Journal of Statistical Physics, Vol. 61, Nos. 3/4, 1990

Fully Parallel Code for Monte Carlo Simulation

T. J. P. Penna[1] and P. M. C. de Oliveira[1]

Received May 3, 1990; final June 1, 1990

We present a fully parallel version of Monte Carlo simulation of the Ising model using the Metropolis algorithm. In the 3-dimensional version the performance can be enhanced by a factor >20 in 16-bit word processors relative to other multispin codes. This factor could be further increased if implemented in 64-bit word computers.

KEY WORDS: Multispin code; Monte Carlo method; phase transitions.

1. INTRODUCTION

Monte Carlo simulations are widely used to study statistical models. A review of applications of this method in statistical physics can be found in refs. 1. The main difficulty found in the use of Monte Carlo simulations is the limitation of computers. This limitation can be overcome by improving the hardware, the software, or still increasing the information obtained from a simulation. Dedicated machines and vector computers have been developed allowing simulations in large systems. The multispin coding technique[2] is an important advance in software, due to memory saving and increasing speed characteristics.

In this paper, we present a new algorithm for Monte Carlo simulation for two- and three-dimensional Ising-spin systems, using the multispin coding technique in its total capacity, bypassing the step of generating one random number and comparing it with a Boltzmann factor for each spin *sequentially*. Our code has been tested in a 16-bit word processor, but the advantage, in comparison with traditional multispin codes, can be better for larger word sizes. In the next section we present the three-dimensional

[1] Instituto de Física, Universidade Federal Fluminense, C.P. 100296, 24000 - Niterói, RJ, Brazil.

0022-4715/90/1100-0933$06.00/0

version of another fast code.[3] In Section 3 we show how to build suitable random numbers for full parallelization.

A full copy of our C-program can be requested from bitnet PMCO at LNCC. The C-compilers are the best ones developed for microcomputers. This code can be easily translated to older languages, such as FORTRAN. We hope that this code can be soon implemented on a vectorized computer.

2. UPDATING THE SPINS

For the sake of simplicity, let us consider a cubic lattice. In this geometry each spin has six neighbors. We consider the following Hamiltonian for the Ising model:

$$H = 2J \sum_{\langle ij \rangle} \sigma_i \otimes \sigma_j \tag{1}$$

where J is the coupling strength, the summation is taken over the nearest neighbor pairs, σ_i can assume the values 0 or 1 and represents the spin in the ith site, and $\otimes$ means the XOR (eXclusive OR) logical operation. In this representation the local energy can assume the values 0, 1, 2,..., 6, taking $2J$ as energy unit. In the Monte Carlo simulation of the Ising dynamics the slowest step is the Metropolis[4] algorithm that introduces the temperature effect. For each spin with local energies 0, 1, and 2 we must choose a random number (between 0 and 1) and compare it with the corresponding Boltzmann factor

$$\exp(-\Delta E/kT) \tag{2}$$

where T is the simulated temperature and ΔE is the change in energy due to spin flip. The spin that is being tested will be flipped if the random number is less than the Boltzmann factor. This recipe guarantees that this factor is the probability of flipping. For the other local energies the Boltzmann factor will be larger than 1, and thus the spins will be always flipped.

In a previous paper,[3] we presented a new technique for storing[5] and updating[3] of spins for a square lattice. Here, we will extend this procedure for the cubic lattice. In this code each bit holds one spin and the local energy value is calculated simultaneously for 16 spins (in a 16-bit word processor), using only bitwise operations, without need of storing it. We divide each plane into four sublattices as defined in ref. 5. We will describe, step by step, the procedure of determination of local energies. We used the following variables: s holds the spins which are being tested, i_j are its

neighbors ($j = 1,..., 6$), and the symbols $\wedge$, $\otimes$, $\vee$, and $\neg$ mean the AND, XOR, OR, and NOT bitwise operations. The variable *imp* defined as

