FPGA-accelerated Monte-Carlo integration using stratified sampling and Brownian bridges

Mark de Jong, Vlad-Mihai Sima and Koen Bertels
Department of Computer Engineering
Delft University of Technology
Delft, The Netherlands

David Thomas
Department of Computing
Imperial College London
London, United Kingdom

***Abstract*—Monte-Carlo Integration (MCI) is a numerical technique for evaluating integrals which have no closed form solution. Naive MCI randomly samples the integrand at uniformly distributed points. This naive approach converges very slowly. Stratified sampling can be used to concentrate the samples on segments of the integration domain where the integrand has the highest variance. Even with stratified sampling, MCI converges very slowly for multidimensional integrals. In this work, we implement an FPGA-accelerated design for MISER, a widely used adaptive MCI algorithm applying stratified sampling. We show how to eliminate the recursion from MISER and partition the algorithm between CPUs and FPGAs. The CPUs manage the control-heavy stratification strategy, while the FPGA is responsible for sampling the integrand. The integrand is compiled into a deep pipeline on the FPGA, producing one function evaluation per clock cycle. We demonstrate the FPGA-accelerated design by pricing a path dependent financial derivative called an Asian option. To make optimal use of the stratification, we implement a Brownian bridge on the FPGA that produces one entire bridge per clock cycle. The FPGA-accelerated design is up to 880 times faster compared to a software reference using the GSL implementation of MISER. Compared to naive MCI in software, our design even requires up to 3572 times less execution time to achieve the same accuracy.**

I. Introduction

Monte-Carlo integration numerically approximates the definite integral of an integrand. This computationally intensive method evaluates the integrand at randomly chosen points in a given domain. The average of these evaluations is the estimate of the integral. Unfortunately, the accuracy of the estimate only scales as the square root of the number of samples. Stratified sampling can be used to enhance the accuracy by concentrating most of the samples on "difficult" areas of the domain. Even with stratified sampling, MCI converges very slowly for multidimensional integrals. These integrations can benefit from acceleration with FPGAs.

Monte-Carlo methods are extensively used in finance. For example, pricing exotic options is done by evaluating high-dimensional integrals with no closed form solution. Stratified sampling can be particularly useful for option pricing in combination with a Brownian bridge. We demonstrate how to improve the execution time of exotic option pricing by combining a hardware implementation of a Brownian bridge with the adaptive stratified sampling algorithm MISER.

The main contributions of this work are:

- A paradigm for splitting stratified MCI between software and hardware, allowing for high utilization of the hardware accelerator.
- A framework for implementing such a system in hardware, allowing selection of custom integrands, that can achieve one simulation per cycle if the integrand can be evaluated in one cycle.
- Application of the framework to an exotic option pricing problem, including a Brownian Bridge, to maximize variance reduction.
- Evaluation in hardware, achieving an acceleration of 880 times over a software reference using the GSL implementation of MISER running on an Intel i7-4770 CPU at 3,40 GHz.
- Evaluation of the combined advantage of stratification and FPGA-acceleration, requiring up to 3572 times less execution time to achieve the same accuracy compared to a software reference using naive MCI.

The remainder of this paper is structured as follows. Section III will first discuss the background topics MCI, stratified sampling, option pricing and Brownian bridges. We adapt MISER to the requirements for a parallel design in Section IV and develop a hardware design in Section V. Section VI discusses the implementation in hardware of an integrand for Asian option pricing. Results for the timing and accuracy of this design are discussed in Section VII. Section VIII concludes this paper.

II. Related Work

Previous work has demonstrated the effectiveness of FPGAs for Monte-Carlo methods in finance. Thomas et al. present FPGA implementations for five different Monte-Carlo simulations of asset price movements in [1]. In [2], an FPGA, a GPU and CPUs are all used to accelerate the computation of the Value at Risk for a portfolio of correlated assets using Monte-Carlo simulation.

FPGA-accelerated Monte-Carlo designs using variance reduction methods are presented in (among others) [3], [4], [5] and [6]. A control variate option pricing framework is used in [3] to determine the price of an Asian option. [4] uses Quasi-random numbers to determine the price of American options. [5] presents a Quasi-Monte-Carlo simulator with a Brownian bridge to generate random walks. In [6], random numbers are generated by using stratified sampling and latin hypercube for a Gaussian random number generator.

In contrast to [6], which applies stratification for improved random number generation, our design uses an advanced

This research is partially supported by Artemis EMC2 project with grant agreement 621429.

978-1-4799-6245-7/14/$31.00 ©2014 IEEE

stratification strategy controlled by CPUs, to adaptively allocate the optimal number of simulations to a specific segment of the entire integration domain. Furthermore, our high-throughput FPGA design produces one entire brownian bridge per clock cycle, where [5] produces one point of the bridge every clock cycle.

III. Background

A. Monte-Carlo Integration

Monte-Carlo integration is a numerical method to determine the value of an integral. Consider the function $y = g(x)$ and the integral

$$\int_0^1 g(x)\,dx, \tag{1}$$

where x is a scalar. If $f(x)$ is the probability density function (PDF) of X which has a uniform distribution on the interval $[0, 1]$, this integral can also be described as

$$\int_0^1 g(x)\,dx = \int_{\mathbb{R}} g(x)f(x)\,dx = \mathbb{E}[g(x)]. \tag{2}$$

An estimate for this expected value can be found by Monte-Carlo simulation. This estimate is $\overline{y}$, which is the average result of N simulations[1]. This method will be referred to as naive MCI. This method can also be applied on multidimensional integrals. In Section III-E, we will integrate an n dimensional function for Asian option pricing.

