A Multi-point Distributed Random Variable Accelerator for Monte Carlo Simulation in Finance

Nicola Bruti Liberati,
Eckhard Platen,
School of Finance & Economics
and Department of Mathematical Sciences,
University of Technology, Sydney,
PO Box 123, Broadway, NSW 2007, Australia

Filippo Martini,
Massimo Piccardi,
Faculty of Information Technology,
University of Technology, Sydney,
PO Box 123, Broadway, NSW 2007, Australia

Abstract

The pricing and hedging of complex derivative securities via Monte Carlo simulations of stochastic differential equations constitutes an intensive computational task. To achieve "real time" execution, as often required by financial institutions, one needs highly efficient implementations of the multi-point distributed random variables underlying the simulations. In this paper a fast and flexible dedicated hardware solution is proposed. A comparative performance analysis demonstrates that the hardware solution is bottleneck-free and flexible, and significantly increases the computational efficiency of the software solution.

1. Introduction

In finance the evolution of security prices underlying derivative contracts is described by stochastic differential equations (SDEs). To price a derivative security it requires to estimate the expectation of a function, the payoff, of the solution of the underlying SDE at a given maturity time. A widespread method, the only feasible one when the underlying securities follow a high dimensional SDE, is the Monte Carlo method. As explained in [**?**], to price an option via Monte Carlo simulation one uses a weak approximation of the underlying SDE. Here only an approximation of the probability distribution is needed, while a pathwise strong approximation is relevant for other problems, such as scenario simulation or filtering. In a weak Taylor scheme, it is then possible to replace the Gaussian random variables approximating the increments of Wiener processes with much simpler multi-point distributed random variables that match certain moments. For instance, one can use two-point distributed random variables that match the first three moments of the Wiener process increments in a simplified Euler scheme. For a second order weak Taylor scheme one needs three-point distributed random variables. Such multi-point distributed random variables can be generated based on random bits, i.e. random variables with only the two possible values 0 and 1, each with probability 0.5.

The main reason for replacing the Gaussian random variables with variables based on random bits is simulation speed. Typical financial simulations can take hours or days to run even on powerful servers, thus making "real time" evaluation unfeasible. In [**?**] it was shown that random bit generators (RBGs) significantly increase the computational efficiency of simplified weak Taylor schemes. However, in many applications a further speedup is required. Therefore, in this paper we present a dedicated hardware solution, based on a Field Programmable Gate Array (FPGA), for the generation of random bits and the associated multi-point distributed random variables that is able to provide a further, significant speedup. The choice of an FPGA as a dedicated hardware solution is mainly due to its flexibility, allowing the user to program different RBGs according to the order of convergence of the weak Taylor scheme employed. Moreover, the "randomness" of the random bits, which is crucial for an effective Monte Carlo simulation, is strictly related to the order and the coefficients of the underlying polynomial [**?**]: on the FPGA, the user is free to program the most suitable polynomial for any given application.

For the purpose of performance analysis, we can identify three different levels where speedup measurement can be performed: a) the generation level, measuring the speedup in the generation of random numbers alone; b) the system level, accounting for additional terms such as the required transfer of the random numbers from the generator to other system units; eventually, c) the application level, measuring the overall speedup on the final application. In particular, in solutions where random numbers are generated by an FPGA and used by a host processor as in our case, system-

level performance must be carefully taken into account in order to prevent efficiency bottlenecks.

In [**?**], an FPGA implementation of random number generators (RNGs) was proposed, but mainly for cryptographical applications. Such applications carry the additional requirement that the generated random sequence should be hard to guess. Therefore, they cannot be based on the simplest and most efficient generators such as the linear shift register generator that we instead use in this work. In [**?**], FPGA implementations of various RNGs were presented which could be appropriately used in financial simulations. However, [**?**] presents no system-level performance analysis. In our work, instead, we propose a fast RBG implementation on an FPGA and we accurately describe its system performance in a PC architecture. The proposed approach proved able to achieve high system-level speedups (up to 10 times) with respect to an optimized software-only implementation for a large range of RBGs differing in polynomial order, number of non-null coefficients and scheme order. Moreover, we also proved a significant application-level speedup in a real application.

