Don't Trust Parallel Monte Carlo!

P. Hellekalek
Dept. of Mathematics
University of Salzburg
A-5020 Salzburg, AUSTRIA

Abstract

Parallel Monte Carlo simulation requires reliable RNGs. For sequential machines, good generators exist. It is not at all trivial to find high-quality RNGs for parallel machines. In this paper we present a short review of the main concepts to produce random numbers on parallel processors and, further, we illustrate some phenomena that occur with parallelization.

1 Introduction

The generation of random numbers is one of the fundamental tasks of numerical practice. Isn't it also one of the most simple tasks we have to deal with? We just call the system random number generator and obtain a finite sequence of random numbers as long as we want. If we want to do parallelization, we just take as many different generators as there are processors.

Alas, life is not that easy! The goal in random number generation is to simulate a realization of a sequence of independent, identically distributed random variables, distributed uniformly in some given domain. The standard domain is the unit interval $[0,1[$. Non-uniform random variates are obtained by transformation methods, see the monograph of Devroye[7] and the software package CRAND of Stadlober and Niederl[39]. Random number generators (RNGs) are basic ingredients of any Monte Carlo simulation on computers. Bad RNGs may ruin your simulation. RNGs are deterministic algorithms that produce "random" numbers in the unit interval $[0,1[$ or "random" points in the d-dimensional unit cube $[0,1[^d$. Sometimes, we use the notion "pseudo"-RNG to underline the deterministic background of these algorithms.

We will discuss now the main concepts to assess RNGs, their relevance for numerical practice, and the (still largely unsolved) question of reliable parallelization techniques for RNGs.

2 Requirements for good RNGs

The numbers or points that are produced by a uniform RNG are supposed to fulfill certain requirements. They should be (i) uniformly distributed, (ii) uncorrelated, (iii) reproducible, in order to debug programs and simulation models, and (iv) portable, in order to get the same random numbers on various platforms. Further, (v) their period should be several magnitudes larger than the largest samples we use, (vi) the inner structure of the output sequence should be analyzable, (vii) parallelization should be possible, and (viii) in arbitrary stochastic simulations, we should get the right results.

Even Santa Claus cannot present us such a "dream" RNG. We are dealing with deterministic algorithms on finite-state machines. Necessarily, the output sequence of an RNG will be periodic, there will exist more or less hidden correlations between these numbers, and there will always be statistical tests or simulations where a given RNG will fail. In this section, we will discuss which of the above conditions can be satisfied.

Reproducibility and portability of an RNG, see (iii) and (iv) above, is a *must.* It is a question of clever scientific programming, see Lendl[24] for an instructive example. Points (i), (ii), (v), and (vi) are the main tasks of the theory of RNGs. Condition (vii) is difficult to satisfy, as we will see later in this paper, and (viii) is an impossible dream.

Compagner[3], an experienced physicist and practitioner, addressed simulation practice as follows: *"Monte Carlo results are misleading when correlations hidden in the random numbers and in the simulated system interfere constructively."* This statement explains our dilemma. The RNGs we use are deterministic algorithms. The numbers they produce will be correlated, they are the output of an iteration of functions. If the inner structure of the random numbers interferes with our particular stochastic simulation, then the simulation results may be useless.

The first consequence of these facts is the insight that there exists nothing like a *safe* RNG. No generator will be appropriate for all tasks. There will always be simulation problems where certain RNGs will fail.

A second consequence might be the wish to employ physical sources of randomness. This choice looks more promising than it is. Such devices tend to be slow and unreliable. We would have to carry out extensive testing of their output to detect technical flaws. Further, it would be very inconvenient to reproduce computations with exactly the same random numbers or to apply parallelization techniques.

The third consequence will be the effort to analyze the period length, the inner structure, and the correlations between the numbers produced by the RNG under study.

2.1 Theoretical Assessment of RNGs

Theoretical research on RNGs is concerned with questions like finding the maximal period length of a particular type of generator, how to choose the parameters of an RNG to obtain maximal period length, the search for algorithms that yield such parameters, the intrinsic structures and regularities (for example lattice structures, etc.) of a particular type of generator, and what can be said about correlations between random numbers.

