

A Parallel Framework for Long-period Random Number Generators in Hardware

Ishaan L. Dalal and Deian Stefan

Department of Electrical Engineering, The Cooper Union, New York, NY 10003, U.S.A.

ABSTRACT

The accuracy of stochastic (Monte Carlo) simulations is critically dependent on the quality of their random number generator (RNG). Recently, such simulations are increasingly implemented in a parallel form on field-programmable gate arrays (FPGAs) for higher performance. Fast, high-quality RNGs with periods long enough for extended simulation (e.g., the Mersenne Twister) have been well-proven in software, but hardware implementations are rare.

In this M.Sc. project, we develop an optimized framework for parallelizing long-period RNGs and implementing them on FPGAs. Both the underlying RNG algorithm as well as FPGA architectural aspects are exploited to provide a flexible trade-off between resource usage (area) and throughput. We also demonstrate three specific RNG implementations that are almost an order of magnitude faster than previous attempts and can significantly accelerate hardware Monte Carlo simulations.

From an educational perspective, the project is an interesting blend of usually disjoint areas such as computer architecture, abstract algebra and applied probability. Some aspects of the project are currently being used in an undergraduate ‘system design’ course to introduce students with only basic knowledge of digital logic design and linear algebra to these advanced concepts via experimentation.

1. INTRODUCTION

Parallelized Monte Carlo simulations are increasingly implemented on field-programmable gate arrays (FPGAs), which are reconfigurable logic devices; FPGA-based simulations exist in fields as diverse as cellular biochemistry, finance and photon transport. Such parallel simulations require fast, high statistical-quality, hardware-based RNGs to maintain both speed and accuracy; there are numerous instances of simulations whose *random number generators* (RNGs) were later discovered to be statistically ‘bad’ [1] and, therefore, their results invalidated.

The RNGs used for simulation are deterministic algorithms whose output is statistically indistinguishable from a true RNG. They are usually implemented as a recurrence equation, and their output sequence eventually repeats (the length is called the *period*). A good RNG algorithm must have a long period, an efficient realization and be portable.

In this M.Sc. project, we focus on RNGs defined by a linear recurrence over the finite (Galois) field with elements $\{0, 1\}$, i.e. in binary. All arithmetic for these $GF(2)$ -linear RNGs can be performed with elementary bitwise operations, making them highly suitable for hardware implementation. This focus on $GF(2)$ -linear RNGs is because they have long periods ($> 2^{1024} - 1$), well-proven statistical properties and are widely used for software simulations; however, efficient hardware implementations of such *long-period* RNGs are almost non-existent. To solve this problem and accelerate hardware simulations, we develop parallelized architectures for *long-period* RNGs and demonstrate specific implementations. Our prototype for long-period RNGs is the popular¹ *Mersenne Twister* [2].

¹The 32-bit Mersenne Twister is the default random number generator for Matlab, Maple and the GNU Scientific Library, among others.

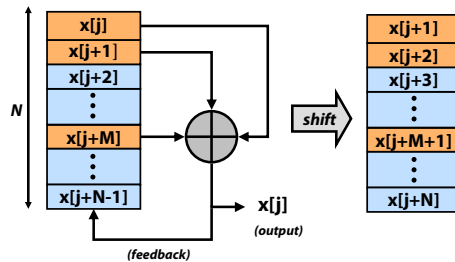


Figure 1: Structure of and transition in a generic GF2SR

This project interested the authors because it ideally balances theory and practice by integrating ideas from disparate fields such as computer architecture, abstract algebra/number theory and probability/operations research within the framework of a single system. Based on the authors’ learning experience during the project, a simplified version is currently being used in an undergraduate course to introduce students to these advanced concepts within an experimental context.

