Journal of Statistical Physics, Vol. 44, Nos. 5/6, 1986

A Fast Algorithm for the Cyber 205 to Simulate the 3D Ising Model

Gyan Bhanot,[1] Dennis Duke,[2] and Román Salvador[2]

Received February 17, 1986; final April 22, 1986

We describe a computer program that performs the Metropolis algorithm for the 3D Ising model at a peak speed of 98 million spin updates per second on a 2-pipe CDC Cyber 205. This speed is achieved using the special vector capabilities of the Cyber 205 and multispin coding techniques.

KEY WORDS: Ising model, Monte Carlo method, multispin coding, vector computer.

INTRODUCTION

This paper describes a new way to implement the Metropolis et al.[1] algorithm for Monte Carlo simulations of statistical systems with a few discrete degrees of freedom per variable. We describe this algorithm as implemented on a CYBER 205 and for the 3D Ising model. However, as should become obvious, our method can also be used on other computers and for other systems.

Each spin update in the Metropolis algorithm involves comparing the exponential of the change in the action with a random number. Since generating a random number takes about 20 ns on the Cyber 205, the best that a normal implementation of the algorithm can achieve is 50 million updates per second. We describe a method that avoids the bottleneck of making a floating point comparison with a random number. (Another method to avoid this bottleneck was proposed in Ref. 2.) Our method, in brief, works as follows:

[1] Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida 32306.

[2] Supercomputer Computations Research Institute and Department of Physics, Florida State University, Tallahassee, Florida 32306.

0022-4715/86/0900-0985$05.00/0

1. In our simulation, we use a variation of the multispin coding method[3] in which a single word contains one spin from n different systems. In our case, n is the word length which for the CYBER 205, $n = 64$. Hence we are simultaneously working on 64 different lattices.
2. We code the necessary information about each random number into two bits.
3. Each bit pair is used exactly once for each of the 64 lattices.
4. We use only logical commands for the update. Thus, 64 spins are updated together.

This method gives an algorithm speed of 98 megaflips in a 2-pipe Cyber 205. The fastest implementation for the 3D Ising model we are aware of does 218 megaflips on a DAP computer. However, this implementation is for the special case of a $128 \times 128 \times 144$ lattice.[4]

Because of inherent Cyber 205 limitations on vector length, our current algorithm can only run on lattices of size up to 50^3. To modify the code for larger lattices is easy. It involves slicing the lattice up into two dimensional planes and making the vector length equal half the number of spins in a plane. The algorithm currently achieves a speed of over 90 million spin updates per second for lattices of size greater than 14^3 on a 2-pipe Cyber 205. Our peak speed of 98 megaflips is achieved on a 20^3 lattice. We have used this program recently to reanalyze finite-size scaling for the 3D Ising model and compute the critical exponent γ to a few parts in a thousand.[5]

THE ALGORITHM

The Ising model is defined as a collection of spins on the sites of the lattice. The action (energy) of the system is given by

$$S(\{s\}) = -\sum_{i,\hat{\mu}} s_i s_{i+\hat{\mu}} \qquad (s = \pm 1) \tag{1a}$$

The spins s_i can have two possible values, ± 1. The aim is to generate configurations of spins with joint probability distribution

$$Z = \sum_{\{s\}} e^{-\beta S(\{s\})} \tag{1b}$$

It is more convenient to store one spin per bit by using, in place of the variables s, the variables $\sigma = (1 - s)/2$, which take values 0 or 1. As a function of these variables, the action is

$$S(\{\sigma\}) = -\sum_{i,\hat{\mu}} (1 - 2x_{i,\hat{\mu}}) \qquad (x_{i,\hat{\mu}} = 0, 1) \tag{2a}$$

with

$$x_{i,\hat{\mu}} = XOR(\sigma_i, \sigma_{i+\hat{\mu}}) \qquad (\sigma = 0, 1) \tag{2b}$$

The change in the action on flipping the spin at site i is

$$\Delta S = S_{\text{final}} - S_{\text{initial}} = 12 - 4 \sum_{\pm\hat{\mu}} x_{i,\hat{\mu}} \tag{3}$$

where the sum goes over all neighbors of s_i. The Metropolis algorithm consists of the following

1. If ΔS is nonpositive the spin flip should be accepted.
2. Otherwise, it should be accepted with probability $e^{-\beta\Delta S}$.