The estimator $\overline{y}$ is unbiased, but suffers from a large sampling error. The $(1-\alpha)$-confidence interval with $0 < \alpha < 1$ is given by

$$\left[\overline{y} - z\left(1-\tfrac{\alpha}{2}\right)\cdot SE_{\overline{y}}, \overline{y} + z\left(1-\tfrac{\alpha}{2}\right)\cdot SE_{\overline{y}}\right], \tag{3}$$

where $SE_{\overline{y}}$ is the standard error of $\overline{y}$ and $z\left(1-\frac{\alpha}{2}\right)$ is the $(1-\frac{\alpha}{2})$-quantile of the standard normal distribution. For example, $z\left(1-\frac{0.05}{2}\right) = 1.96$. For naive MCI, the standard error is given by

$$SE_{\overline{y}} = \frac{\sigma_y}{\sqrt{N}}, \tag{4}$$

where σ_y is the standard deviation of y.

B. Stratified Sampling

Improving the accuracy constitutes to reducing the length of the confidence interval. Increasing N with a factor p decreases the standard error of the estimator only with a factor $\sqrt{p}$. However, variance reduction techniques such as stratified sampling can reduce the size of the confidence interval without increasing N.

Stratified sampling divides the sample space of X into m segments (or 'strata'). The integral for each segment can be determined with a different number of samples for each segment. The integral on the entire domain follows as a weighted sum of the integrals on the segments according to

$$\mathbb{E}[g(x)] = \sum_{i=0}^{m} \mathbb{E}[g(x)|x \in J_i]\cdot \mathrm{P}[x \in J_i], \tag{5}$$

[1]This average should be multiplied by the volume of the integration domain, which is 1 in this situation.

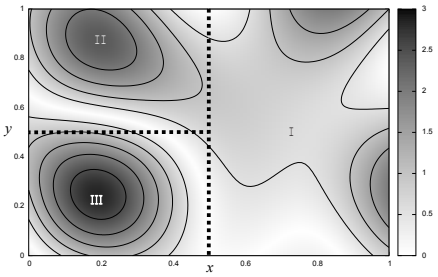


Figure 1: Contour graph of $h(x, y)$. The domain is divided into 3 segments.

where $P[x \in J_i]$ is the probability that a realization x from X is from segment J_i [7]. To increase the accuracy, one should allocate more simulations to segments that are responsible for most of the variance in the Monte-Carlo estimator.

Stratification can also be applied to multidimensional integrals. As an example, consider the graph of function $h(x, y)$ in Figure 1. Naive MCI can determine the integral on the domain $x = [0, 1]$, $y = [0, 1]$. However, dividing the domain in 3 segments as shown by the dotted lines in the figure can increase the accuracy of the Monte-Carlo estimate. The figure shows that the variance of $h(x, y)$ is lowest in segment I. The variance is higher in segment II and highest in segment III. Allocating most simulations to segment III and fewest simulations to I will reduce the variance of the estimate compared to naive MCI.

C. MISER

An adaptive variance reduction algorithm can be used when it is unknown prior to the integration on which domain the integrand has the highest variance. MISER [8] is a widely used adaptive algorithm for high-dimensional integrals based on recursive stratified sampling.

MISER integrates a function with *N* simulations on an integration domain. MISER performs one of two actions when it is executed:

- Determine the integral on the given domain with *N* simulations using naive MCI.
- Determine the optimal bisection of the domain and call MISER on each segment recursively.

Which action MISER performs is based on some user-defined threshold *M*. If $N < M$, MISER performs naive MCI. If $N \geq M$, MISER bisects the domain, leading to two new MISER calls. Figure 2 shows this process for the integration of $h(x, y)$ from Section III-B over the domain $x = [0, 1]$, $y = [0, 1]$ using $N = 1000$ and $M = 200$.

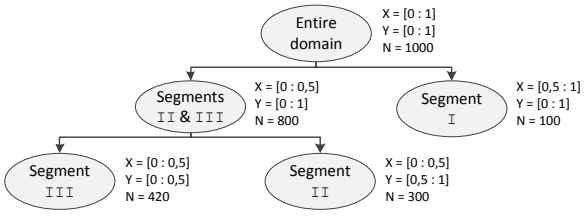


Figure 2: Progress of MISER for function $h(x, y)$.

MISER needs to determine the optimal dimension to bisect, which we will refer to as 'segmentation'. For this purpose, a fraction α of the *N* available simulations is used to sample the integration domain. MISER considers a bisection for each dimension and estimates the variance for each of the $2n$ segments using $\alpha \cdot N$ samples. In Figure 2, $\alpha = 0.1$. For each bisection, the variance estimates of the two segments are accumulated. The bisection which results in the largest accumulated variance is optimal and MISER is used recursively on each segment [9]. The variance estimates are used to determine the optimal number of simulations for each segment. When the recursion finishes, the results for all segments are combined with Equation 5 to compute the entire integral[2].

Segmentation is computationally very intensive for multidimensional integrals. The number of possible bisections equals the number of dimensions n. For each (recursive) MISER call with $N \geq M$, the variance is estimated for all $2n$ considered segments. The execution time to determine the optimal bisection is thus $\mathcal{O}(\alpha \cdot N \cdot n)$.

