JOURNAL OF COMPUTATIONAL PHYSICS 82, 487–491 (1989)

Note

Implementation of a Random Number Generator in OCCAM

In this note we want to communicate the implementation of a shift-bit register random number generator on a transputer system. Such transputer systems have now become widely available and are used for large scale simulations. As a matter of fact, we are currently assembling a large number of these transputers to build a parallel supercomputing structure for the Condensed Matter Theory Group at the University of Mainz.

Among the simulations already performed using transputer-based parallel computing structures are those of neural networks [1], lattice gauge theory [2], and Monte Carlo simulations of the Ising model [3, 4], using the micro canonical ensemble method [5].

Monte Carlo and other simulation methods [6] rely on high-quality random numbers. For such simulations they should have no correlations and the cycle length must be extremely long (for many simulations the cycle length of the linear congruential random number generators is by far too small!). Recently it has become evident that linear congruential random number generators exhibit correlations leading to subtle errors in the results of some Monte Carlo simulations [7–9]. In these simulations one is studying the dynamic evolution of a system in non-equilibrium. An initially stable system is brought to a thermodynamically unstable non-equilibrium state by a (temperature) quench from the one into the two-phase region [9]. During the subsequent time evolution, the physical instability of the system to the long wavelength perturbations is responsible for amplifying not only the thermal fluctuations but also the correlations present in the initial postquench state. If the random number generator used in MC simulations of such systems has some inherent correlations then these get amplified due to the long wavelength instability of the system. In this sense such physical systems act as amplifiers of correlations that seem not to be dealt with in the usual statistical tests.

In the above described physics application one observes, for example, that the system exhibits after some time an unphysical slab structure. Also the order parameter and the energy evolution after a quench exhibit unphysical results.

The linear congruential random number generator [10–12]

$$x_{i+1} \equiv a x_i \bmod m$$

which generates recursively from a seed x_0 a sequence of pseudo random numbers x_i shows dramatic correlations leading to unphysical results [7, 8]. These were observed for mulipliers

0021-9991/89 $3.00

$$a = 65539, \qquad a = 16807$$

with the modulo $m = 2^{31} - 1$ in simulations of the domain growth process. Also the modulo generator RANP

$$a = 1664525, \qquad m = 2^{32}$$

implemented on the transputer development systems TDS [13, 14] for the transputer T414/T800 shows these correlations. There are suggestions for other choices of multipliers [15, 16] which in a certain statistical sense are optimal. It should be noted here that one of these optimal choices indeed is $a = 16807$. However, the simulations show that there are possibly very longrange correlations among the random numbers generated with this multiplier not accounted for in optimality criteria of the statistical tests. Those are the ones creating the unphysical simulation results.

In the studies of the domain growth problem it was found empirically that the shift-bit register generator of Tausworth [17] (or F(r, s, $\oplus$) in the nomenclature of Marsaglia [19]) gave much better results. The simple test of d-space nonuniformity, i.e., the filling of a d-dimensional lattice using consecutive random numbers, reveals that the shift-bit register generator is superior to the linear congruential generator in these applications.

The shift-bit register generator is a generalization of the linear congruential generator. New pseudo random numbers are generated from the previously generated ones by the recursion relation

$$x_n = c_1 x_{k-1} + c_2 x_{k-2} + \cdots + c_n x_{k-n} \bmod 2.$$

Instead of this full relation one usually takes the linear recursion

$$x_k = x_{k-q} + x_{k-p}$$

on the space of $\{0, 1\}$. The operation + is then the exclusive-or operator (>< in OCCAM). The most popular choice for the pair (p, q) is (250, 103) [18] and is known as the "R250."

The program listing shows the implementation of this algorithm in the OCCAM programming language. This program was run and tested on the multitransputer system of the Condensed Matter Theory Group at the University of Mainz.

Table I lists the results of test runs using the random number generator R250. These results are not meant to test the quality of the random number generator, but to give a guidance for those who have implemented the program given here. Given are the seed value, the sample size, and the values obtained for the mean, the standard deviation, the skewness, and the excess. Let s_k with $k = 2, 3, 4$ denote the central moments with respect to the mean; then the sample skewness and the sample excess are defined as

$$\text{skew} = (s_3/s_2)^{3/2}$$

$$\text{excess} = (s_4/s_2)^2 - 3.$$

TABLE I

Results of Test Runs of the OCCAM Version of the Random Number Generator R250 Using the Seed Value of 4711

Sample size	Mean	Std. dev.	Skew	Excess	Time/number (s)	Time(RANP) (s)
10^3	0.4818	0.2909	+0.064	−1.125	2.48×10^{-5}	2.64×10^{-5}
10^4	0.4931	0.2880	−0.041	−1.194	2.31×10^{-5}	2.63×10^{-5}
10^5	0.5007	0.2891	−0.002	−1.202	2.31×10^{-5}	2.63×10^{-5}
10^6	0.5004	0.2888	−0.003	−1.201	2.31×10^{-5}	2.63×10^{-5}

As regards to the timing (note that the results quoted are for a single transputer), we found that the R250 is just as fast as the conventional generator. Hence nothing is lost with respect to time but much is gained with respect to accuracy in Monte Carlo simulations.