Nobody should trust an RNG where the designer does not know the period length. The reason is simple. All RNGs have their regularities. If we do simulation with small samples produced by a good RNG, then practically no statistical test will detect those hidden regularities. Hence, these samples will be safe for most simulations. If the sample size is large in relation to the period length, then the regularities of the RNG will begin to show up. Many tests will now be failed and we should not trust our simulation results any longer. As a rule of the thumb, for linear types of RNGs the maximal usable sample size is at most the square root of the period length. Beyond that sample size, we should check our simulation results with great care.

Correlation analysis is the central problem in the empirical and theoretical assessment of RNGs.

2.2 Figures of Merit

The main idea to assess correlations between random numbers by a theoretical figure of merit is the following.

If $x_0, x_1, \ldots \in [0,1[$ is the output sequence of an RNG, then the points (x_0,x_1), (x_2,x_3), $\ldots$ should behave rather randomly in $[0,1[^2$. The same should hold for the triples $(x_{3n}, x_{3n+1}, x_{3n+2})$, $n \geq 0$, and, in general, for the non-overlapping d-tuples $\mathbf{x}_n := (x_{dn}, x_{dn+1}, x_{dn+d-1})$, $n \geq 0$, in $[0,1[^d$. Correlations between consecutive random numbers should be detected by measuring the equidistribution properties of the point sequences $(\mathbf{x}_n)_{n\geq 0}$.

As an illustration, let us produce $N = 2^{15}$ such random points in the unit square with the still widely used linear congruential generator (LCG) Super-Duper of Marsaglia[25], which has the parametrization LCG(2^{32}, 69069, 0,1). (Sometimes, a combination of this LCG and a shift-register generator is also called Super-Duper.) We will have to zoom into the unit interval to make these points visible.

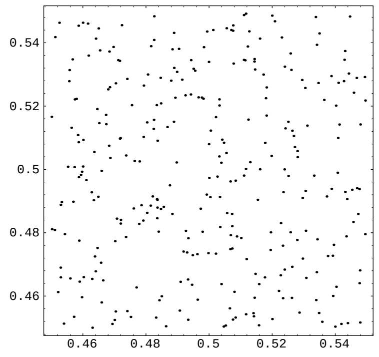

Figure 1. 2^{15} Points of Super-Duper in $[0,1[^2$

These points look randomly distributed. We now produce a larger sample.

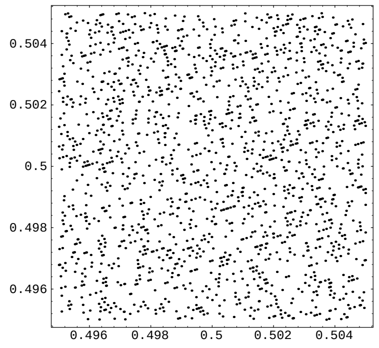

Figure 2. 2^{25} Points of Super-Duper in $[0,1[^2$

Finally, we produce all possible pairs of points in the unit square.

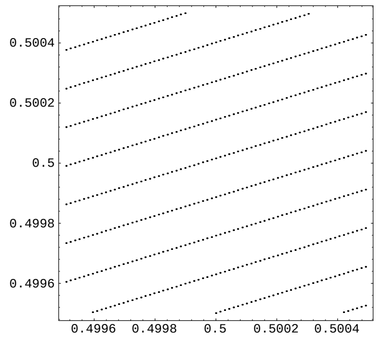

Figure 3. All Points of Super-Duper in $[0,1[^2$

We observe that the regularities begin to shine through in the large sample, see Figure 2. In the last figure, we notice a lattice structure in the full-period point set. This is nothing to be worried about. As we have said before, all RNGs have their inner structure. It is no surprise that a *linear* algorithm like the LCG produces *linear* structures. These structures will only show up for very large samples. They allow us to introduce a powerful figure of merit, the *spectral test*.

This example rises several questions. How to assess such point structures? What will be the influence of these visible correlations for numerical practice? What can happen in parallelization?

The first question has been treated extensively in the literature. There are *figures of merit* for RNGs available to assess such finite point sets $\omega = (\mathbf{x}_n)_{n=0}^{N-1}$ in $[0,1[^d$, even for high dimensions d (i.e. $d \geq 20$). The best-known of these numerical quantities are *discrepancy*, see Niederreiter[31, 33, 35] for details, and the *spectral test* of Coveyou and MacPherson[4], see Knuth[17], L'Ecuyer and Couture[23], and Hellekalek[13] for further information.

