A New Empirical Test for Parallel Pseudorandom Number Generators

Yufeng Liang and P.A. Whitlock

Department of Computer and Information Science, Brooklyn College, Brooklyn, NY 11210

Abstract

Recently, Percus derived probabilities and distributions for parllel, i.i.d. random sequences of integers. This was accomplished by considering s given bit locations in each random variable (represented as a predetermined number of bits) in each sequence. These s bits were used to create a new binary sequence whose expected behavior can be analyzed. Based upon Percus' work, an empirical test for parallel pseudo-random number generators has been devised. For each generator, parallel sequences of various lengths are considered and analyzed as proposed by Percus and the results are statistically compared to the expected behavior for truly random sequences. A variety of parallel pseudo-random number generators from the literature are studied and the usefulness of the new empirical test is discussed.

1 Introduction

Computer simulations use random variables to model the probability distribution functions important to the calculations. The random variables are replaced by values, pseudo-random numbers, provided by deterministic algorithms whose properties mimic those of a truly random sequence [1]. A parallel pseudo-random number generator (PRNG) is used to produce multiple sequences of pseudo-random numbers for parallel simulations [2–8].

To avoid introducing unwanted correlations, sequences of pseudo-random numbers must mimic the properties of the truly random sequences that are most important to the simulations. This is a well-studied topic for single sequences of random variables[1]. However, in the case of parallel sequences less is understood about the type of correlations that could lead to poor results in use. Theoretical tests can be developed which consider the properties of the whole period of a specific generator [6]. This does not necessarily predict the behavior of subsets, *i.e.* samples of smaller size than the whole period, that are used

in actual simulations. As the performance in theoretical tests is no guarantee, but only an indicator of what we may expect in practice, empirical testing of generators is a necessity [7].

Some theoretical tests have been developed for specific PRNG's [6,9]. Our goal was to develop and implement a general empirical test that can be applied to a wide range of PRNG's. The empirical test described here is based upon the theoretical investigations of Percus [10].

Section 2 briefly reviews Percus's theory on the properties of parallel sequences of truly random numbers. The translation of the theory into an empirical test, the parallel sequence test, is described in Section 3. An outline of the implementation of the test as a distributed calculation is given in Section 4. Finally, Section 5 discusses the application of the parallel sequence test to several published PRNG's. We conclude that the test is useful in predicting correlations that affect a distributed or parallel simulation.

2 Theoretical Basis of the Parallel Sequence Test

Consider 2 sequences of random variables

$$\begin{array}{l} X_1^{(1)}, X_2^{(1)}, \cdots, X_l^{(1)} \\ X_1^{(2)}, X_2^{(2)}, \cdots, X_l^{(2)} \quad , \end{array} \tag{1}$$

where the $X_j^{(k)}$ are integers represented by an arbitrary, but fixed number of bits and l is the length of the sequence. Consider the set $I = \{i_1, ..., i_s\}$ of indices of bit locations in each integer, where the total number of bit locations of interest is s. These bit locations do not have to be consecutive. A new binary sequence $\{Y_j \mid 1 \leq j \leq l\}$ is created such that

$$Y_j = \begin{cases} 1 \text{ if } X_j^{(1)} \text{ equals } X_j^{(2)} \text{for } i_1, \cdots, i_s \in I \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

Assume each bit of $X_j^{(k)}$ has the probability of 1/2 to be 0 or 1. If the two sequences of (1) are independent, $\{Y_j\}$ would be a random number sequence with the probability $P\{Y_j = 1\} = 2^{-s}$.

