Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

USING LINEAR CONGRUENTIAL GENERATORS FOR PARALLEL RANDOM NUMBER GENERATION

Mark J. Durst
Lawrence Livermore National Laboratory
Livermore, CA 94550, U.S.A.

ABSTRACT

Linear congruential random number generators are widely used in simulation and Monte Carlo calculations. Because they are very fast, and because they have minimal state space, they remain attractive for use in parallel computing environments. We discuss their use as a source for many streams of pseudo-random numbers. Many authors have discussed splitting the stream of a single CRNG into many substreams; we show spectral calculations for this scheme and compare randomly and regularly spaced selection of starting points. Several authors have suggested using a common multiplier for all streams and a unique additive constant for each. The discrepancies of such schemes are no better than for splitting schemes; we show how they are in a sense equivalent. We also consider the use of distinct multipliers for each stream. Good multipliers are abundant for large enough moduli, but little is known about the multidimensional behavior of such generators. We discuss the use of improvement techniques and larger moduli to overcome the limitations of linear congruential generators.

1. INTRODUCTION

Congruential random number generators (CRNGs) are perhaps the best known and most popular method for generating a stream of pseudo-random numbers. Their limitations are well-known (see, e.g., Marsaglia (1968)), but this is just another way of saying that their structure is well understood. The multiplier for a CRNG can be chosen to provide good behavior in low dimensions (see Fishman 1978, Chapter 8). This, combined with the fast generation speed and minimal state space associated with CRNGs, continues to make them attractive to Monte Carlo and simulation practitioners.

For parallel and distributed computations which use pseudo-random numbers, these advantages of CRNGs will continue to hold. Reproducibility, the guarantee that different runs of a program will give the same result, can be assured if each task or processor has allotted to it a unique state space vector for the pseudo-random number generator. This makes it clear that small state space is still a concern, at least for computations which have many times more tasks than processors. It is also clear that speed is no less a concern in parallel and distributed computations. Finally, the clear structure of CRNGs makes analyses of some strategies for parallel random number generation possible, as will be seen below. Such analyses are not yet available for any other random number generators.

There are at least two ways to use CRNGs to produce many high-quality streams of pseudo-random numbers. One can simply split the sequence from a single CRNG into many small subsequences; these splitting methods will be evaluated below by an extension of the usual spectral apparatus. One can also attempt to provide many distinct generators, one for each stream. Theory for this is not yet complete, but the method has promise.

2. SPLITTING

A splitting method is defined by giving different starting points on a single CRNG for each substream. That is, we have

$$X_{i,n+1} \equiv aX_{i,n} + c \pmod{m},$$

with distinct starting values $X_{i,0}$ (uniform random numbers in the interval $[0,1]$ are obtained as $U_{i,n} = X_{i,n}/m$). Each substream can be assigned a unique set of numbers, stretching from its starting point along the sequence of the CRNG to the next starting point encountered, and we can choose to permit or forbid overlaps. With this framework we can analyze the behavior of t-tuples $(U_{1,n},\ldots,U_{t,n})$ through a variant on the spectral test.

The standard spectral test of Coveyou and MacPherson (1967; a modern treatment is in Knuth 1981, Section 3.3.4) analyzes the behavior of $(U_n, U_{n+1},\ldots,U_{n+t-1})$ by finding the most widely spaced system of parallel hyperplanes which cover all t-tuples of the above form. The maximum hyperplane spacing is $1/\nu_t$, where

$$\nu_t^2 = \min_{\substack{\vec{x}\in \mathbf{Z}^t \\ \vec{x}\neq\vec{0}}} \{\|\vec{x}\|^2 \mid \vec{x}\cdot\vec{w} \equiv 0 \pmod{m}\},$$

with $\vec{w} = (1, a, a^2, \ldots, a^{t-1})$. Low hyperplane spacing corresponds to low integration error (Niederreiter 1978), and Knuth (1981, p. 91) comments that t-tuples characterized by ν_t "will behave essentially as if we took truly random numbers and truncated them to $\log_2 \nu_t$ bits."

