Journal of Statistical Physics, Vol. 48, Nos. 1/2, 1987

A New Random-Number Generator for Multispin Monte Carlo Algorithms

L. Pierre,[1] T. Giamarchi,[2] and H. J. Schulz[2]

Received March 13, 1986; revision received March 25, 1987

We develop a novel multispin coded random number generator algorithm to compute bits equal to 1 with probability p. Compared to previously used algorithms, this generator is at least equally fast and allows for an arbitrary accuracy of the computed probability without any significant increase in time. An explicit implementation of the algorithm is given for a Cray-1 vector computer, and the modifications for other machines are discussed. Finally, the algorithm is tested by computing the magnetization of the two-dimensional Ising model. The measured speed of the program is 57 million spin-flips per second. The agreement with theoretical values is found to remain very satisfying even when quite close ($\simeq 0.5\%$) to the critical temperature.

KEY WORDS: Random numbers; Monte Carlo; multispin coding; parallel computation; Ising models.

1. INTRODUCTION

The Monte Carlo (MC) method has proven to be a very powerful way of studying statistical systems and phase transitions.[1] However, near the critical point relaxation times increase dramatically—roughly as $\min(L, \xi)^z$, where L is the linear size of the system, ξ the correlation length of an infinite system at the same temperature, and z the dynamic critical exponent[2] ($z \simeq 2$ for the Ising model). In order to avoid errors due to the finite size of the system or insufficient statistics, one has to use rather large lattices and compute an important number of configurations. This entails an enormous computational demand. A number of methods have been put forward to overcome this difficulty and increase the accuracy of the

[1] UFR de Sciences Economiques, Université de Paris X Nanterre, 92001 Nanterre, France.
[2] Laboratoire de Physique des Solides, Université de Paris-Sud, 91405 Orsay, France.

result (specially designed computers,[3] Monte Carlo renormalization techniques,[4] multispin coding techniques,[5,6] vector algorithms,[7] but typical configurations always remain to be computed. In all these programs the most important part of computational time is devoted to the generation of the different configurations, the part devoted to the measurements being in most cases negligible. Moreover, in an efficiently written program the time spent computing the gain or loss in energy (if one uses an algorithm of the Metropolis type[8] of a given transformation is also quite small. Much of the time is used to determine bits (i.e., elements of $\{0, 1\}$) equal to 1 with a probability p [$p \neq 1/2$, typically $p = \exp(-\beta \Delta E)$] necessary to decide whether or not the transformation can be accepted. In fact, in many MC programs, at least half of the time is spent computing random numbers! In the first multipsin algorithms[7] the determination of the random bits was not multispin coded. This problem was solved by Williams and Kalos.[6] However, in their algorithm, reducing the time spent in this part of the program implies increasing the systematic error on the Boltzmann factor p, which from the outset limits the accuracy of the results. This may be especially detrimental in models more complicated than the nearest neighbor Ising model, i.e., in cases with more than just one energy parameter. We here give an algorithm to compute bits with a probability p, which is at least as efficient in time as previously used ones, and allows for an *arbitrary* accuracy without any significant increase in time. The program described here has been specially built to take full advantage of the pipeline and vector structure of a Cray-1 computer,[9] but with minor changes in the program, the algorithm can be used on other general machines as well as on specially designed processors or array-processor machines. Finally, to test the random-number generator we use it on a Monte Carlo program for the 2D Ising model. We compute the magnetization and compare it to the exact result.[10] The implementation of the algorithm on a Cray-1 computer is given in the Appendix.

2. SOME SIMPLE RANDOM-NUMBER GENERATORS

A very good algorithm commonly used to generate pseudorandom bits equal to 1 with probability 1/2 is the one advanced by Kirkpatrick and Stoll,[11] which uses exclusive-or (denoted $\oplus$). It generates a sequence of random bits by the recursion relation

$$r(i) = r(i-a) \oplus r(i-b) \tag{2.1}$$

where a and b $(a > b)$ are numbers[12] chosen in order to obtain the maximum period of $2^a - 1$. One now has to use these $b_{1/2}$ bits (i.e., bits equal to 1 with a probability 1/2) to generate b_p bits (i.e., bits equal to 1 with a probability p).

