

Parallel Computations Reveal Hidden Errors of Commonly Used Random Number Generators

Hyo Ashihara, Ai Kuramoto[†]

Department of Computer Science, Kumamoto University

[†]*Currently with Software Vision
Kumamoto City, Kumamoto, Japan*

Isaku Wada, Makoto Matsumoto

Department of Mathematics, Hiroshima University

Higashihiroshima City, Hiroshima, Japan

Hironori Kikuchi[‡], Masato Kiyama

Department of Computer Science, Kumamoto University

[‡]*Currently with I-System Service
Kumamoto City, Kumamoto, Japan*

Abstract

This paper reports that in parallel Monte-Carlo simulations of the 2D Ising-Model, commonly used pseudo-random number generators (PRNG) lead to manifestly erroneous results.

When parallel random number sequences for a parallel simulation are generated by a same PRNG with different initial seeds, the sequences can be strongly correlated with each other if the seeds are selected imprudently.

The error is due to some dependencies of the outputs of the PRNG on the initial seeds. This type of defect is not so problematic in non-parallel computations, but becomes serious in parallel situations. Special care is required when using such PRNG in parallel computations.

Keywords: Monte-Carlo simulation, pseudorandom number generator, parallel computation, interstream correlation, Ising Model, cluster test

1. Introduction

The reliability of Monte-Carlo simulations depends on the randomness of the pseudo-random number generators (PRNG's) used. Such PRNG's have been studied intensively. Many defective generators were used in the past, but these days modern common generators are widely believed to have only minor defects.

We report that this is not the case for parallel computations, by showing dramatic failure in simulating Ising-Model simulations. So far, the PRNG tests mainly consider one single stream of the outputs of a PRNG. In parallel computations, many distinct streams of random numbers are generated by several PRNG's. Often one same algorithm is selected for many PRNG's, and a distinct initial seed is seeded to each PRNG. Thus, dependency of the outputs on the initial seed becomes matters, when the initial seeds are systematically selected.

We conducted a test of randomness called the

cluster test, introduced in [13], to the standard C random number generator in a parallel computing situation. We found strong correlation between the parallel streams of the pseudo-random numbers, if the seeds are systematically given. Most of known defects of randomness have been found in the lower significant bits of generated numbers, and the most significant bits are commonly believed to have good randomness. We found strong correlations between most significant bits of the parallel streams.

2. Cluster test

While there are a number of PRNG tests, till there remain no decisive tests of randomness. This seems due to the lack of a practical definition of randomness.

The cluster test[13] is a test based on the 2D Ising Model, proposed by Vattulainen et.al. Let L be a sufficiently large integer, and consider two dimensional lattice points of size $L \times L$. Suppose that one atom is fixed at each lattice point, and each atom can take two possible states: 0 or 1 (in physics, these states are called the spin). This is a model of phase transition. In finite temperature, the adjacent atoms interact and tend to have the same spin, but here we consider the infinite temperature, where each atom independently takes each state with a probability $1/2$. Thus, the state of the system is uniformly distributed among all the possible 2^{L^2} states.

Take a state of the system. A connected component of atoms with the same spin in the state is called a *cluster*. Its *size* is the number of atoms in it. Then the system is partitioned into a finite set of clusters. The number C_s of clusters of size s in the system is a random variable. In the limit $L \rightarrow \infty$, the distribution of the size s of clusters was studied [11] and explicit values for $s = 1, 2, \dots, 17$ were obtained[12].

The cluster test investigates a PRNG by creating a state by pseudo-random bits from the PRNG, and comparing the observed distribution of the size

of clusters with the theoretical one. This test is very sensitive to dependencies of the bits in medium-size intervals. Because the test itself is a simple typical Monte-Carlo simulation in physics, failure in this test implies that the PRNG may well yield erroneous results in similar simulations in physics.

3. Cluster test in parallel simulation

Parallelism in Monte-Carlo simulations is now an important topic, because of the tendency of larger scale simulations in distributed/parallel computing systems.

One simple way of doing this is to assign a PRNG to each actual processor. However, this breaks the portability of the simulation. Often it is required that a simulation can be reconstructed independently of the number of actual processors, and in this case PRNG's are assigned not to actual processors but to each virtual processor handling one atom (or one sub-system) in the simulated system.

In such simulations, correlations of parallel output streams of PRNG's become the matters. Since the number of virtual processors may be huge, often we use one same type of PRNG with distinct initial seeds. For example, we may select the ID number of each virtual processor as a seed, and thus in the cluster test, the coordinate of each lattice point. Sequences generated from different seeds are expected to be totally independent. If they are not, the correctness of simulations becomes doubtful.