Suppose that n_i indexes the starting point of the i-th sequence with respect to the first stream; i.e., $X_{i,0} = X_{1,n_i}$. Then the theory for our analysis of $(U_n, U_{n+1}, \dots, U_{n+t-1})$ directly follows Knuth (1981, pp. 92–94), but with $\vec{w} = (1, a^{n_2}, \dots, a^{n_t})$. It should be remarked that we will not calculate ν_t for large values of t. Since the computational cost of the spectral test goes as $\mathcal{O}(3^t)$, it would be impractical. Further, since $\log_2(\nu_t)$ is roughly bounded by $\log_2(m)/t$ (Marsaglia 1968), we know that the result of the spectral test for large t will be disappointing. We can at best hope to achieve good behavior across all small subsets of the total number of streams. It should also be remarked that an identical theory can analyze within-stream and across-stream behavior simultaneously. For instance, to analyze the 4-tuples $(U_{1,n}, U_{1,n+1}, U_{2,n}, U_{2,n+1})$, we would use $\vec{w} = (1, a, a^{n_2}, a^{n_2+1})$.

While this theory can analyze any splitting method, it is superfluous for some very badly designed methods. For instance, if m is a power of 2 (as is generally used with vectorized generators), and 2^d starting points are distributed precisely evenly along the sequence of a given CRNG, the resulting 2^d streams differ only in their d most significant bits. Further, for each $c \leq d$, a given stream has 2^c companion streams which differ only in their c most significant bits, including one which simply has the highest bit reversed.

2.1. Random Spacing

The failure of precisely even subdivision suggests that randomly located starting points will be better. Indeed, randomly located starting points do seem to have good (unlagged) spectral behavior. We took the sequence of RANF ($m = 2^{48}$, $c = 0$, $a = 44485709377909$) and generated 1000 random starting points and their two-dimensional spectral values with respect to a fixed starting point. The average value of $\log_2(\nu_2)$ (with units roughly corresponding to bits of randomness) was 23.3, with a standard deviation of 0.71; with a maximum of about 24 bits of randomness possible, over 90% lost less than two bits of randomness, and all 1000 had more than 19 bits of randomness. By contrast, a precisely even split of RANF into 2^{10} streams averaged 7.5 bits of randomness.

But while randomly located starting points show good spectral behavior, there are problems with their practical use. The greatest problem is in the spacing of randomly located starting points. In certain applications, eventual overruns of one stream into another stream's numbers must be forbidden. In this case, each stream can only be allotted the numbers up to the next starting point. With random starting points, some streams will have extremely small allocations. If we approximate k starting points by random points on a circle whose circumference M is the period of the generator, it can be shown (cf. David 1981, pp. 98–99) that the minimum substream length W has the approximate distribution

$$\Pr(W > Mt) = (1 - kt)^{k-1},$$

with expected value M/k^2. Thus if we randomly split the generator RANF (with period $M = 2^{46}$) into 2^{10} streams, we can expect a minimum stream length on the order of 2^{26}, which is acceptable; but if we need 2^{20} streams, the minimum stream length will be on the order of 64, which is not acceptable. That is, randomly located starting points will work for computations with a moderately large number of streams, but will be problematic for computations with a very large number of streams.

2.2. Regular Spacing

For a case requiring many millions of streams, we returned to regularly spaced starting points, but without evenly splitting the sequence (evenly splitting the sequence would probably work well with prime modulus, but our demands for vectorized generation dictated $m = 2^{48}$). We used the same basic generator RANF and separated starting points by 1,000,001. Among the first 1000 starting points, the mean value of $\log_2(\nu_2)$ was 22.3, with a standard deviation of .74. The distribution is quite uniformly shifted down a bit from random starting points; 90% of the comparisons are within three bits of the maximum possible randomness, and all have more than 18 bits of randomness. We have constructed other regularly spaced algorithms which evenly divide up 70 billion numbers from the sequence of RANF (over 99% of its total period) into a user-specified number of streams, and the behavior of these is very similar. It seems that regularly spaced starting points must lose about a bit of randomness compared with randomly spaced starting points in two dimensions (we do not yet have definitive figures in higher dimensions).

The considerations above would seem to dictate the use of randomly spaced starting points when the number of streams required is relatively small (perhaps up into many thousands), with regular spacing being used for much larger numbers of streams. However, regular spacing may also be preferred when great confidence in the interstream spectral values is required (for instance, when a single bad dependency could ruin a computation). Under such circumstances, the use of random starting points must be supported either with extensive statistical analysis of random spectral characteristics or with exhaustive spectral testing of all critical interdependencies. With regular spacing, many different interdependencies will result in identical spectral tests, greatly reducing the number of tests which must be performed.

3. MANY GENERATORS

When attempting to provide a distinct CRNG for each stream, one could vary the modulus, the multiplier, or the additive constant (although the additive constant will typically be zero when the modulus is not a power of 2). With our vectorized generators we are so far limited to moduli which are powers of 2, so varying the modulus is not possible. It could be useful, though, and will be discussed below.