$$imp = i_1 \otimes i_2 \otimes i_3 \otimes i_4 \otimes i_5 \otimes i_6 \tag{3}$$

will hold 1-bits at sites with an odd number of 1-bit neighbors. We also define

$$e3 = [(i_1 \otimes i_2) \wedge (i_3 \otimes i_4) \wedge (i_5 \otimes i_6)] \vee [(i_1 \otimes i_4) \wedge (i_2 \otimes i_5) \wedge (i_3 \otimes i_6)] \vee [(i_2 \otimes i_3) \wedge (i_4 \otimes i_5) \wedge (i_6 \otimes i_1)] \tag{4}$$

which will hold 1-bits at sites with local energy 3 (it is independent of the value of the central spin being tested). Other simple expressions can be constructed for 0 and 6 local energy cases, using the following definitions:

$$up0 = \neg i_1 \wedge \neg i_2 \wedge \neg i_3 \wedge \neg i_4 \wedge \neg i_5 \wedge \neg i_6 \tag{5}$$

and

$$up6 = i_1 \wedge i_2 \wedge i_3 \wedge i_4 \wedge i_5 \wedge i_6 \tag{6}$$

where *up*0 holds 1-bits at sites with all neighbors assuming 0-value and analogously for *up*6 all neighbors assuming 1-value. The variables *e*0 and *e*6 will be defined as

$$e0 = (up6 \wedge s) \vee (up0 \wedge \neg s) \tag{7}$$

$$e6 = (up0 \wedge s) \vee (up6 \wedge \neg s) \tag{8}$$

Other auxiliary variables can be constructed:

$$up4 = \{[(i_1 \vee i_2) \wedge (i_3 \vee i_4)] \vee [(i_1 \vee i_2) \wedge (i_5 \vee i_6)] \vee [(i_3 \vee i_4) \wedge (i_5 \vee i_6)]\} \wedge \{(i_1 \wedge i_2) \vee (i_3 \wedge i_4) \vee (i_5 \wedge i_6)\} \wedge \neg imp \wedge \neg up6 \tag{9}$$

$$up5 = imp \wedge \{(i_1 \wedge i_2 \wedge i_3 \wedge i_4) \vee (i_5 \wedge i_6 \wedge i_1 \wedge i_2) \vee (i_3 \wedge i_4 \wedge i_5 \wedge i_6)\} \tag{10}$$

$$up2 = \neg imp \wedge \neg up4 \wedge \neg e0 \wedge \neg e6 \tag{11}$$

$$up1 = imp \wedge \neg e3 \wedge \neg up5 \tag{12}$$

Starting from (9)–(12) we can define, finally,

$$e5 = (up1 \wedge s) \vee (up5 \wedge \neg s) \tag{13}$$

$$e1 = (up5 \wedge s) \vee (up1 \wedge \neg s) \tag{14}$$

$$e4 = (up2 \wedge s) \vee (up4 \wedge \neg s) \tag{15}$$

$$e2 = (up4 \wedge s) \vee (up2 \wedge \neg s) \tag{16}$$

Although we have no need of evaluating the spins with local energies 3–6, we did it, and thus the performance cited here that we describe can be still improved. We can define the variable t as

$$t = \neg(e0 \vee e1 \vee e2) \tag{17}$$

which will hold 1-bit at sites to be flipped, *a priori*, i.e., those with local energies 3–6. It remains to choose the spins with local energies 0, 1, and 2 that must be flipped following the Metropolis algorithm, according to random numbers.