ALGORITHM. OCCAM version of the random number generator R250.

```
PROC r250 (VAL INT n.f, [10001] REAL32 x.f, [251] INT32 m.f)
— The array m.f has to be initialized with 250 integer32 random
— numbers before the first call of r250 (using for instance the
— built in RANP random number generator). In this example the
— maximum possible number of random numbers to be generated in
— one call is taken to be 100000. n.f is the actual number of
— numbers to be generated.
  INT iloop, num.of.loops, loop.rest, ind:
  INT irand, maxint:
  REAL32 rmax:
  SEQ
    maxint := #7EEEEEEE
    rmax := REAL32 ROUND maxint
    num.of.loops := n.f/250
    loop.rest    := n.f REM 250
    irand        := 1
    SEQ iloop = 1 FOR num.of.loops
      SEQ
        SEQ ind = 1 FOR 147
          SEQ
            m.f[ind] := m.f[ind] >< p.f[ind + 103]
            conversion (x.f[irand], m.f[ind])
            irand := irand + 1
        SEQ ind x 1 FOR 103
          SEQ
            m.f[ind + 147] := m.f[ind + 147] >< m.f[ind]
            conversion (x.f[irand], m.f[ind + 147])
            irand := irand + 1
```

```
IF
  loop.rest = 0
    SKIP
  loop.rest < = 147
    SEQ ind = 1 FOR loop.rest
      SEQ
        m.f[ind] := m.f[ind] > < m.f[ind + 103]
        conversion (x.f.[irand, m.f[ind])
        irand := irand + 1
  loop.rest > 147
    SEQ
      SEQ ind = 1 FOR 147
        SEQ
          m.f.[ind] := m.f[ind] > < m.f[ind + 103]
          conversion (x.f[irand], m.f[ind])
          irand:= irand + 1
      SEQ ind = 1 FOR (loop.rest − 147)
        SEQ
          m.f[ind + 147] := m.f[ind + 147] > < m.f[ind]
          conversion (x.f[irand], m.f[ind + 147])
          irand := irand + 1
```

Acknowledgments

We would like to thank K. Binder for the many stimulating discussions. Also partial support from the *Sonderforschungsberich 41* is gratefully acknowledged.

References

1. B. M. Forrest, D. Roweth, N. Stroud, D. J. Wallace, and G. V. Wilson, Edinburgh Preprint 87/414 (1987).
2. S. Duane, A. D. Kennedy, B. J. Pendelton, and D. Roweth, *Phys. Lett. B* **195**, 216 (1987).
3. R. C. Desai, D. W. Heermann, and K. Binder, *J. Stat. Phys.*, in press.
4. D. W. Heermann and R. C. Desai, *Comput. Phys. Commun.*, in press.
5. M. Creutz, *Phys. Rev. Lett.* **50**, 1411 (1983).
6. For recent reviews on Monte Carlo methods see D. W. Heermann, *Introduction to the Computer Simulation Methods of Theoretical Physics* (Springer-Verlag, Heidelberg, 1986); M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods, Vol. 1* (Wiley, New York, 1986); K. Binder and D. W. Heermann, *The Monte Carlo Method in Statistical Mechanics: An Introduction* (Springer-Verlag, Heidelberg, 1988).
7. A. Milchev, K. Binder, and D. W. Heermann, *Z. Phys. B* **63**, 521 (1986).
8. Th. Filk, M. Marcu, and K. Fredenhagen, *Phys. Lett. B* **165**, 125 (1985).
9. For recent reviews on the first-order phase transitions see J. D. Gunton, M. San Miguel, and P. S. Sahni, in *Phase Transitions and Critical Phenomena, Vol. 8*, edited by C. Domb and J. L. Lebowitz (Academic Press, New York, 1984); K. Binder and D. W. Heermann, in *Scaling Phenomena in Disordered Systems*, edited by R. Pynn and T. Skjeltrop (Plenum, New York, 1985), p. 205; H. Furukawa, *Adv. Phys.* **34**, 703 (1985).
10. D. Knuth, *The Art of Computer Programming, Vol. 2* (Addison–Wesly, Reading, MA, 1969).
11. J. H. Ahrens and U. Dieter, *Pseudo Random Numbers* (Wiley, New York, 1979).

12. D. H. Lehmer, in *Proceedings, 2nd Symposium on Large-Scale Digital Computing Machinery* (Harvard Univ. Press, Cambridge, MA, 1951), p. 142.
13. *OCCAM Programming Maunual* (Prentice–Hall, London, 1984).
14. *TDS Maunual* (INMOS, Cambridge, 1987).
15. I. Borosh and H. Niederreiter, *BIT* **23**, 65 (1983).
16. G. S. Fishman and L. R. Moore, *SIAM J. Sci. Stat. Comput.*, **7**, 7 (1986).
17. R. C. Tausworth, *Math. Comput.* **19**, 201 (1965).
18. S. Kirkpatrick and E. P. Stoll, *J. Comput. Phys.* **40**, 517 (1981).
19. G. Marsaglia and L.-H. Tsay, *Linear Algebra Appl.* **67**, 147 (1985).

Received: January 5, 1988; revised: August 11, 1988

W. Paul
Dieter W. Heermann
Rashmi C. Desai*

Institut für Physik
Johannes-Gutenberg Universität
Staudinger Weg 7, 6500 Mainz
West Germany

* Permanent address: University of Toronto, Department of Physics, Toronto M5S 1A7, Canada.