D. Option Pricing

An option provides the right but not the obligation to buy (call) or sell (put) an asset, such as a stock or a bond. The price for which the asset can be bought or sold is called the *strike price*, denoted by *K*. The date for which the option has to be exercised is the *expiration date* or *maturity*, denoted by *T*. The simplest option is a European option. The payoff for a European option only depends on a fixed strike price and the underlying stock price at maturity. Exotic options, such as Asian options, barrier options and basket options, have more complex payoff structures. For example, the payoff for a discrete Asian call option with fixed strike is determined by the difference between the average stock price at *n* moments in time and the strike price according to

$$Payoff = \max\left(0, \frac{1}{n} \cdot \Sigma_{i=1}^{n} S(t_i) - K\right), \quad (6)$$

where $S(t)$ is the price of the underlying asset at time t and $T = t_n$. Such an option, for which the payoff depends on the price path of the underlying asset, is called path dependent. We refer readers with further interest in options and other financial derivatives to [10].

Determining an option price requires a model for the underlying asset. In the commonly used Black-Scholes model [11], the stock price movement is described by a geometric Brownian motion. The stock price at time *t* is modeled by

$$S(t) = S(t_0) \cdot e^{\left(r - \frac{\sigma^2}{2}\right)t + W(t)}, \text{with } W(t) = \sigma\sqrt{t}Z_t, \quad (7)$$

where *r* is the risk free rate, σ is the volatility of the underlying asset and Z_t is a random variable with a standard normal distribution. The term $\left(r - \frac{\sigma^2}{2}\right)t$ is called the *drift*. Using this model for *S(t)*, the current value of an Asian option is given by

$$V = e^{-rT} \cdot \mathbb{E}_{\mathbb{Q}}\left[\max\left(0, \frac{1}{n} \cdot \Sigma_{i=1}^{n} S(t_i) - K\right)\right], \quad (8)$$

where the operator $\mathbb{E}_{\mathbb{Q}}[\cdot]$ gives the expected value under risk neutral probability measure $\mathbb{Q}$ [10]. Monte-Carlo integration can be used to determine this current value V.

[2]The interested reader is referred to [9] for a more extensive explanation of MISER.

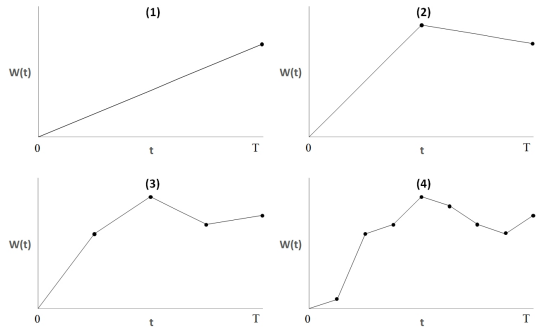


Figure 3: Brownian bridge construction after 1, 2, 4 and 8 points have been sampled.

E. Brownian Bridge

Determining the value of an Asian option with Monte-Carlo integration requires the generation of *N* entire stock price paths with *n* timesteps. A natural way to construct these stock price paths using Equation 7 is by calculating $S(t_1), S(t_2), \ldots, S(t_n)$ sequentially. To do this, the exponent $W(t)$ in Equation 7 can be calculated according to

$$W(t_i) = W(t_i - \tau) + \sigma\sqrt{\tau}Z_t, \quad (9)$$

where τ is the difference between two consecutive timesteps. In this sequential approach, $W(t_i - \tau)$ is calculated before $W(t_i)$. However, the data points can be generated in any order as long as Z_t is sampled from the distribution conditional on the values that are already generated. Furthermore, some parallelism can be exploited since two data points $W(t_i)$ and $W(t_j)$ with $i < j$ can be generated in parallel as long as there is already a data point $W(t_k)$ with $i < k < j$ generated.

To make optimal use of stratified sampling with MISER, a Brownian bridge can be used. A Brownian bridge constructs a point on a Brownian path by conditioning on the known surrounding points. Consider the graph in Figure 3 where $T = t_n$. First, the values of $W(t_0)$ and $W(T)$ are calculated. Conditional on these values, $W(\frac{1}{2}T)$ is calculated. Next, $W(\frac{1}{4}T)$ can be calculated conditional on $W(t_0)$ and $W(\frac{1}{2}T)$. $W(\frac{3}{4}T)$ can be calculated in parallel using $W(\frac{1}{2}T)$ and $W(T)$. Finally, $W(\frac{1}{8}T)$, $W(\frac{3}{8}T)$, $W(\frac{5}{8}T)$ and $W(\frac{7}{8}T)$ are calculated such that the entire stock price path is known.

Each simulation requires the generation of a new stock price path. Data points in a stock price path are independent of other stock price paths. This parallelism can be exploited on an FPGA.

Stratified sampling can be used to generate stock price paths that contribute most to the variance of the Monte-Carlo estimator. The generation of $W(T)$ has a big influence on the entire stock price path. A high value for $W(T)$ will lead

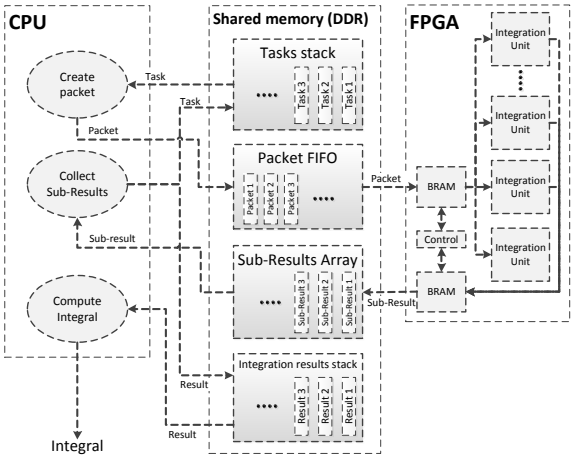


Figure 4: Parallelizable version of MISER

to a high average for the stock price. Therefore, a Brownian bridge is well suited for stratified sampling.

