Journal of Statistical Physics, Vol. 37, Nos. 3/4, 1984

A New Multispin Coding Algorithm for Monte Carlo Simulation of the Ising Model

G. O. Williams[1] **and M. H. Kalos**[1]

Received May 4, 1984

We present a new algorithm for Monte Carlo simulation of the Ising model. The usual serial architecture of a computer is exploited in a novel way, enabling parallel but independent calculations to be carried out on as many spins as there are bits in a computer word in each fundamental move. The algorithm enjoys a substantial increase in execution speed over more usual multispin coding algorithms. By its very nature, the algorithm constitutes a design for a special-purpose processor.

KEY WORDS: Ising model; Monte Carlo; multispin coding; parallel computation.

1. INTRODUCTION

The Ising model remains important in statistical physics for several reasons. The existence of an exact solution in two dimensions[1] and the fact that many of its properties are well known in three and more dimensions allow its use as a testing ground for new methods in the study of critical phenomena. Furthermore, there are many unresolved questions involving both the static and dynamic critical behavior of the Ising model.[2]

Imagine that the Ising model is being simulated by Monte Carlo methods using some variant of the sampling algorithm of Metropolis *et al.*[3-5] (M(RT)^2). It is possible to consider this the simulation of a stochastic dynamic system with time measured by (for example) attempted spin flips per site (Glauber dynamics).[6] It is now well known that, near the critical point, the autocorrelation time ("relaxation time") of the dynamic process increases dramatically ("critical slowing-down"). In fact, the autocorrelation

[1] Courant Institute of Mathematical Science, New York University, 251 Mercer Street, New York, New York 10012.

0022-4715/84/1100-0283$03.50/0

time τ is roughly proportional to $\min(L, \xi)^z$, where L is the linear size of the system being simulated, ξ is the correlation length of an infinite-volume system at the same temperature, and z is the dynamic critical exponent ($z \simeq 2$ for the usual kinetic Ising model in all dimensions[7]).

At the same time, the divergence of the correlation length near the critical point requires that larger lattices be simulated in order to avoid severe systematic errors due to finite-size effects. Specifically, one must take $L \gtrsim \xi$. Thus, one must sweep a lattice of $\gtrsim \xi^d$ sites about ξ^z times (the coefficient is unknown) in order to obtain *one* independent data point. Since statistical errors behave as $N^{-1/2}$, where N is the number of effectively independent samples, it is necessary (for example) to perform on the order of $10^6 \times \xi^{d+z}$ elementary operations in order to obtain an accuracy of $\pm 0.1\%$ at *one* temperature (the accuracy for the critical exponent will, in general, be greatly inferior). This is an enormous computational demand.

Attempts to resolve these difficulties can be divided into two broad classes, according to their focus on the computational or on the physical aspect of the problem. In the former case, one aims to considerably improve the computational speed while remaining within the framework of the standard single-spin-flip Metropolis–Glauber dynamics, and there are three approaches to this problem. One obvious choice is to perform computations on faster machines, and each new generation of super-computers has produced a spate of theretofore unattainable results.[8] A second and more recent approach has been the development of special-purpose computers dedicated to the study of a single problem or class of problems.[9–12][2] A third and more subtle approach is to reorganize the data structure and the computational algorithm so as to produce a signifiant improvement in computational efficiency. The multispin coding algorithms of Friedberg and Cameron,[13] Jacobs and Rebbi,[14] and others[15] fail into this category, as does an unusual approach by Harding.[16] The new algorithm proposed in the present paper is a further improvement in this direction.

A second, and often overlooked, line of attack, is to devise a radically different physical dynamics that would have less severe critical slowing down than the standard dynamics yet would still be computationally feasible. This approach is still in its infancy. An early suggestion was that of Bortz *et al.*,[17] who proposed a method of randomly choosing a spin to flip with a probability proportional to the probability that the given flip be accepted. This has been seen to result in a significant decrease in the computational labor required for $T \lesssim T_c$. Recently, Kalos[18] has proposed a class of

[2] References 9 and 10 describe Ising model processors. Reference 11 describes a special-purpose computer for continuum (nonlattice) particles interacting through a continuous potential. Reference 12 describes a special-purpose computer designed to perform lattice gauge theory calculations.

methods which he calls "collective-mode Monte Carlo," and Schmidt[19] has proposed an unorthodox treatment employing approximate Kadanoff transformations[20] within a Monte Carlo calculation.

In this paper, we introduce a new computational algorithm, which we call "super-spin coding," for Monte Carlo studies of Ising models using the standard Metropolis–Glauber dynamics. For purposes of exposition, we treat the three-dimensional simple-cubic ferromagnetic Ising model in the absence of an external field. The method is easily extended to other dimensions, other lattices, antiferromagnetic couplings, and an external field. Furthermore, our method may also be generalized to other lattice spin models.