The modification introduced in this paper is that we used only one suitable random number for each word of each local energy ($e0$, $e1$, $e2$), instead of one random number for each spin as has been done before.[2,3] Due to this procedure, the larger the computer word size, the larger the time saving. The main idea is: if we succeed in generating suitable random numbers with random bits following predefined probabilities to assume the value 1 (we refer to this probability as *concentration* hereafter) equal to the Boltzmann factors for the simulated temperatures, we can determine all the spins with local energy 2, for example, which must be flipped, with only one AND operation between the word $e2$ and this random number. Moreover, this procedure avoids the use of comparisons, which are the slowest operations, and holds the full parallelization of our algorithm. According to the Hamiltonian (1), the Boltzmann factors will be

$$ex2 = \exp(-4J/kT) \tag{18}$$

$$ex1 = \exp(-8J/kT) = ex2^2 \tag{19}$$

$$ex0 = \exp(-12J/kT) = ex2^3 \tag{20}$$

With two random numbers generated with concentration $ex2$, we can construct another random number with concentration $ex1$ with only one AND operation between these numbers. The AND operation between two numbers with concentration x of bits in state 1 will generate another number with concentration x^2. With other three random numbers and two AND operations, we can construct a fourth random number according to the factor (20). Although this procedure seems to be quite simple, it is not. In the next section we describe how to construct those suitable random numbers.

3. BUILDING THE RANDOM NUMBER

One of the fastest random number generators is the multiplication by 65,539 used in ref. 3 with bitwise operations:

$$r = r + (r \ll 1) + (r \ll 16) \tag{21}$$

where $r \ll b$ means the shift of word r by b bits to the left and it is the same as $r \cdot 2^b$. The number r is a pseudorandom number, but its bits are not. The first bit (or bit 0) always must be 1, in order to avoid that one bit in 0 state propagating along all bits of the word r, after a few iterations. The second bit will be alternately in 0 and 1 states. Due to this restriction, this random number generator cannot be used for the procedure above because it leads to undesired correlations. We checked the repeating period of each bit in long integer (32-bit integer) r, i.e., we measured the cycle of each bit. These periods have a simple formation rule for bits 3, 4,..., 31: the period of the bit n is 2^{n-1}; thus, the more to the left is the bit position in the word, the larger is the repeating period. Due to limitations of our microcomputer, we stored the lattice in short integers (16-bit integers). We used the last eight bits of two numbers generated by (21) in order to construct one 16-bit number rs, using the following procedure:

$$\begin{aligned} r &= r + (r \ll 1) + (r \ll 16) \\ r1 &= r \gg 24 \\ r &= r + (r \ll 1) + (r \ll 16) \\ rs &= r1 \vee ((r \gg 24) \ll 8) \end{aligned} \tag{22a}$$

where r and $r1$ are long and rs is a short integer, in our case, and $r \gg b$ means the shift of word r by b bits to the right. To accept this new random number with long cycle, it remains to test the correlation between each pair of bits. We did it and found that all off-diagonal elements of the correlation matrix are statistically null, i.e., less than $1/\sqrt{N}$, where N is the number of iterations. This random number generator was used here in order to realize a more realistic comparison between this code and the other one,[3] but other random numbers generators (see, for instance, ref. 6) can be tested, improving the performance. We test another, faster procedure to generate suitable 16-bit random words from two independent 32-bit seeds r and $r1$, and a third auxiliary one $r2$:

$$\begin{aligned} r &= [r + (r \ll 1) + (r \ll 16)] \otimes r2 \\ r1 &= r1 + (r1 \ll 1) + (r1 \ll 16) \\ r2 &= (r1 \otimes r) \gg 15 \\ rs &= r2 \end{aligned} \tag{22b}$$

For the sake of comparison, the procedure (22b) generates 10^6 random numbers in 22 sec, while (22a) takes 41 sec in our 80286-based microcomputer running at 12 MHz. Moreover, the (22b) cycles are longer than

those of (22a). At this moment, we have created a random number with uncorrelated bits, each one having equal probability of assuming the 0 or 1 state, i.e., concentration equal to 0.5. We can just simulate the temperature in which the Boltzmann factor $ex2$ is 0.5. It is obvious that this is not interesting, because we want to simulate any temperature.