2. Weak Taylor Schemes

Although the results presented in this section can be extended to multi-dimensional SDEs with time dependent coefficients, let us consider for simplicity the SDE

$$dX_t = a(X_t)dt + b(X_t)dW_t \tag{1}$$

for $t \in [0,T]$, with $X_0 \in \mathbb{R}$, where $W = \{W_t,\, t \in [0,T]\}$ is a standard Wiener process. We address here the problem of computing the expected value of a payoff function $g(X_T)$ of the solution of the SDE (1) at a final time T, as needed, for instance, to obtain the price of a derivative security.

Let us construct a discrete time approximation $Y^\Delta = \{Y_t^\Delta,\, t \in [0,T]\}$ of the solution $X = \{X_t,\, t \in [0,T]\}$ of (1) on an equidistant time discretisation $0 = t_0 < t_1 < \ldots < t_N = T$, where $t_n = \{n\,\Delta,\, n = 0,\ldots,N\}$ and $\Delta = \frac{T}{N}$.

We say that a discrete time approximation Y^Δ *converges weakly* to X at time T with order γ if for each polynomial g there exists a positive constant K, which does not depend on Δ, and a $\Delta_0 > 0$ such that

$$\varepsilon(\Delta) = |E(g(X_T)) - E(g(Y_N^\Delta))| \le K\Delta^\gamma \tag{2}$$

for each $\Delta \in (0, \Delta_0)$.

The first method that we consider for the approximation is the well known *Euler scheme* given by

$$Y_{n+1} = Y_n + a(Y_n)\Delta + b(Y_n)\Delta W_n, \tag{3}$$

where $\Delta W_n = W_{t_{n+1}} - W_{t_n} = \sqrt{\Delta}\,\xi_n$ is the Gaussian increment of the Wiener process W for $n \in \{0,1,2\ldots,N-1\}$, with $\xi_n \sim \mathcal{N}(0,1)$ and $Y_0 = X_0$. The Euler method (3) achieves an order of weak convergence $\gamma = 1.0$. As explained in [**?**], it is possible to replace the Gaussian random variables ΔW_n by two-point distributed random variables $\Delta\widehat{W}_n^2$, where

$$P(\Delta\widehat{W}_n^2 = \pm\sqrt{\Delta}) = \frac{1}{2}, \tag{4}$$

yielding the *simplified Euler scheme*

$$Y_{n+1} = Y_n + a(Y_n)\Delta + b(Y_n)\Delta\widehat{W}_n^2. \tag{5}$$

Since the two-point distributed random variables $\Delta\widehat{W}_n^2$ match the first three moments of the Gaussian random variables ΔW_n, the simplified Euler scheme (5) still achieves an order of weak convergence $\gamma = 1.0$.

When high accuracy is required, it is important to be able to construct approximations with higher orders of weak convergence. As shown in [**?**], if we add more terms from the Wagner-Platen expansion to the Euler scheme (3), then we obtain the *order* 2.0 *weak Taylor scheme*

$$\begin{aligned} Y_{n+1} &= Y_n + a\Delta + b\Delta W_n + \frac{1}{2}b'b\left\{(\Delta W_n)^2 - \Delta\right\} \\ &\quad + \frac{1}{2}\left(aa' + \frac{1}{2}a''b^2\right)\Delta^2 + a'b\,\Delta Z_n \\ &\quad + \left(ab' + \frac{1}{2}b''b^2\right)\{\Delta W_n\,\Delta - \Delta Z_n\}, \end{aligned} \tag{6}$$