Let us suppose that p has a binary representation on n bits:

$$p = 0.p_1\, p_2\, p_3 \cdots p_n \tag{2.2}$$

The usual method is to generate a sequence of n $b_{1/2}$ bits, giving the number

$$seq = 0.b'_1 b'_2 b'_3 \cdots b'_n \tag{2.3}$$

and to compare it with p. The result of the comparison $seq < p$ is obviously equal to 1 with a probability p ("true" will be represented by a bit 1). The accuracy of p is $1/2^n$. In order to obtain Boltzmann factors with a sufficient accuracy, n must be large enough. There are three simple algorithms to perform the comparison:

Algorithm 1. One way (used by Williams and Kalos[6]) to perform the comparison is to make a bit-by-bit subtraction of the two numbers seq and p. The borrow will give the result of the comparison (0 if $seq \geqslant p$). This subtraction of two numbers takes four logical operations per bit of p. We deal here with bits because, as pointed out previously,[5,6] in order to increase the speed of the program, it is convenient to deal with words of independent bits. The logical operation handles of course the whole word ("multispin coding"). This first algorithm can be trivially improved.

Algorithm 2. If p is known *a priori* (i.e., if the temperature remains fixed), one can write a different program for each p (or group of fixed p) and the computation of the borrow takes only *one* logical operation per bit. Another program can generate the instructions before the compilation. This method is suitable, for example, for systems with a discrete symmetry, where the number of independent Boltzmann factors is small (for example, in the 3D Ising model there are only three "*a priori*" Boltzmann factors at a given temperature in the case of a Metropolis-like simulation).

Algorithm 2 needs n $b_{1/2}$ bits plus n/l (l is the length of the words) instructions to generate a b_p bit. Algorithm 1 also uses $n\, b_{1/2}$ bits, but $4n/l$ instructions per b_p bit.

The vectorization of the two algorithms described above is straightforward. One deals with a vector of words instead of a single word, the algorithms remaining unchanged. Let us assume that it takes one cycle to handle a word (this can be sensibly achieved on a vectorized version of the programs by using sufficiently long vectors). Then even the "improved" version of algorithm 1 introduced as algorithm 2 will require $3n$ memory accesses and n logical operations to generate the needed $b_{1/2}$ bits plus n logical operations to compute the $l\, b_p$ bits of a word. On the Cray-1 (and certainly on most vector computers) due to the pipeline structure the $2n$ logical operations can be performed during the $3n$ memory accesses. It will thus take *at least* $3n$ cycles to compute a word of $l\, b_p$ bits with accuracy n. For the 3D Ising model, for example, which needs three different Boltzmann factors, if we want an accuracy of $1/256 \simeq 0.4\%$, at least 72 cycles will be required simply to generate the random b_p bits. This obviously constitutes the main part of the program (for the 2D Ising model program described below the remaining part of the program takes 12 logical operations!). Moreover, time increases linearly with the accuracy.

It is therefore fundamental in MC programs to find a way of reducing the time devoted to the random-number generator without any loss in accuracy.

3. A NEW ALGORITHM

3.1. Single-Bit Version

Let us suppose that we have only a single b_p bit r to generate. We do not need to subtract *seq* and p. We only want to compare them, i.e., to find the first position after the decimal point where p and *seq* differ and to compare the bits at this position. For convenience, we suppose that p has an infinite accuracy, i.e., p_i is defined for all $i > 0$. This can be achieved by assuming $p_i = 0$ if i is large. One then has the following algorithm:

Algorithm 3. We compare b'_1 with p_1, b'_2 with p_2,..., until we find N such that $b'_N \neq p_N$. The result is $r = (p_N > b'_N)$, i.e., $r = p_N$, since $b'_N \neq p_N$.

Let $b_i = \text{not } (b'_i \oplus p_i)$. The algorithm simplifies as follows:

Algorithm 4. We test b_1, b_2,... until we find N such that $b_N = 0$. The result is then $r = p_N$.

Since the b'_i are indkependent $b_{1/2}$ bits, so are the b_i. We thus do not need to generate the b'_i and then to compute the bits b_i. It is equivalent and simpler to generate directly the bits b_i as $b_{1/2}$ bits.

Thus N is the number of b_i bits used and the position of the first zero b_i (note that the probability that no zero b_i exists is zero). Now, $N = i$ if $(b_1, b_2,..., b_i) = (1, 1,..., 1, 0)$, which happens with a probability $1/2^i$. The value of N is 1, 2,..., n,... with probabilities 1/2, 1/4,..., $1/2^n$,... . The expected number of b_i used is the average value of N: $\langle N \rangle = 2$.