In contrast with contemporary multispin coding algorithms, which eventually require each spin to be processed individually, our algorithm proceeds entirely in parallel, simultaneously performing standard $M(RT)^2$ Monte Carlo on as many spins as there are bits in a computer word. Our algorithm has the further advantage of using nothing but Boolean operations,[3] and as such constitutes a design for a special-purpose computer, the implications of which we will discuss elsewhere.

In the next section, we will present our algorithm in some detail. The third section will be devoted to a comparative timing analysis of this algorithm and a modern multispin coding algorithm.[15] In an appendix, we will discuss requirements on the pseudorandom numbers needed for our algorithm and decribe the pseudorandom number generator we have selected together with some details about its testing. A second appendix contains a listing of our program.

2. THE METHOD

Since Ising spins are two-state objects, they can be represented by a single bit in a computer word. This has long been recognized, but has not been fully exploited, except in a remarkable paper by Friedberg and Cameron[13] which appears to have been overlooked by many practitioners of Ising Monte Carlo. Their technique, which was illustrated for a two-dimensional Ising model, is a forerunner of multispin coding. Their algorithm

[3] Although not part of the ANSI standard, almost all FORTRAN compilers support direct linkage to the Boolean operations available on an assembly language level. We make use of the logical operations AND, OR, XOR (exclusive or), each of which act in bitwise fashion on two full-word arguments (some compilers allow more than two arguments), and COMPL, which returns the bitwise Boolean complement of its single argument. We also make use of the nonarithmetic SHIFT instruction in applying periodic boundary conditions; this instruction varies greatly from machine to machine, and may appear as either left or right and as either logical (circular) or arithmetic (zero fill).

and ours both proceed entirely in parallel and are expressed in Boolean operations. Our method, however, satisfies ergodicity.

Recall that in one variant of the $M(RT)^2$ algorithm one proposes to flip a given spin, computes the change in energy of the system ΔE due to this flip, and accepts or rejects the move with probability p according to the Metropolis function

$$p = \min[1, \exp(-\beta \Delta E)] \tag{1}$$

where $\beta = 1/k_B T$, k_B being the Boltzmann constant and T being the temperature. While contemporary multispin coding algorithms are able to perform a portion of the procedure on several spins in parallel, they require that the decisions to accept or reject the proposed moves be made serially. We stress that our algorithm performs all steps of the Monte Carlo procedure entirely in parallel on as many spins as there are bits in a word.

The algorithm has three essential elements which we shall describe in turn. The first is a particular data structure in which spin variables are assigned to bits in words. The key feature is that successive spins in a direction, say, parallel to the x axis are coded into the same bits in successive words. Then determining the changes in energy which follow parallel but independent attempts at flipping all spins represented in one word simply entails comparing all bits pairwise with bits in six other words. If the acceptance probabilities can be represented by a finite number of bits, then the decision whether to independently accept or reject the flip of each spin can be carried out by a succession of Boolean operations over a finite number of bits.

In what follows, we shall assume we are dealing with a computer having N_B bits per word (in practice, we have used machines with $N_B = 32$, 60, and 64). Consider a simple-cubic Ising model in three dimensions of size $L \times L \times L$, where $L = kN_B$, $k \geqslant 2$. We store the spins in memory as follows: Visualize a row of spins in the lattice parallel to the x axis (Fig. 1). The row of spins is then placed in k successive words, each spin occupying a single bit, with the jth word containing spins $j, j + k, j + 2k, \ldots, j + (N_B - 1)k$. An up spin is represented as a 1, while a down spin is represented as a 0.

This procedure is continued until all the spins are stored in $k^3 N_B^2$ computer words. In a FORTRAN program, these spins would most naturally be stored in an INTEGER array IS(k, kN_B, kN_B) with the second and third indices, respectively, labeling the y and z coordinates of a given row.

Consider the spins in the word IS(I1, I2, I3) (whose value we shall also call ICI). Each spin has six nearest neighbors, two within the same row and four in other rows. Under the imposition of periodic boundary conditions, these other four spins lie, modulo kN_B, in the words IS(I1, I2 − 1, I3), IS(I1,

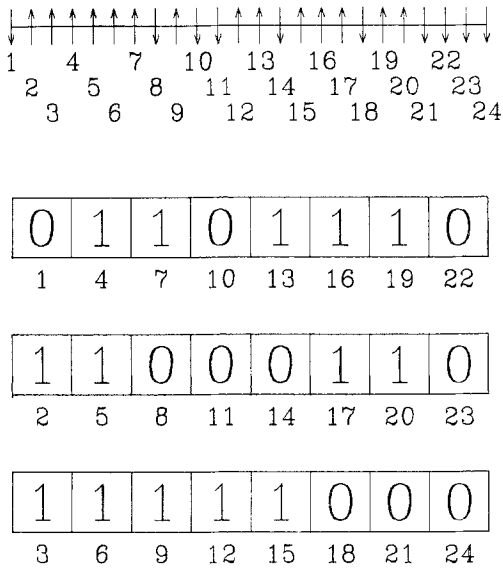

Fig. 1. One row of spins in the lattice and its representation in memory for the particular choices $k = 3$, $N_B = 8$, and $L = 24$. An up spin is represented as a one and a down spin is represented as a zero.