What is important to know for a practitioner is the fact that these figures of merit require only the parameters to assess a given RNG. If we obtain satisfactory values for these figures of merit, then the RNG will perform well in many simulations. This statement is based upon more than twenty years of numerical evidence, but not upon some mathematical theorem. Those figures of merit act like a very reliable weather-forecast. In most cases, the prediction will be correct. Sometimes reality will defy prediction.

There is an important difference between discrepancy and the spectral test. The spectral test can only be computed if the points in $[0,1[^d$, $d \geq 2$, have a lattice structure. It computes the maximal distance between successive parallel hyperplanes covering all possible points $\mathbf{x}_n$ that the generator can produce. This limits the spectral test to linear types of RNGs, but those are by far the most popular and best understood class of generators. Discrepancy cannot be computed for an RNG, we can only determine its order of magnitude. This order will serve to compare RNGs and to find promising ones. Discrepancy is not limited to linear RNGs, discrepancy estimates are known for almost all important types of RNGs, see Niederreiter[35]. Both figures of merit yield very reliable RNGs.

2.3 Empirical Tests for RNGs

The performance of RNGs in theoretical tests is no guarantee for successful simulation. It is only a prediction of what we may expect in practice.

The importance of empirical (statistical) testing of RNGs is beyond question. If an empirical test is well-designed, then it will cover a large class of practical simulation problems.

Together with Marsaglia's[26] DIEHARD collection, Knuth's[17] statistical tests constitute the current standard for empirical testing. Important additions to this battery are under construction, see L'Ecuyer et al. [21, 20, 18, 15].

If we want to parallelize an RNG, then, first of all, this RNG should perform well on a single processor.

L'Ecuyer and Andres[19] have proposed a combined LCG with period length near 2^{121}. We will call it "CLCG4". The authors present an implementation in C in a package that provides for multiple virtual generators working in parallel. With its solid background in theory, this RNG is certainly a very promising candidate for a parallel generator. We will now check the distribution of the bits in its output stream.

The "load test" is a variant of the overlapping serial test of Marsaglia[26] that has been studied by Wegenkittl[40]. Wegenkittl constructs the test statistics as follows. We consider overlapping d-tuples $(x_n, x_{n+1}, \ldots, x_{n+d-1})$ of random numbers, $n \geq 0$, where the dimension d ranges between 1 and 5. For every component of this vector, we consider the first four digits. Each block represents a four-digit integer in $\{0, \ldots, 15\}$. We apply Marsaglia's [26] M-tuple test to these d-tuples of integers. This yields one value of the test statistics for a given sample size N and a given dimension d. In our setup, the sample size varies between 2^{18} and 2^{26}, and we compute 32 samples. In the graphical presentation, the bar chart shows the value of the two-sided KS-test statistics applied to the empirical distribution of the upper-tail probabilities of these 32 values. We plot the dimension d versus the dyadic logarithm of the sample size N. At the 0.01 level of significance we use here, the critical value equals 1.59. The target distribution is $U[0,1]$, i.e. uniform distribution on $[0,1]$. The pattern plot shows the 32 values of the upper-tail probabilities in a grey scale. Irregular patterns should appear. If a box becomes all white, then the samples approximate the target distribution too evenly. If a box becomes all black, then the distance between the empirical distribution function and the cummulative distribution function of the target distribution is too large. In both of these extreme cases, the RNG fails the test. It produces the wrong results. Its samples are unable to randomize the test statistics in a proper way.

