A Hardware Generator of Multi-point Distributed Random Numbers for Monte Carlo Simulation

Nicola Bruti-Liberati [a], Filippo Martini [b], Massimo Piccardi [b] and Eckhard Platen [a]

[a] *School of Finance & Economics and Department of Mathematical Sciences*

[b] *Faculty of Information Technology,*

University of Technology, Sydney, PO Box 123, Broadway, NSW 2007, Australia

Abstract

Monte Carlo simulation of weak approximations of stochastic differential equations constitutes an intensive computational task. In applications such as finance, for instance, to achieve "real time" execution, as often required, one needs highly efficient implementations of the multi-point distributed random number generator underlying the simulations. In this paper a fast and flexible dedicated hardware solution on a field programmable gate array is presented. A comparative performance analysis between a software-only and the proposed hardware solution demonstrates that the hardware solution is bottleneck-free, retains the flexibility of the software solution and significantly increases the computational efficiency. Moreover, simulations in applications such as economics, insurance, physics, population dynamics, epidemiology, structural mechanics, chemistry and biotechnology can benefit from the obtained speedups.

Key words: random number generators, random bit generators, hardware implementation, field programmable gate arrays (FPGAs), Monte Carlo simulation, weak Taylor schemes, multi-point distributed random variables.
PACS: 02.50.Ng, 89.65.Gh; MSC2000: 65C10, 65C30.

1 Introduction

In many applied sciences the dynamics of key quantities can be described by *stochastic differential equations* (SDEs). In finance, for instance, the evolution of security prices underlying derivative contracts is described by SDEs. To price a derivative security it is necessary to estimate the expectation of a

function, the payoff, of the solution of the underlying SDE at a given maturity date. A widespread method and the only feasible one when the underlying securities follow a high dimensional SDE, is the Monte Carlo method. As explained in [5], 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.

Another important application of Monte Carlo simulation of weak approximations of SDEs is the solution of multi-dimensional *partial differential equations* (PDEs). In mathematical physics, for instance, multi-dimensional PDEs arise in many models. A feasible approach to their solution is the Monte Carlo simulation via their probabilistic representation, see [8].

In a Monte Carlo simulation of a weak Taylor scheme, it is 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, in a simplified Euler scheme one can use two-point distributed random variables that match the first three moments of the Wiener process increments. 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 [1] 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 efficient generation of random bits and the associated multi-point distributed random numbers. 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 [9]. On the FPGA, the user is free to program the most suitable polynomial for any given application.

This paper provides a twofold contribution to the literature, namely in the areas of Monte Carlo simulation of weak Taylor schemes and random number generation. In the following, we address the main works related to both areas. In [1] simplified weak Taylor schemes up to weak order two based on RBGs

are proposed and a study on the efficiency of a related software implementation is reported. [3] provides a four-point distributed random variable for a simplified weak Taylor scheme of weak order three. However, such a four-point distributed random variable cannot be efficiently implemented via RBGs. Instead, in this paper we propose a five-point distributed random variable suitable for an efficient implementation of a simplified weak Taylor scheme of weak order three based on RBGs. In finalizing this paper the authors learned that the idea of using software-based RBGs for the simulation of simplified weak Taylor schemes and the proposal of a five-point distributed random variable have been independently suggested in [8]. However, no analysis of the computational efficiency nor implementation design are presented in [8], whereas they are the objectives of this paper.

Random number generators (RNGs) can be divided into the two categories of true and pseudo random number generators. The former are based on "true" random physical phenomena while the latter are based on deterministic numerical algorithms. The main advantage of true RNGs is that the generated random numbers are independent and thus impossible to predict. This proves to be a crucial factor in applications such as encryption. On the other hand, the non reproducibility of the generated sequence makes it difficult to asses its statistical properties [9]. Moreover, the generation speed is unlikely to rival that of the fastest pseudo RNGs. In [6] an FPGA true RNG is proposed based on the jitter of a clock signal. The generation speed reported of 0.5 Mbit/s is far less than that achievable by a pseudo RNG. For these reasons, in Monte Carlo simulations pseudo RNGs are usually preferred to true RNGs.