We implement this by logical commands as follows:

Define three bit variables $B3V$, $B2V$, $B1V$ initialized to 0, 0, 1, respectively. Add the six values of $x_{i,\mu}$ to the bits $B3V$, $B2V$, $B1V$, thinking of them as the third, second, and first bits of an integer. Note that this can be done with logical instructions since each $x_{i,\mu}$ is either 0 or 1. The seven final values of the set $B3V$, $B2V$, $B1V$ are shown in Table I. Notice that when the spin flip is to be accepted $B3V = 1$. Using $B3V$ as the acceptance criterion implements the first part of the Metropolis algorithm. If $B3V$ is still zero after the additions, then we are dealing with the cases where $\sum_{i,\hat{\mu}} x_{i,\hat{\mu}}$ is 0, 1, or 2 and in this case, the spin should flip with probability $e^{-\beta\Delta S}$. To do this, we define two additional bits $D2V$, $D1V$, called *demons* (*demon* variables similar to ours were first used by Creutz in the context of the microcanonical ensemble.[6]) which take the values (0, 1), (1, 0), and (1, 1) with probabilities p_{01}, p_{10}, and p_{11} given by

$$\begin{aligned} p_{01} &= e^{-4\beta} - e^{-8\beta} \\ p_{10} &= e^{-8\beta} - e^{-12\beta} \\ p_{11} &= e^{-12\beta} \end{aligned} \tag{4}$$

Table I. Logical Implementation of the Metropolis Algorithm

ΔS	$\sum_{\pm\hat{\mu}} x_{i,\hat{\mu}}$	$B3$	$B2$	$B1$
		0	0	0
12	0	0	0	1
8	1	0	1	0
4	2	0	1	1
0	3	1	0	0
−4	4	1	0	1
−4	5	1	1	0
−12	6	1	1	1

The integer formed with $D1V$ and $D2V$ as the first and second bit is now added to the integer formed by $B3V$, $B2V$, and $B1V$. Once again, if $B3V$ is unity, the flip is accepted. Notice that this will happen with probability $e^{-4\beta}$, $e^{-8\beta}$, and $e^{-12\beta}$ when ΔS is 4, 8, or 12, respectively. This implements the second part of the Metropolis algorithm.

Now we discuss the implementation of this algorithm on the Cyber 205. (The code is shown in Fig. 2.) As mentioned earlier, our program updates 64 different lattices simultaneously. Spins occupying the same coordinates on each of the 64 lattices are stored in the same word. The spins are labeled even/odd in a checkerboard pattern and updated in two steps: first all even spins are updated and then all the odd ones. The vector length of the pipelined arrays is half the total number of spins on a lattice. The variables $B1V$, $B2V$, and $B3V$, as well as the demons $D1V$ and $D2V$, are also arrays of the same length. Clearly, the algorithm can be pipelined in the Cyber 205 if one can arrange the arrays $D1V$ and $D2V$ such that their entries (in pairs), take on the values (0, 1), (1, 0), and (1, 1) with probabilities given by eq. 4.

The crucial part of the algorithm is to get such a distribution of demon bits. This is done in subroutine DEMETRO. First, 64 vectors of random numbers are generated one after the other using a shift register random number generator.[7] Each of these vectors is used to define one string of demon pairs ($D1V$, $D2V$) distributed according to eq. 4. The final vectors for $D1V$ and $D2V$ are obtained by merging together, in the 64-bit positions of the words, the 64 strings thus obtained. After each half sweep (update of the odd or even sites of all the 64 lattices), the demon pairs ($D1V$, $D2V$) are subjected to a random GATHERand a random shift. The random shifts are arranged so that each string of demons is used in each of the lattices exactly once but in a different order for each lattice. After 64 half sweeps, the demons are reinitialized. This means that we use one random number once for each of the 64 lattices.

In *any* Metropolis run, one uses a finite sequence of random numbers to compare to $e^{-\beta\Delta S}$. In such a finite run, the set of random numbers used is not uniformly distributed in the interval (0, 1) and this means that the β value is not the desired one but rather, some other value β_{eff}. For our simulation, these effective values of β are given by

$$
\begin{aligned}
n_1/n &= e^{-4\beta_{\text{eff},1}} \\
n_2/n &= e^{-8\beta_{\text{eff},2}} \\
n_3/n &= e^{-12\beta_{\text{eff},3}}
\end{aligned}
\tag{5}
$$

where n is the total number of random numbers used and n_1, n_2, and n_3 are the number of times these random numbers are smaller than $e^{-4\beta}$, $e^{-8\beta}$, and $e^{-12\beta}$, respectively.