3.2. Multispin Version

The simultaneous computation of several bits with the Algorithm 4 seems difficult: N (the number of steps for a given bit) depends randomly on the bit computed and is not bounded *a priori*. We will thus have to stop the comparison for different bits at different times and terminate only when all the different bits are determined. The test given in Algorithm 4 ($b_i = 0$) does not straightforwardly generalize to several bits in parallel. We give in this section a different realization of the same algorithm which can be multispin coded.

In order to determine the termination of the algorithm, let us define the sequence s_i of bits by

$$s_i = 1 \Leftrightarrow i < N \tag{3.1}$$

i.e., $s_i = 1$ if $b_1 = b_2 = \cdots = b_i = 1$. Hence, $s_i = b_1 \wedge b_2 \wedge \cdots \wedge b_i$ (where $\wedge$ means the logical product AND). The s_i can thus be computed using the recursion relation:

$$\begin{aligned} s_0 &= 1 \\ s_i &= s_{i-1} \wedge b_i \end{aligned} \tag{3.2}$$

and $s_{i-1} - s_i = 1$ if $i = N$ and 0 otherwise. Hence, r, which is p_N, can be expressed as

$$r = \sum_{i=1}^{\infty} (s_{i-1} - s_i)\, p_i \tag{3.3}$$

$$= p_1 + \sum_{i=1}^{\infty} (p_{i+1} - p_i)\, s_i \tag{3.4}$$

Equation (3.4) gives the algorithm to compute r:

Algorithm 5. We start with $r = p_1$ and $s_0 = 1$. For $i = 1$ to ∞:

1. We compute $s_i = s_{i-1} \wedge b_i$.
2. If $p_i = 0$ and $p_{i+1} = 1$, then we add s_i to r.
3. If $p_i = 1$ and $p_{i+1} = 0$, then we subtract s_i from r.

As long as the b_i are equal to 1, s_i remains equal to 1, and r is set at each step to p_{i+1}. During this first phase r oscillates between 0 and 1 for each transition $0 \rightarrow 1$ or $1 \rightarrow 0$ in p. When b_i is zero for the first time, s_i becomes and remains zero. r does not change any more, and keeps its last value p_N. We can thus stop at any time when $s_i = 0$.

This version of the algorithm straightforwardly generalizes to a word of independent bits or to vectors of words. One introduces a word (or vector) of independent bits s_i in order to treat a word (vector) of independent bits r. The comparison must stop only when all the r's are fixed, which is the case if and only if all the s_i are zero. This test (a word of bits or a vector of words is zero) can easily be implemented on any computer.

Our algorithm permits the use of the arithmetical operations (i.e., addition and subtraction) even if we deal with words of independent bits, since no carry occurs during the operations involved Algorithm 5 [cf. Eq. (3.4) and steps 2 and 3 of Algorithm 5]. We here use the arithmetical operations, because on the Cray-1 their cost in time is the same as the cost of logical operations, and due to the pipeline structure both kinds of operations can be done *simultaneously* (see Appendix). If on other computers the arithmetical operations are slower, they can be replaced by a logical "exclusive-or" without any change in the algorithm.

Let us consider the number of operations needed by our algorithm to generate in parallel t random b_p bits. For a single bit, the number of steps needed is the random variable N, taking the values 1, 2,... with probabilities $\frac{1}{2}, \frac{1}{4}, \ldots$, respectively. For t b_p bits generated in parallel, we have the independent and identically distributed random variables $N_1, N_2, \ldots, N_t$, where N_j is the number of steps needed to compute the jth bit. The algorithm stops when all the b_p bits are generated, which takes a (random) time $M = \max(N_1, N_2, \ldots, N_t)$. As $M \leqslant k$ if and only if $\forall j\ N_j \leqslant k$, we have

$$P(M \leqslant k) = \prod_{j=1}^{t} P(N_j \leqslant k) = (1 - 2^{-k})^t \tag{3.5}$$

It follows that the expected value of M is

$$\langle M \rangle = \sum_{k=0}^{\infty} P(M > k) = \sum_{k=0}^{\infty} [1 - (1 - 2^{-k})^t] \tag{3.6}$$

As tests are expensive, it is a bad idea to put tests of s_i too soon. Let us suppose we start testing for completion ($s_i = 0$) only at the $\lfloor \log_2 t \rfloor$th step ($\lfloor x \rfloor$ means the integer part of x). If M is lower than $\lfloor \log_2 t \rfloor$, useless steps will be made. The number of useless steps is, on the average,

$$U = \lfloor \log_2 t \rfloor - \langle \min(M, \lfloor \log_2 t \rfloor) \rangle = \lfloor \log_2 t \rfloor - \sum_{k=0}^{\lfloor \log_2 t \rfloor - 1} P(M > k) \tag{3.7}$$

