Journal of the South Carolina Academy of Science 1(1):28-32 Fall 2003

SOME NOTES ON MULTIPLICATIVE CONGRUENTIAL RANDOM NUMBER GENERATORS WITH MERSENNE PRIME MODULUS 2^{61}-1

Dr. James Harris*

*Department of Mathematics and Computer Science, Georgia Southern University, Statesboro, GA 30460
jkharris@gasou.edu

ABSTRACT

Multiplicative congruential random number generators of the form $s_n = a*s_{n-1} \bmod m$ using the Mersenne prime modulus 2^{61}-1 are examined. Results show that they can provide sufficiently long pseudo-random sequences that can be implemented efficiently using 64 bit accumulators without the need of a costly division operation.

INTRODUCTION

Random number generators are widely used in computer simulations to provide approximate solutions to statistical problems. Hardware devices can efficiently generate non-uniform random sequences of infinite periodicity (Fort et al., 2003), which can be transformed into uniform distributions (Ya et al., 2002). However, all algorithmically implemented random number generators have a period length associated with the random sequences they generate. They are typically described as "pseudo-random" generators (Mascagni and Srinivasan, 2000). Ideally, the period length of the sequence should be long enough so that the period does not repeat within the simulation. In most cases, having the period repeat can severely compromise the results of the simulation. For a good reference on pseudo-random number generators and simulation, see L'Ecuyer, (1990).

Pseudo-random sequences should also be "statistically" random. There are barrages of tests that can be performed to determine if a random number generator falls within acceptable limits. The most popular of these tests is the spectral test (Tezuka, 1987; Tang and Kao, 2002; L'Ecuyer and Couture, 1997). In his classical book "The Art of Computer Programming", Donald Knuth (1997) states, "Not only do all good generators pass this test, all generators now known to be bad actually fail it". Random number generators should also be fast. They should be implemented using a minimum number of CPU clock cycles.

There are many types of pseudo-random number generators in use today. Common ones include generalized frequency shift-register generators (Wu, 2001), linear congruential generators (Lehmer, 1949) and matrix-congruential generators (Deng et al., 1992).

One of the oldest and most commonly implemented random number generators is the linear congruential random number generator (LCG) or Lehmer generator as proposed by D.H. Lehmer (1949) which uses a recurrence relation of the form

$$s_n = (a*s_{n-1} + c) \bmod m$$

to generate a sequence of pseudo-random integers where the integer **a** is called the multiplier, the integer **c** the increment, and the integer **m** the modulus . The initial integer value of the sequence (s_0) is called the random seed and is typically provided by a hardware device such as the system clock. If the value of c is taken to be zero, the resulting generator is called a multiplicative linear congruential random number generator (MLCG).

In order to generate a pseudo-random sequence of periodicity **m**-1, the multiplier must be chosen to be relatively prime to **m**. However, MLCG's with non-prime moduli tend to exhibit non-random characteristics (Knuth, 1997), therefore a natural choice for the modulus is a prime number. Since all elements, other than the identity, of the multiplicative group of integers mod p, where p is prime, are generators of the group, choosing **m** to be prime guarantees a period length of **m**-1 independent of the multiplier (other than those multipliers congruent to 0 or 1 mod p).

A Mersenne prime is a prime number of the form 2^n-1. MLCG's with Mersenne prime moduli have many nice properties and have been studies extensively (Entacher, K. 1998; Fishman and Moore, 1986; Wu, 1997; Matsumoto and Nishimura, 1998; Smith, 1971). It has been shown that the Mersenne prime 2^{31}-1 is a good choice for MLCG's implemented on CPU's with 32 bit accumulators (Fishman and Moore, 1986; Smith, 1971; Park and Miller, 1988). In fact, Carta (1990) shows how to implement MLCG's with modulus **m** = 2^{31}-1 on 32 bit CPU's without using a costly division operation.