Let $q = 1 - p$, and $P(r, l)$ be the probability of having r ones in succession in $\{Y_j\}$, where $(1 \leq r \leq l)$. Percus [10] derived that the probability,

$P\{\text{length of longest run } \geq r\}$ is [11]

$$P(r,l) = 1 - \sum_{j=0}^{\frac{l}{r+1}} (-1)^j (qp^r)^j \binom{l-jr}{j} + p^r \sum_{j=0}^{\frac{l-r}{r+1}} (-1)^j (qp^r)^j \binom{l-(j+1)r}{j}. \quad (3)$$

It is challenging to compute $P(r,l)$ with (3). Alternatively, let

$$\alpha = p^r(1-p) \quad (4)$$

It can be proved [10] that the equation

$$\xi - 1 - \alpha\xi^{r+1} = 0 \quad (5)$$

has at least one positive real root. Denote the smallest one as ξ_0 . Then $P(r,l)$ has the following estimate

$$P(r,l) = 1 - \frac{1-p\xi_0}{(1-p)\xi_0} \frac{\xi_0^{-1}}{r+1-r\xi_0} + \theta \frac{r}{1-p} p^{l+2} \quad (6)$$

where $\mid \theta \mid < 1$. The root ξ_0 can be found either by direct solution of the trinomial equation following Gauss' method, or by application of Lagrange's series.

Percus's analysis can be extended to the case of t parallel sequences

$$\begin{array}{cccc} X_1^{(1)}, & X_2^{(1)}, & \cdots, & X_l^{(1)} \\ X_1^{(2)}, & X_2^{(2)}, & \cdots, & X_l^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{(t)}, & X_2^{(t)}, & \cdots, & X_l^{(t)}. \end{array}$$

A binary sequence $\{Y_j \mid 1 \leq j \leq l\}$ can be defined such that

$$Y_j = \begin{cases} 1 \text{ if at least w of the } \{X_j^{(k)}\}_{k=1}^t \text{ are equal for all } i_1, \cdots, i_s \in I \\ 0 \text{ otherwise} \end{cases} \quad (7)$$

where $(\max(1, \mathrm{t} - 2^{\mathrm{s}}) \leq \mathrm{w} \leq \mathrm{t})$. Then the probability p is

$$p = P\{Y_j = 1\} = \sum_{j=w}^{t} \binom{t}{j} \frac{1}{2^{s(j-1)}} (1 - \frac{1}{2^s})^{t-j} \quad (8)$$

and the analysis continues as in (3).

3 The Parallel Sequence Test

The basis of the empirical test is to consider many groups of pseudo-random number sequences derived from a particular PRNG. The number of sequences, t, tested in each group can be two or more. Each group of sequences is converted to a new binary sequence with (2), if $t = 2$ or (7), if $t \geq 3$. The longest run of 1's in each new binary sequence is determined. A chi-square statistic, defined below, is calculated for each group. The values of the chi-square statistic are used in a Kolmogorov-Smirnov test to judge whether the PRNG produces sequences whose properties mimic those of a truly random sequence.

3.1 Calculate Longest Runs

The set of bit locations, I, is chosen prior to the test and are fixed throughout a single replication of the test. The number of classes, N_c, representing the distribution of possible longest runs depends upon the number of bits considered. Table 1 illustrates the observed distributions of probabilities, p, for a longest run of a random binary sequence when $t = 2$, l, the length of the sequence = 1,000,000 and $1 \leq s \leq 6$ bit locations are examined. It can be seen that N_c becomes smaller as the number of bits to be tested increases. To be meaningful, s should be chosen so that $l * p$ is greater than 5.

From each group of sequences using the s bits, a new binary sequence

$$Y_1^{(j,k)}, Y_2^{(j,k)}, \cdots, Y_l^{(j,k)} \quad (1 \leq j \leq L_\chi \; 1 \leq k \leq L_{ks}) \tag{9}$$

is obtained from (2) if $t = 2$ or (7) if $t \geq 3$. Each binary sequence is formed from independent, non-overlapping sequences. L_χ is the number of groups used to form the χ^2 statistic and L_{ks} is the maximum number of χ^2 values used in the Kolmogorov-Smirnov test. The longest run of 1's in each sequence of (9) is determined and is denoted by $r_j^{(k)}$.