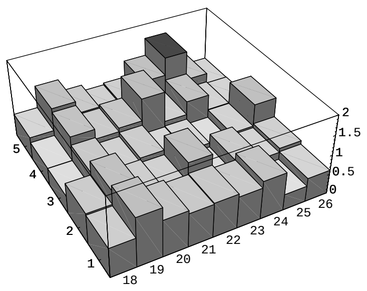

Figure 4. CLCG4, KS Values

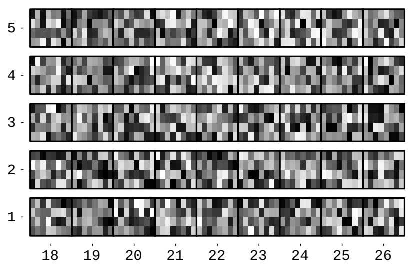

Figure 5. CLCG4, Upper Tail Probabilities

CLCG4 gives a flawless performance. In our discussion of the leap-frog technique below, we will encounter a totally different behavior with Super-Duper. Our empirical results will illustrate two facts, that small RNGs like Super-Duper are now out-of-date, and that the samples should not be chosen too large compared to the period length of the generator we use. The second conclusion holds in general.

3 Parallel RNGs: Requirements

Parallel random number generation requires great care. Much less is known than in the single processor case.

For parallel RNGs, we will have to enlarge our list of properties of a good RNG, see Section 2. The additional requirements are (ix) that the RNG should be usable for any (reasonable) number of processors, (x) that the parallel streams of random numbers produced on the different processors should be uncorrelated, and (xi) that they are generated independently, for reasons of efficiency.

We will now discuss how to meet all these criteria and we will also indicate which phenomena may occur in practice.

3.1 Parallelization Methods

There are two basic parallelization techniques to produce random numbers. Method I assigns different RNGs to different processors. Method II assigns different substreams of one large RNG to different processors. We will not talk about a third method to generate substreams of random numbers, so-called *pseudorandom trees*, see Anderson[1] for a concise discussion, because there is no information available about the correlations we may get.

The danger with the first method is the following. There might be unknown correlations between the different RNGs we use. If we happen to use the same type of RNG but with different parameters, then we might encounter very unpleasant surprises. There is one family of RNGs where theoretical support is available, explicit-inversive congruential generators (EICG), see Niederreiter[32, 34]. The EICG was introduced by Eichenauer-Herrmann[9]. A very efficient implementation is due to Lendl[24], both the code in C and the master's thesis are available from the web-site `http://random.mat.sbg.ac.at`.

Method II can be controlled better, although its risks should not be forgotten. There are two variations. Method II.a, the "leap-frog" technique, assigns the substream $(x_{nL+j})_{n\geq 0}$ to the j-th processor, $0 \leq j \leq L-1$. In other words, we use substreams of *lag* L of the original sequence $(x_n)_{n\geq 0}$.

Method II.b, the "splitting technique", partitions the original sequence into L (very long) consecutive blocks. Each of our L processors is assigned a different block, where every block is defined by a unique seed. This approach is a very efficient way to assign different streams of random numbers to different processors. Splitting is particularly easy for linear types of RNGs, see L'Ecuyer[22, 19]. It has to be used with caution, as we will exhibit below.

Matsumoto's[28, 29, 30] twisted feedback shift-register generators (tGFSR) TT800 and MT19937 have the extremely long periods $2^{800}-1$ and $2^{19937}-1$ and an extensive theoretical background. Due to their long period, we could choose the initial values (in other words, the seeds) randomly and obtain as many substreams as we need. It is highly improbable that two processors will use the same seeds or that two substreams will overlap.

Not much is known about correlations between disjoint substreams of consecutive random numbers. One thing is for sure, this subject is dangerous territory. We refer to De Matteis and Pagnutti[5, 6] for further information and cautionary tales.

3.2 Example I: Leap-Frog

As a first example of what can happen with these parallelization techniques, we study lagged subsequences of Super-Duper. As we have said before, this RNG is out-of-date, but the results below exhibit intrinsic problems of the parallelization method that apply to any RNG, in particular to linear ones.

The original Super-Duper generator starts to fail the load test if we increase the sample size or the dimension.