3.1. Different Additive Constants

Percus and Kalos (1989), and independently Halton (in press), have considered using a unique additive constant for each stream, with a common multiplier and modulus (which is a power of 2). The theory they develop is helpful, but the idea that different additive constants produce truly distinct sequences is somewhat misleading. To see this, let X_n be a CRNG with

$$X_{n+1} \equiv aX_n + c \pmod{2^e},$$

and let

$$Y_n \equiv X_n - r \pmod{2^e}.$$

Then Y_n satisfies

$$Y_{n+1} \equiv aY_n + c + (a-1)r \pmod{2^e}.$$

That is, the two additive constants c and $c + (a-1)r$ produce the same sequence (except for a shift modulo 2^e) for any r. This produces a division of the possible additive constants into equivalence classes. In the important case of maximum potency ($a \equiv 5 \pmod{8}$), there are two such classes, and their sequences are in fact antithetic. Reducing the potency of the multiplier increases the number of equivalence classes, but correspondingly degrades the within-stream behavior. However, attempting a tradeoff of potency for equivalence classes is probably a red herring. Simple calculations show that for any multiplier, all choices of additive constant produce the same second difference sequence.

It should be noted that the results of Percus and Kalos and Halton are of interest even if different additive constants are just a masked form of sequence splitting. Percus and Kalos (1989, Sections 4 and 5) show how to obtain additive constants with good mutual two-dimensional spectral values and provide some help with three-dimensional values. When translated into starting points, their recommended constants resemble random starting points, but come with deterministic guarantees on spectral values. Halton's work is presented for computations which produce a natural tree structure for the streams. In this case there is a well-defined sense of closeness of streams which can even take lagged behavior into account with a reasonable number of spectral tests.

3.2. Different Multipliers

While using different additive constants for each stream adds no power over splitting methods, it is possible that using different multipliers for each stream may. The streams from CRNGs with different multipliers do not have a spectral theory analogous to those for splitting and different additive constants. Rather than being scattered on a single lattice, t-tuples generated from t different multipliers lie on one of many intertwined lattices, as in Beyer (1972). Bad interdependencies seem to result when many of these lattices collapse together. Little theory exists so far, but this method looks promising.

First, there is a sufficient stock of very good multipliers at large moduli like 2^{48}. Fishman (in press) searched 2^{26} multipliers with $m = 2^{48}$ for multipliers with hyperplane spacings within 25% of the minimal possible spacing in dimensions 2 through 6. From the results, Fishman estimated that about 11 million distinct multipliers meet his stringent conditions. Measured on the scale of $\log_2(\nu_2)$, Fishman's conditions require that a multiplier lose no more than about half a bit of randomness from the maximum possible in the tested dimensions. We can get an even larger stock of multipliers by relaxing this requirement. We generated 1000 randomly chosen maximum-potency multipliers, and found that the mean value of $\log_2(\nu_2)$ was 23.3, with a standard deviation of .76. Over 90% of the multipliers lost less than two bits of randomness. This would suggest that over 3 trillion multipliers at this modulus are within two bits of the maximum possible randomness in two dimensions.

The more difficult issue is the interdependency of sets of multipliers, and it is here that theory is lacking. We have looked at pairwise dependencies for $m = 2^e$ for various values of e, and have found necessary conditions for good interdependencies, but so far no sufficient conditions. Any two maximum potency multipliers a_1 and a_2 will satisfy $a_1 = a_2^j$ for some odd j. It is clear that j should not take on a very small value. A little calculation shows that if

$$j \equiv 1 \pmod{2^{e-r}}, \ r > 2, \qquad (1)$$

then the pairs (a_1^n, a_2^n) lie on at most 2^{r-2} lines. Thus we should look for a_i which are individually good multipliers and which are related by large powers j which do not satisfy (1) for any small r. For moduli small enough to examine exhaustively, such pairs of multipliers seem to behave well at all lags. For $m = 2^{48}$, we have found up to a thousand multipliers with no j satisfying (1) for $r < 36$. We only tested pairwise behavior; these sets of multipliers may be faulty in higher dimensions, although tests similar to those in Fishman and Moore (1982) do not detect problems.

4. IMPROVEMENTS

While splitting schemes seem adequate for current use, the spectral limitations on CRNGs are severe. We mentioned above that $\log_2(\nu_t)$ is roughly bounded by $\log_2(m)/t$; thus CRNGs with the highest commonly used moduli (around 2^{48}) cannot produce good behavior much above a dozen dimensions. For truly general parallel use, more would be required. Further, routine spectral tests could fail to uncover a problem with a specific splitting scheme in a specific application; such a problem could only be diagnosed by using a better parallel random number generator.