2. MATHEMATICAL BACKGROUND

Long-period RNGs are a type of *Generalized-Feedback Shift-Register* (GF2SR). While the standard linear-feedback shift-register (LFSR) uses a bitwise recurrence, a GF2SR uses *word-wise* recurrences (16-bit, 32-bit, etc.) to achieve longer periods with higher efficiency. To illustrate, consider the GF2SR of Fig. 1, which consists of a N -word shift-register (or *state vector*) \mathbf{x} . At each step, a new word $\mathbf{x}[j]$ is computed based on the following recurrence equation:

$$\mathbf{x}[j] \leftarrow \mathbf{x}[j + M_1] \oplus \mathbf{x}[j + M_2], \quad (1)$$

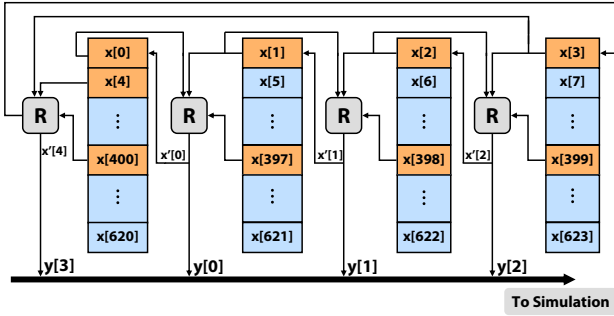
where $M_1 = 1$ and $M_2 = M$ are integer offsets (or ‘taps’) from the current index j and \oplus is the exclusive-OR (XOR) operation. The recurrence of typical long-period RNGs is usually slightly more complicated than (1); for example, the recurrence function $R(\cdot)$ for computing the new word $\mathbf{x}[j]$ in the Mersenne Twister is a function of three inputs, as follows:

$$\mathbf{x}[j] \leftarrow R(\mathbf{x}[j], \mathbf{x}[j + 1], \mathbf{x}[j + M]), \quad (2)$$

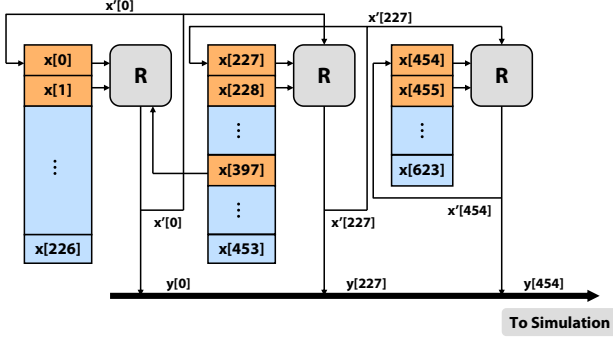
where the offset $M = 397$ and the length of the Mersenne Twister state vector (shift-register) \mathbf{x} is $N = 624$ words (32-bits each). Long-period RNGs have periods of $2^k - 1$, with k being a *Mersenne Prime* number; the Mersenne Twister has a period of $2^{19937} - 1$. In practice, long-period RNGs implemented in hardware do not consist of N individual registers; for higher efficiency, RAM (to store the state vector \mathbf{x}) and modulo address counters (to provide the recurrence offsets) are used.

3. THE HARDWARE FRAMEWORK

We have developed two parallelization methodologies for long-period RNGs: *Interleaved Par.* (IP) and *Chunked Par.* (CP) whose goal is to minimize I/O operations and resource usage (area) while maximizing throughput.



(a) 4-Interleaved (4-IP) Parallelization



(b) 3-Chunked (3-CP) Parallelization

Figure 2: Demonstrating two kinds of parallelizations for the Mersenne Twister (32-bit) long-period RNG

The optimizations that make this possible include pipelining/buffering of the recurrence offsets and the exploitation of FPGA architectural features, such as asynchronous ‘read-before-write’ memory operations. IP and CP have different constraints and possible degrees of parallelization depending on the RNG, which gives the user the choice of various area/throughput/power trade-offs for implementation.

A. Interleaved Parallelization (IP)

Since the recurrence offsets $\{M_1, M_2, \dots\}$ are *constant* relative to each other, IP generates multiple random numbers in parallel by *interleaving* the RNG’s N -word state vector across multiple equal-sized memory banks. This scheme is only practical if the number of banks (i.e., the degree of parallelization) β is a factor of N . Fig. 2a illustrates 4-interleaved parallelization for the Mersenne Twister (eq. (2); $N = 624$, $M = 397$) with four 32-bit outputs $y[0]$, $y[1]$, $y[2]$ and $y[3]$ generated per clock.

