Report on
Random Number Generation for Petascale Quantum Monte Carlo

Summary: The quality of random number generators can affect the results of Monte Carlo computations [2], especially when a large number of random numbers are consumed. Furthermore, correlations present between different random number streams in a parallel computation can further affect the results [2]. The SPRNG software, which the author had developed earlier, has pseudo-random number generators (PRNGs) capable of producing large numbers of streams with large periods. However, they had been empirically tested on only thousand streams earlier [2]. In the work summarized here, we tested the SPRNG generators with over a hundred thousand streams, involving over 10^{14} random numbers per test, on some tests. We also tested the popular Mersenne Twister. We believe that these are the largest tests of PRNGs, both in terms of the numbers of streams tested and the number of random numbers tested. We observed defects in some of these generators, including the Mersenne Twister, while a few generators appeared to perform well. We also corrected an error in the implementation of one of the SPRNG generators.

1. SPRNG MLFG bug: SPRNG can use either 32-bit integer types or 64-bit integer types for representing the underlying states of the PRNGs. A bug was discovered in the 64-bit version of the MLFG. This does not affect the quality of the random numbers generated. However, some of the random number streams in the 64-bit version will not be identical to those in the 32-bit version. Line 163/164 of *SRC/mlfg/mlfg.c* should be `unsigned int i, j, k, temp[2], length`, instead of `int i, j, k, temp[2], length`. This fixes the error.

2. Tests results: Most of the tests in the SPRNG test suite were used to test the SPRNG generators – 48-bit LCG, 64-bit LCG, Additive LFG, Multiplicative LFG, and CMRG – and also a parallel Mersenne Twister[1]. Some of the tests, which had not been very effective in detecting PRNG defects in [2], were not used. The sequential and parallel tests used are mentioned below, along with their parameters, numbers of streams, and number of random numbers tested. The details of the tests and meaning of the parameters are available in [2]. Each SPRNG generator comes with several variants, which are distinguished by a value of one parameter to that generator. In the tests below, we used the default parameter, which typically gives a variant with the best quality. The exception was the multiplicative LFG, in which we use the variant with the smallest state space, which also implies the worst quality. However, even with this, this generator typically has very good quality and had passed all of Le'cuyer's tests [1], which most other generators, including the Mersenne Twister, had failed.

The parallel tests interleaved 100 streams at a time to form a new stream, and tested the new streams with the usual statistical tests. (The 'blocking/sum' test and the Ising model application-based tests – Metropolis and Wolff algorithms – were exceptions.) We also modified the existing SPRNG implementation of the above two application-based tests to permit greater parallelism, in order to make more effective use of the Jaguar machine.

[1] Mersenne Twister. The parallel implementation was based on the one by Geoff Kuenning, which can be obtained from http://www.lasr.cs.ucla.edu/geoff/mtwist.html.

We test one or more blocks of random numbers from each stream. We can compare the result from each block of number with the value expected from a truly random sample from the uniform distribution. We typically get some percentile from this, which indicates, in some sense, the fraction of truly random samples that are likely to have a smaller deviation from the true expected value. For example, we may obtain the percentile from the chi-square statistic. We may wish to consider blocks with percentiles lower than 2.5 % or greater than 97.5 % (5 % of all streams) as defective. However, in a large test, several blocks (5 % of the total) can fall outside these thresholds, even for a truly random sample. We wish to determine if the distribution of percentiles is abnormal. We therefore, typically, perform a Kolmogorov-Smirnov test to determine if the distribution of percentiles is abnormal. If the percentile in this is below 2.5% or above 97.5%, then we consider the test as having failed. However, when we perform a large number of tests, we can expect some tests (5% of the total) to fail. In order to determine if the random number generator is truly defective, we repeat the test with a different random number seed, when the streams fail the test. If the tests fail with a different seed too, then it is quite likely that the random number generator is truly defective with respect to that test.

