding the FPGA reconfiguration, adds up to 16.94 ms for one option pricing.

Table 7.2 FPGA resources breakdown

Step	Block	LUT	FF	DSP	BRAM
1	Path generation forward	18,404	17,376	188	88
	Paths @ Maturity	1,752	1,648	0	64
2	Reconfiguration	–	–	–	–
3	Longstaff-Schwartz	28,296	33,468	212	108
	Path generation reversed	24,048	23,932	164	88

Table 7.3 Power, runtime and energy consumption breakdown

Block	Dynamic power (mW)	Runtime (ms)	Energy (mJ)
Path generation forward	1,239	6.53	8.09
Paths @ Maturity	334	0.02	0.01
Paths storage full WR	1,265	6.53	8.26
MT communication overhead 1/2	160	1.21	0.19
Reprogramming [1]	1,860	50.00	93.00
MT communication overhead 2/2	160	0.62	0.09
Paths storage full RD	1,526	6.53	9.97
Path generation reversed	1,392	6.53	9.09
Regression data collection	795	6.53	5.19
Regression coefficients (CPU)	160	2.03	0.32
Update cash-flow	374	6.53	2.44
Cash-flow	174	6.53	1.14
Paths @ Current-Step	334	6.53	2.18
Accumulation (Average)	103	0.02	0.00

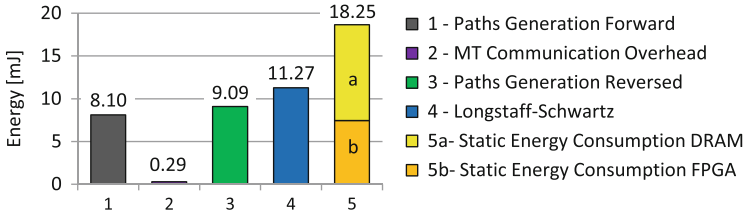

Fig. 7.7 Dynamic energy consumption breakdown

7.8.1 Dynamic Energy Consumption Breakdown

Based on the dynamic power and the runtime, it is possible to derive the dynamic energy consumption of every building block shown in Fig. 7.4, as detailed in Table 7.3.

Figure 7.7 presents the dynamic energy consumption breakdown of the whole architecture when the novel Reverse LS approach is implemented. *MT communication overhead* refers to the energy consumed to initialize the internal states of the forward MT modules, read the final states and index, and initializing the reversed MT modules. The *LS* column in Fig. 7.7 includes the energy consumption in FPGA (10.95 mJ) and in CPU (0.32 mJ). The latter includes the computation of the regression coefficients, as well as the communication overhead when reading the accumulated matrices and writing back the coefficients.

An optimized CPU implementation of the entire algorithm in Matlab on an Intel i5-2450M (2.50 GHz) core with, 6 GB of RAM, requires, for the given setup, 270 ms and an energy consumption of 12.70 J. The latter has been obtained at the power-plug with all unnecessary components in the computer disabled. In contrast, our implementation in Zynq requires 16.94 ms and consumes approximately 47 mJ, providing a speedup of 16× in runtime and 268× in energy consumption.

7.8.2 Reverse LS Versus Paths Storage

When comparing the regeneration of the paths in FPGA against the storage of all paths in DRAM (both when writing and reading data), there is a reduction in the energy consumption of 2×, as depicted in Fig. 7.8. All values shown are based on the given setup and methodology. To make a fair comparison, only the additional (dynamic) energy consumption is taken into account. This is due to the fact that in a hybrid CPU/FPGA device, like the Xilinx Zynq running Linux on the ARM cores, the DRAM is already being used by the operating system.

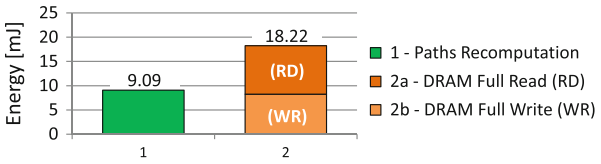

Fig. 7.8 Dynamic energy consumption when regenerating all paths in FPGA and when storing the paths in an external memory (DRAM)

7.8.3 Comparison to Related Work

The reference work [18] presented a dedicated FPGA implementation targeted for one specific option and setting. It further uses a number format specialized for this usecase based on 26/32-bit fixed-point operations. With our proposed architecture we show how it is possible to target high-dimensional options. We further use single-precision floating-point operations, so that the user does not have to take care of the accuracy of the solution.

The main inconvenience in comparing our work to the reference resides in the fact that both architectures target different devices at different technology nodes. Under these circumstances, it was decided to run the comparison on the basis of energy efficiency, by porting their work [18] to the same Xilinx Zynq device based on their published resources utilization. Although this approach is just a coarse estimation, it could still be considered a valid setup for a comparison purposes. For their work, one option pricing consumes, according to XPE, 2.46 mJ dynamic power including one DRAM chip. Our downscaled architecture to one-dimension and the same number of paths and steps only requires 1.85 mJ dynamic power, being a 33 % improvement. This means that we achieve higher energy-efficiency while providing higher accuracy. We make this possible with FPGA reconfiguration in combination with an optimized scheduling, and our novel Reverse LS approach.