Using $e^{-x} \geqslant 1 - x$ for $0 \leqslant x \leqslant 1$, we get

$$1 - e^{-t2^{-k}} \leqslant P(M > k) \leqslant 1 \tag{3.8}$$

From (3.8), we bound U:

$$0 \leqslant U \leqslant \sum_{k=1}^{\infty} e^{-2^k} \simeq 0.154 \tag{3.9}$$

which shows that it is really useless to perform tests before the $\lfloor \log_2 t \rfloor$th step. The average number of steps performed is, in this case,

$$\begin{aligned} \langle M_2 \rangle &= \langle \max(M, \lfloor \log_2 t \rfloor) \rangle \\ &= \lfloor \log_2 t \rfloor + \sum_{k=\lfloor \log_2 t \rfloor}^{\infty} P(M > k) \\ &\leqslant \sum_{k=0}^{\infty} \min(1, t2^{-k}) \leqslant \log_2 t + 2 \end{aligned} \tag{3.10}$$

As the number of steps without tests is exactly $\lfloor \log_2 t \rfloor - 1$, the average number of tests performed is at most $(\log_2 t + 2) - (\lfloor \log_2 t \rfloor - 1) \leqslant 4$. In short, in order to generate one vector of t b_p bits, we use on the average fewer than $\log_2 t + 2$ steps and we perform fewer than four tests. Actually, on the Cray-1 we used $t = 4096$ (64 words of 64 bits), and we started testing at the 12th step.

The expected number of useful steps is $\langle M \rangle = \sum_{k=0}^{\infty} 1 - (1 - 2^{-k})^{4096} \simeq 13.33$.

The expected number of performed steps is $12 + \sum_{k=12}^{\infty} 1 - (1 - 2^{-k})^{4096} \simeq 13.49$.

The expected number of performed steps with test is $1 + \sum_{k=12}^{\infty} 1 - (1 - 2^{-k})^{4096} \simeq 2.49$.

The expected number of useless performed steps is $\sum_{k=0}^{11} (1 - 2^{-k})^{4096} \simeq 0.15$.

It is also useful to notice that the generation of one $b_{1/2}$ bit costs three memory accesses and one logical operation, while the comparison itself costs only one logical operation per bit. It is therefore important to limit the number of $b_{1/2}$ bits necessary for the computation of the different b_p bits. In the 3D Ising model, for example, three Boltzmann factors are needed,

$$p = e^{-\beta J}, \qquad p' = p^2, \qquad p'' = p^3 \tag{3.11}$$

Supposing p to be represented on n bits, the exact values of p' and p'' are respectively represented on $2n$ and $3n$ bits. With the method currently used (Algorithms 1 and 2), the direct computation of three independent bits r, r', and r'' with probabilities p, p', and p'' results either in a loss of accuracy or in a waste of time. However, for a given spin, only one of the three

Boltzmann factors p, p', p'' is selected. It is thus possible to use the *same* sequences of $b_{1/2}$ factors to compute the b_p associated to a given spin. In fact, it is more convenient to generate three independent bits ρ_1, ρ_2, ρ_3 with the same probability p. The bits r, r', r'' are generated by the logical products of one, two, or three of these bits ρ_i. It takes three sequences of $n\, b_{1/2}$ bits because the ρ_i have to be independent. More generally, let us suppose that we have K independent Boltzmann factors for a given spin, that we are able to compute simultaneously $d\, b_p$ bits with the same sequence of $b_{1/2}$ bits, and that we find I sets of d numbers $\{a(i, 1),..., a(i, d)\} \subset [0, 1[$, i varying from 1 to I, such that all the Boltzmann factors can be obtained by

$$p_k = \prod_{i=1}^{I} a(i, j_{i,k}) \tag{3.12}$$

In this formula we use the convention that $j_{i,k} \in [0, d]$ and $a(i, 0) = 1$. We can then compute the b_p bits associated to the $a(i, j)$, $j \in [1, d]$, for a fixed i with the same sequence of $b_{1/2}$ bits (a different sequence for each i). We can obtain the b_p bits associated with the Boltzmann factor p_k by the logical product of the b_p bits associated with the $a(i, j_{i,k})$, since they are independant. The determination of the b_p bits will thus take $3n \times I$ memory accesses at most. For example, let us consider a model (chiral Potts model[13]) where the needed Boltzmann factors are $p = \exp(-x_i)$. The x_i assume the values