Warnings on the danger of interstream correlation have been given in many places (e.g. [4], [10]), but to our knowledge, known problems are (i) incidental overlapping of sequences by birthday-collisions and (ii) long-range correlations [7] [8] which become short-range correlations in parallel situations. They are considered not to be so serious in a short simulation, whereas we report a dramatic failure even in a very short usage of random number sequences.

4. PRNGs in C library

We selected two standard PRNG's in the C-language library: `random()` and `lrand48()`. The standard PRNG in the C library is named `rand()`, but its implementation depends on the system, and it has often been replaced. In the old BSD UNIX, a linear congruential generator

$$x_{1+j} = 110351515245x_j + 12345 \pmod{2^{31}}$$

was used. This generator is unsatisfactory by today's standard. In particular, due to the well known fact that the lower k bits of x_j have period 2^k , it has poor randomness for small k .

As of April 2006, `rand()` in the GNU C library is identical to `random()`, which is one of the lagged-Fibonacci generators. As a default, it generates a pseudo-random 31-bit integer sequence by the recursion

$$x_{31+j} = x_{28+j} + x_j \pmod{2^{31}}.$$

The initial values x_0, x_1, \dots, x_{30} were given from initial seed x_0 by the linear congruential method

$$x_{1+j} = 1103515245x_j + 12345 \pmod{2^{32}}$$

in early versions of `random()`. This initialization scheme is replaced with

$$x_{1+j} = 16807x_j \pmod{(2^{31} - 1)}$$

in `glibc2`. In the following simulation, this current version of `random()` is used. We also tested the older version and found it to be more erroneous.

Another standard PRNG in `gcc` is the `drand48()` family, which is a linear congruential generator with 48 bit integers. `lrand48()` is one of the functions in the family which returns a non-negative long integer. The recurrence is

$$x_{1+j} = 25214903917x_j + 11 \pmod{2^{48}}$$

and the most significant 31 bits are used. Discarding the lower significant bits improves the lower-bits problem mentioned above. This generator is now rather obsolete, but it is still widely used.

5. Experimental Results

Here we evaluate `random()` and `lrand48()` by the cluster test both in sequential and parallel situations.

Each of these two generators is used for a 2D Ising-Model simulation with $L \times L$ lattice and periodic boundary condition, in the following two ways. Select an integer m as the initial seed for the entire simulation, and then the state of the atom at (i, j) ($0 \leq i, j < L$) is determined to be

- (A) sequential: the k -th bit ($k = 0$:LSB, $k = 30$:MSB) of the $(i + jL + nL^2)$ -th output of a single PRNG with initial seed m
- (B) parallel: the k -th bit of n -th output of the PRNG with initial seed $(i + jL + m)$.

By moving $n = 0, 1, \dots, N - 1$, we obtain N distinct random states. We enumerate the number of the clusters of size s for $s = 1, 2, \dots, 17$ and obtain the average cluster size \bar{C}_n for each n , i.e. for an arbitrary atom a , as expected for a cluster size that contains a . (If the size is greater than 17, it is treated as zero.) Then we compute the mean \bar{C} and the standard deviation σ of \bar{C}_n among N iterations. C_T , the theoretical value of \bar{C} , is about 2.51. We obtain $\Delta = C_T - \bar{C}$.

The result of (A) is listed in Table 1. The initial seed m is 1, $L = 100$ and the number of iterations is $N = 100$. Both `random()` and `lrand48()` conform well to the theoretical expectations derived from [11], while old BSD `rand()` shows abnormal behaviour at lower bits in Table 2, as is already known.

If L is too small, the finiteness of the system will produce errors. Table 3 shows the values for $L = 10, 20, 50, 200, 500$ for the most significant bit (i.e. $k = 30$) of `random()`. To uniformize the average sample sizes, we selected the number of iterations N to be $10^6/L^2$. As L increases, both Δ and σ decrease. $L = 100$ is shown to be adequate.

