A Hardware Generator for Multi-point Distributed Random Variables

Filippo Martini*, Massimo Piccardi*, *Member, IEEE*, Nicola Bruti Liberati†, and Eckhard Platen†

*Faculty of Information Technology,

†School of Finance & Economics and Department of Mathematical Sciences,

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

***Abstract*— In Monte Carlo simulation of weak approximation of stochastic differential equations, multi-point distributed random variables play an important role. However, they need highly efficient implementations to meet the "real-time" requirements of applications such as the pricing of complex derivative securities. In this paper a fast and flexible dedicated hardware generator of multi-point distributed random variables on a Field Programmable Gate Array (FPGA) is presented. A comparative performance analysis demonstrates that the hardware solution is bottleneck-free, retains the flexibility of a traditional software implementation and significantly increases the computational efficiency of the overall simulation.**

I. Introduction

In finance, the dynamics of key quantities is described via stochastic differential equations (SDEs) [1]. When the required SDEs are high dimensional, the only feasible approach to their solution is the Monte Carlo method. In many cases of practical interest such as the pricing of a derivative security, it is possible to replace the Gaussian random variables used in Monte Carlo simulations with much simpler multi-point distributed random variables that match certain moments. For instance, for a first order weak Taylor scheme, one can use two-point distributed random variables. For a second order weak Taylor scheme, one needs three-point distributed random variables; and so on. 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 0.5 probability.

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 [2] it was shown that random bit generators (RBGs) significantly increase the computational efficiency of simplified weak Taylor schemes. However, the speedup achieved is still not sufficient in many cases. 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, which is able to provide a further, significant speedup. The choice of an FPGA as the 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 [3]: on the FPGA, the user is free to program the most suitable polynomial for any given application.

In simulation, random numbers are often generated from numerical algorithms rather than "true" random physical phenomena. On a modern PC, typical times are in the order of only tens of nanoseconds per number. As a consequence, speeding up the generation by a dedicated hardware solution requires not only a powerful random number generator (RNG), but also a careful, bottleneck-free system design. In [4], a hardware RNG is presented, but based on outdated discrete logic. In [5], an FPGA implementation is proposed, but mainly as a hardware core for cryptographical applications. In [6], detailed results are presented for FPGA implementation of various RNGs which could also be used in financial simulations. However, [6] presents no system-level performance analysis. In this work, instead, we propose a fast implementation on FPGA and we describe its system performance in a PC architecture. The proposed approach is able to achieve high speedups (up to 10 times) with respect to an optimized software-only implementation for a large range of generators differing in polynomial order, number of non-null coefficients and scheme order.

The rest of the paper is organized as follows: section 2 describes the multi-point distributed random variables and RBGs. Section 3 describes the FPGA implementation and the system architecture. Section 4 presents the comparative performance analysis. Finally, we present the conclusions.

II. Multi-point distributed random variables and random bit generators

In a Monte Carlo simulation, for any given weak Taylor scheme, it is possible to replace the Gaussian random variables with simpler multi-point distributed ones. As explained in [1], if in a weak Taylor scheme of order γ the multi-point distributed random variables match the first $2\gamma + 1$ moments of the Gaussian random variables, 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 based on random bits. For this reason, we consider here Taylor schemes with order of weak convergence up to $\gamma = 3.0$.

For the order $\gamma = 1.0$, the Gaussian random variables $\Delta W_n \sim n(0, \Delta)$, can be replaced by two-point distributed random variables $\Delta \widehat{W}_n^2$, where

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

with Δ the step size of the discretised time interval. However, when high accuracy is required, it is important to be able to construct approximations with higher orders of weak convergence. For $\gamma = 2.0$, ΔW_n can be replaced by three-point distributed random variables $\Delta \widehat{W}_n^3$, with

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

For the order $\gamma = 3.0$, the required multi-point distributed random variables need to match the first seven moments of the Gaussian ones. In [7], a four-point distributed random variable was proposed. However, such a random variable cannot be efficiently implemented based on RBGs. For this reason, we present here a five-point distributed random variable $\Delta\widehat{W}_n^5$, with

$$P(\Delta\widehat{W}_n^5 = \pm\sqrt{6\Delta}) = \frac{1}{30}, \quad P(\Delta\widehat{W}_n^5 = \pm\sqrt{\Delta}) = \frac{9}{30}, \\ P(\Delta\widehat{W}_n^5 = 0) = \frac{1}{3}, \tag{2}$$