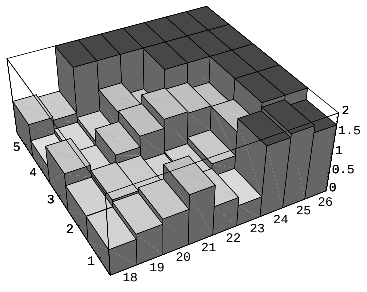

Figure 6. Super-Duper, KS Values

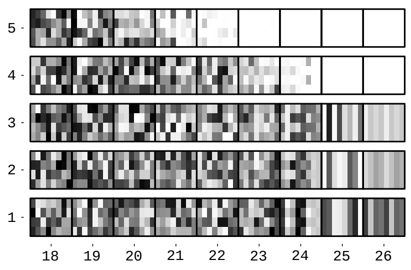

Figure 7. Super-Duper, Upper Tail Probabilities

If we consider the leap-frog subsequence with lag 135, i.e. the sequence of random numbers $(x_{135n})_{n\geq 0}$, then we observe a phenomenon that is common to linear RNGs: A marked decrease in the statistical quality. This unpleasant phenomenon occurs also for reliable long-period generators like the CRAY system generator `Ranf`, which is LCG(2^{48}, 44485709377909, 0, 1). Here the subsequences with lag 128, as they appear in recommended vectorization and parallelization techniques, perform terribly. We refer the reader to Entacher[11] for a whole collection of cautionary tales of this sort.

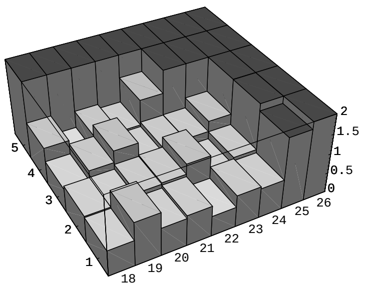

Figure 8. Super-Duper, Lag 135, KS Values

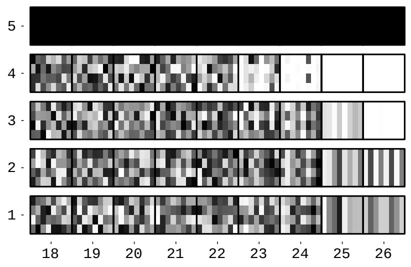

Figure 9. Super-Duper, Lag 135, Upper Tail Probabilities

3.3 Example II: MC Integration

The test design is as follows. We work with a given test function f in dimension d. We produce two sets of integration nodes in $[0,1[^d$ of roughly the same size. The first set ω_1 stems from the original stream of random numbers, the second set ω_1^* is produced by combining L disjoint leap-frog substreams that have the same lag L into one big finite sequence. The exact procedure will be discussed below. For each of those two sets of integration nodes, we compute the integration errors ϵ_1 and ϵ_1^* i.e. the absolute value of the difference between the integral of f and the mean over the values of f at the nodes. Then we generate the next two sets of integration nodes, with the same number of elements as before. In total, we produce 64 set of integration nodes of each type and obtain 64 integration errors ϵ_k and ϵ_k^*, $1 \leq k \leq 64$. We use disjoint consecutive streams of random numbers for these computations. Finally, we compute the sample mean $\hat{\mu}$ ($\hat{\mu}^*$) and variance $\hat{\sigma}^2$ ($\hat{\sigma}^{*2}$) of these 64 values ϵ_k and ϵ_k^*.

As a test function, we choose the polynomial

$$f(\mathbf{x}) := \prod_{i=1}^{d} g(x_i), \quad \mathbf{x} = (x_1, \ldots, x_d) \in [0,1[^d,$$

where $g(x) := x^{20} - \frac{1}{21}$, $x \in [0, 1[$. The functions g and f integrate to zero. We consider the mean errors in dimension $d = 6$.

We then compute the 99% confidence interval for the true value zero of the integral over f by Student's t-distribution. This interval is given by $]\hat{\mu} - 0.331\hat{\sigma}, \hat{\mu} + 0.331\hat{\sigma}, [$, where $0.331 = t_{63,0.995}$. Here, the number $t_{63,0.995}$ denotes the 0.005-quantile of Student's t-distribution with 63 degrees of freedom.

The precise setup of our test follows Entacher, Uhl and Wegenkittl[12]. With the Super-Duper generator, we produce the first consecutive $d \cdot N$ random numbers x_n in $[0, 1[$ and construct N nonoverlapping d-tuples $\mathbf{x}_n = (x_{nd}, \ldots, x_{nd+d-1})$ in $[0, 1[^d$, $0 \leq n < N$. This yields the first set ω_1 of integration nodes and the first value ϵ_1 of the integration error. Then, we take the next $d \cdot N$ random numbers x_n, construct the associated N points in the d-dimensional unit cube and obtain the second integration error ϵ_2. In total, we repeat this procedure 64 times to get the integration errors ϵ_k, $1 \leq k \leq 64$.