7.9 Conclusion

American option pricing is a computational challenge for financial institutions, which operate huge clusters. In this work a high-throughput and energy-efficient pricing system for American options has been presented, targeting hybrid CPU/FPGA devices. Compared to the state-of-the-art, this is the first FPGA-based implementation targeting the full range of high-dimensional American options.

Our main contribution is Reverse Longstaff-Schwartz, a bit-true algorithmic transformation where recomputation is exploited. Paths storage is minimized by means of recomputation, removing any bandwidth limitation and significantly improving energy-efficiency. By additionally making use of runtime reconfiguration and utilizing an optimized scheduling to amortize the reconfiguration times, we are

able to deliver higher energy-efficiency. In this regard, the resulting architecture is 16× faster and 268× more energy-efficient than an optimized Intel i5 implementation in Matlab.

Acknowledgements We gratefully acknowledge the partial financial support from the Center of Mathematical and Computational Modelling $(CM)^2$ of the University of Kaiserslautern, from the German Federal Ministry of Education and Research under grant number 01LY1202D, and from the Deutsche Forschungsgemeinschaft (DFG) within the RTG GRK 1932 "Stochastic Models for Innovations in the Engineering Sciences", project area P2. The authors alone are responsible for the content of this work.

Appendix

Algorithm 1 Longstaff Schwartz MC method to price American maximum call option on two stocks

Input: discounted payoff $g(S)$
Output: option price V_N

1: Generate N independent paths for two stocks at all possible exercise dates: $\{S_1^n(t_0), S_1^n(t_1), \ldots, S_1^n(t_m)\}$ and $\{S_2^n(t_0), S_2^n(t_1), \ldots, S_2^n(t_m)\}$, with $n = 1, \ldots, N$, $t_i = \frac{T}{m} \times i$, $i = 1, \ldots, m$ and $S_1^n(t_0) \equiv S_1(0)$, $S_2^n(t_0) \equiv S_2(0)$ as follows:

$$S_1(t_i) = S_1(t_{i-1}) e^{((r-q_1-\frac{1}{2}\sigma_1^2)\Delta t + \sigma_1\sqrt{\Delta t} Z_1)}$$

$$S_2(t_i) = S_2(t_{i-1}) e^{((r-q_2-\frac{1}{2}\sigma_2^2)\Delta t + \sigma_2\sqrt{\Delta t} Z_2)}$$

with $Z_1 = u_1$ and $Z_2 = \rho u_1 + \sqrt{1-\rho^2} u_2$, where $u_1, u_2 \sim N(0,1)$.

2: At maturity $t_m = T$, fix the discounted terminal values of the American option for each path $n = 1, \ldots, N$:

$$V^n(t_m) = e^{-rT} (\max\{S_1^n(t_m), S_2^n(t_m)\} - K)^+$$

3: Compute backward at each potential exercise date t_i for $i = m-1, m-2, \ldots, 1$:

1. Choose k basis functions: $\{H_1, \ldots, H_k\}$.
2. Consider the subset of paths $\Theta_{\hat{N}} \subset \{1, \ldots, N\}$ for which the option is ITM, i.e. $\max\{S_1^n(t_m), S_2^n(t_m)\} > K$ holds for $n \in \Theta_{\hat{N}}$.
3. Solve the least-square linear regression problem:

$$\min_{a_l \in \mathbb{R}} \frac{1}{\hat{N}} \sum_{n=1}^{\hat{N}} (V^n(t_i) - \sum_{l=1}^{k} a_l H_l(S_{1,2}^n(t_i)))^2$$

$$S_{1,2}^n(t_i) := \{S_1^n(t_i), S_2^n(t_i)\} \quad \text{for simplicity}$$

and obtain the optimal coefficient a^*:

$$a^* := [a_1^*, \ldots, a_k^*]^\top$$
$$= (X^\top X)^{-1} X^\top Y \in \mathbb{R}^{k \times 1}$$

(continued)

Algorithm 1 (continued)

with $Y := [V^1(t_i), \ldots, V^{\hat{N}}(t_i)]^\top \in \mathbb{R}^{\hat{N}\times 1}$,

$$X := \begin{pmatrix} H_1(S^1_{1,2}(t_i)) & \ldots & H_k(S^1_{1,2}(t_i)) \\ \vdots & \ldots & \vdots \\ H_1(S^{\hat{N}}_{1,2}(t_i)) & \ldots & H_k(S^{\hat{N}}_{1,2}(t_i)) \end{pmatrix} \in \mathbb{R}^{\hat{N}\times k}$$

4. Calculate the approximation of the value for continuing the option $C^n(t_i)$ and the value for exercising the option $E^n(t_i)$ for each path $n \in \Theta_{\hat{N}}$:

$$C^n(t_i) = \sum_{l=1}^{k} a_l^* H_l(S^n_{1,2}(t_i))$$

$$E^n(t_i) = e^{-rt_i}(\max\{S^n_1(t_i), S^n_2(t_i)\} - K)^+$$

5. Compare the value of $C^n(t_i)$ and $E^n(t_i)$ to decide whether to exercise or to continue the option:

$$V^n(t_i) = \begin{cases} E^n(t_i), & \text{if } n \in \Theta_{\hat{N}} \text{ and } E^n(t_i) \geq C^n(t_i) \\ V^n(t_{i+1}), & \text{otherwise} \end{cases}$$