where ΔZ_n represents the double Itô integral

$$\Delta Z_n = \int_{t_n}^{t_{n+1}} \int_{t_n}^{s_2} dW_{s_1} ds_2.$$

For the sake of simplicity, in equation (6) and in the following we suppress in the notation the dependence of the coefficients on the numerical approximation Y_n, that means $a = a(Y_n)$ and $b = b(Y_n)$ for $n \in \{0,1,2\ldots,N-1\}$. In this case we can replace the Gaussian random variables ΔW_n and ΔZ_n with expressions involving the three-point distributed random variables $\Delta\widehat{W}_n^3$, with

$$P(\Delta\widehat{W}_n^3 = \pm\sqrt{3\Delta}) = \frac{1}{6}, \qquad P(\Delta\widehat{W}_n^3 = 0) = \frac{2}{3}, \tag{7}$$

and we obtain the *second order simplified method*

$$\begin{aligned} Y_{n+1} &= Y_n + a\Delta + b\Delta\widehat{W}_n^3 + \frac{1}{2}bb'\left\{\left(\Delta\widehat{W}_n^3\right)^2 - \Delta\right\} \\ &\quad + \frac{1}{2}\left(aa' + \frac{1}{2}a''b^2\right)\Delta^2 \\ &\quad + \frac{1}{2}\left(a'b + ab' + \frac{1}{2}b''b^2\right)\Delta\widehat{W}_n^3\Delta. \end{aligned} \tag{8}$$

Since the multi-point distributed random variables appearing in scheme (8) match the first five moments of those appearing in (6), method (8) still achieves an order of weak convergence $\gamma = 2.0$.

For any given weak Taylor scheme it is possible to replace the Gaussian random variables with simpler multi-point distributed ones. As explained in [**?**], if the multi-point distributed random variables match the first $2\gamma + 1$ moments of the Gaussian random variables in a weak Taylor scheme of order γ, then the resulting simplified Taylor scheme achieves the same order of weak convergence γ. For higher order schemes, however, it might be more efficient to generate a Gaussian random variable rather than a multi-point distributed one that matches the first $2\gamma+1$ moments based on random bits. For this reason, we consider here up to the order 3.0 weak Taylor scheme.
In the *order* 3.0 *weak Taylor scheme*, detailed in [**?**], the required multi-point distributed random variables need to match the first seven moments of the Gaussian ones. In [**?**] a corresponding four-point distributed random variable $\Delta\widehat{W}_n^4$ was proposed, where

$$P(\Delta\widehat{W}_n^4 = \pm\sqrt{3+\sqrt{6}}\,\sqrt{\Delta}) = \frac{1}{12+4\sqrt{6}},$$
$$P(\Delta\widehat{W}_n^4 = \pm\sqrt{3-\sqrt{6}}\,\sqrt{\Delta}) = \frac{1}{12-4\sqrt{6}}. \quad (9)$$

However, such a four-point distributed random variable (9) cannot be efficiently implemented with the method based on random bit generation described below. Instead, we present here a five-point distributed random variable $\Delta\widehat{W}_n^5$, with

$$P(\Delta\widehat{W}_n^5 = \pm\sqrt{6\Delta}) = \frac{1}{30}, \quad P(\Delta\widehat{W}_n^5 = 0) = \frac{1}{3},$$
$$P(\Delta\widehat{W}_n^5 = \pm\sqrt{\Delta}) = \frac{9}{30}, \quad (10)$$

that still matches the first seven moments and is suitable for a highly efficient implementation.

The multi-point distributed random variables and the corresponding RBGs to be presented can be applied to any weak scheme, including for instance derivative free and implicit schemes. Moreover, many other applied sciences such as economics, insurance, physics, population dynamics, epidemiology, structural mechanics, chemistry and biotechnology employ models often specified via SDEs and thus can greatly benefit from the above methods.