I2 + 1, I3), IS(I1, I2, I3 − 1), and IS(I1, I2, I3 + 1). Furthermore, the neighboring spins occupy the same bit positions as the spins in ICI. The remaining two neighbors lie, modulo k, in IS(I1 − 1, I2, I3) and IS(I1 + 1, I2, I3), again in corresponding bit positions. When I1 = 1 or I1 = k it is necessary to perform a left or right circular shift of one of these two adjacent words in order to maintain the correspondence among bit positions.

We now describe how to compute the change in energy associated with the proposed flip of each spin in ICI. The exclusive or (XOR) of two bits returns one if the bits are different and zero if they are the same. Thus, performing the XOR of all six neighboring words with ICI produces six words (which we shall call NN(I), I = 1, 2,..., 6) which characterize the energy of the N_B spins in ICI.

Counting the number of one bits in a given bit position over all six of the NN(I) then gives the number of unsatisfied bonds for that spin in ICI. This is accomplished by a "Boolean Bubble Sort" on these six words. The following FORTRAN fragment illustrates this procedure. The variables NN(I) represent the results of the six XOR operations, and the variables N(I), I = 1, 2,..., 6 represent the sorted words:

```
      N(1) =  OR(NN(1),NN(2))
      N(2) = AND(NN(1),NN(2))
         DO 20 I = 3,6
         N(I) = AND(N(I-1),NN(I))
            DO 10 J = I-1,2,-1
            N(J) = OR(N(J),AND(NN(I),N(J-1)))
10          CONTINUE
         N(1) = OR(N(1),NN(I))
20       CONTINUE
```

(F.1)

N(1) is seen to be the logical or of all the NN(I), N(2) the logical or of all pairwise ands of the NN(I), and so on, so that the result of executing this code is that the lth bit of N(J) is a one if the spin in the lth bit position of ICI is in a different state from at least J of its neighbors. Figure 2 illustrates the effect of this code. In practice, it is not necessary to compute N(5) and

NN(1)	0	0	0	0	1	1	1
NN(2)	0	0	0	1	1	0	1
NN(3)	0	0	1	1	0	1	1
NN(4)	0	1	0	0	1	1	1
NN(5)	0	0	0	1	0	1	1
NN(6)	0	0	1	0	1	1	1

N(1)	0	1	1	1	1	1	1
N(2)	0	0	1	1	1	1	1
N(3)	0	0	0	1	1	1	1
N(4)	0	0	0	0	1	1	1
N(5)	0	0	0	0	0	1	1
N(6)	0	0	0	0	0	0	1

Fig. 2. An example of the effect of the Boolean Bubble Sort code. Above are shown the original words NN(I), I = 1, 2,..., 6. Below are shown the sorted words N(I), I = 1, 2,..., 6. N(1) is constructed as the logical or of all the NN(I), N(2) the logical or of all pairwise ands of the NN(I), and so on, with the result being that the bits of each column in the figure are rearranged so that all the one bits precede all the zero bits.

N(6), and N(4) needs to be computed only in the presence of an external field.

We now address the method of accepting or rejecting the proposed move. For simplicity, we will discuss only the zero-field algorithm for the ferromagnetic three-dimensional Ising model. The generalizations to nonzero field, antiferromagnetic couplings and other dimensions are straightforward and are briefly discussed at the end of this section.

In the usual $M(RT)^2$ algorithm, one proposes a move (a spin flip in our case) and computes the change in evergy ΔE which would result from the move. If the energy of the system is lowered, the move is accepted with probability 1. If, on the other hand, the change in energy ΔE is positive, the move is accepted with probability $p = \exp(-\beta \Delta E)$. This is merely a restatement of Eq. (1).

For a three-dimensional simple-cubic Ising model, ΔE can take on only certain values. If n is the number of unsatisfied bonds a given spin has in its initial state, then

$$\Delta E = 4K(n - 3) \tag{2}$$

where K is the coupling constant (assumed positive for now) and n can take on the values $0, 1, 2, \ldots, 6$. Note that the move will be accepted with probability $p = 1$ if $n \geqslant 3$, and, if $n \leqslant 2$, the move will be accepted with probability $p = w^{3-n}$, where $w = \exp(-4\beta K)$. In practice, one draws a random number x, uniformly distributed on $[0, 1]$, and accepts the move if $x \leqslant p$.