$$\begin{matrix} E, & 2E, & 3E, & 4E, & E-\varepsilon \\ E+\varepsilon, & 2E-\varepsilon, & 2E+\varepsilon, & 3E-\varepsilon, & 3E+\varepsilon \end{matrix} \tag{3.13}$$

where E and ε are two positive quantities and $\varepsilon < E$. The p can be obtained by the product of two numbers $a(i, j) = \exp[-x(i, j)]$ with two different i's. The $x(i, j)$ take the values

$$0, \quad E-\varepsilon, \quad \varepsilon, \quad 4E \tag{3.14}$$

for $i = 1$, and

$$0, \quad E, \quad 2E, \quad 3E \tag{3.15}$$

for $i = 2$. This corresponds to $I = 2$, $d = 3$, and $K = 10$ in the above formulation. All the probabilities for the same i can be generated with the same set of $b_{1/2}$ bits.

3.3. Summary

We use a different program for each temperature. Thus, in the generation of a vector of V words of W bits (we have $VW = t = 4096$ on Cray-1) a step consists in (1) the computation of a vector of $b_{1/2}$ bits, which takes $3V$ memory accesses and V logical operations; and (2) V logical operations and V arithmetical operation (or "exclusive-or") according to Algorithm 5.

As previously pointed out, the algorithm will stop after at most $\log_2(t)+2$ steps on the average ("at most" will not be repeated in the following). If we can compute simultaneously d sequences of probabilities with the same $b_{1/2}$-bit sequence, the complete determination of all the Boltzmann factors will thus take $3VI[\log_2(t)+2]$ memory accesses, $VI[\log_2(t)+2]$ logical operations, and $dVI[\log_2(t)+2]$ arithmetical operations. Notice that at most $I = \lceil K/d \rceil$, where K is the number of different Boltzmann factors ($\lceil \;\; \rceil$ denotes nearest upper integer).

3.4. Discussion and Comparison of the Different Algorithms

Let us consider a machine where the time of a scalar (involving a single word) operation is S, whereas it is $D + V/r$ for a V-word vector operation; D is the startup time and r the vector to scalar speedup ratio. We consider the time T needed to compute VWb_p bits using the different algorithms:

1. Improved version of the Williams–Kalos algorithm (Algorithm 2).

We assume that all the logical operations can be performed during the memory accesses needed to compute the $b_{1/2}$ bits. This is the case for a Cray-1. One then has

$$T_2 = 3n(D + V/r) \qquad (3.16)$$

2. Single-bit version of our algorithm (Algorithm 3),

$$T_3 = SVW(2) \qquad (3.17)$$

3. Multispin (scalar) version of our algorithm (Algorithm 5).

Due to the structure of the algorithm, all the logical and arithmetic operations (for one step) can be done simultaneously with the memory accesses (cf. the Appendix) needed to compute one $b_{1/2}$ bit. One has

$$T_{5S} = SV[\log_2(W) + 2] \qquad (3.18)$$

4. Vectorial version of our algorithm (Algorithm 5).

The same remark as made in point 3 applies and one has

$$T_{5V} = (D + V/r)[\log_2(V) + \log_2(W) + 2] \tag{3.19}$$

Equation (3.17) shows that the bit implementation is not efficient on a general-purpose computer, due to the gain of time in generating simultaneously W bits $\{$there is a ratio $T_3/T_{5S} = 2W/[\log_2(W) + 2] > 1$ between the bit and scalar versions$\}$.

Equations (3.18) and (3.19) show that the time gained in the vectorization is partially lost because we are dealing with many more bits in parallel and are thus making a greater number of useless operations before completion. There is an optimal size of the vector that maximizes the improvement due to the vectorization. On the Cray-1, we have $D = S = 11$, $r = 1$, and $W = 64$. This gives an optimal size of $\simeq 101$ for V. We therefore choose the maximum size of vectors allowed on Cray-1, $V = 64$. This gives a ratio of ~ 5 between the vector and the scalar versions on a Cray-1. This ratio will of course be machine-dependent.