3. Multi-point Distributed Random Variables Based on Random Bit Generators

In [**?**] a highly efficient software implementation of simplified schemes based on RBGs has been proposed. The two-point distributed random variables $\Delta\widehat{W}_n^2$ that constitute the core of the simplified Euler scheme (5) can be efficiently obtained with a single RBG, which is an algorithm that generates a bit 0 or 1 with probability 0.5. Random bits can be obtained via a linear shift register generator. This generator, used in digital communications [**?**], relies on the theory of primitive polynomials modulo 2. These are special polynomials of the form

$$y(x) = 1 + c_1 x + \ldots + c_{n-1}x^{n-1} + x^n, \quad (11)$$

with coefficients $c_i \in \{0, 1\}$. A primitive polynomial modulo 2 of order n defines a recurrence relation for obtaining a new bit from the n preceding ones with maximal period, which is $2^n - 1$. The recurrence is given by:

$$a_{n+1} = c_1 a_n \oplus c_2 a_{n-1} \oplus \ldots \oplus c_{n-1} a_2 \oplus a_1, \quad (12)$$

where a_{n+1} is the new bit obtained from the $\{a_i, i \in (n, \ldots, 1)\}$ preceding ones and $\oplus$ is the "exclusive or" operator. Thus, RBGs can be efficiently implemented in C via bitwise operations, see [**?**] and [**?**]. In [**?**] a Monte Carlo simulation of an option pricing problem using the simplified first order scheme (5) with the RBG (12), provided a speed up of about 28 times when compared to a first order scheme based on Gaussian random variables.

For a first order simplified scheme, each bit obtained from the RBG is used to generate a value for the two-point distributed random variable by a simple look-up operation ($0 \to \sqrt{\Delta}, 1 \to -\sqrt{\Delta}$). For a second order simplified scheme (8), one bit is not sufficient to generate a value for the required three-point distributed random variable, $\Delta\widehat{W}_n^3$. However, a sequence of three generated random bits is first used to generate 8 equiprobable values. Then, with an acceptance-rejection method we discard 2 of them, use 4 to generate the 0 value for the random variable and use 1 each for values $+\sqrt{3\Delta}$ and $-\sqrt{3\Delta}$. For the third order simplified scheme, the random variable is five-point distributed with the probability distribution (10). In this case, a sequence of five random bits is used to generate 32 equiprobable combinations. The acceptance-rejection method discards 2 of them, uses 10 to generate the 0 value, 9 each for values $+\sqrt{\Delta}$ and $-\sqrt{\Delta}$, and 1 each for values $+\sqrt{6\Delta}$ and $-\sqrt{6\Delta}$.

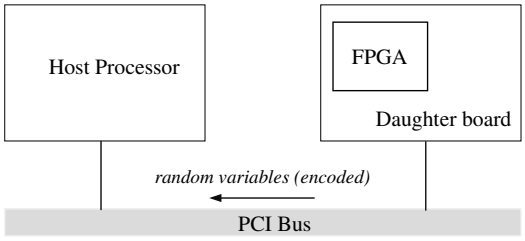

Figure 1. The system architecture

Given the variety of schemes and the compelling need for fast generation, we have decided to implement the multi-point distributed Random Variable Generator (RVG) on a high-performance FPGA, the Altera Stratix

EP1S1OB672C6. By using an FPGA, the user can obtain high speeds and easily adapt the generator to the changing application requirements. Fig. 1 shows our proposed system architecture for a personal computer platform. The FPGA device is the core of a PC daughter card connected to the host processor via the PCI bus. We can describe the system's operations as divided in three phases: in phase 1, the FPGA generates a new set of random values (in a T_{FPGA} average time per value). In phase 2, the FPGA transfers such a set in a compact, encoded format to the host memory via burst bus cycles for maximum communication efficiency (in a T_{comm} average time per value). In phase 3, the host processor is ready to serve the requests for random values from the simulation software. At each request, the host processor decodes one encoded value and returns it to the caller (in a T_{dec} average time per value). In this way, the simulation software sees the system through the same function interface of a conventional software-only implementation and requires no further modifications. More importantly, we coordinate these phases so that the simulation software uses the current set of generated random values while the FPGA concurrently produces a new set (phase 1), thus obtaining a significant speedup. The complete time models for the simulation are given in the following. In the general case, we can write:

$$T_{\text{exe}} = T_{\text{use}} + T_{\text{gen}}, \tag{13}$$

where T_{gen} is the average time spent for generating a multi-point distributed random value, T_{use} is the average time spent by the rest of the simulation software in using it and their sum, T_{exe}, is the average total execution time per value.