IV. Parallel Architecture

We develop an efficient FPGA-accelerated implementation of MISER. CPUs control the stratification. FPGAs perform the data intensive work: naive MCI and segmentation. We need to solve two key problems to develop this parallel design. First, we need to remove the recursion from MISER to expose parallel tasks. Next, we need a paradigm that maximizes the utilization of pipelined FPGA operators. An important factor here is the communication overhead. The size of the data required by the FPGA for integration and segmentation is $\mathcal{O}(n)$. The size of the result communicated to the CPU is $\mathcal{O}(n)$ too. Communicating this data is a bottleneck for highdimensional integrals, stalling the FPGAs.

To eliminate the recursion, we consider each node in the graph of Figure 2 to be a Task. A Task is either a Segmentation Task (ST) or an Integration Task (IT). The leaf nodes in Figure 2 result in ITs, while the other nodes result in STs. Unhandled Tasks are stored on a stack. The CPU pops unhandled Tasks and creates packets that hold the minimal amount of information required to process a Task: the type of task, the number of simulations, the integration domain and possibly other integrand specific parameters. The CPU communicates the packets to an FPGA, which performs the integration or segmentation on an integration kernel. One FPGA may hold multiple integration kernels. The FPGA communicates one Sub-Result per packet to the CPU. If the FPGA processed an ST, the CPU pushes two new Tasks on the stack. Execution of an IT yields an Integration Result, stored on a separate stack. Execution finishes when all Tasks are handled. Figure 4 shows the flow of data in this parallel design.

The design hides the communication overhead by performing the data communication in parallel to the work on the FPGA. Packets and Sub-Results are stored in a double buffer in a shared memory. We maximize the throughput by loading data to and from the FPGA while the FPGA processes Tasks in parallel. Since the FPGA can execute packets in any order, this paradigm allows a scheme where the Task with most simulations is communicated first. While the FPGA is processing that Task, smaller Tasks may be communicated in parallel without stalling the FPGA.

V. Integrand independent hardware design

We implement pipelined integration kernels on an FPGA to process STs and ITs in single precision floating-point format. Figure 5 shows the structure of an integration kernel. Various integrands can be used with this design. The implementation of the pricing function for an Asian option using a Brownian bridge is discussed in Section VI. This section describes the integrand independent hardware.

A random number generator (RNG) produces n random numbers from an arbitrary distribution per clock cycle. The random numbers are used to sample the integrand. For stratified sampling, the random numbers are generated in the correct segment. Our RNG generates random numbers in four steps. First, uniformly distributed fixed-point random numbers are generated using the RNG from [12]. Second, these numbers are transformed into uniformly distributed floating-point numbers on the domain [0,1] with the technique described in [13]. Third, the floating-point numbers are converted to the correct integration domain. Fourth, the uniformly distributed numbers are transformed to random numbers from the target distribution using the Inverse Cumulative Distribution Function (ICDF) as described in [14]. The authors of [14] allow us to use their design and adapt it to our requirements.

The integrand is sampled with new random numbers every clock cycle. The integrand should be compiled into a deep pipeline, generating one simulation on every clock cycle.

For ITs, the FPGA calculates the average and variance of the simulations. For STs, the FPGA finds the optimal segmentation. Each segmentation creates a left and a right segment. For all segments, an estimate of the standard deviation is required. Following the implementation of MISER in [9], the estimate of the standard deviation of a segment is implemented as the difference between the minimum and maximum value for that segment[3]. Figure 6 shows how the minimum and maximum are calculated.

[3]This method leads to a stable design according to [9]. Using the sample standard deviation for each segment gives a better estimate, but results in a much larger resource utilization on the FPGA

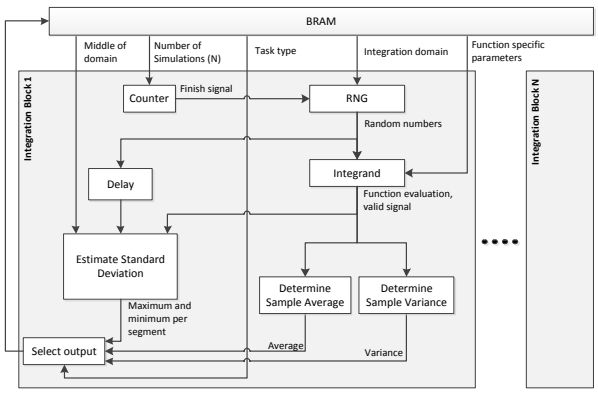


Figure 5: Structural overview of the integration kernel.

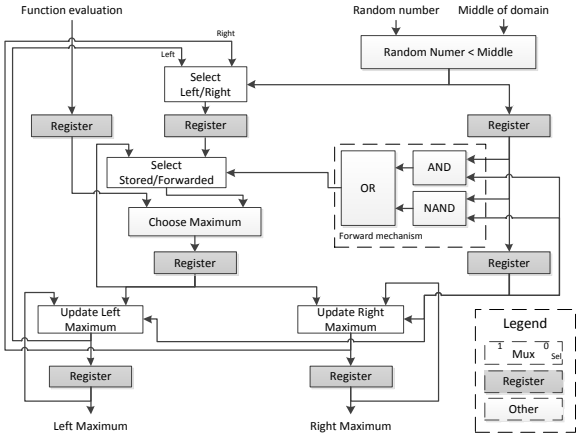


Figure 6: Structural overview of the component Estimate Standard Deviation *from Figure 5. The figure shows how the maximum function evaluation is found efficiently for a right and left segment. The minimum values are determined in a similar fashion.*

VI. Integrand for Asian option pricing in hardware

Asian option pricing requires an integrand that can generate one Brownian bridge per clock cycle. Such a design was presented in [15]. Every simulated Brownian bridge yields one simulated option payoff. To construct the Brownian bridge, we use the algorithm from figure 3.2 in [16].