3.2 Forming the Chi-square Statistic

Longest runs exceeding a length of 100 values are unlikely and are grouped together in the statistical analysis. In general, we have $0 \leq r_j^{(k)} \leq 100$. Let n_i be integers where $(1 \leq i \leq N_c)$, and $n_0 = 0$, $n_{N_c} < 101$, $n_{i-1} < n_i$. There

Longest	Number of Bits					
Run	1	2	3	4	5	6
1	-	-	-	-	-	
2	-	-	-	-	-	0.023
3	-	-	-	-	0.397	0.920
4	-	-	-	0.409	0.575	0.056
5	-	-	0.036	0.537	0.028	0.001
6	-	-	0.623	0.051	0.001	-
7	-	-	0.290	0.003	-	-
8	-	0.057	0.044	-	-	-
9	-	0.432	0.006	-	-	-
10	-	0.347	0.001	-	-	-
11	-	0.120	-	-	-	-
12	-	0.033	-	-	-	-
13	-	0.008	-	-	-	-
14	-	0.002	-	-	-	-
15	-	0.001	-	-	-	-
16	0.022	-	-	-	-	-
17	0.126	-	-	-	-	-
18	0.237	-	-	-	-	-
19	0.235	-	-	-	-	-
20	0.167	-	-	-	-	-
21	0.100	-	-	-	-	-
22	0.055	-	-	-	-	-
23	0.028	-	-	-	-	-
24	0.015	-	-	-	-	-
25	0.007	-	-	-	-	-
26	0.004	-	-	-	-	-
27	-	-	-	-	-	-

Table 1
The distributions of lengths of longest runs in binary sequences of length 1,000,000 with the number of bits considered varying from 1 to 6.

are N_c mutual exclusive classes $[n_{i-1}, n_i)$, $(1 \le i \le N_c)$. The probability that the longest run, $r_j^{(k)}$, falls in the mutually exclusive class, $[n_{i-1}, n_i)$ can be obtained from $P(r, l)$, (4) - (6), the probability that a longest run exceeds length r. That is,

$$P\{r_j^{(k)} \in [n_{i-1}, n_i)\} = P(n_{i-1}, l) - P(n_i, l) \tag{10}$$

The expected number of longest runs falling into an interval, $l * P\{r_j^{(k)} \in [n_{i-1}, n_i)\} \ge 5$, to yield a good approximation to the asymptotic chi square distribution. Let $Z_i^{(k)}$ be the number of $r_j^{(k)}$, $(1 \le j \le L_\chi), (1 \le k \le L_{ks})$ such that $n_{i-1} \le r_j^{(k)} \le n_i$, then

$$V_k = \chi^2 = \sum_{i=1}^{N_c} \frac{(Z_i^{(k)} - e_i)^2}{le_i} \quad , \tag{11}$$

where $e_i = l * P\{r_j^{(k)} \in [n_{i-1}, n_i)\}$. V_k is asymptotically chi-square distributed with $N_c - 1$ degrees of freedom, $V_k \approx \chi^2_{N_c-1}$, see [12].

3.3 Performing the Kolmogorov-Smirnov Test

Each time a new value of χ^2 is calculated, it is added to the set, $V_1, \cdots, V_q$, obtained from (11). The V_i are sorted in ascending order and denoted by $\bar{V}_1 \le \cdots \le \bar{V}_q$. A Kolmogorov-Smirnov test is performed on the values and the quantities K_q^+ and K_q^- are formed:

$$K_q^+ = \sqrt{q} \max_{1 \le j \le q} \left(\frac{j}{q} - F_{\chi^2_{N_c-1}}(\bar{V}_{j-1}) \right) \tag{12}$$

$$K_q^- = \sqrt{q} \max_{1 \le jqlek} \left(F_{\chi^2_{N_c-1}}(\bar{V}_{j-1}) - \frac{j}{q} \right) \tag{13}$$

When q is large, the distribution of K_q^+ and K_q^- is given by

$$F_{K_q^\pm}(t) = 1 - e^{2t^2} \left(1 - \frac{2t}{3\sqrt{q}} \right), \quad t \ge 0, \tag{14}$$

The distribution functions (14) can be used to transform the sequences of realizations of K_q^+ and K_q^- into sequences of numbers $\{F_{K^+}(K_q^+)\}$ and $\{F_{K^-}(K_q^-)\}$,

$1 \leq q \leq L_{ks}$, which are uniformly distributed on [0,1]. For any integer q_0 much smaller than L_{ks}, *e.g.* $q_0 < \frac{1}{2}L_{ks}$, if $q > q_0$ such that

$$\left| \{F_{K^{\pm}}(K_q^{\pm})\} - 0.5 \right| > d\% \tag{15}$$

for any positive number $d < 50$, we have $(50+d)$ percent of confidence to reject the hypothesis that any t sequences of pseudo-random variables generated by the PRNG are independent.