RESULTS

Over the years the clock speeds of processors have increased to the point where MLCG's can quickly generate all 2^{31} - 2 numbers in the sequence generated by MLCG's with modulus 2^{31}-1, causing the period to repeat. When the Mips R4000 RISC CPU was introduced in 1991, 64-bit CPU's became commercially available for use in high-end workstations. The newer desktop CPU's, such as the Intel Itanium and the Motorola GigaProcessor Ultralite have 64 bit registers. Prime moduli close to, but smaller than 2^{64} are natural choices. The largest Mersenne prime less than 2^{64} is 2^{61}-1. This prime number is particularly attractive choice for the modulus. First, it has a very long period length. To get an idea of the sequence length, a processor generating 1 trillion random numbers per second would take over 11 years to repeat the sequence. Second, Carta's method for implementing MLCG's with modulus 2^{31}-1 in 32 bit registers without using a division operation (Carta, 1990) can be modified to work for the modulus 2^{61}-1 in 64 bit registers. The construction is given below:

Given the MLCG $s_{n+1} = as_n \bmod (2^{61}-1)$
Express as_n and $2^{63}p+q$

This form has a representation in hardware. The product of two signed 63 bit integers (**a** and s_n) is stored in two 64 bit accumulators, one storing the high order bits (p) and one storing the low order bits (q). We can then write:

$$s_{n+1} = as_n \bmod (2^{61}-1)$$
$$= (2^{63}p+q) \bmod (2^{61}-1)$$
$$= [(2^{61}-1)\,(2^{63}p+q)(1/(2^{61}-1))] \bmod (2^{61}-1)$$

Expanding $1/(2^{61}-1)$ as an infinite series gives

$$[(2^{61}-1)(2^{63}p + q)(2^{-61} + (2^{-122}) + (2^{-183}) + ...)] \bmod (2^{61}-1)$$

$$= [(2^{61}-1)((4p + (4p + q)(2^{-61}) + (4p + q)(2^{-122}) + ...] \bmod (2^{61}-1)$$

$$= [(2^{61}-1)(4p) + (4p + q) - (4p + q)(2^{-61}) + (4p + q)(2^{-61}) - (4p + q)(2^{-122}) + ...] \bmod (2^{61}-1)$$

The series telescopes to

$$[(2^{61}-1)(4p) + (4p + q)] \bmod (2^{61}-1)$$

$$= (4p + q)] \bmod (2^{61}-1)$$

Therefore, if $4p + q < 2^{61}-1$, then $s_{n+1}=4p + q$

and if $4p + q \geq 2^{61}-1$ then $s_{n+1}=4p + q - (2^{61}-1)$

Since a and s_n are both smaller than $2^{61}-1$ and q never exceeds $2^{63}-1$, p is never greater than 2^{59} and 4p is never greater than 2^{61}. Therefore, if addition is done as unsigned integers, the sum 4p+q can always fit into a 64-bit register. The division needed by the mod operation can be implemented using two shifts, one integer addition and one integer comparison in one case and an extra integer subtraction in the second case. This allows the MLCG $s_{n+1} = as_n \bmod (2^{61}-1)$ to be implemented without a costly division operation.

DISCUSSION

The method shown above allows for efficient software generation of sufficiently long sequences of random numbers. The choice of a "good" multiplier is left as an open question (Carta, 1990). With current CPU speeds, it is impossible to do an exhaustive statistical evaluation of all possible multipliers for $2^{61}-1$ as was done for $2^{31}-1$ (Fishman and Moore, 1986). Wu (1997) shows that MLCG's with multipliers of $\pm 2^{k1} \pm 2^{k2}$ and Mersenne prime moduli can be implemented efficiently. However, L'ecuyer and Simard (1999) demonstrated that the resulting sequences exhibited non-random behavior. Compagner (1995) and L'Ecuyer (1998) found that generators should not be based on simple recurrences relations. In the case of MLCG's, multipliers should have a good "mix" of ones and zeros. This illustrates a tradeoff between efficiency and randomness. MLCG's whose binary representations of their multipliers have large clusters of ones or zeros can have the multiplication operation efficiently implemented through shift and add operations, but do a poor job of exhibiting random behavior. The same assumption cannot be made for the modulus. There are examples of MLCG's with Mersenne prime moduli, whose binary represenations consist essentially of a giant clump of ones, that generate statistically random sequences. One that has been studied extensively is **a** = 16807 and **m** = $2^{31}-1$ [17].