We transform this algorithm in order to produce one Brownian bridge per clock cycle on an FPGA. Listing 1 shows the transformed (MATLAB-based) pseudocode for timesteps of size $\frac{1}{n}$ and $T = t_n = 1$. The algorithm first computes n random numbers, then computes $W(t_i)$ conditional on $W(t_l)$ and $W(t_r)$ for all i and finally computes the stock prices $S(t_i)$. $W(t_l)$ and $W(t_r)$ are the known surrounding points of $W(t_i)$.

Gaussian numbers are generated in line 2. These numbers are converted to the correct conditional distribution by multiplying them with the coefficients $c_i = \sqrt{\frac{1}{4}(t_r - t_l)}$ (line 4). These coefficients only depend on the order in which the data points are generated and can be precomputed. Finally, the random numbers are multiplied with σ as in Equation 7 (lines 8 and 9) and stored in the first row of matrix W.

At this point, $W(t_0)$ and $W(t_n)$ are known. On each iteration of the outer loop, all matrix elements in the middle of two known elements are calculated by the inner loop in line 16 and stored in the next row of W. All other elements are copied to the next row (line 20). Thus, one row of W is completely filled in each iteration of the outer loop. When the loops finish, $S(t_i)$ is found with Equation 7 (line 25).

Figure 7 shows how to move this algorithm into hardware for $n = 4$. The RNG provides a stream of n Gaussian numbers. These numbers are multiplied with the precomputed coefficients (stored in registers) and σ, as in lines 4, 8 and 9 of Listing 1. Next, $W(t_{n/2})$ is calculated conditional on $W(t_0)$ and $W(t_n)$ using an ADA (add-divide-add). ADAs are the hardware implementation of line 16 of Listing 1. They add the nearest known left (L in the figure) and right (R) data point, divide them by 2 and add the random number

```
1  n = 2^m
2  Generate Z(1:n) ~ N(0, I)
3  c = precomputed_coefficients
4  cond_random_numbers = c .* Z(1:n-1)
5
6  W = empty_matrix(m+1, n+1)
7  W(1,1) = 0
8  W(1, 2:n) = σ * cond_random_numbers
9  W(1, n+1) = σ * Z(n)
10
11 for k = 1:m
12   for j = 1:n
13         if (modulo(j, n/2^(k-1))==n/2^k)
14           l = -2^(m-k)
15           r = 2^(m-k)
16           W(k+1, j+1) = W(k, j+1)+
17                         W(k, j+1+l)/2+
18                         W(k, j+1+r))/2
19         else
20           W(k+1, j+1) = W(k, j+1)
21         end
22   end
23 end
24 drift = (r - σ^2/2) * [1/n : 1/n : 1]
25 S = S(t_0) * exp(W(m+1, :) + drift)
26 return(S)
```

Listing 1: Transformed algorithm to generate a Brownian bridge with n equally sized timesteps from $t_0 = 0$ to $t_n = 1$.

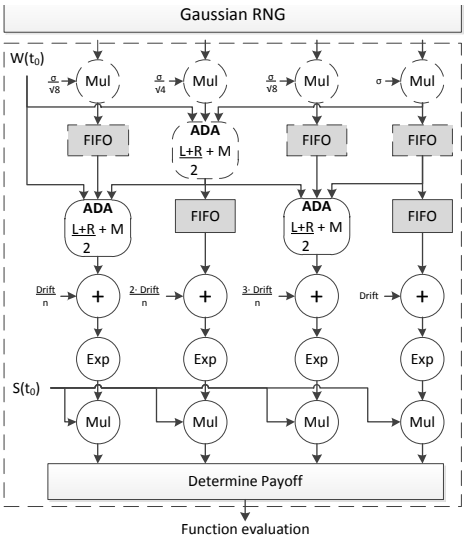


Figure 7: Integrand with Brownian bridge in hardware for n = 4.

(M). One ADA is found in each column. All other numbers are stored in FIFOs until they are required for an operation. The FIFOs are the hardware representation of line 20.

All[4] $W(t_i)$ are known after addition of the correct drift (line 25). The actual asset prices can be found with Equation 7. From the asset price, the payoff can be determined with a hardware implementation of Equation 6.

Some optimizations may be implemented to decrease the resource usage. These optimizations include:

1) Reducing the number of FIFOs
2) Making the ADAs more efficient
3) Reducing the number of multipliers
4) Reducing the number of adders

Figure 8 shows the improved stucture.

[4]The hardware has to be slightly restructured when the number of timesteps *n* is not a power of two.

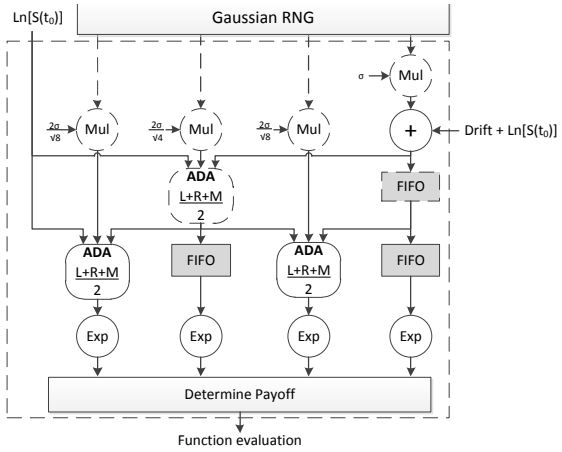


Figure 8: Optimized Brownian bridge design for n = 4.