In the case of a conventional software implementation $T_{\text{gen}} = T_{\text{gen}_{\text{SW}}}$ is the time taken by the execution of a function that generates and returns one multi-point distributed random value to the caller. In order to provide the most unbiased performance comparison, we have implemented such a function as a highly speed-optimised C macro.

With our system, instead the simulation software uses the current set of random values while the FPGA concurrently generates a new set. In this way, if the generation time on the FPGA, T_{FPGA}, is shorter than the use time, T_{use}, the former does not add up to the total execution time. This constraint was largely satisfied in all our experiments. Hence, $T_{\text{gen}} = T_{\text{gen}_{\text{HW}}}$ can be expressed as:

$$T_{\text{gen}_{\text{HW}}} = T_{\text{comm}} + T_{\text{dec}}, \text{ if } \quad T_{\text{FPGA}} < T_{\text{use}}. \tag{14}$$

4. Experimental Results and Performance Analysis

In order to provide a comparative performance analysis, we have implemented both software and hardware versions of the RVG for a comprehensive variety of parameters. In particular, the three dimensions of the polynomial order, number of non-null coefficients, and order of weak Taylor scheme are considered. All software components have been compiled with the MS Visual C++ 6.0 compiler unless otherwise stated and time measurements performed on a Pentium 4 Mobile 2.0 GHz personal computer.

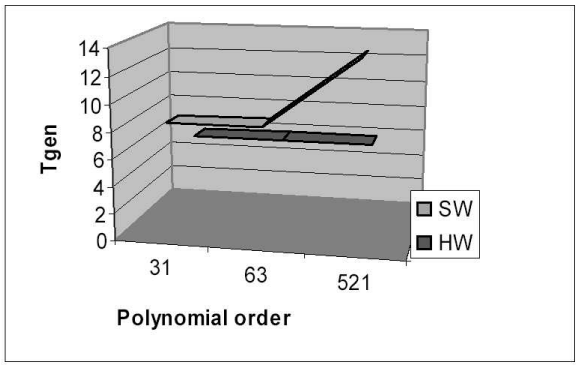

Figure 2. The generation time as a function of the polynomial order

4.1. Variable polynomial order

For a primitive polynomial modulo 2, a polynomial order n guarantees a period of $2^n - 1$ for the generated random sequence. It is known that the accuracy of a simulation based on a pseudo-random sequence is compromised when the sequence length is substantial when compared with the period of the RNG. In the light of this, high order polynomials should be preferred. However, in a software implementation one faces an increase in generation time when using high order polynomials, since they cannot be mapped onto a single primitive-type operand. Instead, the hardware implementation does not suffer from any predefined operand size. Fig. 2 shows the generation time T_{gen}, in nanoseconds, for the software and hardware implementations as a function of the polynomial order. T_{gen} grows steadily for a software implementation as it requires multi-operand operations. Instead, the time for hardware remains constant. In our tests, even larger polynomial sizes proved not to introduce any further delay in FPGA performance.

4.2. Variable number of non-null coefficients

The "randomness" of the random bits, which is crucial for an effective Monte Carlo simulation, is strictly related not only to the order of the generating polynomial but also to the choice of its (non-null) coefficients [**?**]. However, in a software implementation a programmer is tempted to use

polynomials with few non-null coefficients, as each introduces an additional computational load. Fig. 3 shows that the software implementation suffers from a proportional delay. Again, the time instead remains constant for our hardware implementation as T_{FPGA} remains less than the T_{use} in all cases of practical interest.