In a software implementation on a modern personal computer (PC), the generation of a random number from a pseudo RNG can take as little time as a few nanoseconds. A dedicated hardware solution can certainly provide faster generation, but requires careful, bottleneck-free system design to prove really useful for the overall application. In [10], a hardware design for a pseudo RNG is presented but based on outdated discrete logic. In [12], an FPGA implementation of pseudo RNGs is proposed but specifically for cryptographical applications. In [7], detailed results are presented for FPGA implementations of various pseudo RNGs which could also be used in Monte Carlo simulations. However, none of the above papers presents system-level integration or discusses system-level performance. In the current paper, instead, we propose a fast generator of multi-point distributed random numbers on an FPGA and describe its system performance in a PC architecture. The proposed approach has been tested over a wide variety of parameters, including different multi-point random variables and corresponding weak Taylor schemes, proving capable of achieving speedups of up to ten times with respect to an optimized software-only implementation.

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. Such expectation is 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 $t_n = \{n\,\Delta,\, n \in \{0, \ldots, N\}\}$ 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

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

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 [4], 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 [4], 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*

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

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 the dependence of the coefficients on the numerical approximation Y_n from the notation, meaning, for instance, we write a for $a(Y_n)$, where $n \in \{0, 1, 2 \ldots, N-1\}$. In this case we can replace the Gaussian random variables ΔW_n and ΔZ_n by expressions involving the three-point distributed random variables $\Delta\widehat{W}_n^3$, where

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

to obtain the *second order simplified method*

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

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

By adding more terms from the Wagner-Platen expansion and approximating the arising multiple stochastic integrals with Gaussian random variables, we obtain the following *order* 3.0 *weak scheme* given by

$$\begin{aligned} Y_{n+1} = & Y_n + a\Delta + b\Delta W_n + \frac{1}{2}L^1b\left\{(\Delta W_n)^2 - \Delta\right\} \\ & + L^1a\Delta Z_n + \frac{1}{2}L^0a\Delta^2 + L^0b\left\{\Delta W_n\,\Delta - \Delta Z_n\right\} \\ & + \frac{1}{6}\left(L^0L^0b + L^0L^1a + L^1L^0a\right)\left\{\Delta W_n\,\Delta^2\right\} \\ & + \frac{1}{6}\left(L^1L^1a + L^1L^0b + L^0L^1b\right)\left\{(\Delta W_n)^2 - \Delta\right\}\Delta \\ & + \frac{1}{6}L^0L^0a\,\Delta^3 + \frac{1}{6}L^1L^1b\,\left\{(\Delta W_n)^2 - 3\,\Delta\right\}\Delta W_n, \end{aligned} \tag{9}$$

where L^0 and L^1 are differential operators defined by

$$L^0 = a\,\frac{\partial}{\partial x} + \frac{1}{2}b^2\,\frac{\partial^2}{\partial x^2} \quad \text{and} \quad L^1 = b\,\frac{\partial}{\partial x}. \tag{10}$$

This scheme achieves an order of weak convergence $\gamma = 3.0$.

To construct a third order simplified method, the required multi-point distributed random variables need to match, in general, the first seven moments of the Gaussian ones. In [3] a corresponding four-point distributed random variable $\Delta\widehat{W}_n^4$ was proposed, where

$$\begin{aligned} P(\Delta\widehat{W}_n^4 = \pm\sqrt{3+\sqrt{6}}\,\sqrt{\Delta}) &= \frac{1}{12+4\sqrt{6}}, \\ P(\Delta\widehat{W}_n^4 = \pm\sqrt{3-\sqrt{6}}\,\sqrt{\Delta}) &= \frac{1}{12-4\sqrt{6}}. \end{aligned} \tag{11}$$