In Figure 10 below, the ticks on the abscissa represent the dual logarithm of N. The sample size N ranges between 2^{18} and 2^{24}. The associated value of $\hat{\mu}_N$ is given on the ordinate by the dotted line. The shaded area indicates the 99% confidence interval. The horizontal line at level 0 represents the expected value, i.e. the value of the integral of f over $[0, 1]^d$.

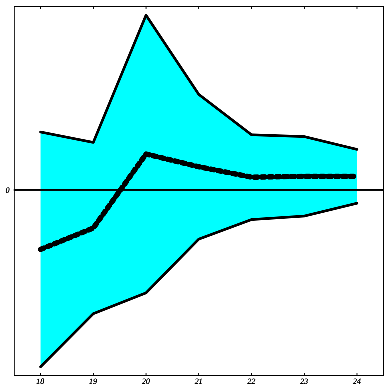

Figure 10. Integration Error, Dimension d=6.

We observe that the mean error is simulated correctly. The Super-Duper generator works as it may be expected in such a harmless Monte Carlo integration.

As we have stated before, even in this simple Monte Carlo application, leap-frogging must be done with care. The Super-Duper generator is very sensible to leap-frogging with lag $L = 135$, as we will show by comparison to the performance of the original generator. In our example above, we have used 64 samples ω_k, $1 \leq k \leq 64$, each of size N. Every sample has consumed $d \cdot N$ consecutive random numbers to produce N points in dimension d. Now, we will again produce 64 samples ω_k^* of a size $\tilde{N}$ that is comparable to N, but each sample will be the union of 135 point sets that stem from 135 leap-frog subsequences with lag $L = 135$. We will see that these 64 samples perform much worse than the original Super-Duper samples although they do not stem from *pure* leap-frog sequences but are just unions of such subsequences. Even in this "diluted" leap-frog case, wrong simulation results appear. The situation would have been much worse if we had compared the samples of the first setup to node sets of similar size that were constructed from *just one* leap-frog sequence and not from a union of 135 such subsequences.

We proceed as follows. We produce the first block of $d \cdot N$ random numbers x_n in $[0, 1[$ with the Super-Duper generator as before. Now, in the first block, we take the leap-frog random numbers with lag $L = 135$. This yields x_0, x_{135}, x_{270}, $\ldots$, i.e. the numbers $x_{n \cdot 135}$, $0 \leq n < N'$, $N' := \lfloor d \cdot N/135 \rfloor$. From these N' numbers, we construct $N'' := \lfloor N'/d \rfloor$ nonoverlapping points $\mathbf{x}_n$ in $[0, 1[^d$. In the next step, we use the leap-frog sequence $x_{n \cdot 135+1}$, $0 \leq n < N'$, in the same manner to construct another N'' points in $[0, 1[^d$, then the subsequence $x_{n \cdot 135+2}$, $0 \leq n < N'$ and so on. Finally, we unite all these 135 point sets of size N'' in one big point set ω_1^*. This set will have $\tilde{N} = 135 \cdot N''$ elements with $\tilde{N} \leq N$. It is elementary to see that $N - \tilde{N} < 135 - \frac{135}{d}$. This difference can be neglected in numerical computations if N is large in comparison to the lag $L = 135$. The next sample ω_2^* is constructed in the same way from the second block of $d \cdot N$ random numbers and so on, until we have 64 node sets ω_k^*, each of size $\tilde{N}$. The simulation results for the mean integration error are *off target*, as is clearly visible in Figure 11.

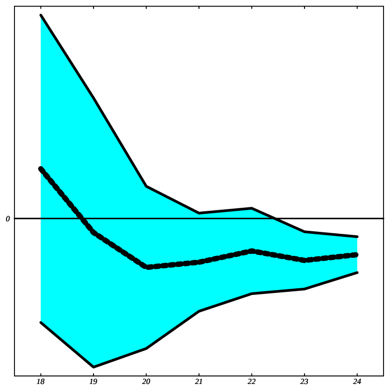

Figure 11. Integration Error, Lag 135, d=6.