Test	Parallel /Serial	Parameters	# of streams	# of RNs
Blocking	Parallel	28000, 1, 0, 0, 1, 0, 1000000, 1024	28000	2.8×10^{13}
Collisions	Serial	28000, 1, 0, 0, 100, 0, 200000, 4, 7	28000	2.2×10^{12}
Collisions	Parallel	280, 100, 0, 0, 100000, 0, 200000, 4, 7	28000	2.2×10^{13}
Coupon	Serial	28000, 1, 0, 0, 1, 0, 5000000, 30, 10	28000	$\approx 4\times10^{12}$
Coupon	Parallel	280, 100, 0, 0, 100, 0, 5000000, 30, 10	28000	$\approx 4\times10^{12}$
Gap	Serial	28000, 1, 0, 0, 1, 0, 200, 0.5, 0.51, 1000000	28000	2.8×10^{12}
Gap	Parallel	280, 100, 0, 0, 1000, 0, 200, 0.5, 0.51, 1000000	28000	2.8×10^{13}
Maxt[2]	Serial	28000, 1, 0, 0, 100, 0, 50000, 16	28000	2.2×10^{12}
Maxt	Parallel	280, 100, 0, 0, 10000, 0, 50000, 16	28000	2.2×10^{12}
Perm	Serial	28000, 1, 0, 0, 1, 0, 7, 14000000	28000	2.7×10^{12}
Perm	Parallel	280, 100, 0, 0, 1000, 0, 7, 14000000	28000	2.7×10^{13}
Poker	Serial	28000, 1, 0, 0, 1, 0, 9000000, 10, 10	28000	2.5×10^{12}
Poker	Parallel	280, 100, 0, 0, 100, 0, 9000000, 10, 10	28000	2.5×10^{12}
Runs	Serial	28000, 1, 0, 0, 1, 0, 10, 50000000	28000	2.1×10^{12}
Runs	Parallel	280, 100, 0, 0, 100, 0, 10, 50000000	28000	2.1×10^{12}
Serial	Serial	28000, 1, 0, 0, 1, 0, 100, 50000000	28000	2.8×10^{12}
Serial	Parallel	280, 100, 0, 0, 100, 0, 100, 50000000	28000	2.8×10^{12}
Metropolis	Parallel	0, 0, 16, 1000, 100, 1000000, lattices=513	131,328	$\approx 1.3\times10^{14}$
Wolff	Parallel	0, 0, 16, 1000, 100, 1000000, lattices=513	131,328	$\approx 10^{14}$

Table 1: *Summary of tests*

Test	Parallel	MLFG	LFG	CMRG	LCG	LCG64	MT

[2] In this test, we have taken 16 random numbers at a time, and determined the largest. The cumulative distribution for a truly random sequence should be x^{16}, $0 \le x \le 1$.

	/Serial						
Blocking	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Collisions	Serial	Passed	Passed	Passed	Passed	Passed	Passed
Collisions	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Coupon	Serial	Passed	Passed	Passed	Passed	Passed	Passed
Coupon	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Gap	Serial	Passed	Passed	Passed	Passed	Passed	Passed
Gap	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Maxt	Serial	Failed	Failed	Failed	Failed	Failed	Failed
Maxt	Parallel	Failed	Failed	Failed	Failed	Failed	Failed
Perm	Serial	Passed	Passed	Passed	Passed	Passed	Passed
Perm	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Poker	Serial	Passed	Passed	Passed	Passed	Passed	Passed
Poker	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Runs	Serial	Passed	Passed	Passed	Passed	Passed	Passed
Runs	Parallel	Passed	Passed	Passed	Failed	Failed	Failed
Serial	Serial	Passed	Passed	Passed	Passed	Passed	Passed
Serial	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Metropolis	Parallel	Passed	Passed	Passed	Passed	Passed	Passed
Wolff	Parallel	Passed	Passed	Passed	Passed	Passed	Passed

Table 2: Summary of PRNG results

The parallel *runs* test was failed by the Mersenne Twister and both the SPRNG LCGs. The SPRNG LCGs failed this test with several different values of the parameters. This is not entirely unexpected, because these generators have certain parallel correlations [2] that can cause errors. Earlier tests were not large enough to detect these errors, while the new ones are. The Mersenne Twister failed with four of six seeds on this test. The reason for this is that the parallelization of the Mersenne Twister does not guarantee that each stream is independent. The *maxt* test was failed by all PRNGs. This is discussed in further detail below.

3. *Maxt* results: We discuss results with a multiplicative LFG (MLFG) as example. Other generators show similar trends. Both parallel and sequential tests fail, with both small and large lags. In fact, all SPRNG generators fail this test, as does the Mersenne Twister.

We created the histogram for the Kolmogorov-Smirnov percentiles for the individual results from the original sequential test (each data point is the percentile for the combined result of ten streams). In figure 1, we compare the cumulative distribution for the histograms with the exact distribution. We can see that the cumulative distribution obtained is always below the exact one (except at 0 and 1). This suggests that the number of streams with a percentile close to 100 is higher than it ought to be. Note that no stream actually has a percentile of 100 (though some come close to it), as often happens when a test fails badly. Instead, we have a situation where the distribution of the percentiles is not satisfactory.