However, the four-point distributed random variable appearing in (11) cannot be efficiently implemented by the method based on random bit generation described below because the probability values in (11) are not rational numbers. Instead, we present here a five-point distributed random variable $\Delta\widehat{W}_n^5$, with

$$\begin{aligned} P(\Delta\widehat{W}_n^5 = \pm\sqrt{6\Delta}) = \frac{1}{30}, \qquad & P(\Delta\widehat{W}_n^5 = \pm\sqrt{\Delta}) = \frac{9}{30}, \\ & P(\Delta\widehat{W}_n^5 = 0) = \frac{1}{3}, \end{aligned} \tag{12}$$

that still matches the first seven moments and is suitable for a highly efficient implementation based on RBGs. Therefore, we can present the *third order simplified method*

$$\begin{aligned} Y_{n+1} = Y_n &+ a\Delta + b\Delta\widehat{W}_n^5 + \frac{1}{2}L^1 b\left\{\left(\Delta\widehat{W}_n^5\right)^2 - \Delta\right\} + \frac{1}{2}L^0 a\Delta^2 \\ &+ \frac{1}{2}L^1 a\left\{\Delta\widehat{W}_n^5 + \frac{1}{\sqrt{3}}\Delta\widehat{W}_n^2\right\}\Delta + \frac{1}{2}L^0 b\left\{\Delta\widehat{W}_n^5 - \frac{1}{\sqrt{3}}\Delta\widehat{W}_n^2\right\}\Delta \\ &+ \frac{1}{6}\left(L^0L^0b + L^0L^1a + L^1L^0a\right)\Delta\widehat{W}_n^5\,\Delta^2 \\ &+ \frac{1}{6}\left(L^1L^1a + L^1L^0b + L^0L^1b\right)\left\{(\Delta\widehat{W}_n^5)^2 - \Delta\right\}\Delta \\ &+ \frac{1}{6}L^0L^0a\Delta^3 + \frac{1}{6}L^1L^1b\left\{(\Delta\widehat{W}_n^5)^2 - 3\,\Delta\right\}\Delta\widehat{W}_n^5, \end{aligned} \tag{13}$$

that involves the multi-point distributed random variables $\Delta\widehat{W}_n^5$ and $\Delta\widehat{W}_n^2$ and achieves an order of weak convergence $\gamma = 3.0$, see [3].

For any given weak Taylor scheme it is possible to replace the Gaussian random variables by simpler multi-point distributed ones. As explained in [4], if the multi-point distributed random variables match the first $2\gamma+1$ moments of the multiple stochastic integrals in a weak Taylor scheme of order γ, then the resulting simplified Taylor scheme achieves the same order γ of weak convergence. For higher order schemes, however, the complexity of multi-point distributed random variables that match the required $2\gamma+1$ moments increases. For this reason, we consider here up to third order simplified methods, with (13) being the only third order scheme implemented.

Furthermore, the multi-point distributed random variables and the corresponding RBGs to be presented can be applied to any weak scheme, including derivative free and implicit schemes. For instance, the two-point distributed random variables $\Delta\widehat{W}_n^2$ could be used in the *first order simplified predictor-corrector method* with corrector

$$Y_{n+1} = Y_n + \frac{1}{2}\{a(\bar{Y}_{n+1}) + a\}\Delta + b\Delta\widehat{W}_n^2, \tag{14}$$

and predictor

$$\bar{Y}_{n+1} = Y_n + a\Delta + b\Delta\widehat{W}_n^2. \tag{15}$$

The simplified predictor-corrector scheme (14)-(15) achieves an order of weak convergence $\gamma = 1.0$ and shows, in general, better numerical stability properties than the Euler scheme (5).

Moreover, simulations in economics, insurance, physics, population dynamics, epidemiology, structural mechanics, chemistry and biotechnology for models specified via SDEs can greatly benefit from the above methods.