Note that by writing p as the product of factors $p = p_1 p_2 p_3$ one can alternatively draw the random numbers x_1, x_2, and x_3, and accept the move if $x_i \leqslant p_i$, $i = 1, 2, 3$. Thus all probabilities can be written as products of w and 1. For certain choices of βK, w can be represented exactly in M bits. These properties are fully exploited by our algorithm.

Suppose that $\hat{w} = 2^M w$ is an integer between zero and $2^M - 1$. Assume further that $\hat{x}$ is a sequence of M uniformly distributed random bits. Then $\hat{x}$ represents a random integer uniformly distributed between zero and $2^M - 1$. We seek to determine if $\hat{x} < \hat{w}$. This can be accomplished by successively comparing the bits of $\hat{x}$ and $\hat{w}$, beginning at the most significant bit. As soon as two bits are found to differ, it is known that $\hat{x}$ and $\hat{w}$ differ. If the given bit in $\hat{w}$ be a one, then $\hat{x} < \hat{w}$, else $\hat{x} > \hat{w}$. If no decision be reached upon exhausting all M bits, then $\hat{x} = \hat{w}$.

This primitive approach, not unlike the bit-serial subtraction techniques used in the earliest digital computers, becomes attractive when it is considered for the parallel determination of N_B such inequalities. Consider M successive words in a computer. Let each bit in the jth word be set to the

value of the jth bit of $\hat{w}$. Then, if M successive words of random bits are generated, the above bit-serial comparison can be applied in parallel, the lth bits in each of the M succeeding random words together defining one of the N_B random numbers $\hat{x}$.

The following FORTRAN fragment illustrates one implementation of this procedure. A bit within ID is set to one once the corresponding inequality is decided, and a bit of IC is set to one if the corresponding $\hat{x} < \hat{w}$. The INTEGER function NYUBIT, a feedback shift register pseudorandom number generator described in Appendix I, returns a full word of random bits. IW(J), J = 1, 2,..., M, represents the bitwise transpose representation of $\hat{w}$ introduced above:

```
      IC = 0
      ID = 0
         DO 10 J = 1,M
         IX = XOR(IW(J),NYUBIT(J))
         IC = OR(IC,AND(COMPL(ID),IX,IW(J)))
         ID = OR(ID,IX)
10       CONTINUE
```
(F.2)

The "change" word IC which results from this code is equivalent to a "thermal" word of Friedberg and Cameron.[13] This procedure is actually performed three times, generating change words IC1, IC2, and IC3. The final change word can be constructed as follows:

```
      M0 = COMPL(N(1))
      M1 = XOR(N(1),N(2))
      M2 = XOR(N(2),N(3))
      M3 = N(3)
      IC = OR(AND(M0,IC1,IC2,IC3),
     +        AND(M1,IC1,IC2),
     +        AND(M2,IC1),
     +        M3)
```
(F.3)

This expression, while not the simplest that can be written, reveals the content of IC. At this point, each bit of IC reflects the outcome of the Monte Carlo move of the corresponding spin in ICI. A one bit in IC implies that the move has been accepted, so that the statement

```
      IS(I1,I2,I3) = XOR(IC,ICI)
```
(F.4)

generates the updated word of spins.

2.1. Generalizations

The extensions required for simulations in $d > 3$ are obvious. In $d = 2$, our algorithm runs in the same time as that of Friedberg and Cameron[13] and is to be preferred since their algorithm does not satisfy ergodicity.

Lattices other than simple cubic require only a reorganization of the data structure. This is essentially defined by the unit cell, although underlying sublattices will impose natural simplifications. Care must be taken in implementing the desired boundary conditions.

The only change necessary to simulate an antiferromagnetic Ising model is in determining the energy. Since here antiparallel bonds are of lower energy, it is necessary to count the original number of parallel bonds. Thus, instead of forming the exclusive ors of the six neighboring words with ICI, one forms them with the Boolean complement of ICI. The remainder of the algorithm proceeds as in the ferromagnetic case.

In the presence of an external field, if $w \leqslant \exp(-h)$, then the probability of accepting a move can be expressed in terms of three factors, w, $\exp(-h)$, and $w\exp(h)$. These must simultaneously be written in the form $w = \hat{w}/2^M$, $\exp(-h) = \hat{h}/2^{M_h}$, and $w\exp(h) = \hat{w}_+/2^{M_+}$, where $M_+ = M - M_h$ and $\hat{w}_+ = \hat{w}/\hat{h}$ is an integer. This would seem to imply that only a few fields may be studied at any temperature. However, if one takes M sufficiently large, the error introduced in an approximate representation of these factors becomes negligible, and computations can be made at many values of the field.