The number of FIFOs can be reduced (1) by moving the random number generators and multipliers closer to the ADAs. The ADAs can be implemented more efficiently (2) when three floating-point numbers are added first and the result is divided by two. To allow this, the random numbers need to be multiplied by two since they are divided by two as well. Multiplying all exponentiated values (3) with $S(t_0)$ can be replaced by adding $\ln(S(t_0))$ to the exponents. We set the left-most data point equal to $\ln(S(t_0))$ instead of zero and add $\ln(S(t_0))$ to the final data point $W(t_n)$ like Figure 8 shows. Due to the ADAs, $\ln(S(t_0))$ will be added to all $W(t_i)$ automatically. Finally, addition (4) of the drift after generating all data points can be eliminated similarly. The drift can simply be added to the final data point right after the random number is generated. The ADAs make sure the drift is added to all data points automatically.

VII. Results

We implement the design on the Convey HC2-ex platform [17]. The Convey HC2-ex has four Xilinx Virtex-6 LX760 FPGAs and two Intel Xeon X5670 6-core processors running at 2,93 GHz. However, this work only uses one of the FPGAs. The CPU could send Tasks to any FPGA, but we did not implement this yet in the interest of time. The FPGA resource utilization for the Asian option is shown in Table I. The design runs at 150 MHz. Floating-point operators were generated with the FloPoCo generator [18].

As a reference, we use the MISER implementation from the GNU Scientific Library [19]. The software is compiled with ICC 10.1 and executed on an Intel i7-4770 CPU running at 3,40 GHz.

We compare three different methods to compute the value of an Asian option: (i) naive Monte-Carlo integration in software, (ii) MISER integration in software and (iii) MISER integration in hardware (this work). The parameters from Table II define the Asian option. These parameters are

	Design	Available	Percentage
Flip Flop	343,176	948,480	36.2%
LUT	319,563	474,240	67.4%
DSP	649	864	75.1%
RAMB36E1	192	720	26.7%
RAMB18E1	113	1,440	7.8%

Table I: FPGA utilization on a Virtex-6 LX760.

$\mathbf{S(t_0)}$	**K**	**r**	$\boldsymbol{\sigma}$	**T**	**n**
50	55	0.1	0.25	1	64

Table II: Parameters of the Asian option.

economically justified, but many other sets of parameters could have been used.

To analyze the performance of the three methods, we vary the number of simulations *N* from 1000 to 150 million. We measure the execution time and the accuracy of the result. We configure MISER to use 5 percent of the available number of simulations to evaluate the integration domain for segmentation. Segmentation stops when less then 900 simulations or less then 10 percent of the original amount of simulations is left for a Task. The minimal amount of simulations that is used to integrate or segment an integration domain is 100 simulations. In the following tests, the software and hardware programs are run 15 times.

Figure 9 demonstrates the results of these integrations. Figure 9(a) shows that the graphs converge to the same option value when the sample count is increased. Convergence is faster for MISER software and hardware compared to naive Monte-Carlo software. Figure 9(b) shows the advantage of the hardware implementation over the software implementation[5]. The hardware is already converging in the same time it takes the software to perform a Monte-Carlo integration with 1000 simulations. When the same amount of simulations is used, the hardware design is up to 880 times faster compared to the MISER software reference.

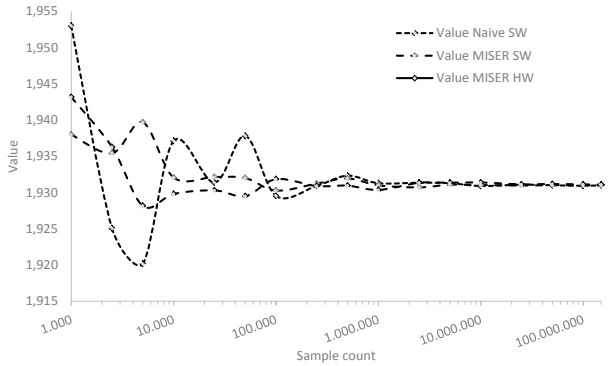
Figure 9(c) shows the root mean squared error[6] (RMSE) in percentages of the option price that was found with Monte-Carlo integration. The error decreases when the sample count increases. The RMSE is always smaller for the designs that use MISER's stratified sampling. The small difference in RMSE between the software and hardware version of MISER is caused by Monte-Carlo noise and the difference in estimating the variance of each segment. Figure 9(d) gives Pareto curves that show the trade-off between the execution time and the error. The MISER hardware is able to limit the error to 0.01% in only one second of execution time. The MISER software requires 860 seconds to reach this accuracy. The naive Monte-Carlo software is unable to reach such an accuracy with $1.5 \cdot 10^8$ simulations, which is the maximum sample count we used. Between 0.001 and 0.01 seconds, different errors occur for the same execution time. The communication overhead is the main cause of this pattern. We explain this pattern in depth in [20].