3 Multi-point Random Variables and Random Bit Generators

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

$$y(x) = 1 + c_1 x + \ldots + c_{n-1}x^{n-1} + x^n, \tag{16}$$

with coefficients $c_i = \{0, 1\}$. 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:

$$a_{k+1} = c_1 a_k + c_2 a_{k-1} + \ldots + c_{n-1} a_{k-n+2} + a_{k-n+1} \, (mod \, 2), \tag{17}$$

where a_{k+1} is the new bit obtained from the preceding ones, a_i, with $i \in \{n, \ldots, 1\}$ and $k > n$. Equation (17) can be rewritten as

$$a_{k+1} = c_1 a_k \oplus c_2 a_{k-1} \oplus \ldots \oplus c_{n-1} a_{k-n+2} \oplus a_{k-n+1}, \tag{18}$$

where $\oplus$ is the "exclusive or" operator. Thus, RBGs can be efficiently implemented in C via bitwise operations, see [11] and [1]. In [1] a Monte Carlo simulation of an option pricing problem using the simplified first order scheme (5) with the RBG (18), 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 (5), 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 ($0 \rightarrow +\sqrt{\Delta}, 1 \rightarrow -\sqrt{\Delta}$). 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 obtain eight equiprobable values. Then, with an acceptance-rejection method we discard two of them, use four to generate the 0 value for the random variable and use one each for values $+\sqrt{3\Delta}$ and $-\sqrt{3\Delta}$. For the third order simplified scheme (13), the random variable is five-point distributed with the probability distribution (12). In this case, a sequence of five random bits is used to generate 32 equiprobable combinations. The acceptance-rejection method discards two of them, uses ten to generate the 0 value, nine each for values $+\sqrt{\Delta}$ and $-\sqrt{\Delta}$, and one each for values $+\sqrt{6\Delta}$ and $-\sqrt{6\Delta}$.

4 System Architecture and FPGA Implementation

4.1 System Architecture

Our ultimate goal is to substantially speed up the above described Monte-Carlo simulations by moving the random number generation from software to a dedicated hardware platform. More precisely, we aim to move the whole generation of multi-point distributed random numbers from the host processor to a dedicated hardware unit. Since the percentage of time typically taken by the generation of such numbers can be as high as 60-70% of the total execution time [1], the above appears to be a promising approach. However, there exist critical performance challenges at the system level. As the typical generation time for a software implementation can be as short as a few nanoseconds

per number, the dedicated hardware solution must avoid any system-level bottlenecks to prove competitive.

Fig. 1. The system architecture.

Our basic idea for a PC environment is that of implementing the hardware "accelerator" as a daughter board on the PCI (peripheral component interconnect) bus, Revision 2.2 [13]. The daughter board hosts the RNG, not to be confused with the RBG which is just a part of it. The RNG is implemented on an FPGA and returns the generated numbers to the simulation software through the PCI bus. Fig. 1 shows our proposed system architecture for a PC platform. We can divide the system's operations in four phases. In phase 1, the FPGA generates a set of random numbers (in a T_{FPGA} average time per number). In phase 2, the FPGA transfers such a set in a compact, combinatorially encoded format to the host memory via burst bus cycles operated under DMA (direct memory access) for maximum communication efficiency (in a T_{comm} average time per number). The combinatorial encoding works as follows. Any given multi-point distributed random variable has a small finite set of n possible values, with each value typically represented as a 32-bit floating point datum. We can encode each value by combinatorial encoding with $\lceil \log_2 n \rceil$ bits, where $\lceil a \rceil$ denotes the smallest integer greater than or equal to a. Accordingly, the amount of random numbers that we are able to pack and transfer in a single PCI data phase (32 bits of data over 30 ns) is much larger than that possible with the native floating point representation (only one number per data phase). In phase 3, the host processor is ready to serve the requests for random numbers from the simulation software. At each request, the host processor decodes one encoded number and returns it to the caller (in a T_{dec} average time per number). In phase 4, the host processor uses the random numbers (in a T_{use} average time per number). 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 into a pipeline so that the simulation software uses the current set of generated random numbers (phase 4) while the FPGA concurrently produces a new set (phase 1), thus obtaining a significant speedup. Fig. 2 shows how the various phases occur with respect to time. It can be seen that if T_{FPGA} is less than T_{use}, then phase 1 is completely hidden by phase 4 and thus adds no time to the total execution time. With a more aggressive implementation, also phases 2 and 3 could have been considered for pipelining with other phases. In particular, phase 2 could be overlapped with phase 1 by means of a double-buffer implementation on the FPGA. At its turn, phase 3 could be overlapped with phases 1 and 2 by a double-buffer implementation in the host memory. Note that phase 3 cannot overlap with phase 4 as they both require the same resource, the host processor. It can be shown that such changes could result in hiding T_{comm} completely in the overall execution time. On the other hand, T_{dec} will instead increase due to the