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

As described in [2], the generation of multi-point distributed random variables can be efficiently performed based on RBGs. Random bits can be generated by a so-called shift register generator. This generator, long used for digital communications [8], relies on the theory of primitive polynomials modulo 2. These are special polynomials of the form

$$y(x) = 1 + c_1 x + \ldots + c_{n-1}x^{n-1} + x^n, \tag{3}$$

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:

$$a_{n+1} = c_1 a_n \oplus c_2 a_{n-1} \oplus \ldots \oplus c_{n-1} a_2 \oplus a_1, \tag{4}$$

where a_{n+1} is the new bit obtained from the $a_i, i = n, \ldots, 1$ preceding ones and $\oplus$ is the "exclusive or" operator. Thus, RBGs can be efficiently implemented in C via bitwise operations [2], [9]. In [2] a Monte Carlo simulation of an option pricing problem using a simplified first order scheme with the RBG (4) provided a speedup of about 28 times compared with a first order scheme using Gaussian random variables.

A value for a two-point distributed random variable $\Delta\widehat{W}_n^2$ can be directly obtained from a single random bit. Instead, a single bit is not sufficient to generate a value for the three-point distributed random variable $\Delta\widehat{W}_n^3$ required for a second order simplified scheme. However, with three calls to the RBG we obtain 8 equiprobable values. With an acceptance-rejection method we can then discard two of them and group the six remaining ones so as to obtain the three-point distributed random variable $\Delta\widehat{W}_n^3$ with the probabilities described in (1). For the third order simplified scheme, we have implemented the five-point distributed random variable (2), with rational probabilities that can be efficiently generated with five calls to the RBG (4).

Fig. 1. A simplified scheme of the multi-point distributed random variable generator

III. Random value generator and system architecture

A fast and flexible implementation of the multi-point distributed random variable generator (RVG hereafter) is the main requirement in this application. Accordingly, we have chosen to implement the RVG on a high-performance FPGA, the Altera Stratix EP1S1OB672C6. Simulation tools for this device are available in the Altera Quartus II development environment. By using an FPGA, the user can easily upgrade the generator according to the changing application requirements. All the circuits for the FPGA have been developed in VHDL (Very-High Definition Language).

Fig. 1.a shows the RVG together with the output FIFO queue (some signals have been omitted for simplicity). The generation of the random values is synchronous with the main clock signal (CK), with one value generated per clock cycle over signals RV[0:2]. Each random value RV[0:2] generated by the RVG 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 synchronised with CK. However, the queue can suspend the random value generation when full by raising FIFO_FULL. The generator needs an initial seed which can be uploaded asynchronously through the SEED[0:31] and WR signals. Fig. 1.b shows details of the generator. It is based on a shift register 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 1 to 3 (although higher orders are also straightforward to implement). When the selected order is 1, the generator must generate the values $+\sqrt{\Delta}$ and $-\sqrt{\Delta}$ for a two-point distributed random variable. Each single bit (0 or 1 with probability 0.5 each) generated by the RBG carries one such value. When the selected order is 2, the generator must produce values for a three-point distributed random variable with the probability distribution (1). In this case, a sequence of three generated random bits X1:X3 is used to generate 8 equiprobable combinations. The accept/group logic discards 2 of them, uses 4 to generate a 0 value for the random variable and uses 1 each for values $+\sqrt{3\Delta}$ and $-\sqrt{3\Delta}$. When the selected order is 3, the random variable is five-point distributed with the probability distribution (2). In this case, a sequence of five random bits X1:X5 is used to generate 32 equiprobable combinations. The accept/group logic discards 2 of them, uses 10 to generate a 0 value, 9 each for values $+\sqrt{\Delta}$ and $-\sqrt{\Delta}$, and 1 each for values $+\sqrt{6\Delta}$ and $-\sqrt{6\Delta}$. Typically, a value of a multi-point distributed variable is represented by a 32-bit floating point datum. In our implementation, instead, each value is encoded by combinatorial encoding (requiring up to 3 bits in the case of a five-point distributed variable) in order to limit the communication time with the host processor. This choice proved fundamental to achieve high performance also at the system level.