4: Compute $V_N = \left(\frac{1}{N}\sum_{i=1}^{N} V^n(t_1)\right)$ as an approximation for the American option price

References

1. Brugger, C., de Schryver, C., Wehn, N.: HyPER: a runtime reconfigurable architecture for Monte Carlo option pricing in the Heston model. In: 2014 IEEE 24th International Conference on Field Programmable Logic and Applications (FPL), Munich (2014). doi: 10.1109/FPL.2014.6927458
2. Brugger, C., Varela, J.A., Wehn, N., Tang, S., Korn, R.: Reverse Longstaff-Schwartz American option pricing on hybrid CPU/FPGA systems. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, pp. 1599–1602. ACM (2015)
3. Hagita, K., Takano, H., Nishimura, T., Matsumoto, M.: Reverse generator MT19937. Fortran source code (2000). http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/FORTRAN/REVmt19937b.f. Last access 16 Sept 2014
4. Hull, J.C.: Options, Futures, and other Derivatives, 8th edn. Pearson, Harlow (2012)
5. Jin, Q., Luk, W., Thomas, D.B.: On comparing financial option price solvers on FPGA. In: 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Salt Lake City, pp. 89–92 (2011). doi:10.1109/FCCM.2011.30
6. Kandemir, M., Li, F., Chen, G., Chen, G., Ozturk, O.: Studying storage-recomputation tradeoffs in memory-constrained embedded processing. In: 2005 Proceedings in Design, Automation and Test in Europe, Munich, pp. 1026–1031. IEEE (2005)
7. Koc, H., Kandemir, M., Ercanli, E., Ozturk, O.: Reducing off-chip memory access costs using data recomputation in embedded chip multi-processors. In: Proceedings of the 44th Annual Design Automation Conference, San Diego, pp. 224–229. ACM (2007)
8. Korn, R., Korn, E., Kroisandt, G.: Monte Carlo Methods and Models in Finance and Insurance. CRC, Boca Raton (2010)

9. Longstaff, F.A., Schwartz, E.S.: Valuing American options by simulation: a simple least-squares approach. Rev. Financ. stud. **14**(1), 113–147 (2001)
10. Matsumoto, M.: Mersenne twister home page (2007). http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. Last access 02 July 2014
11. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (1998). doi:http://doi.acm.org/10.1145/272991.272995
12. Monetary, Economic Department: Statistical release: OTC derivatives statistics at end-December 2013. Technical report, Bank for International Settlements (2014). http://www.bis.org/publ/otc_hy1405.pdf
13. de Schryver, C., Schmidt, D., Wehn, N., Korn, E., Marxen, H., Korn, R.: A new hardware efficient inversion based random number generator for non-uniform distributions. In: Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, pp. 190–195 (2010). doi:10.1109/ReConFig.2010.20
14. de Schryver, C., Shcherbakov, I., Kienle, F., Wehn, N., Marxen, H., Kostiuk, A., Korn, R.: An energy efficient FPGA accelerator for Monte Carlo option pricing with the Heston model. In: Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, pp. 468–474 (2011). doi:10.1109/ReConFig.2011.11
15. Sridharan, R., Cooke, G., Hill, K., Lam, H., George, A.: FPGA-based reconfigurable computing for pricing multi-asset barrier options. In: Proceedings of Symposium on Application Accelerators in High-Performance Computing PDF (SAAHPC), Argonne (2012)
16. Thomas, D.B., Bower, J.A., Luk, W.: Hardware architectures for Monte-Carlo based financial simulations. In: Proceedings of the IEEE International Conference on Field Programmable Technology FPT 2006, Bangkok, pp. 377–380 (2006). doi:10.1109/FPT.2006.270352
17. Tian, X., Benkrid, K.: Fixed-point arithmetic error estimation in Monte-Carlo simulations. In: 2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, pp. 202–207 (2010). doi:10.1109/ReConFig.2010.14
18. Tian, X., Benkrid, K.: Implementation of the Longstaff and Schwartz American option pricing model on FPGA. J. Signal Process. Syst. **67**(1), 79–91 (2012). doi:10.1007/s11265-010-0550-1
19. Tse, A.H., Thomas, D.B., Tsoi, K.H., Luk, W.: Efficient reconfigurable design for pricing Asian options. SIGARCH Comput. Archit. News **38**(4), 14–20 (2011). doi:10.1145/1926367.1926371. http://doi.acm.org/10.1145/1926367.1926371
20. Varela, J.A.: Embedded architecture to value American options on the stock market. Master's thesis, Microelectronic Systems Design Reseach Group, Department of Electrical Engineering and Information Technology, University of Kaiserslautern (2014)
21. Xilinx: XPower estimator (XPE) (2014). http://www.xilinx.com/products/design_tools/logic_design/xpe.htm. Last access: 16 Sept 2014