4 Some Useful Links and References

For readers who are interested in the theoretical analysis of RNGs, we recommend Niederreiter[33, 35], and Hellekalek[13]. Eichenauer-Herrmann et al.[10] contains a comprehensive theoretical discussion of inversive RNGs.

Useful survey papers on parallel RNGs are Coddington[2], Srinivasan, Ceperley, and Mascagni[38], Anderson[1], and Eddy[8]. Helpful web-sites are `http://random.mat.sbg.ac.at` and `http://www.ncsa.uiuc.edu/Apps/CMP/RNG/-www-rng.html`.

For parallel RNG libraries we refer to Masuda and Zimmermann[27], Hennecke[16], and Pryor et al.[37]. The code of Matsumoto's TT800 and Mersenne Twister MT19937, a tGFSR with the incredible period length $2^{19937} - 1$, is available from the web-site `http://random.mat.sbg.ac.at`. The combined LCG of L'Ecuyer and Andres[19] is also a very promising RNG for parallelization.

Acknowledgments

I would like to thank my research assistants Stefan Wegenkittl and Karl Entacher, who have carried out the computations for all figures in this paper. This research has been supported by the grants FWF–P11143 and FWF–P12654 from the Austrian Science Foundation.

References

[1] S.L. Anderson. Random number generation on vector supercomputers and other advanced architectures. *SIAM Review*, **32**:221–251, 1990.

[2] P. Coddington. Random number generators for parallel computers. NHSE Review, 2nd issue, Northeast Parallel Architectures Center, 1996. Available from `http://nhse.cs.rice.edu/NHSEreview/RNG/`.

[3] A. Compagner. Operational conditions for random-number generation. *Phys. Review E*, **52**:5634–5645, 1995.

[4] R.R. Coveyou and R.D. MacPherson. Fourier analysis of uniform random number generators. *J. Assoc. Comput. Mach.*, **14**:100–119, 1967.

[5] A. De Matteis, J. Eichenauer-Herrmann, and H. Grothe. Computation of critical distances within multiplicative congruential pseudorandom number sequences. *J. Comp. Appl. Math.*, **39**:49–55, 1992.

[6] A. De Matteis and S. Pagnutti. Controlling correlations in parallel Monte Carlo. *Parallel Comput.*, **21**:73–84, 1995.

[7] L. Devroye. *Non-Uniform Random Variate Generation.* Springer-Verlag, New York, 1986.

[8] W.F. Eddy. Random number generators for parallel processors. *J. Comp. Appl. Math.*, **31**:63–71, 1990.

[9] J. Eichenauer-Herrmann. Statistical independence of a new class of inversive congruential pseudorandom numbers. *Math. Comp.*, **60**:375–384, 1993.

[10] J. Eichenauer-Herrmann, E. Herrmann, and S. Wegenkittl. A survey of quadratic and inversive congruential pseudorandom numbers. In Niederreiter et al. [36], pages 66–97.

[11] K. Entacher. A collection of selected pseudorandom number generators with linear structures. Technical report series, ACPC - Austrian Center for Parallel Computation, 1997.

[12] K. Entacher, A. Uhl, and S. Wegenkittl. Linear Congruential Generators for Parallel Monte-Carlo: the Leap-Frog Case. Preprint, Department of Mathematics, University of Salzburg, Austria, submitted for publication.

[13] P. Hellekalek. On the assessment of random and quasi-random point sets. In Hellekalek and Larcher [14]. To appear.

[14] P. Hellekalek and G. Larcher, editors. *Random and Quasi-Random Point Sets*, Springer Lecture Notes in Statistics. Springer-Verlag, New York, 1998. To appear.

[15] P. Hellekalek and P. L'Ecuyer. Testing random number generators. In Hellekalek and Larcher [14]. To appear.

[16] M. Hennecke. Random number generators homepage. Available from `http://www.uni-karlsruhe.de/~RNG/`.

[17] D.E. Knuth. *The Art of Computer Programming, Vol. 2.* Addison-Wesley, Reading, Mass., second edition, 1981.

[18] P. L'Ecuyer. Random number generators and empirical tests. In Niederreiter et al. [36], pages 124–138.