4 Implementation of the Empirical Test

A prototype of the parallel sequence test was implemented and run on a cluster of workstations distributed across a network. The test was implemented in the C language on a UNIX operating system environment with the message passing facility, DP [13], installed. DP is a library of process management and communication tools for facilitating writing portable distributed programs on MIMD systems. It supports dynamic process creation and message passing with a variety of semantics.

To perform the test upon a PRNG, three modules were created, the test module, the provider module and the configuration module. The test module is a executable program consisting of all of the major functionality for testing a particular PRNG. The provider module is another executable program providing the test module with pseudo-random variables generated by the specific PRNG to be tested. The configuration module is a text file with the parameters for a test.

4.1 The Provider Module

The provider module consists of one or more PRNG's to be tested and the interface between the PRNG's and the test module. The interface transfers pseudo-random variables from the PRNG specified in the configuration module to the test module. It is hence called the provider while the test module acts as a consumer of pseudo-random variables. The program begins execution when a process running the test module requests pseudo- random variables. The provider module can be maintained by users of the testing software without the need to know the details of the test module or the DP facility.

4.2 The Test Module

The test module uses the DP library to distribute processes to carry out the testing across a network of workstations. A primary process on the initial host begins the execution of test module processes on the other network hosts. It collects the longest run statistics, calculates the χ^2 values and performs the Kolmogorov-Smirnov test.

Each host on the network executes two processes, one that provides the needed pseudo-random variables and another that carries out the details of the test. That is, the distributed test module process obtains the t sequences of random variables, produces the new binary sequence and calculates the longest run of the new sequence. The DP message passing facilities are used to communicate with the primary process.

4.3 The Configuration Module

The configuration module is used to set the parameters needed by the test and the provider modules. The parameters include the PRNG to be tested (multiple PRNG's can be contained in the provider module), the bit locations to be tested, the length of a sequence l , the number of longest runs for a chi-square test L_χ , the total number of χ^2 values to be observed for the Kolmogorov-Smirnov test, L_{ks} , and the number of sequences in a group t . The configuration module can be modified any time before performing the test without having to recreate the executable programs.

5 Results of Applying the Parallel Sequence test to Specific Generators

5.1 R250

The R250 pseudo-random number generator [14] as implemented by W. L. Maier [15] was used. Since 16 R250 streams can be obtained from the generator at the same time, two of them were randomly chosen for the test.

The test was repeated four times and the results are shown in Table 2, for $q = 150$ replications. Only one of the experiments exhibited acceptable behavior. The other 3 experiments showed poorer performance; the $F_{K^\pm}(K_q^\pm)$ were out of the range $[15\%, 85\%]$. According to (15), we have a 85% confidence to reject

Generators	R250		Nested Weyl		SNWS	
q	150		10		15	
run	K_q^+	K_q^-	K_q^+	K_q^-	K_q^+	K_q^-
(1)	47.8	37.3	0.0	100	0.0	100
(2)	85.5	1.6	0.0	100	0.0	100
(3)	89.6	3.4				
(4)	98.3	14.8				

Table 2
The quantity $\left|\{F_{K^\pm}(K_q^\pm)\} - 0.5\right|$ obtained from running the parallel sequence test for the pseudo-random number generators R250, the nested Weyl sequence and the shuffled nested Weyl sequence (SNWS).

R250. In practice, R250 was found to introduce correlations into some classes of parallel simulations [16].

