ARGONNE NATIONAL LABORATORY
Engineering Research — Reactor Analysis Division **Intra-Laboratory Memo**

June 16, 1994

To:	**D. C. Wade**			
From:	**Forrest B. Brown**			
Subject:	**Random Number Generation with Arbitrary Strides**			
CC:	E. K. Fujita	C. H. Adams	H. S. Khalil	E. M. Gelbard
	D. J. Hill	R. N. Blomquist		

Special random number generators are one of the principle requirements for performing Monte Carlo calculations on parallel computers. To permit reproducibility of calculations, regardless of the number of processors used, each particle in a batch is given its own random seed, and the random number generation is performed in parallel for a number of active particles. Last year I extended and generalized some of the random number routines used in the RACER code, and implemented them in the parallel version of VIM on the RA Network and the SP1 computer.

To provide each particle with a unique random seed, I have developed special routines which permit fast and arbitrary "skip-ahead" or "strides" in the random sequence obtained from linear congruential random number generators. Basically, these routines evaluate 2 quantities,

$$g^k \bmod 2^m \qquad \text{and} \qquad c\,\frac{g^k - 1}{g - 1} \bmod 2^m$$

for integers c and g comprised of up to m bits. Only $O(m)$ operations are required, rather than $O(k)$ for a naive implementation. For m=48 and k in $[0,2^{48}]$, k could be as large as about 7×10^{14}, so that these routines can provide speedups as large as $\sim 10^{13}$ over a "brute force" approach. More important, use of these new algorithms permits the fast and convenient use of arbitrary strides, completely user-selectable rather than "hard-wired" into the Monte Carlo codes.

In the past, I did not publish this material since I thought it was straightforward and well-known. I have since found out that apparently no one but me knows of these techniques (at least in the nuclear applications areas). The LANL Monte Carlo code MCNP, for example, uses a brute-force method which could take up to several years of workstation CPU time, as compared to around 100 microseconds for my method.

The attached paper titled *"Random Number Generation with Arbitrary Strides"* is based on these techniques for random number generation. The summary will be submitted to the 1994 American Nuclear Society Winter meeting.

Paper for Contributed Presentation

**1994 ANS Winter Meeting —Washington, D. C.
(November 13-17, 1994)**

Random Number Generation with Arbitrary Strides*

Forrest B. Brown

Argonne National Laboratory
Reactor Analysis Division
9700 S. Cass Ave. — RA/208
Argonne IL 60439-4842

* Work supported by the U. S. Department of Energy, Nuclear Energy Programs, under Contract W-31-109-ENG-38.

(Summary prepared for the 1994 ANS Winter Meeting, Washington D. C., Nov. 13-17, 1994)

Random Number Generation with Arbitrary Strides

Forrest B. Brown (ANL)

In this paper, we present techniques for the generalized initialization of random number generators for parallel Monte Carlo calculations. In particular, we develop and prove an algorithm for a fast, direct method for skipping-ahead in the random sequence by an arbitrarily large positive or negative "stride." This algorithm is currently used in three different production-level Monte Carlo codes[1-3] on a variety of parallel, vector, and distributed computer systems to generate the initial seeds for different particle histories. In the extreme cases, the algorithm can reduce several years of "brute-force" computation to a few hundred microseconds.

Most of the production-level Monte Carlo codes used for radiation transport analysis rely on linear congruential generators for producing a uniform-(0,1) distribution of basic random numbers. These generators have the form

$$s_{i+1} = s_i \cdot g + c \mod 2^m , \tag{1}$$

where g and c are integer constants in the range $[0, 2^m]$, and m is the number of bits used for the "seeds" s_i. If the values of m, g, and c are chosen properly[4], then the period of the sequence is 2^m for $c \neq 0$, or 2^{m-2} for $c=0$. For a particular code or computer, m is typically 32 or 48. The seeds are converted to fractions on (0,1) by multiplying by 2^{-m}. Generators of the form of Equation (1) have been used for more than 40 years in Monte Carlo codes for radiation transport, and are considered well-proven and very robust for these applications.

Special attention to random number generation is one of the principle requirements for performing Monte Carlo calculations on parallel computers. To permit reproducibility of calculations[5] regardless of the number of processors used, each particle in a batch is given its own random seed. Random number generation is then performed in parallel for a number of active particles. The most common procedure for choosing the seeds for each particle is to skip-ahead in the random sequence by a fixed number of steps or "stride"[6] for each particle. That is, given an initial seed of s_0 and a stride of L, the initial seed for particle N would be found by iterating Equation (1) for $k = NL$ times, to get the k-th term of the sequence:

$$s_k = s_0 \cdot g^k + c \frac{g^k - 1}{g - 1} \mod 2^m \tag{2}$$

Equation (2) cannot be naively evaluated using ordinary computer arithmetic, since roundoff and truncation effects would rapidly corrupt the random sequence. If m is chosen as 48, then exact 96-bit integer arithmetic is required. In many Monte Carlo codes[6], Equation (2) is evaluated in practice by a "brute-force" method, simply iterating with Equation (1) L times for each successive particle seed. This can be expensive for large strides. For a fixed value of L, the constants needed for Equation (2) are often pre-computed and "hard-wired" into a code, thus prohibiting any investigation of alternate strides.