The trick used to change the concentration of 1-bits in a word is also very fast because we used only binary operations. Let us consider, *a priori*, two numbers with concentrations x and y. One AND operation between these numbers will generate another with concentration xy, because the AND operation gives 1 as a result if and only if both operands are 1. The OR operation will give as a result a number with concentration $1-(1-x)(1-y)$. The OR operation gives a result 0 if and only if both operands are 0. The XOR operation will give as a result a number with concentration $x(1-y)+y(1-x)$. We have tested the validity of these rules for more than one million iterations and they were statistically confirmed. Using these properties, we can construct several concentrations and temperatures. Let us consider, for example, the Boltzmann factor $ex2=0.25$ or $J/kT \approx 0.35$. For construction of a short integer random number $re2$ we used two random numbers, $rs1$ and $rs2$ generated by (22a) or (22b):

$$re2 = rs1 \wedge rs2 \tag{23}$$

After this step we can do

$$t = t \vee (e2 \wedge re2) \tag{24}$$

With this operation we introduce 1-bits in the word t at positions corresponding to local-energy-2 sites, with probability 0.25, as required by the Metropolis algorithm. We used four random numbers generated by (21) to construct $re2$. To construct $re1$ with concentration $ex1 = ex2^2$ we use eight numbers. For $re0$ with concentration $ex0 = ex2^3$, 12 numbers are necessary. For the two-dimensional case the gain is increased because we need not generate these last 12 numbers. For the sake of time saving, we can generate these 12 numbers at the beginning, using 4 of them to create $re2$, the other 8 for $re1$, and all 12 for $re0$. There is no danger of introducing correlations, because $re0$, $re1$, and $re2$ will influence distinct bits in the word t when composed with $e0$, $e1$, and $e2$, respectively; see Eq. (26) below. We tried also to construct $re0$, $re1$, and $re2$ by combining only six random numbers rs generated by (22a), but we abandoned this idea due to the correlations introduced in this process. In this temperature, we increased the performance of ref. 3 by a factor 3. Had this code been implemented in 64-bit processor, this factor would be larger than 12. In combination with the factor 8 obtained in the same reference, we can obtain a net gain by a

factor 10^2, at least, compared with other multispin codes.[2] We can simulate other temperatures close to $J/kT = 0.35$ if we modify the number $rs1$ with the following procedure:

$$rs1 = rs1 \vee (1 \ll (rs2 \wedge 15)) \tag{25}$$

Using Eq. (23), we find that $re2$ will have concentration equal to 0.266 or $J/kT \approx 0.33$. The inner expression is necessary in order to add 1-bit in a random position. When the Metropolis algorithm step finishes, we have

$$t = t \vee (e2 \wedge re2) \vee (e1 \wedge re1) \vee (e0 \wedge re0) \tag{26}$$

and t will hold 1-bits at all sites that must be spin flipped. The final updating is done with one XOR operation between s (the word being tested) and t.

4. THE RESULTS

The thermodynamic quantities of the 3-dimensional ferromagnetic Ising model shown here reproduce the original results. They were extracted in order to test the fully-parallel algorithm. We simulated 32^3 spins. The rule employed to simulate close to the critical temperature for spins with local energy 2 was

$$re2 = (rs1 \wedge rs2) \vee (rs3 \wedge rs4) \tag{27}$$

where rsi are numbers generated by procedure (22a), i.e., concentration 0.5. The corresponding Boltzmann factor is $ex2 = 0.4375$ or $J/kT = 0.20667$.

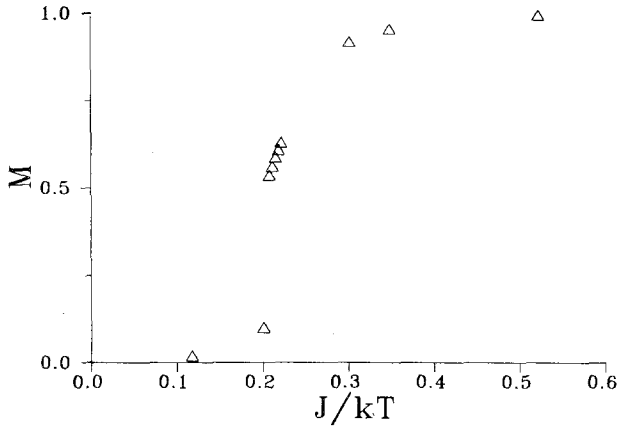

Fig. 1. Results for the spontaneous magnetization $\langle |m| \rangle$.