Next, the parallel simulation (B) is executed. Table 4 shows its results. The 25th-29th bits in

Table 1: Sequential cluster test

k	random()		lrand48()	
	Δ	σ	Δ	σ
0	-2.50e-3	1.83e-1	-8.93e-3	1.78e-1
1	2.77e-2	1.99e-1	-3.77e-2	1.74e-1
2	1.46e-3	1.92e-1	2.73e-2	2.04e-1
3	-3.83e-2	1.85e-1	1.74e-2	1.88e-1
4	-1.55e-3	1.70e-1	1.13e-2	1.91e-1
5	3.77e-2	2.02e-1	1.82e-2	1.91e-1
6	9.27e-3	1.85e-1	-2.59e-2	1.86e-1
7	3.96e-2	1.53e-1	-1.77e-2	1.72e-1
8	1.41e-2	1.82e-1	6.02e-3	1.76e-1
9	1.74e-2	1.66e-1	2.15e-2	1.75e-1
10	-1.42e-2	1.81e-1	8.88e-3	2.01e-1
11	-1.44e-2	1.72e-1	1.89e-2	1.57e-1
12	6.20e-2	1.87e-1	1.74e-2	1.88e-1
13	1.49e-2	1.61e-1	-4.10e-3	1.77e-1
14	3.77e-3	1.95e-1	3.39e-2	1.81e-1
15	2.70e-3	1.80e-1	2.46e-2	2.13e-1
16	-8.70e-3	1.94e-1	1.05e-2	1.70e-1
17	-1.43e-2	1.74e-1	-2.60e-2	1.67e-1
18	-3.30e-3	1.93e-1	-8.94e-3	1.98e-1
19	5.68e-3	1.72e-1	3.42e-2	1.58e-1
20	-4.71e-3	1.67e-1	3.22e-2	1.81e-1
21	-1.04e-2	1.54e-1	-2.00e-2	1.80e-1
22	1.09e-2	1.96e-1	-1.61e-2	1.78e-1
23	3.86e-3	1.85e-1	2.34e-2	1.71e-1
24	7.12e-3	2.04e-1	2.64e-2	1.55e-1
25	3.13e-2	1.81e-1	1.23e-2	1.77e-1
26	-9.90e-4	1.93e-1	-1.66e-3	1.67e-1
27	3.41e-3	1.75e-1	2.07e-2	1.79e-1
28	-1.25e-3	1.71e-1	7.26e-3	1.89e-1
29	4.11e-2	1.76e-1	3.36e-2	1.55e-1
30	3.52e-2	1.77e-1	-3.69e-4	1.59e-1

Table 2: Cluster test for a defective PRNG

k	old rand()	
	Δ	σ
0	2.51e+0	0.00e+0
1	2.51e+0	0.00e+0
2	-1.81e+0	1.77e-7
3	-5.37e+0	1.16e-7
4	2.39e+0	5.00e-9
5	1.70e+0	5.23e-3
6	1.55e+0	9.98e-3
7	-2.22e+0	1.19e-2
8	4.76e-1	2.46e-2
9	6.99e-1	3.77e-2
10	-3.42e-1	5.20e-2
11	-4.51e-1	6.94e-2
12	-3.12e-1	7.08e-2
13	8.89e-2	9.05e-2
14	-6.35e-2	1.01e-1
15	7.86e-2	1.59e-1
16	-1.30e-1	1.78e-1
17	-1.69e-2	1.68e-1
18	1.94e-2	1.67e-1
19	-1.06e-2	2.06e-1
20	2.21e-2	1.75e-1
21	1.49e-2	1.74e-1
22	-3.17e-3	1.62e-1
23	-3.23e-3	1.93e-1
24	-1.55e-3	2.00e-1
25	4.62e-3	1.73e-1
26	1.21e-2	2.01e-1
27	-8.23e-6	1.76e-1
28	-1.06e-3	1.81e-1
29	-4.32e-3	1.82e-1
30	1.55e-2	1.76e-1

Table 3: Finite-size effects

L	Δ	σ
10	3.30e-1	1.71e+
20	7.08e-2	8.77e-1
50	2.89e-2	3.56e-1
100	3.52e-2	1.77e-1
200	-7.12e-3	7.10e-2
500	4.23e-3	1.92e-2