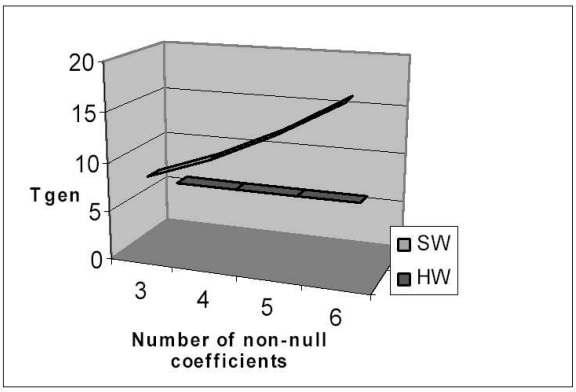

Figure 3. The generation time as a function of the number of non-null coefficients (for a polynomial order of 31)

4.3. Variable order of the weak Taylor scheme

When high accuracy is required, higher order of the weak Taylor schemes will eventually increase the computational efficiency, even though both the scheme and the multi-point distributed random variables are more complex. In any case, speeding up the computation of the random variables has a dramatic impact on the simulation time. Fig. 4 shows the generation time, T_{gen}, in nanoseconds, for the software and hardware implementations as a function of the order of the weak Taylor scheme. Once again, the software time grows steadily, up to 80 ns per value in a third order scheme. The hardware time, instead, increases negligibly. Actually, the increase in T_{gen} is due only to the larger size of the encoded random values (which grows as $\lceil \log_2 n \rceil$ with the n number of points of the multi-point distributed variable) which has an impact on the transfer time, T_{comm}, and the decoding time, T_{dec}.

Table 1 reports the speedup of the proposed hardware solutions with respect to the optimised software implementation for both the generation time, T_{gen}, and the total simulation time, T_{exe}, for the pricing of a European call option with the underlying security following a geometric brownian motion, see [**?**]. For the generation time, the speedup is simply computed as $S_{\text{gen}} = T_{\text{gen}_{\text{SW}}}/T_{\text{gen}_{\text{HW}}}$. For the execution time, the speedup is instead computed as $S_{\text{exe}} = (T_{\text{use}} + T_{\text{gen}_{\text{SW}}})/(T_{\text{use}} + T_{\text{gen}_{\text{HW}}})$. We recall here that T_{use} accounts for the simulation software other than the generation of the random values and is the same in both cases. Moreover, for this application, T_{use} is always several times greater than T_{FPGA}, proving that the time model in (14) easily holds in such a real case. Table 1 shows that not only is the generation time improved, but the overall application strongly benefits from the hardware acceleration. This is due to the large percentage of the total execution time typically spent on the random value generation by the software. Moreover, the speedup increases with the order of the polynomial (and the number of its non-null coefficients – not shown in table), thus becoming even more significant in the case of high accuracy simulations.

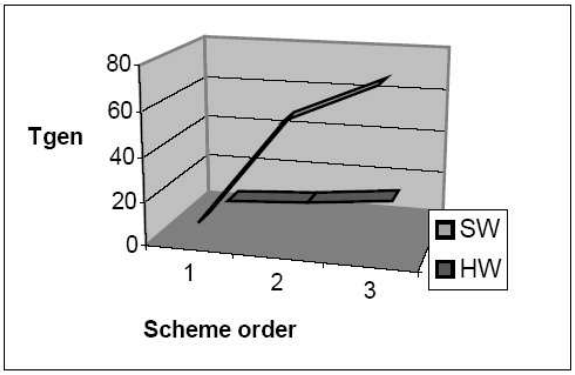

Figure 4. The generation time as a function of the scheme order (for a polynomial order of 31)