There is a ratio $T_2/T_{5V} = n/[\log_2(V) + \log_2(W) + 2]$ between Algorithm 2 and our algorithm. Thus, if a good accuracy (large n) is needed, our algorithm is faster. Furthermore, due to the structure of our algorithm, we are able to compute *simultaneously* $d = 3$ (on Cray-1) Boltzmann factors. This means that in the case where K Boltzmann factors are needed the ratio becomes

$$T_2/T_{5V} = Kn/l[\log_2(WV) + 2] \tag{3.20}$$

For example, for the 3D Ising model (three Boltzmann factors) our algorithm is (for an infinite accuracy) 24/14 times as fast as Algorithm 2 (for only 1/256 of accuracy).

As far as the number of $b_{1/2}$ bits needed is concerned, our algorithm is not optimal. Fourteen $b_{1/2}$ bits are required to produce a random b_p bit. Algorithm 3 (or Algorithm 4) would use only two $b_{1/2}$ bits per b_p bit. As already seen, it is not worth implementing it on a general-purpose computer. But as it involves logical operations only, and necessitates a very short sequence of $b_{1/2}$ bits, it would be perfectably suitable for specially designed computers. Furthermore, Knuth and Yao(14) have given an optimal method which requires only one $b_{1/2}$ bit per random b_p bit. However, this method handles trees. This entails either conditional branches or search in tables, i.e., memory accesses, which are both slow operations on the Cray-1 as well as on most other machines.

4. TESTS

In order to test the random-number generator, we used it in a MC program of the two-dimensional Ising model. The program was written in Cray-1 assembler language for reasons of efficiency (and also due to the poor optimization of the Fortran compiler). The instructions of the random-number generator were generated by a Fortran program for each temperature. We limited ourselves to 40 bits of accuracy ($\sim 10^{-12}$) in the determination of the probabilities. All computations were made on a 256×256 lattice with periodic boundary conditions. As our purpose was not to make real physical measurements on the 2D Ising model, we limited ourselves to the magnetization. The initial configuration is the totally ordered one ($T = 0$). After an equilibration of approximately 2×10^4 spin-flip attempts for each spin, the magnetization of the whole lattice is measured after each passage through the whole system. The results are given in Table I with the exact values[10] and an estimate of error bars. The errors bars are estimated by

$$\delta m \simeq (2\tau\chi/nL^2)^{1/2} \tag{4.1}$$

where L^2 is the number of spins in the lattice, n the number of measurements, χ the static susceptibility, and τ the autocorrelation time for the magnetization. We have 10^5 measurements at each temperature. The numerical values of χ and τ are taken respectively from Refs. 15 and 16. Computed and exact values of the magnetization agree, even when close to the critical temperature. In order to measure the speed of the program, we used the crude formula

$$\text{rate of spin} - \text{flip} = \frac{(\text{number of spins})(\text{number of passes})}{\text{time}} \tag{4.2}$$

Table I. Results of the Simulation on the 2D Ising Model[a]

Temperature	Exact magnetization	Measured magnetization	Estimated absolute error
1	0.911319	0.911337	5×10^{-5}
1.111111	0.749323	0.748497	10^{-3}
1.127395	0.648460	0.656104	9×10^{-3}
1.132502	0.556154	0.58429	8×10^{-2}

[a] In our units the critical temperature is 1.1345927.

The time includes the measurements, which are expected to be negligible. On the Cray-1 on which the program was run, the measured rate was 57 million flips per second. This value is to be compared with the speed of the other programs in the literature. As we are able to compute three Boltzmann factors simultaneously, a 3D Ising model with this random-number generator will reach nearly the same speed. For practical reasons, we cannot perform simulations on another vector machine than the Cray-1. We therefore give a comparison of the speed of the different vector machines(17): the estimated speed of a Cray-1 is 0.16 gigaflops, whereas it is 0.4 gigaflops for a Cyber 205 6×2 and 1.3 gigaflops for a NEC SX-2. We can use these numbers to estimate the speed of our program on these machines. This gives an extrapolated speed of 142 million flips per second on a Cyber 205 and 463 on a SX-2 computer. Of course this estimation has to be taken with great care: we assume that this speed is a good indication of the performance of the machines when performing vector operations, even if logical rather than floating-point operations are concerned. It is not obvious that this will hold in every case. The specific instructions or structure of each machine can also affect the performances. One can nevertheless see that our program should be, if not faster than, at least comparable in speed with the other canonical algorithms when implemented on the same machine [98 million flips per second on a two-pipe Cyber 205(18); 251 on a Nec SX-2(19); 218 on an Array processor(20) (the ICL array processor has a cycle time of 200 nsec and handles 64×64 bits simultaneously; the cycle time of the Cray-1 is 12.5 nsec and 64 bits are updated in one cycle; so we expect a ratio of four between the array processor and the Cray-1)]. Moreover, the performances of our program are not related to a specific size of the lattice, and we can evaluate the Boltzmann factors with an infinite precision. We are also free from the problem of correlation of random numbers that arises in Ref. 18. Of course for the Ising model it is much more efficient to use a microcanonical algorithm(21,22) (the best way to solve the random-number generator problem!). But it is not obvious that for more complicated cases such an efficient and simple implementation can be found.(21) In addition, the microcanonical algorithms have longer relaxation time than the canonical ones.