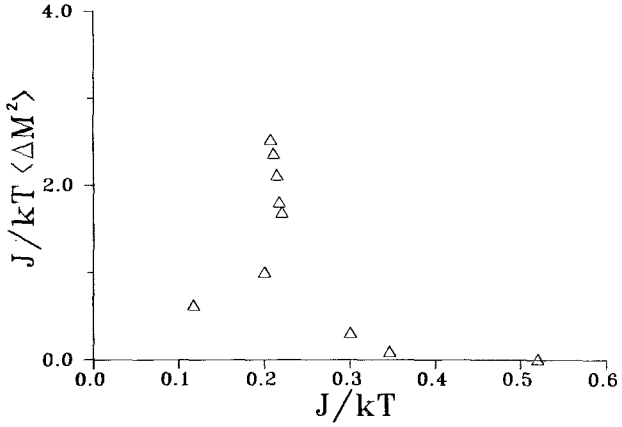

Fig. 2. Results for the susceptibility.

The phase transition point is estimated numerically to be $J/kT = 0.22165$.[7] Closer temperatures were obtained by modifying the *rsi* numbers through

$$rs1 = rs1 \wedge \neg(1 \ll (rs2 \wedge 15)) \tag{28}$$

The results for spontaneous magnetization and susceptibility are shown in Figs. 1 and 2, extracted from different simulations for each temperature. We performed 100,000 Monte Carlo steps after 1000 steps without measurement for thermalization, for each temperature.

For performance comparison, in our best case, including the energy and magnetization calculus at every step, the updating rate was 6×10^4 spins/sec, in an 80286-processor-based microcomputer running at 12 MHz.

5. SUMMARY

In this paper we present a new strategy for Monte Carlo simulation without generating one random number for each spin in the Metropolis algorithm. Instead, we generate random numbers with uncorrelated random bits with different concentrations in order to simulate different temperatures. We increase this performance by a factor of 20 relative to original multispin codes, but this factor could be considerably increased if this procedure is implemented in 64-bit processors. Another important feature of this procedure is that it can be applied also for stochastic cellular automata.[8]

ACKNOWLEDGMENTS

We are grateful to Prof. J. A. e Souza for reading the manuscript and offering suggestions for its improvement. This work was partially supported by Brazilian agencies FINEP, FAPERJ, CAPES, and CNPq.

REFERENCES

1. K. Binder, ed., *Monte Carlo Methods in Statistical Physics*, 2nd ed. (Springer-Verlag, Berlin, 1986); *Applications of Monte Carlo Methods in Statistical Physics*, 2nd ed. (Springer-Verlag, Berlin, 1987).
2. R. Zorn, H. J. Herrmann, and C. Rebbi, *Comp. Phys. Comm.* **23**:337 (1981); L. Jacobs and C. Rebbi, *J. Comp. Phys.* **41**:203 (1981); C. Kalle and V. Winkelmann, *J. Stat. Phys.* **28**:639 (1982).
3. P. M. C. de Oliveira and T. J. P. Penna, *Rev. Bras. Fís.* **18**:502 (1988).
4. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *J. Chem. Phys.* **21**:1087 (1953).
5. H. J. Herrmann, *J. Stat. Phys.* **45**:145 (1986).
6. S. Kirkpatrick and E. P. Stoll, *J. Comp. Phys.* **40**:517 (1980).
7. G. Pawley, D. Wallace, R. Swendsen, and K. Wilson, *Phys. Rev.* **29**:4030 (1984).
8. W. Kinzel, *Z. Phys. B* **58**:229 (1985).

Communicated by D. Stauffer