Table 2 reports comparative results for a conventional software implementation based on Gaussian variables and the proposed hardware implementation. The first column of Table 2 shows the T_{use} values for the aforementioned application. The next two columns report the generation times for the software implementation based on Gaussian variables and the proposed hardware implementation for a period of $2^{31} - 1$. As Gaussian random number generator we use the routine *gasdev*, see [**?**]. Times are reported in nanoseconds per random value and T_{use} and $T_{\text{gen}_{\text{Gauss}}}$ have been measured by using the Mingw port of the GCC compiler for convenience. Eventually, the last column reports the corresponding speedup on the execution time computed as $S_{\text{exe}} = (T_{\text{use}} + T_{\text{gen}_{\text{Gauss}}})/(T_{\text{use}} + T_{\text{gen}_{\text{HW}}})$. Such an application-level speedup proves very high, ranging between 8 and 26 for different scheme orders. For the Gaussian-based implementation, the generation time of a random number is independent of the scheme order, while for the proposed hardware implementation based on multi-point distributed random variables the generation time of a random number grows with the scheme order. However, Table 2 shows

	$S_{gen}(31)$	$S_{exe}(31)$	$S_{gen}(521)$	$S_{exe}(521)$
Order 1	1.43	1.11	2.32	1.32
Order 2	6.84	2.26	9.94	2.93
Order 3	5.88	1.76	8.01	2.09

Table 1. The speedup between hardware and software solutions

	T_{use}	$T_{\mathrm{gen}_{\mathrm{Gauss}}}$	$T_{\mathrm{gen}_{\mathrm{HW}}}$	S_{exe}
Order 1	14.00	509.62	5.85	26.38
Order 2	47.00	509.62	8.40	10.05
Order 3	57.00	509.62	12.57	8.13

Table 2. Execution times and speedup for a conventional software implementation based on Gaussian variables and the proposed hardware solution

that the generation time for the hardware implementation is generally much lower and grows only slowly with the scheme order, thus retaining a significant speedup even for the higher orders.

5. Conclusions

In this paper we have presented a dedicated hardware solution on an FPGA for the generation of multi-point distributed random variables for use in a PC environment. The proposed solution uses a high-performance FPGA for fast and flexible generation of the random bits and the associated multi-point distributed random numbers. Moreover, the random numbers are transferred to the host processor in an encoded format with PCI burst cycles for maximum communication efficiency. Thanks to this bottleneck-free system design, the obtained speedups in the generation of multi-point distributed random variables range from 1.4 up to about 10 times, with the higher speedups corresponding to higher polynomial orders. We also report relevant speedups of up to 2.41 in the total simulation time for an application in finance. It is important to note that other applied sciences (such as economics, insurance, physics, population dynamics, epidemiology, chemistry and biotechnology) whose models are often specified via SDEs and solved by Monte Carlo simulations, can benefit greatly from the proposed hardware solution.

References

[1] N. Bruti-Liberati and E. Platen. On the efficiency of simplified weak Taylor schemes for Monte Carlo simulation in finance. In *Computational Science - ICCS 2004*, volume 3039, pages 771–778. Springer, 2004.

[2] S. W. Golomb. *Digital Communications with Space Applications*. Prentice-Hall, Englewood Cliffs, N.J., 1964.

[3] N. Hofmann. *Beiträge zur schwachen Approximation stochastischer Differentialgleichungen*. PhD thesis, Dissertation A, Humboldt Universität Berlin, 1994.

[4] P. E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*, volume 23. Springer, 1999. Third printing.

[5] P. E. Kloeden, E. Platen, and H. Schurz. *Numerical Solution of SDE's Through Computer Experiments*. Universitext. Springer, 2003. Third printing.

[6] P. Martin. An analysis of random number generators for a hardware implementation of genetic programming using FPGAs and Handel-C. Technical Report CSM-358, University of Essex, 2002.

[7] H. Niederreiter. *Random Number Generation and Quasi-Monte-Carlo Methods*. SIAM, Philadelphia, PA, 1992.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++. The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 2002.

[9] K. H. Tsoi, K. H. Leung, and P. H. W. Leong. Compact FPGA-based true and pseudo random number generators. In *Proc. of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM03)*, pages 1–13, 2003.