5.2 Nested Weyl Sequence

The nested Weyl sequence (NWS) is a natural extension of the Weyl sequence [17] and is defined as $x_n = \{n\{n\alpha\}\}$, where $\{\}$ indicates the fractional part of the enclosed number, within the precision of the computer used. When used as a PRNG, the k^{th} sequence is defined by

$$x_n^{(k)} = \{k\{n\{n\alpha\}\}\} \tag{16}$$

Because our empirical test only accepts integers, the above formula was replaced by

$$X_n^{(k)} = \left\lfloor 2^{s_0} x_n^{(k)} \right\rfloor = \lfloor 2^{s_0}\{k\{n\{n\alpha\}\}\}\rfloor \tag{17}$$

where $\lfloor y \rfloor$ is the floor of y, and s_0 is an integer such that $x_n^{(k)}$ are converted to integers in $[0, 2^{s_0})$. In our tests, $\alpha = \sqrt{2}$, s_0 was 4 and all bits in the whole number part were tested. Two experiments were done and for $q = 10$ (larger values of q showed no improvement), Table 2, the $F_{K^\pm}(K_q^\pm)$ in each experiment are outside of the range $[1\%, 99\%]$. Hence we have 99% of confidence to reject it. This result was not unexpected since previous work had found the generator correlated [18].

This generator was an attempt to create a PRNG with better properties than the nested Weyl sequence [18]. An integer $M >> 1$ is selected and a sequence of processor numbers ν_n is given as

$$\nu_n = M\{n\{n\alpha\}\} + \frac{1}{2} \tag{18}$$

The shuffled nested Weyl sequence is defined by $x_n = \{\nu_n\{\nu_n\alpha\}\}$. To create multiple sequences, the k_{th} sequence is defined by

$$x_n^{(k)} = \{k\{\nu_n\{\nu_n\alpha\}\}\} \tag{19}$$

and (17) is modified to

$$X_n^{(k)} = \left\lfloor 2^{s_0} x_n^{(k)} \right\rfloor = \lfloor 2^{s_0}\{k\{\nu_n\{\nu_n\alpha\}\}\}\rfloor \tag{20}$$

In this case, s_0 was 4, $\alpha = \sqrt{2}$ and $M = 1234567$. All bits in the whole number part are tested. Two experiments were done with $q = 15$, Table 2. The $F_{K^{\pm}}(K_q^{\pm})$ in each experiment are outside of $[1\%, 99\%]$. Hence we have 99% of confidence to reject the PRNG.

5.4 Explicit Inversive Congruential Method

When used as an individual RNG, the Explicit Inversive Congruential Method [5,7,19] is defined as

$$X_n = \overline{an + c} \quad mod\ p \quad \text{for n} \geq 0 \tag{21}$$

where p is a prime and $\bar{x} = x^{-1}$ is the inverse of the element x in the finite field F_p .

As a PRNG, a different pair of a and c is assigned for each different pseudo-random sequence. So the generator for the k^{th} process is given by

$$X_n^{(k)} = \overline{a_k n + c_k} \quad mod\ p \quad \text{for n} \geq 0 \tag{22}$$

The pairs of a_k and c_k should be selected properly such that $c_0\overline{a_0}, c_1\bar{a_1}, \cdots, c_{N-1}\overline{a_{N-1}} \in F_p$ are distinct.

run	a_0	bits tested	q	K_q^+	K_q^-
(1)	2027812808	0x30000000	60	0.7	99.9
(3)	1323257245	0x30000000	60	8.7	94.0
(4)	764261123	0x38000000	60	12.9	96.3

Table 3
The parallel sequence test results for the Explicit Inversive Congruential Generator

The version of the generator used is based on the implementation by Otmar Lendl [20] with $p = 2147483647 = 2^{31} - 1$. Three experiments were performed. Table 3 lists the values of the multipliers a_0 ($a_1 = a_0$) used in each experiment and the number of replications. The additives c_0 and c_1 were $(5 * m)$ and $(5 * m + 91)$ respectively for the m^{th} pair of sequences. The 2 most significant bits (bits 28 and bit 29) in 3 experiments and the 3 most significant bits (bits 27 - 29) in the other were tested. In all experiments, the $F_{K^{\pm}}(K_q^{\pm})$ have the trend to go out of the range of $[6\%, 87\%]$. We have 87% confidence to reject it as a PRNG.