CONCLUSIONS

The Lehmer generators are still one of the best ways for software generation of statistically random sequences. With the increased register sizes in newer CPUs, random sequences with significantly longer period lengths can be efficiently generated. It is because of the increase in CPU register sizes that a minimal standard Lehmer generator (Fishman and Moore, 1986) is needed for larger moduli. It has been shown in this paper that the 2^{61}-1 is a good choice for the modulus when implemented on 64-bit CPU's. However, as the size of the modulus in Lehmer generators increases, the number of possible multipliers increases, making the selection process for the "best" multiplier more difficult. A method different than exhaustive analysis is needed.

LITERATURE CITED

Brunner, D. Uhl, A. 1999. Optimal Multipliers for Linear Congruential Pseudo-Random Number Generators with Prime Moduli: Parallel Computation and Properties. BIT: Numerical Mathematics 39(2) 193-209.

Carta, D. 1990. Two Fast Implementations of the "Minimal Standard" Random Number Generator. Communications of the ACM 33(1) 87-88.

Compagner, A. 1995. Operational conditions for random number generation. Physics Review E52, 5-B, 5634–5645

Deng L., Rousseau C., and Yuan Y. 1992. Generalized Lehmer-Tausworthe random number generators. Proceedings of the 30th annual Southeast Regional Conference 108 - 115.

Entacher, K. 1998. Bad Subsequences of Well-Known Linear Congruential Pseudorandom Number Generators. ACM Transactions on Modeling and Computer Simulation 8(1) 61–70.

Fishman, G. and Moore, L. 1986. An Exhaustive Analysis of Multiplicative Congruential Random Number Generators with Modulus 231 - 1. SIAM Journal of Scientific Statistics Computing 7 24–45.

Fort , A.; Cortigiani, F;. Rocchi, S. and Vignoli, V. 2003. Very High-Speed True Random Noise Generator. Analog Integrated Circuits and Signal Processing 34 (2): 97 - 105.

Knuth, D. E.. 1997. The Art of Computer Programming. Vol. 2, Seminumerical Algorithms. 3rd edition. Addison-Wesley.

L'Ecuyer, P. 1990. Random Number Generators For Simulation. Communications of the ACM 33(1) 85 - 97.

L'Ecuyer, P. 1998. Random number generation. Handbook of Simulation, J. Banks ed., John Wiley & Sons, Inc., New York, NY, 93–137.

L'Ecuyer, P. and Couture, R. 1997. An Implementation of the Lattice and Spectral Tests for Multiple Recursive Linear Random Number Generators. Journal on Computing 9(2) 206-217.

L'Ecuyer, P. and Simard, R. 1999. Beware of Linear Congruential Generators with Multipliers of the Form a = ± 2q ± 2r . ACM Transactions on Mathematical Software 25(3) 367-374.

Lehmer, D.H. 1949. Mathematical methods in large-scale computing units. In Proceedings 2nd Symposium on Large-Scale Digital Calculating Machinery. Cambridge, MA, 141-146.

Mascagni, M. and Srinivasan, A. 2000. Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. ACM Transactions on Mathematical Software 26(3) 436-461.

Matsumoto, M. and Nishimura, T. 1998. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM Transactions on Modeling and Computer Simulation 8(1) 3–30.

Park, S. and Miller, K. 1988. Random number generators: Good ones are hard to find. Communications of the ACM 31 33(10) 1192–1201.

Smith, C. S. 1971. Multiplicative Pseudo-Random Number Generators with Prime Modulus. Journal of the Association for Computing Machinery 18(4) 586-593.

Tang, H. and Kao, C. 2002. Lower bounds in spectral tests for vectors of nonsuccessive values produced by multiple recursive generator with some zero multipliers. Computers & Mathematics with Applications 43(8) 1153-1159.

Tezuka, S. 1987. Walsh-Spectral Test for GFSR Pseudorandom Numbers. Communications of the ACM 30(8) 731-735.

Wu, P. C. 1997. Multiplicative Congruential Random-Number Generators with Multiplier = ±2k1 ± 2k2 and modulus 2p-1. ACM Transactions on Mathematical Software 23(2) 255–265.

Wu, P. 2001. Random Number Generation with Primitive Pentanomials. ACM Transactions on Modeling and Computer Simulation 11(4) 346–351.

Ya, Ryabko, B. and Matchikina, E. 2002. An Efficient Generation Method for Uniformly Distributed Random Numbers. Problems of Information Transmission 38(1): 20-25.