B. Chunked Parallelization (CP)

In contrast to IP, CP *sequentially* splits chunks of the state vector across multiple memory banks. Fig. 2b illustrates degree-3 chunked parallelization for the Mersenne Twister. CP-parallelization usually results in fewer read accesses and lower RAM usage than a similar degree of IP. The downside is that one of the memory banks is usually smaller than the others (see Fig. 2b), which results in a variable output rate (throughput); this may or may not be acceptable depending on the application.

C. IP vs. CP and Formalizing the Parallel Framework

In general, for the same degree p , a p -IP parallelization has higher throughput than a p -CP, while the p -CP uses comparatively fewer resources than the p -IP. There are exceptions when CP is both faster and more efficient than IP [3], but we recommend IP-parallelization for most users.

Table 1: Various Interleaved (IP) and Chunked (CP) Parallelizations for the 32-bit Mersenne Twister

Par. Type	None	2-IP	3-IP	8-IP	3-CP	11-CP
Slices	78	159	222	566	207	1038
18k BlockRAMs	2	2	3	8	2	7
Freq (MHz)	348.4	349.4	265.1	283.5	258.3	235.7
Thruput (Gbps)	11.15	22.36	25.45	72.58	24.03	82.96

To formalize the framework, the outputs of the parallelized RNGs are connected directly to the simulation via the FPGA logic fabric. A soft *MicroBlaze* microprocessor controls the RNGs/simulations and allows debugging/data transfer (over Ethernet-TCP/IP).

D. Test Results and Analysis

We used the framework to implement various IP- and CP-parallelizations of three different long-period RNGs on the Xilinx *Virtex-II Pro* FPGA. Five of these (three IP, two CP; for 32-bit Mersenne Twister) are shown in Table 1. On average, their throughput is $\approx 4.8\times$ that of the fastest hardware long-period RNGs reported to date in the literature [4], while using $\approx 30\%$ fewer resources. The 2-IP is $7.1\times$ faster (22.36 Gbps) than the software version on a 3 GHz Pentium 4 (3.14 Gbps). We also demonstrate the first hardware implementations of the 64-bit and 128-bit Mersenne Twister variants [3].

The framework was tested with two real-world applications: a Monte Carlo simulation of photon transport and the Ziggurat algorithm for generating normal random variables. The statistical quality of the parallelized RNG outputs was also verified with the standard RNG test batteries *diehard* and *Crush*.

4. CONCLUSION

We have developed two novel parallelization methodologies and a hardware framework for generating multiple random-number streams using long-period RNGs. Also presented are parallelized variants of the *Mersenne Twister* RNG that, to our knowledge, are the fastest and/or first hardware implementations in the literature to date. The goal is to accelerate hardware-based Monte Carlo simulations without compromising on accuracy.

An educational benefit of this project is that its multidisciplinary aspects have been integrated into a course, allowing undergraduates with basic digital logic and linear algebra knowledge to learn about concepts in computer architecture, abstract algebra and applied probability through ‘system design’ experiments.

REFERENCES

- [1] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed. Reading, Mass.: Addison Wesley, 1997.
- [2] M. Matsumoto and T. Nishimura, “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Trans. on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, Jan. 1998.
- [3] I. L. Dalal and D. Stefan, “A hardware framework for the fast generation of multiple long-period random number streams,” in *Proc. 16th Intl. ACM/SIGDA Symposium on Field Programmable Gate Arrays*, Monterey, CA, Feb. 2008, pp. 245–254.
- [4] S. Konuma and S. Ichikawa, “Design and evaluation of hardware pseudo-random number generator MT19937,” *IEICE Trans. on Info. and Systems*, vol. 88, no. 12, pp. 2876–2879, 2005.