increased complexity of a multiple-buffer implementation, thus compromising the speedup. For this reason, we decided to limit pipelining to the two main phases, 1 and 4.

Fig. 2. The various phases with respect to time.

The complete time models for the simulation are given in the following. First, we can define T_{gen} as the average time spent for generating a multi-point distributed random number and T_{use} as the average time spent by the rest of the simulation software in using it. If generation and use are sequential, we can write:

$$T_{\text{exe}} = T_{\text{use}} + T_{\text{gen}}, \tag{19}$$

where T_{exe} is the average total execution time per number.

In the case of a conventional software implementation the above model holds and $T_{\text{gen}} = T_{\text{gen}_{\text{SW}}}$ is the time taken by the execution of a function that generates and returns one multi-point distributed random number to the caller.

With our system, instead, the simulation software uses the current set of random numbers 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. Such a constraint was largely satisfied in all our experiments. Hence, (19) still holds with $T_{\text{gen}} = T_{\text{gen}_{\text{HW}}}$ given by:

$$T_{\text{gen}_{\text{HW}}} = T_{\text{comm}} + T_{\text{dec}}, \text{ if } \quad T_{\text{FPGA}} < T_{\text{use}}. \tag{20}$$

4.2 FPGA Implementation

A fast and flexible implementation of the RNG is the main requirement in this application. FPGAs enjoy several features such as quasi-ASIC (application specific integrated circuit) speed and programmer-level flexibility, which makes them the most suitable option for the hardware platform. Accordingly, we have chosen to implement our generator on a high-performance FPGA, the Altera Stratix EP1S1OB672C6. Simulation tools for this device are available in the Altera Quartus II development environment. We have used the Web Edition Software Version 4.2 of such tools. Moreover, all the circuits for the FPGA have been developed in VHDL (Very High Speed Integrated Circuit Hardware Description Language).

Fig. 3. A simplified scheme of the random number generator.