2.2. Testing

In order to assure ourselves of the correctness of the algorithm, we have simulated Ising models in two and three dimensions. We have observed the energy and magnetization over a wide range of temperatures and found agreement with existing results.[1,4,21] We have also observed the exponential decay of magnetization for $T > T_c$.[15]

3. COMPARISON WITH MULTISPIN CODING

We have prepared FORTRAN benchmarking programs to carry out both our algorithm, which we shall refer to as super-spin coding, and the usual multispin coding algorithm as most recently refined by Kalle and Winkelmann.[15] We have made comparisons of execution speed on a VAX-11/780 with floating point accelerator ($N_B = 32$ and $L = 64$), a Floating Point Systems FPS-164 ($N_B = 64$ and $L = 128$), and a PUMA computer[22] built at the Courant Institute ($N_B = 60$ and $L = 120$). This last machine

executes the instruction set of the CDC 6600, although it is somewhat slower.

In the multispin coding algorithm, $N_B' = N_B/n_b$ spins are coded into a single word (n_b can be three in the case of zero field, but must be four in the presence of an external field). The change in energy due to N_B' proposed spin flips is computed in parallel, but the determination whether to accept or reject each proposed move must be made serially. The spin-flip rate can be expressed as

$$R_m = \frac{R_m^0 N_B'}{1 + \alpha_m N_B'} \tag{3}$$

where α_m is the ratio of the work done per spin in the serial part of the code to the work done per word of spins in the parallel part of the code. R_m^0 is the number of words processed per second by the parallel part of the code. For the computers we have used in our tests, $\alpha_m \simeq 0.2$, as may be determined by using different values of n_b or by directly examining the machine instructions generated by the FORTRAN compiler.

Because the VAX and FPS-164 have word lengths which are not divisible by three, we have made multispin coding computations there in four bits ($n_b = 4$), so that the execution times we quote for these machines are close to what they would be in the presence of an external field. The PUMA allows both $n_b = 3$ and $n_b = 4$, and we have compared execution times for these two values there. Since the PUMA emulates the CDC 6600, no changes were necessary in the program published by Kalle and Winklemann,[15] which was written for a CDC 7600. Minor changes were made to tune their code to the VAX and FPS-164 architectures.

The super-spin coding algorithm has a rate given by

$$R_s(M) = \frac{R_s^0 N_B}{1 + \alpha_s(M - 1)} \tag{4}$$

where M is the number of bits needed to represent w, and α_s is the ratio of the work required per bit in making the N_B parallel decisions whether or not to accept the proposed moves to the work required for the remainder of the algorithm. R_s^0 is the number of full words of spins processed per second outside the acceptance determination, and is within a factor of two of R_m^0. Note that $R_s^0 N_B$ is the rate at which we can flip spins at $w = 1/2$. α_s is seen to vary between about 0.4 and 0.6 for the machines we have used. In Table I we present R_m, $R_s(8)$, and $R_s^0 N_B$ for each of the machines used. In addition, we give M_c, for which $R_s(M_c) = R_m$.

The surprising discrepancy in speeds on the FPS-164 is due in part to the branching inherent in the multispin coding algorithm and the consequent

Table I

Machine	N_B [a]	N'_B [b]	R_m [c]	$R_s(8)$ [d]	$R_s^0 N_B$ [e]	M_c [f]
VAX-11/780	32	8	25,000	40,000	204,000	13
FPS-164	64	16	128,000	601,000	2,180,000	44
PUMA[g]	60	20	24,500	71,000	271,000	26
PUMA	60	15	22,700	71,000	271,000	28

[a] Word size.
[b] Number of spins per word in multispin coding.
[c] The multispin coding flip rate.
[d] The super-spin coding flip rate when $w = \exp(-4\beta K)$ is represented in $M = 8$ bits.
[e] The super-spin flip rate for $M = 1$.
[f] The value of M for which multispin and super-spin coding execute in the same time.
[g] The PUMA computer[22] executes the instruction set of the CDC 6600. For that machine, we give flip rates for multispin calculations done in three ($N'_B = 20$) and four ($N'_B = 15$) bits.

inability of the optimizing compiler to take full advantages of the pipelined architecture, as it was able to with the super-spin code. However, the primary reason for this difference may be seen in the dependence of R_m and R_s on word size. If one doubles the word size from 32 to 64 bits, then, for given R^0, the super-spin rate will double, while with $\alpha_m = 0.2$, the multispin rate will only increase by about 7%.