The standard deviation of the Monte-Carlo integration indicates the variability of the result. If there is no proxy for the 'true' option value[7], this is the only piece of information available to determine the significance of the integration result. Figure 9(e) shows the standard deviation achieved by

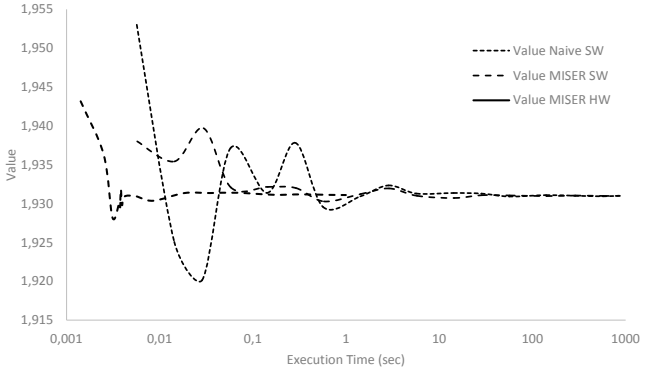
[5]Notice the logarithmic scale of the horizontal axes. The graphs of MISER HW stop at 1 second, since the FPGA requires only 1 second for 150 million simulations. No tests have been performed for larger *N*.

[6]The error is the difference between the option price found by the Monte-Carlo integration and the 'true' option price. Since the 'true' option price is unknown, we use the result of naive MCI with $N = 1,5 \cdot 10^{10}$ as a proxy for the 'true' value.

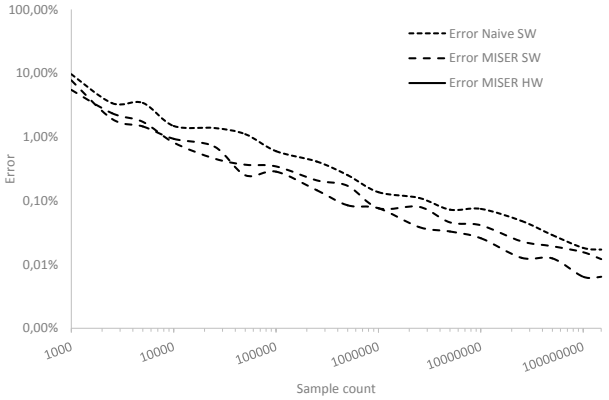
[7]Which is usually the case, otherwise the integration would be pointless.



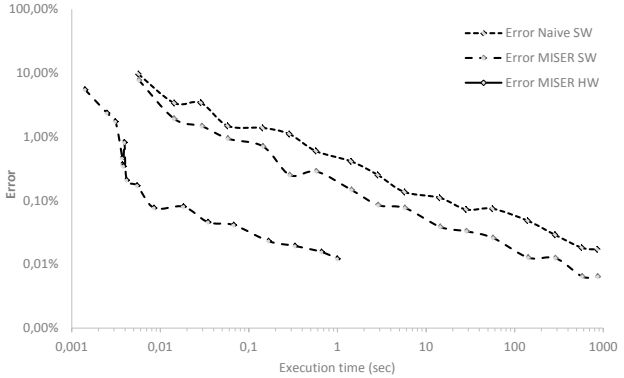
(a) Price of an Asian option using Monte-Carlo integration.



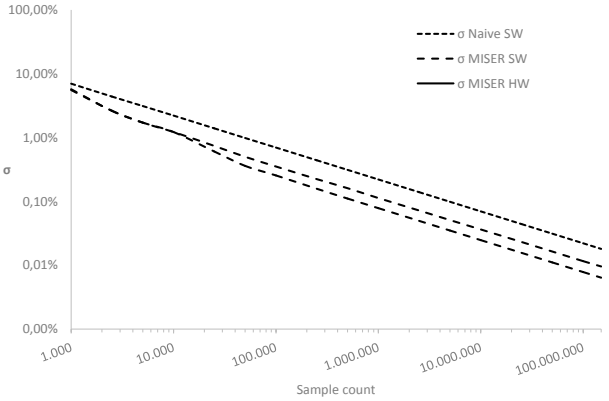
(b) Price of an Asian option using Monte-Carlo integration.



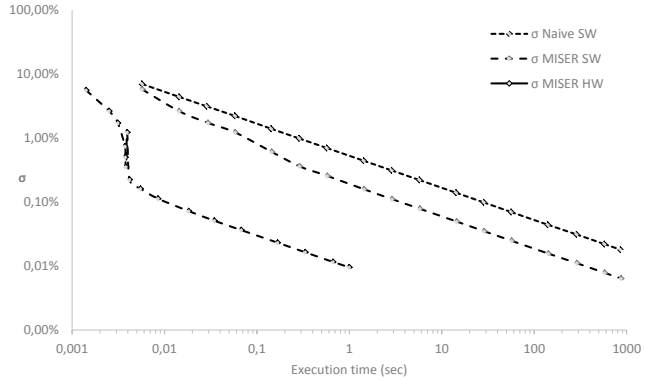
(c) RMSE of the Monte-Carlo integration for an Asian option, given as a percentage of the integration result.



(d) RMSE of the Monte-Carlo integration for an Asian option, given as a percentage of the integration result.



(e) Standard deviation of the Monte-Carlo integration for an Asian option, given as a percentage of the integration result.



(f) Standard deviation of the Monte-Carlo integration for an Asian option, given as a percentage of the integration result.

Figure 9: Results of Monte-Carlo integrations to price an Asian option. The number of simulations N *is varied from 1000 to 150.000.000 simulations.*

σ	Execution time naive MC SW (sec)	Execution time MISER HW (sec)	Effective speed-up
0,1%	28,09	0,0092	3069,48
0,05%	112,36	0,0353	3179,23
0,01%	2809,00	0,8143	3449,43
0,005%	11236,00	3,1449	3572,77

Table III: Effective speed-up of the hardware design over a naive Monte-Carlo software reference.

the software and hardware implementations. For all designs, the standard deviation decreases slowly with the sample count. The MISER software performs slightly better than the MISER hardware, due to the different method to estimate the variance of each segment. Figure 9(f) gives Pareto curves that show the trade-off between the sample count and the standard deviation. From the graphs can be seen that the option price determined by the hardware has a standard deviation smaller than 0.1% within 10 milliseconds.