5. CONCLUSION

We have here presented a new random-number generator to compute bits equal to 1 with a probability p. This generator is adapted to multispin coding and offers a small computational time as well as an arbitrary accuracy in probability without any significant increase in computational time. Although it was designed to run on vector computers, since it uses logical operations only, it is perfectly implementable on other systems. Its performances could even be improved on specially designed machines or array processors where one can dispose of more than one logical unit (which is not the case on general-purpose processors).

APPENDIX

We give here the schematic implementation of the algorithm on a Cray-1 computer. Table II shows how vector resources of a Cray-1 are used to compute three nonindependent vectors of b_p bits from the same sequence of $b_{1/2}$ bits.

In boldface we represent one step of the algorithm described in Section 3.2.

M_i, M_{i-b}, and M_{i-a} stand for parts of the memory of addresses i, $i-b$, and $i-a$.

Each of the B_i, B_i', B_i'', S_i, R_i, R_i', R_i'',... stands for one of the eight vector registers.

On respectively first, second, and third lines are the memory accesses, the logical operations, and the arithmetical operations.

Three operations in a same column can be simultaneous, since they do not use the same operands. However, the result of an operation can be used immediately in another operation as it occurs for B_i'' and S_i.

$S_i \overset{?}{=} 0$ means that we check that *all* bits of vector S_i are zero and stop the computation if true.

Table II. A Schematic Description of the Implementation of the Algorithm on a Cray-1 Computer

Clockcycle $3i$	Clockcycle $3i+1$	Clockcycle $3i+2$
$\mathbf{B_i'} \leftarrow \mathbf{M_{i-b}}$	$\mathbf{B_i''} \leftarrow \mathbf{M_{i-a}}$	$\mathbf{M_i} \leftarrow \mathbf{B_i}$
$S_{i-1} \leftarrow S_{i-2} \wedge B_{i-1}$	$\mathbf{B_i} \leftarrow \mathbf{B_i'} \oplus \mathbf{B_i''}$	$(S_{i-1} \overset{?}{=} 0)$
$(R_i \leftarrow R_{i-1} \pm S_{i-1})$	$(R_i' \leftarrow R_{i-1}' \pm S_{i-1})$	$(R_i'' \leftarrow R_{i-1}'' \pm S_{i-1})$
Clockcycle $3i+3$	Clockcycle $3i+4$	Clockcycle $3i+5$
$B_{i+1}' \leftarrow M_{i+1-b}$	$B_{i+1}'' \leftarrow M_{i+1-a}$	$M_{i+1} \leftarrow B_{i+1}$
$\mathbf{S_i} \leftarrow \mathbf{S_{i-1}} \wedge \mathbf{B_i}$	$B_{i+1} \leftarrow B_{i+1}' \oplus B_{i+1}''$	$(\mathbf{S_i} \overset{?}{=} \mathbf{0})$
$(\mathbf{R_{i+1}} \leftarrow \mathbf{R_i} \pm \mathbf{S_i})$	$(\mathbf{R_{i+1}'} \leftarrow \mathbf{R_i'} \pm \mathbf{S_i})$	$(\mathbf{R_{i+1}''} \leftarrow \mathbf{R_i''} \pm \mathbf{S_i})$

The sequences R_i, R_i', R_i'' have, at the end of the calculus, the values R, R', R''. Let p, p', and p'' be the probabilities that the bits of R, R', and R'' are 1. If $p = 0$, $p_1 p_2$,..., then $(R_{i+1} \leftarrow R_i \pm S_i)$ stands for

$R_{i+1} \leftarrow R_i + S_i$	if $p_i = 0$ and $p_{i+1} = 1$
$R_{i+1} \leftarrow R_i - S_i$	if $p_i = 1$ and $p_{i+1} = 0$
nothing	if $p_i = p_{i+1}$ (which happens in half of the cases)

Actually, the three optional arithmetical operations are performed as soon as possible, so no arithmetical operation occurs during the last third of most of the steps.