Since the super-spin algorithm contains no branching, it is ideally suited to a vector machine, such as a CRAY. Vectorizing the algorithm would allow one to simulate, for example, 64 lattices simultaneously. Comparing the instruction cycle time of the CRAY 1 to that of the FPS-164 gives a conservative estimate of $R_s^0 N_B = 3.2 \times 10^7$ spin flips per second. It is heartening to predict a spin flipping rate from a FORTRAN program run on an admittedly very fast, but still general purpose, computer that meets the design speed of the faster of the two Ising model processors in use today.[9]

ACKNOWLEDGMENTS

We wish to thank A. D. Sokal for his many contributions and continued encouragement. This work was supported in part by a grant from the National Science Foundation, No. CHE-800 112 85, and a contract from the United States Department of Energy, No. DE-AC02-76 ERO 3077.

APPENDIX I: PSEUDORANDOM NUMBER GENERATORS

Our algorithm requires Md sequential words y_i of random bits of length N_B to generate $N_B d$ random integers x_j between 0 and $2^M - 1$ for each word

of spins processed in a d-dimensional simulation. We require that the x_j pass any usual test of randomness. One simple such test is a check for uniformity and apparent lack of correlation in one- and two-dimensional distributions.

Note that our requirements impose stringent requirements on the lack of sequential correlation among the y_i, which in a more usual computation scheme would themselves constitute the random numbers. Thus a pseudorandom number generator (RNG) which meets our requirements would, if computationally efficient, be an ideal choice for any programming application requiring one.

Linear congruential RNGs are perhaps the most common ones in use today. These generate pseudo-random numbers by the first-order recursion

$$y_{i+1} = \lambda y_i + r (\text{mod } m) \tag{A.1}$$

However, their less significant bits are highly correlated. We note in passing that the particular choice of $\lambda = 11^{13}$, $r = 0$, and $m = 2^{48}$ has seen long service at the Courant Institute and has been extensively tested without negative results. We refer the reader to a paper by Borosh and Niederreiter[23] for a tabulation of many other choices together with their properties.

A second class of RNG are a special case of those introduced by Tausworthe,[24] which are given the name feedback shift register RNGs. These generate pseudorandom numbers by the recursion

$$y_i = XOR(y_{i-q+p}, y_{i-p}) \tag{A.2}$$

which is itself generated by the primitive trinomial $X^p + X^q + 1$. A large number of primitive trinomials have been identified by Zierler and Brillhart,[25] and the properties of some of the resulting RNGs have been obtained by Lewis and Payne,[26] Niederreiter,[27] and Fushimi and Tezuka.[28] Feedback shift register RNGs are computationally efficient, address computations exceeding the single exclusive or entailed.

While there are fewer exact results available on the statistical properties of RNGs of this type than there are for linear congruential RNGs, a number of such results do obtain.[26–28] For example, linear independence among the initial p words of the sequence is necessary and sufficient to obtain the maximum period $(2^p - 1)$ of the generator. We recommend the initialization procedure attributed to J. A. Greenwood by Kirkpatrick and Stoll,[29] and direct the reader to Ref. 28 for a formal treatment of this approach. A second initialization procedure which we have found satisfactory, although we can offer no proof of its validity, is a permutation of the first p elements obtained by any other initialization procedure.

We have tested three pairs (q, p): (15, 127), (30, 127), and (103, 250). The first two pairs are currently in use in the two Ising model spcial-purpose computers, the former at Delft,[10] and the latter at Santa Barbara.[9] The third pair was investigated by Kirkpatrick and Stoll.[29] All three are satisfactory for our purposes, although we note that the pair (30, 127) has slightly better statistical properties than the other two and is to preferred as well in terms of a loose criterion stated by Lewis and Payne.[26]

When simulating a d-dimensional model, we construct a random integer $\hat{x}$ from M bits taken from the same bit position of the random words $y_0, y_d, \ldots, y_{(M-1)d}$. We have observed the x_j to exhibit no apparent nonuniformity in one and two dimensions for $d = 2$, 3, and 4 and for $M = 1, 2, \ldots, 16$. These empirical results, together with the results both analytical and empirical of other authors, suggest that feedback shift register RNGs are both reliable and efficient. We urge their adoption, particularly on small word size machines, where good linear congruential RNGs are usually computationally inefficient, and computationally efficient linear congruential RNGs are usually bad.

APPENDIX 2: FORTRAN CODE FOR THE SUPER-SPIN CODING ALGORITHM

We present here the central portion of our algorithm as coded for the PUMA computer. The program is designed to simulate an Ising model on a simple-cubic lattice of side $L = 120$ in three dimensions in zero field. The number of sweeps of the lattice (NPASS), the number of bits used to represent w (M), and $\hat{w}$ (NW) are read in and (in this version of the code) the lattice is set to a random configuration, NYUBITV returning a vector of words of random bits. The bitwise transpose of $\hat{w}$ is then constructed in the DO 10 loop.