We use Figure 9(f) to determine the required execution time to reach a certain level of accuracy. Table III shows these execution times for the naive MC software and the MISER hardware. We define the effective speed-up as the ratio of these execution times. The table shows an effective speed-up of 3069 to 3572 for high levels of accuracy. This is the combined effect of the hardware acceleration and the stratified sampling. Furthermore, the speed-up increases for a higher accuracy. This occurs because the sample count is higher for these accuracy levels. As the sample count increases, the communication between the CPU and FPGA can be performed more and more in parallel with the integration on the FPGA.

Taking the size of the FPGA and the multithreading of the software reference into account, we estimate the hardware acceleration of our design to be comparable to the Control Variate (CV) Monte-Carlo design for Asian option pricing in [3]. The main advantage of our work over the design in [3] is the different range of applications. Like stratified sampling, CV is a variance reduction method. Asian option pricing is an application that benefits from both methods. Our work and the work in [3] demonstrate that both methods can be very powerful in combination with FPGAs.

VIII. Conclusion and Future Work

We demonstrate how Monte-Carlo integration can be performed on a heterogeneous multicore platform with CPUs and FPGAs while using the general purpose algorithm MISER to apply stratified sampling and reduce the variance of the Monte-Carlo estimate. The parallel architecture for MISER can be applied to various integrands. We demonstrate the advantages of our design by pricing an Asian call option. We implement a Brownian bridge rather than a Brownian motion to make optimal use of the stratification.

By implementing stratified sampling in the hardware design, an accuracy that requires minutes with software programs can be achieved in a fraction of a second. Pricing an Asian option using the FPGA-accelerated design is up to 880 times faster than a software implementation using MISER from the GSL library running on an Intel i7-4770 CPU at 3,40 GHz. Furthermore, our design requires up to 3572 times less execution time to achieve the same accuracy as a software implementation without stratification.

Future work includes further research into the stratification performed by MISER. Our hardware uses a simplified estimator for the variance of each segment. The simplification works well for this integrand, but other integrands may require the sample standard deviation as an estimator. Furthermore, the Brownian bridge could be implemented to use fixed-point internally, in order to fit bridges of higher dimensions on the FPGA. Finally, the latency of the design can be minimized for smaller integrations, increasing the speed-up for integrations with less than 100.000 simulations.

References

[1] D. Thomas, J. Bower, and W. Luk, "Hardware architectures for monte-carlo based financial simulations," in *Proc. FPT*, 2006, pp. 377 –380.

[2] N. Singla et al., "Financial monte carlo simulation on architecturally diverse systems," in *High Performance Computational Finance, Workshop on*. IEEE, 2008, pp. 1–7.

[3] A. Tse, D. Thomas, K. Tsoi, and W. Luk, "Reconfigurable control variate monte-carlo designs for pricing exotic options," in *Proc. FPL*. IEEE, 2010, pp. 364–367.

[4] X. Tian and K. Benkrid, "American option pricing on reconfigurable hardware using least-squares monte carlo method," in *Proc. FPT*. IEEE, 2009, pp. 263–270.

[5] N. Woods and T. VanCourt, "Fpga acceleration of quasi-monte carlo in finance," in *Proc. FPL*. IEEE, 2008, pp. 335–340.

[6] P. Echeverria, M. López-Vallejo, and J. M. Pesquero, "Variance reduction techniques for monte carlo simulations. a parameterizable fpga approach," in *Electronics, Circuits and Systems*. IEEE, 2008, pp. 1296–1299.

[7] R. Korn, E. Korn, and G. Kroisandt, *Monte Carlo Methods and Models in Finance and Insurance*, ser. CRC financial mathematics series. Chapman & Hall, 2010.

[8] W. H. Press and G. R. Farrar, "Recursive stratified sampling for multidimensional monte carlo integration," *Computers in Physics*, vol. 4, no. 2, pp. 190–195, 1990.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[10] J. Hull, *Options, Futures, & Other Derivatives*, 6th ed., ser. The Prentice Hall Series in Finance. Prentice Hall, 2006.

[11] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *The journal of political economy*, pp. 637–654, 1973.

[12] D. Thomas and W. Luk, "Fpga-optimised uniform random number generators using luts and shift registers," in *Proc. FPL*. IEEE, 2010, pp. 77–82.

[13] D. B. Thomas and W. Luk, "Resource efficient generators for the floating-point uniform and exponential distributions," in *Proc. ASAP*. IEEE, 2008, pp. 102–107.

[14] C. De Schryver et al., "A hardware efficient random number generator for nonuniform distributions with arbitrary precision," *International Journal of Reconfigurable Computing*, p. 12, 2012.

[15] D. Thomas, "Exploiting spatial parallelism to improve both speed and accuracy in financial computing," workshop Design Methods & Tools for FPGA-Based Acceleration of Scientific Computing, DATE 2011.

[16] P. Glasserman, *Monte Carlo methods in financial engineering*. Springer, 2004, vol. 53.

[17] Convey Computer Corp, "Convey computer hc series," 3 2014. [Online]. Available: http://www.conveycomputer.com/products/hcseries/

[18] F. De Dinechin and B. Pasca, "Designing custom arithmetic data paths with flopoco," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2011.

[19] M. Galassi et al., *GNU Scientific Library Reference Manual*, 3rd ed. Network Theory Ltd, 2009. [Online]. Available: http://www.gnu.org/software/gsl/

[20] M. de Jong, "Hardware acceleration of monte-carlo integration in finance," Master's thesis, Delft University of Technology, April 2014.