Any practitioner who needs a parallel RNG with thousands of streams should have *ad hoc* improvement techniques on hand. We have examined shuffling and combination techniques (Durst 1988). We concluded that shuffling buffers should be quite large (on the order of a hundred) if bad interdependencies are to be shuffled away, and that streams from a CRNG must be combined with quite different streams to effect improvement of bad dependencies. We used a stream from a lagged Fibonacci to improve poor CRNG performance since we found that using other streams from the same CRNG did not help. However, combination of one CRNG with a sufficiently different CRNG (e.g., one with a different modulus) might work.

Improvements for sensitive applications can also be obtained by increasing the modulus. Our calculations above showed that generators with modulus 2^{48} cannot produce acceptable random starting points for 2^{20} streams; but $m = 2^{64}$ would produce 2^{20} streams with expected smallest stream length well over a million. Still larger moduli would require multi-word implementation on current computers, and naive multiplication of k-word numbers takes $\mathcal{O}(k^2)$ steps (although Fourier methods can reduce this to $\mathcal{O}(k \log k)$). But $k = 2$ would more or less square the number of streams we can get by splitting a CRNG, and so might well repay the extra computation.

5. CONCLUSIONS

The best current methods for parallel pseudo-random number generation using CRNGs are splitting methods. For a small number of streams, random starting points provide the best interstream behavior, but the problem of short minimum substream lengths mandates regularly spaced starting points, whose interstream behavior is only somewhat worse, for thousands to millions of streams. In the future, the use of different multipliers may prove acceptable; this may provide many long streams with good interstream behavior at all lags. For the moment, troublesome applications and diagnostic checks should use either well-known *ad hoc* improvement techniques or larger moduli.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

REFERENCES

Beyer, W.A. (1972). Lattice Structure and Reduced Bases of Random Vectors Generated by Linear Recurrences. *Applications of Number Theory to Numerical Analysis*, ed. S.K. Zaremba, New York: Academic Press, 361–370.

Coveyou, R.R., and MacPherson, R.D. (1967). Fourier Analysis of Uniform Random Number Generators. *Journal of the Association for Computing Machinery* **14**, 100–119.

David, H.A. (1981). *Order Statistics* (second edition), New York: John Wiley.

Durst, M. J. (1988). Improving Parallel Random Number Generators. *Proceedings of the Statistical Computing Section*, American Statistical Association, Alexandria, Virginia.

Fishman, G. S. (1978). *Principles of Discrete Event Simulation*. John Wiley, New York.

Fishman, G. S. (in press). Multiplicative Congruential Random Number Generators with Modulus 2^β: An Exhaustive Analysis for $\beta = 32$ and a Partial Analysis for $\beta = 48$. *Mathematics of Computation*.

Fishman, G.S., and Moore, L.R. (1982). A Statistical Evaluation of Multiplicative Congruential Random Number Generators with Modulus $2^{31} - 1$. *Journal of the American Statistical Association* **77**, 129-136.

Halton, J. H. (in press). Pseudo-Random Trees. *Journal of Computational Physics*.

Knuth, D.E. (1981), *The Art of Computer Programming*. Vol. 2: *Seminumerical Algorithms*, 2nd edition, Reading, MA: Addison-Wesley, Chapter 3.

Marsaglia, G. (1968). Random Numbers Fall Mainly in the Planes. *Proceedings of the National Academy of Sciences* **61**, 25–28.

Niederreiter, H. (1978). Quasi-Monte Carlo Methods and Pseudo-Random Numbers. *Bulletin of the American Mathematics Society* **84**, 957–1041.

Percus, O. E., and Kalos, M. (1989). Random Number Generators for MIMD Parallel Processors. *Journal of Parallel and Distributed Computation* **6**, 477–497.

AUTHOR'S BIOGRAPHY

MARK DURST is a Mathematician at the Lawrence Livermore National Laboratory, where he has been involved in Monte Carlo calculations since 1986. He received an A.B. degree in mathematics from the University of California in 1974, and a Ph.D. degree in mathematics from the Massachusetts Institute of Technology in 1980. His current research interests include random number generation, parallel processing, and variance reduction techniques. He is a member of ASA.

Mark J. Durst
Lawrence Livermore National Laboratory, L-307
Livermore, CA 94550, U.S.A.
(415) 422-4272