The DO 40 loop processes one word of spins. First the exclusive or of each of the six neighboring words is constructed (NN1, NN2,..., NN6). Note that the SHIFT instruction provided by CDC's Extended FORTRAN is a circular shift. Other architectures provide only arithmetic shifts, and one must construct a circular shift from two arithmetic shifts, a logical or, and perhaps a logical and. These words are then sorted. Since this is a zero field case, we only construct N1, N2, and N3. The masks Mi, $i = 0, 1, 2$, and 3, are then words each of whose bits are one if the corresponding spin has exactly i unsatisfied bonds in its initial state.

The acceptance determination of each of the moves is carried out in the DO 30 loop. We have applied de Morgan's laws to the formulation discussed in Section 2 in order to reduce the computational overhead. The variables IC1, IC2, and IC3, as before, have bits which are set to one if the

corresponding inequalities $\hat{x}_i < \hat{w}$, $i = 1$, 2, and 3, are satisfied. The variables ND1, ND2, and ND3 are the Boolean complements of the variables IDi discussed in Section 2. That is, a given bit of NDi is a one if the corresponding inequality has not yet decided. In these expression, we also view the random bit strings returned by NYUBITV in the vector RAN as the Boolean complements of the random bit strings defining the random numbers $\hat{x}$, the complement of a random string of bits being a random string of bits.

Finally a simplified expression for the final change word IC is given, and the updated word of spins is generated and stored in memory.

```
      PROGRAM PBENCH(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
C
C      BENCHMARK PROGRAM FOR PARALLEL PROCESSING
C      OF THE ZERO-FIELD ISING MODEL.
C
CCC
C
      INTEGER IS(120,120,2),W(10),IP(120),IM(120),RAN(3600)
C
      DATA NB,NK,L /60,2,120/
C
CCC
C
      READ (5,*) NPASS,M,NW
      NR = 3*M*L
C
      CALL NYUBITV(IS,L*L*NK)
C
         DO 10 J = 1,M
         W(J) = 0
         IF (AND(1,SHIFT(NW,J-M)).EQ.1) W(J) = COMPL(0)
   10    CONTINUE
C
         DO 20 I = 1,L
         IP(I) = I+1
         IM(I) = I-1
   20    CONTINUE
      IP(L) = 1
      IM(1) = L
```

```
C
      KSH = 1
         DO 70 IPASS = 1,NPASS
            DO 60 K = 1,NK
            KP1 = 3-K
            KSH = NB-KSH
               DO 50 J = 1,L
               JP1 = IP(J)
               JM1 = IM(J)
               CALL NYUBITV(RAN,NR)
               JJ = 0
                  DO 40 I = 1,L
                  IP1 = IP(I)
                  IM1 = IM(I)
                  ICI = IS(I,J,K)
                  NN1 = XOR(ICI,IS(IP1,J,K))
                  NN2 = XOR(ICI,IS(IM1,J,K))
                  NN3 = XOR(ICI,IS(I,JP1,K))
                  NN4 = XOR(ICI,IS(I,JM1,K))
                  NN5 = XOR(ICI,IS(I,J,KP1))
                  NN6 = XOR(ICI,SHIFT(IS(I,J,KP1),KSH))
C
                  N2 = AND(NN1,NN2)
                  N1 =  OR(NN1,NN2)
                  N3 =       AND(N2,NN3)
                  N2 = OR(N2,AND(N1,NN3))
                  N1 = OR(N1,       NN3)
                  N3 = OR(N3,AND(N2,NN4))
                  N2 = OR(N2,AND(N1,NN4))
                  N1 = OR(N1,       NN4)
                  N3 = OR(N3,AND(N2,NN5))
                  N2 = OR(N2,AND(N1,NN5))
                  N1 = OR(N1,       NN5)
                  N3 = OR(N3,AND(N2,NN6))
                  N2 = OR(N2,AND(N1,NN6))
                  N1 = OR(N1,       NN6)
C
                  M0 = COMPL(N1)
                  M1 = XOR(N1,N2)
                  M2 = XOR(N2,N3)
                  M3 = N3
```

```
C
                  IC1 = AND(W(1),RAN(JJ+1))
                  ND1 = XOR(W(1),RAN(JJ+1))
                  IC2 = AND(W(1),RAN(JJ+2))
                  ND2 = XOR(W(1),RAN(JJ+2))
                  IC3 = AND(W(1),RAN(JJ+3))
                  ND3 = XOR(W(1),RAN(JJ+3))
                  JJ  = JJ+3
C
                     DO 30 KK = 2,M
                     IC1 = OR(IC1,AND(ND1,AND(W(KK),RAN(JJ+1))))
                     ND1 =        AND(ND1,XOR(W(KK),RAN(JJ+1)))
                     IC2 = OR(IC2,AND(ND2,AND(W(KK),RAN(JJ+2))))
                     ND2 =        AND(ND2,XOR(W(KK),RAN(JJ+2)))
                     IC3 = OR(IC3,AND(ND3,AND(W(KK),RAN(JJ+3))))
                     ND3 =        AND(ND3,XOR(W(KK),RAN(JJ+3)))
                     JJ  = JJ+3
  30                 CONTINUE
C
                  IC = OR(M3,AND(IC3,
    +                  OR(M2,AND(IC2,
    +                  OR(M1,AND(IC1,M0))))))
C
                  IS(I,J,K) = XOR(ICI,IC)
  40              CONTINUE
  50           CONTINUE
  60        CONTINUE
  70     CONTINUE
C
      STOP
C
      END
```