5.5 Split Multiplicative Linear Congruential Generator

The generator is based on L'Ecuyer's implementation [21] of a combination generator using two different linear congruential generators:

$$X_{j+1} = aX_j + c \quad mod\ p \quad j \geq 0 \tag{23}$$

The (a_1, a_2, c_1, c_2, p) are chosen to yield the maximum period. Then the period of the sequence is divided up into N subsequences whose starting points are far apart. Each process using the generator is assigned a non-overlapping supsequence with a period of $l = p/N$. The structure of this generator was analyzed in [22].

Three experiments were performed. One of the experiments tested the 3 most significant bits (mask 0x70000000) and the others tested the 4 most significant bits (mask 0x78000000). The results of each experiment are shown in Table 4. The $F_{K^{\pm}}(K_q^{\pm})$ are in the range of $[40\%, 60\%]$. We can say this generator passes our test.

5.6 SUN Workstation C library function, srand48

Two pseudorandom number streams were formed from *srand*48 [23], one consists of all of its odd terms $\{X_1(k) = srand48(2k-1) \mid k = 1, 2, \cdot\}$ and the

Generators	Split Combination				Srand48			
run	q	bits	K_q^+	K_q^-	q	bits	K_q^+	K_q^-
(1)	150	0x70000000	42.9	63.3	60	0x3C000000	0.3	99.0
(2)	60	0x78000000	57.0	59.7	60	0x78000000	8.1	94.3
(3)	60	0x78000000	61.4	47.9				

Table 4
The parallel sequence test results for the Split Multiplicative Linear Congruential Combination geneator and Srand48

other consists of the even terms $\{X_2(k) = srand48(2k) \mid k = 1, 2, \cdot\}$. Each stream is further split into sequences of length of 1,000,000. One sequence from each stream in the corresponding order is chosen to form a pair of sequences.

Two experiments were performed and described in Table 4. One experiment examined bits $26-29$ and in the other, bits $27-30$ were tested. The $F_{K^\pm}(K_q^\pm)$ in each experiment for $q = 60$ were outside the interval $[10\%, 90\%]$. Hence we have 90% of confidence to reject it. The generator *srand*48 was not claimed to be a PRNG [23]; so it is not surprising that it fails the empirical test.

6 Conclusions

We present a new empirical test for PRNG's based on Percus' theory. It is a generic test that can be used to study both bit stream generators and full word size generators. The test was implemented on a distributed network and used to study several published PRNG. The results confirmed the expectation that the nested Weyl generator and the C library function, *srand*48, would fail the test. Both of these generators have well-documented shortcomings, even when used as serial pseudorandom number generators and also have parallel correlations that the parallel sequence test detects. The results of the applying the parallel sequence test to R250 is also consistent with published reports. Researchers have reported both successes and failures using it in parallel simulations.

The failure of the shuffled, nested Weyl generator and the explicit inversive congruential method were not anticipated and are under further investigation. Finally, multiplicative congruential generators are known to have many types of serial correlations [24], [25]. However, when used as parallel sequences, the most significant bits do not exhibit across sequence correlations as tested by the parallel sequence test. The parallel sequence test appears to predict correlations that are important in some classes of simulations and should be a useful tool in examining new parallel pseudo-random number generators.

Acknowledgments

This work was supported in part by NSF Grant No. ASC-912 1428 and by PSC/ CUNY Grant No. 665312. P.A.W. is also supported by ONR Grant No. N00014-96-1-1-1057.

References

[1] D.E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms,* $2^n d$ edition (Addison-Wesley, Reading, MA, 1981).

[2] A. De Matteis and S. Pagnutti, A Class of Parallel Random Number Generators, *Parallel Comput.* **13** (1990) 193–198.