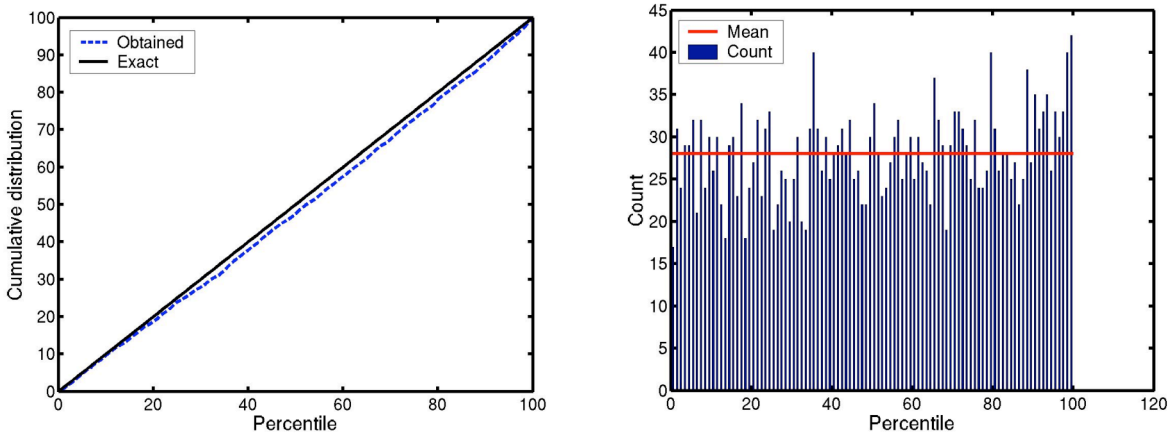

Figure 1: *(Left) Cumulative distribution function for percentiles. (Right) Histogram for percentiles.*

We need to find out if there are some streams that are consistently bad, in which case, it may be possible to eliminate them. In figure 2 (left), we try to get a qualitative idea of whether there is an pattern in the streams that have poor quality. For example, are the lower numbered streams bad, or are the higher numbered streams bad? The results here do not show any particular trend. Next, we wish to see if there are particular streams that are bad. If we could identify certain bad streams, then we could omit them, and obtain a random number generator with good streams. We expect that a bad stream would give bad results consistently, even if choose different subsequences. In fig. 2 (right), we plot the percentile for each stream group with two different subsequences used along the x and y direction respectively. Bad streams would give poor results in both cases (points on the top right corner).

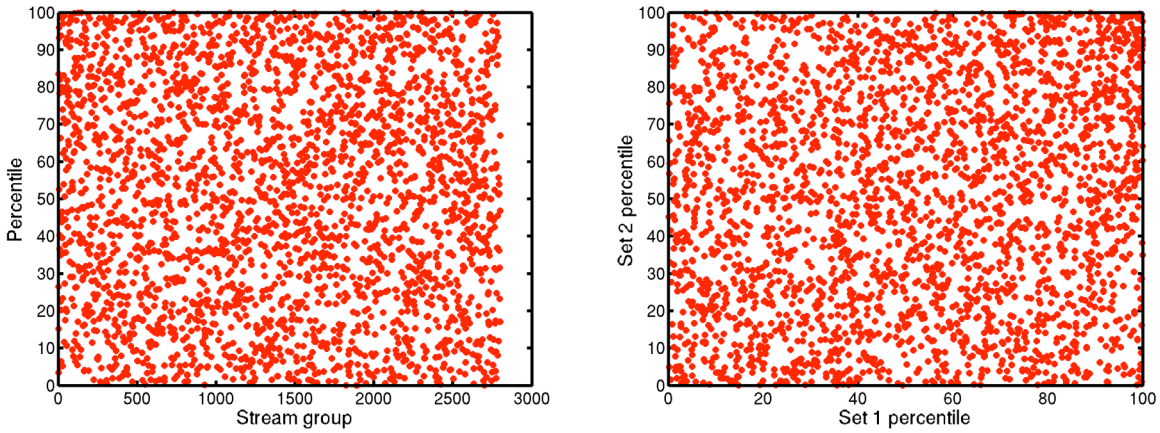

Figure 2: *(Left) Percentile for each group of streams. Each group of streams consists of ten streams. (Right) Correlation between subsequences of the same streams. Set 1 contains the results for each stream group of the MLFG. Set 2 contains the results of the same streams, except that we skip 10,001 random numbers between each block.*

When we test a large number of streams, one would expect a few streams to have percentiles close to 100 in both sets of tests, even if the streams were truly random. For example, we can expect 1% of the streams to have percentiles over 90% on each of the two tests. That is, we

would expect 28 streams groups to have percentiles of 90% in both the tests and seven stream groups to have percentiles over 95% on both. In practice, we obtain 70 and 23 streams having these properties, which is a little larger than expected. This is reflected in more points in the top right, compared with other regions. If the streams were truly random, then we would expect correlation coefficient close to 0 between the two sets of results. However, the correlation coefficient is 0.12, which is a little higher than what we would expect. We next wish to identify which streams are truly bad, by performing more sets of tests, and seeing which streams are consistently bad. We performed four additional sets of tests, with gaps of 10K, 20K, 30K, and 40K between blocks respectively. However, only one group of streams was consistently bad in all cases (with thresholds of either 95% or 97.5%). Omitting this group of streams does not significantly affect the results.

Consequently, it appears that the results are poor because the set of streams taken together perform a little worse than would be expected, and not because there are a few streams in particular that give poor results. Considering that all generators fail this test, we need to examine further whether this is due to the inherent nature of this tests with large number of random numbers used, or due to approximations in the test implementation, because it is unusual for a single test to be sensitive to the different types of correlations in the different generators.

References

1. P. L'Ecuyer and R. Simard. *TestU01: A C library for empirical testing of random number generators*, ACM Transactions on Mathematical Software, Vol. 33, 2007.

2. A. Srinivasan, M. Mascagni, and D.M. Ceperley, *Testing parallel random number generators*, Parallel Computing, Vol. 29, 2003, 69-94.