The FPGA device containing the RVG and the FIFO queue is designed to be interfaced with the PCI bus. The data contained in the FIFO queue are transferred to the host memory via burst PCI cycles for maximum communication efficiency. The host processor is responsible for requesting the data transfer, decoding the received data into the actual random values, and returning them *on-demand* to the simulation software. In this way, the rest of the simulation software requires no modifications with respect to a conventional software-only implementation. More importantly, the simulation software and the generation of the next set of random values operate concurrently. The time model for the simulation can be given as:

$$T_{\text{exe}} = T_{\text{use}} + T_{\text{gen}} \tag{5}$$

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 application in using it and T_{exe} is the average total execution time *per random value*. In the case of a conventional software implementation T_{gen} is the time taken by a function that computes and returns one multi-point distributed random value to the caller. For the sake of performance comparison, we have implemented such a function as a speed-optimised C macro on a P4 2 GHz PC. Instead, in our system T_{gen} is given by:

$$T_{\text{gen}} = T_{\text{comm}} + T_{\text{dec}}, \;\text{ if } T_{\text{FPGA}} < T_{\text{use}} \qquad (6)$$

where T_{FPGA} is the average time for generating an encoded random value on the FPGA generator, T_{comm} is its average transfer time over the PCI bus, and T_{dec} is its average decoding time on the host processor. Thanks to the concurrency between generation and use, (6) shows that T_{gen} reduces to the sum of T_{comm} and T_{dec} – provided the generation time remains less than the use time (this constraint was largely satisfied in all our experiments).

IV. 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.

A. 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} for the software and hardware

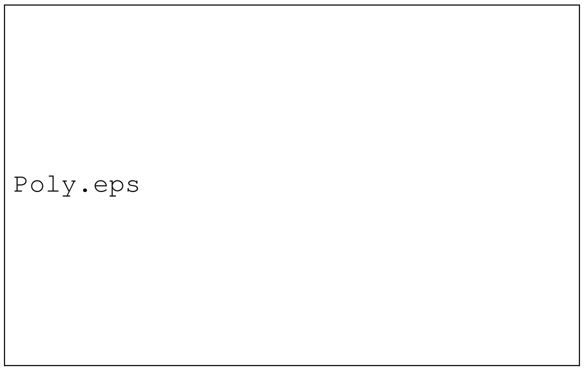

Fig. 2. The generation time as a function of the polynomial order

implementations as a function of the polynomial order. T_{gen} grows steadily for a software implementation as it requires multi-operand operations. Yet, the time for hardware remains constant. In our tests, even higher polynomial orders did not introduce any further delay in the FPGA operations.

B. 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 [3]. 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. 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 T_{use} in all cases of practical interest.

C. Variable order of the weak Taylor scheme

When high accuracy is required, higher orders of the weak Taylor schemes will eventually increase the computational efficiency, even though both the scheme

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

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

TABLE I

THE SPEEDUP BETWEEN HARDWARE AND SOFTWARE SOLUTIONS

	S_{gen} (31)	Sp_{exe} (31)	S_{gen} (521)	Sp_{exe} (521)
Order 1	1.43	16%	2.32	48%
Order 2	6.84	84%	9.94	128%
Order 3	5.88	98%	8.01	141%

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}, 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 decoding time, T_{dec}.

Table I reports the speedup of the proposed hardware solutions with respect to the optimised software implementation for both the generation time T_{gen} (as absolute speedup, $S_{\text{gen}} = T_{\text{gen}_{\text{SW}}}/T_{\text{gen}_{\text{HW}}}$), and the total simulation time T_{exe} (as percentage speedup, $Sp_{\text{exe}} = (T_{\text{exe}_{\text{SW}}}/T_{\text{exe}_{\text{HW}}}$ - 1) * 100), when an option pricing problem reported in [2] is considered. Table I 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 weak Taylor scheme and 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.

V. CONCLUSION

In this paper we have presented a dedicated hardware solution on an FPGA for the generation of multi-point distributed random variables (the first to date in the literature). With a careful, bottleneck-free system design, the proposed solution achieves relevant speedups, when compared to an optimised software-only implementation. These speedups range from 1.4 up to about 10 times in the generation of multi-point distributed random variables. We also reported relevant speedups up to 141% in total simulation time when an application in finance was considered. It is important to note that other applied sciences (such as economics, insurance, physics, population dynamics, epidemiology, structural mechanics, 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] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*. Springer, 1999, vol. 23, third printing.

[2] 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*. Springer, 2004, vol. 3039, pp. 771–778.

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

[4] A. P. Paplinski and N. Batacharjee, "Hardware implementation of the Lehmer random number generator," in *IEE Proc. Comput. Digit. Tech.*, vol. 146, no. 1, 1996, pp. 93–95.

[5] 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)*, 2003, pp. 1–13.

[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] N. Hofmann, "Beiträge zur schwachen Approximation stochastischer Differentialgleichungen," Ph.D. dissertation, Dissertation A, Humboldt Universität Berlin, 1994.

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

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