Fortunately, simple and fast algorithms have been developed for direct evaluation of Equation (2), greatly simplifying the initialization of parallel random number generators. Denoting the j-th bit of the binary representation of k as $k_{(j)}$ (with j=0 the least-significant bit), then the multiplier in the first term of Equation (2) may be computed by:

$$G = g^k \bmod 2^m = g^{\sum_{j=0}^{m-1} k_{(j)} 2^j} \bmod 2^m = \prod_{j=0}^{m-1} \left(g^{2^j} \right)^{k_{(j)}} \bmod 2^m \qquad (3)$$

A computational algorithm using recursion to evaluate G according to Equation (3) is

Algorithm G:

```
$G \leftarrow 1, \quad h \leftarrow g, \quad i \leftarrow k+2^m \bmod 2^m$
while $i>0$
    if $i$=odd:   $G \leftarrow Gh \bmod 2^m$
    $h \leftarrow h^2 \bmod 2^m$
    $i \leftarrow \lfloor i/2 \rfloor$
```

Thus Algorithm G can be evaluated in $O(m)$ steps, rather than $O(k)$ steps, which can be a significant savings when k is a very large number. Using periodicity, negative strides can be trivially handled as the equivalent positive stride. In a similar manner (although the proof is more complicated), the second term

$$C = c \frac{g^k - 1}{g - 1} \bmod 2^m = c \left[1 + g + g^2 + \ldots + g^{k-1} \right] \bmod 2^m \qquad (4)$$

from Equation (2) can be evaluated in $O(m)$ steps via the algorithm:

Algorithm C:

```
$C \leftarrow 0, \quad f \leftarrow c, \quad h \leftarrow g, \quad i \leftarrow k+2^m \bmod 2^m$
while $i>0$
    if $i$=odd:   $C \leftarrow Ch + f \bmod 2^m$
    $f \leftarrow f(h+1) \bmod 2^m$
    $h \leftarrow h^2 \bmod 2^m$
    $i \leftarrow \lfloor i/2 \rfloor$
```

For many Monte Carlo codes the constant c is chosen to be 0, so that Algorithm C is not needed. When $c \neq 0$, Algorithms G and C are readily combined.

To demonstrate the usefulness of Algorithms G and C, consider the specific random generator used in VIM and MCNP, where $g=5^{19}$, c=0, m=48. To compute a new particle seed directly using Algorithm G for the default MCNP stride of 152,917 requires about 80 μsec on an IBM rs6000/350 workstation, which is greater than the 2 μsec required if a pre-computed value of G is used. Changing the stride to 1,152,917 requires about 90 μsec for Algorithm G, versus about 2.5 sec for the brute-force iteration of Equation (1). A more extreme difference is seen for negative strides: To use a stride of -152,917 (i.e., $k \sim 7 \times 10^{13}$) would require about 215 μsec using Algorithm G, but about 5 years using brute-force. Timing measurements and estimates for other computers show similar advantages of Algorithm G when large or negative strides are desired. More important, use of these new algorithms permits the fast and convenient use of arbitrary strides, completely user-selectable rather than "hard-wired" into the Monte Carlo codes.

For "ordinary" Monte Carlo calculations (if there are any), the algorithms presented here for initializing the particle seeds will have little or no impact on computer running times. For specialized calculations where a nonstandard, unusually large stride or negative stride is needed, the present algorithms can provide considerable flexibility and computer time savings. In addition, they are very convenient for initializing the particle seeds on different processors in a parallel calculation — sequential portions of the calculation are eliminated and no special communication is required. The techniques described in this paper for initializing the Monte Carlo random number generators are currently being used successfully in a number of production-level Monte Carlo codes on a variety of computers: RACER[1] (KAPL) on a Cray-C90 vector/parallel supercomputer and a Meiko CS1 parallel computer, a parallel version of VIM[2] (ANL) on a workstation network and the IBM-SP1 parallel computer, and a parallel version of KENO-Va[3] (CSN-Spain) for a Convex-C3440 vector/parallel computer.

References

[1] F. B. Brown & T. M. Sutton, “Parallel Monte Carlo for Reactor Calculations,” proc. ANS topical meeting on *Advances in Reactor Physics*, April 11-15, 1994, Knoxville TN (1994).

[2] R. N. Blomquist & F.B. Brown, “Parallel Monte Carlo Reactor Neutronics,” Proc. Soc. Comp. Sim. meeting on *High Performance Computing '94*, April 11-15, 1994, La Jolla, CA (April 1994).

[3] J. Pena, Consejo de Seguridad Nuclear — Spain, private communication (May 9, 1994).

[4] D. E. Knuth, The Art of Computer Programming — Vol. 2, 2nd ed., pp. 9-25, Addison-Wesley (1981).

[5] F. B. Brown & T. M. Sutton, “Reproducibility and Monte Carlo Eigenvalue Calculations,” *Trans. Am. Nucl. Soc.*, **65**, 234 (1992).

[6] J. S. Hendricks, “Random Number Stride in Monte Carlo Calculations,” *Trans. Am. Nucl. Soc.*, **62**, 283 (1990)