Fig. 3 shows a simplified scheme of the RNG. In Fig. 3.(a) the RNG is shown together with the output FIFO (first in first out) queue (some signals have been omitted for simplicity). The generation of the encoded random numbers is synchronous with the main clock signal (CK), with one number generated per clock cycle over signals RN[0:2]. Each encoded random number generated by the RNG is input in the FIFO queue which, in turn, allows for asynchronous reading from an external master with 32-bit data parallelism. The writing on the queue is clocked by the BUFF_WR signal, which is synchronous with CK. However, the queue can suspend the random number generation when full by raising FIFO_FULL. The generator needs an initial seed of arbitrary length which can be uploaded asynchronously, in one or more steps, through the SEED[0:31] and WR signals. Fig. 3.(b) shows details of the generator. The generation of the random bits is performed by a shift register generator of programmable length equal to that of the generating polynomial. The "active" (i.e. non-null) coefficients can also be programmed by the user. The orders considered for the weak Taylor scheme range from $\gamma = 1$ to $\gamma = 3$, although higher orders can also be straightforwardly implemented. When the selected order is $\gamma = 1$, the RNG generates numbers sampled from a two-point distributed random variable with the probabilities described in (4). Each single bit generated by the shift register generator represents a valid encoded number. When the selected order is $\gamma = 2$, the RNG generates numbers sampled from a three-point distributed random variable with the probability distribution (7). In this case, a sequence of three generated random bits, X1:X3, is used to generate eight equiprobable combinations. As described in Section 3, the accept/group logic discards two of them, uses four to generate a 0 value for the random variable and uses one each for values $+\sqrt{3\Delta}$ and $-\sqrt{3\Delta}$. When the selected order is $\gamma = 3$, the random variable is five-point distributed with the probability distribution (12). In this case, a sequence of five random bits, X1:X5, is used to generate 32 equiprobable combinations. The accept/group logic discards two of them, uses ten to generate a 0 value, nine each for values $+\sqrt{\Delta}$ and $-\sqrt{\Delta}$, and one each for values $+\sqrt{6\Delta}$ and $-\sqrt{6\Delta}$. All combinatorial functions in the accept/group logic are optimised.

5 Experimental results and performance analysis

Table 1 shows the main performance results of the proposed implementation for a polynomial order of 31 and the different weak Taylor scheme orders. F_{ck} (in MHz) is the maximal clock frequency obtained for the RNG. T_{FPGA}, the time (in ns) for generating one multi-point distributed random number, is computed as $\alpha * 10^3/F_{\text{ck}}$. The α term accounts for the fact that some of

Table 1
Performance results of the proposed implementation, polynomial order 31.

Scheme Order	$F_{\mathtt{ck}}$	T_{FPGA}	T_{use}	T_{comm}	T_{dec}
$\gamma = 1$	162	6.17	18.06	2	3.85
$\gamma = 2$	158	8.42	30.44	4	4.4
$\gamma = 3$	116	9.12	68.60	6.4	6.27

the clock cycles generate a random number that should be rejected; such a factor is 8/6 for the three-point distributed random variable and 32/30 for the five-point distributed one. T_{use}, the time spent by the application in using a random number, is measured on the option pricing problem reported in [1]. From Table 1, it is possible to see that the constraint of (20) is always easily satisfied. Although T_{use} obviously depends on the application, its range will be similar for comparable Monte Carlo simulations. T_{comm} is the average time for transferring one random number from the FIFO queue to the host memory over the PCI bus. This time increases proportionally to the size in bits of the encoded random numbers. Moreover, in some cases the data require extra-alignment bits to match the 32-bit PCI data size. For instance, this applies to the case of the 3-bit encoded numbers sampled from a five-point distributed random variable. In Table 1, T_{comm} is computed based on a transfer rate of 66 MB/s. However, there exist several implementations over the PCI bus which can almost saturate its peak rate of 132 MB/s; hence, even smaller values for T_{comm} are achievable. Moreover, the upcoming PCI Express$^{\text{TM}}$bus carries the potential to further decrease T_{comm} by at least a factor of 4. Based on these parameters and thanks to our design choice of combinatorial encoding for the generated random numbers, we have proved herewith that data communication is not a performance bottleneck in our system. Moreover, we have implemented highly-optimised C macros to perform the decoding operation on the host side, thus also limiting T_{dec}, the average time that the host processor takes to decode one encoded random number and return it to the requesting application.

Table 2 shows the main performance results for a much higher polynomial order of 521. T_{use}, T_{comm}, and T_{dec} are not influenced by the polynomial order. It can also be seen that the FPGA performance does not suffer from the increased polynomial length and in some cases even slightly exceeds that of the polynomial order 31. As the implementation still uses a very small fraction of the FPGA resources, we cannot see any practical upper bound on the choice of the polynomial length.