Table 4: Parallel cluster test

k	rand()		lrand48()	
	Δ	σ	Δ	σ
0	-1.43e-2	1.82e-1	2.51e+0	0.00e+0
1	5.20e-3	1.65e-1	-2.68e-1	1.27e+0
2	3.25e-2	1.81e-1	-2.81e-1	3.34e+0
3	1.52e-2	1.79e-1	-1.44e+0	4.75e+0
4	2.58e-3	1.84e-1	1.15e+0	1.94e+0
5	-2.28e-3	1.75e-1	6.87e-1	2.63e+0
6	-2.41e-2	1.82e-1	8.17e-1	2.65e+0
7	7.52e-3	1.76e-1	3.62e-1	2.99e+0
8	1.27e-2	1.78e-1	9.60e-1	2.43e+0
9	1.55e-2	2.10e-1	4.35e-1	2.91e+0
10	2.11e-2	1.93e-1	-1.44e-1	3.42e+0
11	1.96e-2	1.69e-1	8.89e-1	2.62e+0
12	-2.38e-2	1.77e-1	7.09e-1	2.83e+0
13	1.15e-2	1.68e-1	4.58e-1	3.02e+0
14	-5.40e-3	1.74e-1	1.90e-1	3.32e+0
15	-7.92e-5	1.71e-1	-2.46e-2	3.23e+0
16	-1.51e-2	2.06e-1	4.43e-1	2.74e+0
17	2.24e-2	1.57e-1	-4.06e-1	3.66e+0
18	-3.51e-2	1.79e-1	3.87e-1	3.02e+0
19	-1.77e-2	2.04e-1	6.67e-1	2.53e+0
20	-1.14e-4	1.89e-1	4.68e-1	2.96e+0
21	-2.52e-2	1.78e-1	3.22e-1	2.87e+0
22	-4.52e-3	1.78e-1	6.18e-1	3.14e+0
23	-7.03e-4	1.75e-1	5.58e-1	2.42e+0
24	-2.41e-2	2.81e-1	3.34e-1	3.05e+0
25	-4.86e-2	7.53e-1	1.02e+0	2.25e+0
26	1.26e-1	1.01e+0	8.70e-1	2.71e+0
27	-1.51e-1	1.32e+0	5.97e-1	2.81e+0
28	1.65e-1	1.48e+0	6.00e-1	2.54e+0
29	-3.94e-2	2.07e+0	2.99e-1	3.01e+0
30	3.13e-3	1.85e-1	7.11e-1	2.69e+0

random() and all bits in lrand48() are obviously erroneous. Both Δ and σ are far from the expected values, and it turns out that for many (k, n) there exist NO clusters of a size less than or equal to 17. We conducted the same simulation for several different parameter values L, N and m , and obtained similar results.

One may claim that the initialization in (B) is too naive. However, the symptom is observed whenever the seeds are selected by some linear function. Such a situation will often occur if programmers do not have adequate knowledge and vigilance, which is unnecessary if the PRNG is correct.

The problem is not limited to random() and lrand48(). Many modern PRNGs which are widely believed to be reliable, such as ran_array [2, P.186], RAN3[9, P.283] and ranlux[3][1], have the same defects.

6. Conclusion

We showed that the standard PRNG's yield dramatically erroneous results in a parallel Monte-Carlo simulation, if their initial seeds are selected in a simple manner. This phenomenon can be avoided by selecting the seeds more carefully, or by using more sophisticated parallelizing techniques like parameterizing [4][6]. However, these burden programmers unnecessarily. Another simpler solution might be to use more reliable PRNG's, such as Mersenne Twister [5].

We are now testing other PRNG's using this type of simulation, and analyzing the reason for these dependencies theoretically.

References

- [1] F. James. Ranlux: A fortran implementation of the high-quality pseudorandom number generator of Lüscher. *Computer Physics Communications*, 79(1):111–114, February 1994.
- [2] Donald E. Knuth. *The Art of Computer Programming. Vol.2. Seminumerical Algorithms*. Addison-Wesley, Reading, Mass., 3 edition, 1997.
- [3] M. Lüscher. A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Communications*, 79(1):100–110, February 1994.
- [4] M. Mascagni and A. Srinivasan. Parameterizing parallel multiplicative lagged-Fibonacci generators. *Parallel Computing*, 30(7):899–916, July 2004.
- [5] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation*, 8(1):3–30, January 1998.
- [6] M. Matsumoto and T. Nishimura. The dynamic creation of distributed random number generators. In *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing*, 1999.
- [7] A. De Matteis and S. Pagnutti. Parallelization of random number generators and long-range correlations. *Numerische Mathematik*, 53:595–608, 1988.
- [8] A. De Matteis and S. Pagnutti. Long-range correlations in linear and non-linear random number generators. *Parallel Computing*, 14(2):207–210, June 1990.
- [9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 2 edition, 2002. Acrobat Edition, version 2.10.
- [10] SPRNG. Sprng home. <http://sprng.cs.fsu.edu/>.
- [11] M. F. Sykes and M. Glen. Percolation processes in two dimensions. i. low-density series expansions. *Journal of Physics A: Mathematical and General*, 9(1):87–96, January 1976.
- [12] I. Vattulainen. New tests of random numbers for simulations in physical systems. Report series in theoretical physics HU-TFT-IR-94-4, Reserch Institute for Theoretical Phisics, University of Helsinki, November 1994. cond-mat@babbage.sissa.it/No.9411062.
- [13] I. Vattulainen, T. Ala-Nissila, and K. Kankaala. Physical models as tests of randomness. *Physical Review E*, 52(3):3205–3214, September 1995.