REFERENCES

1. L. Onsager, *Phys. Rev.* **65**:117 (1944).
2. *Phase Transitions* (Cargèse, 1980), M. Lévy, J. C. Le Guillou, and J. Zinn-Justin, eds. (Plenum Press, New York, 1982).
3. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *J. Chem. Phys.* **22**:881 (1954).
4. L. D. Fosdick, *Phys. Rev.* **116**:565 (1959); L. D. Fosdick, in *Methods in Computational Physics*, Vol. 1, B. Alder, S. Fernbach, and M. Rotenberg, eds. (Academic Press, New York, 1963).

5. C. P. Yang, *Proceedings of Symposia in Applied Mathematics*, Vol. 15, 351 (American Mathematical Society, Providence, Rhode Island, 1963).
6. Yu. Ya. Gotlib, *Fiz. Tverd. Tela* **3**:2170 (1961) [*Sov. Phys.-Solid State* **3**:1574 (1962)]; R. J. Glauber, *J. Math. Phys.* **4**:294 (1963).
7. 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).
8. D. Stauffer, *J. Appl. Phys.* **53**:7980 (1982).
9. R. B. Pearson, J. L. Richardson, and D. Toussaint, *J. Comp. Phys.* **51**:241 (1983).
10. A. Hoogland, J. Spaa, B. Selman, and A. Compagner, *J. Comp. Phys.* **51**:250 (1983).
11. A. F. Bakker, C. Bruin, F. van Dieren, and H. J. Hilhorst, *Phys. Lett.* **93A**:67 (1982).
12. N. H. Christ and A. E. Terrano, *IEEE Trans. Comp.* **C-33**:344 (1984).
13. R. Friedberg and J. E. Cameron, *J. Chem. Phys.* **52**:6049 (1970).
14. L. Jacobs and C. Rebbi, *J. Comp. Phys.* **41**:203 (1981).
15. C. Kalle and V. Winkelmann, *J. Stat. Phys.* **28**:629 (1982), and references therein.
16. M. P. Harding, *J. Comp. Phys.* **44**:227 (1981).
17. A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, *J. Comp. Phys.* **17**:10 (1975).
18. M. H. Kalos, in Proceedings of the Brookhaven Conference on Monte Carlo Methods and Future Computer Architectures, May 1983 (unpublished).
19. K. E. Schmidt, *Phys. Rev. Lett.* **51**:2175 (1983).
20. K. G. Wilson, *Rev. Mod. Phys.* **4**:773 (1975).
21. D. P. Landau, *Phys. Rev. B* **13**:2997 (1976); D. P. Landau, *Phys. Rev. B* **14**:255 (1976).
22. R. Grishman, "The structure of the PUMA computer system," Courant Mathematics and Computing Laboratory Report COO-3077-157 (1978).
23. I. Borosh and H. Niederreiter, *BIT* **23**:65 (1983).
24. R. C. Tausworthe, *Math. Comput.* **19**:201 (1965).
25. N. Zierler, *Inform. Contr.* **15**:67 (1969); N. Zierler and J. Brillhart, *Inform. Contr.* **13**:541 (1968); N. Zierler and J. Brillhart, *Inform. Contr.* **14**:566 (1969).
26. T. G. Lewis and W. H. Payne, *J. ACM* **20**:456 (1973).
27. H. Niederreiter, in *Probability and Statistical Inference*, W. Grossman and G. Pflug, eds. (D. Reidel, Dordrecht, 1982).
28. M. Fushimi and S. Tezuka, *Commun. ACM* **26**:516 (1983).
29. S. Kirkpatrick and E. P. Stoll, *J. Comp. Phys.* **40**:517 (1981).