To provide a comparative analysis between software and hardware performance, we have implemented both software and hardware versions of the RNG for a comprehensive variety of parameters. In order for the performance comparison to be unbiased, we have implemented all software functions as highly speed-optimised C macros. The reference PC is a Mobile Pentium 4 2.0 GHz

Table 2
Performance results of the proposed implementation, polynomial order 521.

Scheme Order	$F_{\mathtt{ck}}$	T_{FPGA}	T_{use}	T_{comm}	T_{dec}
$\gamma = 1$	167	5.99	18.06	2	3.85
$\gamma = 2$	157	8.47	30.44	4	4.4
$\gamma = 3$	135	7.90	68.60	6.4	6.27

Fig. 4. The generation time as a function of the polynomial order.

Fig. 5. The generation time as a function of the number of non-null coefficients for a polynomial order of 31.

Fig. 6. The generation time as a function of the number of points of the random variable for a polynomial order of 31.

Fig. 7. The generation time as a function of the number of points of the random variable for a polynomial order of 521.

and the C compiler used is the Microsoft Visual Studio 6.0 with -O2 optimizations. In the following, the three dimensions of the polynomial order, number of non-null coefficients, and number of points of the random variable are discussed.

5.1 Polynomial order

A polynomial order n, for a primitive polynomial modulo 2, 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 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. 4 shows the generation time, T_{gen}, for the software and hardware implementations as a function of the polynomial order. For the software implementation, T_{gen} remains approximately stable up to 63 and then starts to grow with the polynomial order. Yet, the time for the hardware implementation always remains constant. In our tests, even larger polynomial sizes did not introduce any further delay in the FPGA operations.

5.2 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 [9]. However, in a software implementation a programmer is tempted to use the polynomial with the smallest number of coefficients, as each introduces an additional computational load. Fig. 5 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 T_{use} in all cases of interest.

5.3 Multi-point random variables

When high accuracy is required, higher orders 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. 6 shows the generation time, T_{gen}, for the software and hardware implementations as a function of the number of points in the multi-point distributed random variable, which refers to $\Delta\widehat{W}_n^2$, $\Delta\widehat{W}_n^3$ and $\Delta\widehat{W}_n^5$, defined in (4),(7) and (12), for a polynomial order of 31. Once again, the software time grows steadily, up to 80 ns per value for a five-point distributed random variable. The hardware time, instead, increases negligibly. Actually, the increase in T_{gen} is due only to the larger size of the encoded random numbers. The size of the encoded random numbers grows as $[\log_2 n]$, where n is the number of points representing the possible values of the multi-point distributed variable, and this has an impact on the transfer time, T_{comm}, and the decoding time, T_{dec}, see Tables 1 and 2. Fig. 7 shows that the trend is similar for a polynomial order of 521.

5.4 Speedup

Table 3 reports the speedups achieved with the proposed hardware solutions with respect to the optimised software implementation, when considering the multi-point distributed random variables $\Delta\widehat{W}_n^2$, $\Delta\widehat{W}_n^3$ and $\Delta\widehat{W}_n^5$ and polynomial orders of 31 and 521. $S_{\mathrm{gen}} = T_{\mathrm{gen_{SW}}}/T_{\mathrm{gen_{HW}}}$ is the speedup between the generation in hardware and that in software. As explained in (20), $T_{\mathrm{gen_{HW}}}$ does not account for the generation time on the FPGA device, but it consists, rather, of communication and decoding times. The units responsible for such times are mainly the PCI bus and the host processor. While S_{gen} is the main performance figure in our system, it is important to report also

Table 3
The generation speedup between hardware and software.

	$S_{\texttt{gen}}(31)$	$S_{\texttt{FPGA}}(31)$	$S_{\texttt{gen}}(521)$	$S_{\texttt{FPGA}}(521)$
$\Delta\widehat{W}_n^2$	1.44	1.36	2.33	2.27
$\Delta\widehat{W}_n^3$	6.84	6.83	9.94	9.86
$\Delta\widehat{W}_n^5$	5.88	8.18	8.01	12.84

Table 4
The application speedup between hardware and software.