[19] P. L'Ecuyer and T.H. Andres. A random number generator based on the combination of four LCGs. *Mathematics and Computers in Simulation*, **44**:99–107, 1997.

[20] P. L'Ecuyer, A. Compagner, and J.-F. Cordeau. Entropy tests for random number generators. Manuscript, 1996.

[21] P. L'Ecuyer, J.-F. Cordeau, and R. Simard. Close-point spatial tests for random number generators. Submitted, 1996.

[22] P. L'Ecuyer and S. Coté. Implementing a random number package with splitting facilities. *ACM Trans. Math. Software*, **17**:98–111, 1991.

[23] P. L'Ecuyer and R. Couture. An implementation of the lattice and spectral tests for multiple recursive linear random number generators. *INFORMS J. on Comput.*, **9**:206–217, 1997.

[24] O. Lendl. Explicit inversive pseudorandom numbers. Master's thesis, Institut für Mathematik, Universität Salzburg, Austria, 1996. Available from `http://random.mat.sbg.ac.at/`.

[25] G. Marsaglia. The structure of linear congruential sequences. In S. K. Zaremba, editor, *Applications of Number Theory to Numerical Analysis.* Academic Press, New York, 1972.

[26] G. Marsaglia. A current view of random number generators. In L. Brillard, editor, *Computer Science and Statistics: The Interface*, pages 3–10, Amsterdam, 1985. Elsevier Science Publishers B.V. (North Holland).

[27] N. Masuda and F. Zimmermann. PRNGlib: a parallel random number generator library. Technical report, Swiss Center for Scientific Computing, 1996. Available from `http://www.cscs.ch/Official/Publications.html`.

[28] M. Matsumoto and Y. Kurita. Twisted GFSR generators. *ACM Trans. Model. Comput. Simul.*, **2**:179–194, 1992.

[29] M. Matsumoto and Y. Kurita. Twisted GFSR generators II. *ACM Trans. Model. Comput. Simul.*, **4**:254–266, 1994.

[30] M. Matsumoto and T. Nishimura. A new generation: 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. Model. Comput. Simul., to appear*, 1998.

[31] H. Niederreiter. Quasi-Monte Carlo methods and pseudo-random numbers. *Bull. Amer. Math. Soc.*, **84**:957–1041, 1978.

[32] H. Niederreiter. New methods for pseudorandom number and pseudorandom vector generation. In J.J. Swain et al., editor, *Proc. 1992 Winter Simulation Conference (Arlington, Va., 1992)*, pages 264–269. IEEE Press, Piscataway, N.J., 1992.

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

[34] H. Niederreiter. On a new class of pseudorandom numbers for simulation methods. *J. Comp. Appl. Math.*, **56**:159–167, 1994.

[35] H. Niederreiter. New developments in uniform pseudorandom number and vector generation. In H. Niederreiter and P.J.-S. Shiue, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, volume 106 of *Lecture Notes in Statistics*, pages 87–120. Springer-Verlag, New York, 1995.

[36] H. Niederreiter, P. Hellekalek, G. Larcher, and P. Zinterhof, editors. *Monte Carlo and Quasi-Monte Carlo Methods 1996*, volume 127 of *Springer Lecture Notes in Statistics.* Springer-Verlag, New York, 1997.

[37] D.V. Pryor, S.A. Cuccaro, M. Mascagni, and M.L. Robinson. Implementation and usage of a portable and reproducible parallel pseudorandom number generator. Technical report, Supercomputing Research Center, Institute for Defense Analyses, 1994.

[38] A. Srinivasan, D.M. Ceperley, and M. Mascagni. Random Number Generators for Parallel Applications. In D. Ferguson, J.I. Siepmann, and D.G. Truhlar, editors, *Monte Carlo Methods in Chemical Physics*, Advances in Chemical Physics series. Wiley, New York, 1998, to appear.

[39] E. Stadlober and F. Niederl. C-Rand: a package for generating nonuniform random variates. In *Compstat '94, Software Descriptions*, pages 63–64, 1994.

[40] S. Wegenkittl. Empirical testing of pseudorandom number generators. Master's thesis, Institut für Mathematik, Universität Salzburg, Austria, 1995. Available from `http://random.mat.sbg.ac.at/`.