$(S_i \overset{?}{=} 0)$ is performed only when $i \geqslant 13$. Thus, as explained before, there will be only an average of two steps with tests during the calculation.

These last two points are fortunate, because $R''_{i+1} \leftarrow R''_i \pm S_i$ and $S_i \overset{?}{=} 0$ cannot be simultaneous, since they both use S_i. In bad cases, the steps are longer.

ACKNOWLEDGMENTS

We thank the Centre de Calcul Vectoriel pour la Recherche, Palaiseau, for a grant of computer time, and a referee for valuable comments.

REFERENCES

1. K. Binder, in *Phase Transitions and Critical Phenomena*, Vol. 5b, C. Domb and H. S. Green, eds. (Academic Press, New York, 1976), p. 2.
2. G. F. Mazenko and O. T. Valls, *Phys. Rev. B* **24**:1419 (1981); R. Bausch, V. Dohm, H. K. Janssen, and R. K. P. Zia, *Phys. Rev. Lett.* **47**:1837 (1981).
3. R. B. Pearson, J. L. Richardson, and D. Toussaint, *J. Comp. Phys.* **51**:241 (1983); A. Hoogland, J. Spaa, B. Selman, and A. Compagna, *J. Comp. Phys.* **51**:250 (1983); A. F. Bakker, C. Bruin, F. van Dieren, and H. J. Hilhorst, *Phys. Lett.* **93A**:67 (1982); N. H. Christ and A. E. Terrano, *IEEE Trans. Comp. C* **33**:344 (1984).
4. R. H. Swendsen, in *Real Space Renormalization* (Topics in Current Physics, Vol. 30), T. W. Burkhardt and J. M. T. van Leeuwen, eds. (Springer-Verlag, 1982), p. 57.
5. R. Friedberg and J. E. Cameron, *J. Chem. Phys.* **52**:6049 (1970); L. Jacobs and C. Rebbi, *J. Comp. Phys.* **41**:203 (1981); C. Kalle and V. Winkelmann, *J. Stat. Phys.* **28**:639 (1982).
6. G. O. Williams and M. H. Kalos, *J. Stat. Phys.* **37**:283 (1984).
7. S. Wansleben, J. G. Zabolitzky, and C. Kalle, *J. Stat. Phys.* **37**:271 (1984).
8. N. Metropolis, A. W. Rosenbluth, A. H. Teller, and E. Teller, *J. Chem. Phys.* **22**:881 (1954).
9. *Cray-1 Hardware Reference Manual.*
10. C. N. Yang, *Phys. Rev.* **85**:809 (1952).
11. S. Kirkpatrick and E. P. Stoll, *J. Comp. Phys.* **40**:517 (1981); R. C. Tausworthe, *Math. Comput.* **19**:201 (1965); S. W. Golomb, *Shift Register Sequences* (Holden Day, San Francisco, 1967).

12. N. Zierler and J. Brillhart, *Inform. Contrib.* **14**:566 (1969).
13. D. Huse, *Phys. Rev. B* **24**:5180 (1981); S. Ostlund, *Phys. Rev. B* **24**:398 (1981).
14. D. E. Knuth and A. C. Yao, The complexity of nonuniform random number generation, in *Algorithms and Complexity*, J. F. Traub, ed. (Academic Press, 1976), pp. 357–428.
15. A. J. Guttmann, *J. Phys. A* **8**:1236 (1976).
16. E. Stoll, K. Binder, and T. Schneider, *Phys. Rev. B* **8**:3266 (1973).
17. *Le Monde de l'Informatique* (summer 1986).
18. G. Bhanot, D. Duke, and R. Salvador, *Phys. Rev. B* **33**:7841 (1986); *J. Stat. Phys.* **44**:985 (1986).
19. Y. O. Kabe and M. Kikuchi, Contrib. CP 5093 at the 16th International Conference on Thermodynamics and Statistical Mechanics. (Boston, August 1986).
20. S. F. Reddaway, D. M. Scott, and K. A. Smith, *Comp. Phys. Comm.* **37**:351 (1985).
21. H. J. Herrmann, preprint.
22. M. Creutz, *Phys. Rev. Lett.* **50**:1411 (1983).