Scheme Order	$S_{\texttt{exe}}(31)$	$S_{\texttt{exe}}(521)$
$\gamma = 1$	1.11	1.32
$\gamma = 2$	2.26	2.93
$\gamma = 3$	1.76	2.09

$S_{\text{FPGA}} = T_{\text{gen}_{\text{SW}}}/T_{\text{FPGA}}$, which is the speedup between the generation on the FPGA alone and that in software. This speedup is important to express the relative performance of the FPGA device and the host processor in the generation of multi-point distributed random variables in view of a possible transfer of the whole simulation to FPGAs. Table 3 shows that such a speedup is as high as 12.84 and could possibly increase by using FPGA development tools providing further optimizations.

Table 4 reports the application speedup of the proposed hardware solutions with respect to the optimised software implementation $S_{\text{exe}} = T_{\text{exe}_{\text{SW}}}/T_{\text{exe}_{\text{HW}}}$ when an option pricing problem reported in [1] is considered. Table 4 shows that the overall application strongly benefits from the hardware acceleration, up to almost three times in some cases. This is due to the large percentage of the total execution time typically spent on the random number generation by the software. Moreover, the speedup increases with the order of the polynomial and also with the number of its non-null coefficients, which is not shown in the table. Therefore, these speedups become more significant in the case of high accuracy simulations.

From Table 4, it appears that the application speedup for a weak Taylor scheme of order $\gamma = 2$ is greater than that for order $\gamma = 3$. However, such a result is not general since the measured times and speedups can depend significantly on the compiler used. To verify that, we measured T_{use} also with another compiler, the Mingw port of GCC. Here we obtained 14, 47 and 57 ns per random number for a weak Taylor scheme of order 1, 2, and 3, respectively. Such times, when compared to those obtained by the Microsoft compiler and reported in tables 1 and 2, seem to be in better proportion with the complexity of the operations in (5), (8) and (13). With such times, the application speedup for the weak Taylor scheme of order $\gamma = 2$ is equivalent to that of order $\gamma = 3$.

6 Conclusion

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 proposed solution achieves relevant speedups in generating the random numbers for the application when compared to an optimised software-only solution, ranging from 1.4 up to about 10 times. As the typical percentage of time employed by random number generation in Monte Carlo simulations is relevant, this speedup, in turn, provides a significant improvement on the efficiency of the overall Monte Carlo simulation. In this paper, we reported a case of a finance application where the speedup on the total simulation time reaches 2.93. Moreover, when high accuracy in the solution is required, thus involving higher order schemes and larger polynomial orders, the speedup on the total simulation time proves even greater. It is important to note that other applications such as in economics, insurance, physics, population dynamics, epidemiology, structural mechanics, chemistry and biotechnology, with models specified via SDEs and solved by Monte Carlo simulations, can benefit greatly from the proposed solution.

Acknowledgement

The authors wish to express their thanks to Hardy Hulley for valuable comments and suggestions in preparing this paper.

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*, ser. Lecture Notes in Comput. Sci. Springer, Vol. 3039, pp. 771–778, 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. Vol. 23 of Appl. Math.*, Springer, 1999. Third corrected printing.

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

[6] P. Kohlbrenner and K. Gaj. An embedded true random number generator for FPGAs. In *Proc. of the 2004 ACM/SIGDA Twelfth International Symposium on Field Programmable Gate Arrays (FPGA 2004)*, pages 71–78, 2004.

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

[8] G. N. Milstein and M. V. Tretyakov. *Stochastic Numerics for Mathematical Physics.* Springer, 2004.

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

[10] A. P. Paplinski and N. Batacharjee. Hardware implementation of the Lehmer random number generator. In *IEE Proc. Comput. Digit. Tech.*, Vol. 146, pages 93–95, 1996.

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

[12] 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.

[13] *PCI Local Bus Specification Revision 2.2.* PCI-SIG, 2000.