[3] Jian Chen and Paula A. Whitlock, Implementation of A Distributed Pseudorandom Number Generator, in: H. Niederreiter and P. J-S. Shiue, eds., *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, Lecture Notes in Statistics* **106** (Springer-Verlag, Berlin, 1995) 168–185.

[4] Mark J. Durst, Using Linear Congruential Generators for Parallel Random Number Generation, in: Edward A. MacNair, Kenneth J. Musselman, Philip Heidelberger, eds., *Proceedings of the 1989 Winter Simulation Conference* (Society for Computer Simulation, 1989) 462–466.

[5] Harald Niederreiter, New Developments in Pseudorandom Number and Vector Generation, in: H. Niederreiter and P. J-S. Shiue, eds., *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing, Lecture Notes in Statistics* **106** (Springer-Verlag, Berlin, 1995) 87–120.

[6] O.E. Percus and M.H. Kalos, Random Number Generators for MIMD Parallel Processors, *J. Parallel and Dist. Comput.* **6** (1989) 477–497.

[7] P. Hellekalek, Inversive pseudorandom number generators: concepts, results, and links, in: C. Alexopoulos, K. Kang, W.R. Lilegdon and D. Goldsman, eds., *Proceedings of the 1995 Winter Simulation Conference* (Society of Computer Simulation, 1995) 255–262.

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

[9] Michael Mascagni, Steven A. Cuccaro, Daniel V. Pryor and M.L. Robinson, A Fast, High Quality, and Reproducible Parallel Lagged-Fibonacci Pseudorandom Generator, *J. Comp. Phys.* **119** (1995) 211–219.

[10] Ora E. Percus, Testing for Correlations Between Independent Parallel Random Number Generators, New York University (1995).

[11] J.V. Uspensky, *Introduction to Mathematical Probability* (McGraw Hill, New York, 1937) 77–84.

[12] William G. Cochran, *The χ^2 Test of Goodness of Fit* (John Hopkins University, Department of Biostatistics, Paper No. 282).

[13] David M. Arnow, DP: A Library for Building Portable, Reliable, Distributed Applications, in: *Proceedings of the USENIX Winter '95 Technical Conference* (Usenix Association, New Orleans,1995).

[14] Scott Kirkpatrick and Erich P. Stoll, A Very Fast Shift-Register Sequence Random Number Generator, *J. Comput. Phys.* **40** (1981) 517–526.

[15] W.L. Maier, A Fast Pseudorandom Number Generator, *Dr. Dobb's Journal* **176** (1991).

[16] Alan M. Ferrenberg and D.P. Landau, Monte Carlo Simulations: Hidden Errors from "Good" Random Number Generators, *Phys. Rev. Lett.* **69** (1992) 3382–3384.

[17] H. Weyl, Über die Gleichverleilung von Zahlan mod. Eins, *Math. Ann.* **77** (1916) 313–352.

[18] Brad Lee Holian, Ora E. Percus, Tony T. Warnock and Paula A. Whitlock, Pseudorandom Number Generator for Massivel Parallel Molecular-Dynamics Simulations, *Phys. Rev. E* **50** (1994) 1607–1615.

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

[20] Otmar Lendl, prng2.2, A library for the generation of pseudorandom numbers, obtained January 27, 1997, http://random.mat.sbg.ac.at/ftp/pub/software/gen/

[21] Pierre L'Ecuyer and Serge Côté, Implementing a Random Number Package with Splitting Facilities, *ACM Trans. on Math. Software* **17** (1991) 98–111.

[22] Pierre L'Ecuyer and S. Tezuka, Structural properties for two classes of combined random number generators, *Math. of Compu.* **57** (1991) 735–746.

[23] SunOS 5.6 manual pages, "man srand48", 22 Jan 1993.

[24] G. Marsaglia, Random numbers fall mainly in the planes, *Proc. Nat. Acad. Sci. USA* **60** (1968) 25–28.

[25] P. L'Ecuyer, R. Simard and S. Wegenkittl, Sparse serial tests of uniformity for random number generators, preprint, 1998.