This expected shift in β can be easily understood theoretically. Given n random numbers in the interval (0, 1), the probability for n_i of them being in an interval of length $p_k = e^{-4k\beta}$ is given by the binomial distribution

$$P_k(n_i) = \binom{n}{n_i} p_k^{n_i}(1-p_k)^{n-n_i} \tag{6}$$

If we define $x \equiv n_i/n$, then this probability distribution has mean $\bar{x} = p_k$ and standard deviation $\sigma_k = \sqrt{p_k(1-p_k)/n}$. Note that the error in β is statistical and proportional to $1/\sqrt{n}$ and disappears in the limit of an infinite simulation. It is easy to correct for this shift in β. This is done by correcting each order parameter O according to

$$O(\beta) = O(\beta_{\text{eff}}) + \left.\frac{\partial O}{\partial \beta}\right|_{\beta_{\text{eff}}} (\beta - \beta_{\text{eff}}) \tag{7}$$

The reason we need to worry about this error (which is actually present even in standard Metropolis updating but is less relevant there) is the following: In our simulation, each of the 64 lattices uses the same set of random numbers. Therefore, each of the lattices has the same β_{eff} on average and the effects of the shift in β add coherently. In a standard Metropolis run on 64 lattices using different random number streams on each, the effective length of the random number stream is 64 times our length. The shift in β then is smaller by a factor of 8.

We obtained one single value of the effective β using the average demon action E_{demon} via the formula

$$\frac{4e^{-4\beta_{\text{eff},1}} + 8e^{-8\beta_{\text{eff},2}} + 12e^{-12\beta_{\text{eff},3}}}{1 + e^{-4\beta_{\text{eff},1}} + e^{-8\beta_{\text{eff},2}} + e^{-12\beta_{\text{eff},3}}} = \langle E_{\text{demon}} \rangle$$
$$= \frac{4e^{-4\beta_{\text{eff}}} + 8e^{-4\beta_{\text{eff}}} + 12z^{-12\beta_{\text{eff}}}}{1 + e^{-4\beta_{\text{eff}}} + e^{-8\beta_{\text{eff}}} + e^{-12\beta_{\text{eff}}}} \tag{8}$$

We have tested these ideas on the two-dimensional Ising model where exact answers are known. This is shown in Fig. 1. The 40 points in the graphs correspond to different runs, each of them coming from 20,000 sweeps in a set of 64 lattices of size 20^2 at $\beta = 0.44$ and averaging over the last 10,000 sweeps. In Fig. 1 we show average values of $\langle S^2 \rangle$ before and after the correction. The errors were obtained by using the 64 average values from the 64 independent lattices. The uncorrected data values (Fig. 1a), as expected, are scattered around a shifted β with a Gaussian

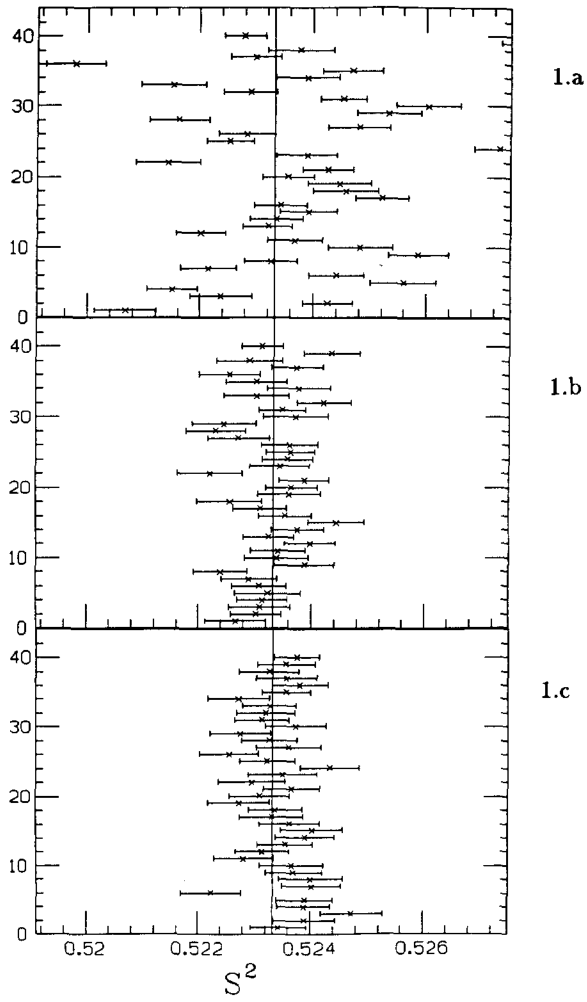

Fig. 1. 40 data points for $\langle S^2 \rangle$ on 20^2 lattices at $\beta = .44$. Each point corresponds to averaging for 10,000 sweeps on the set of 64 lattices after thermalizing for 10,000 sweeps. The error bars were obtained from fluctuations between the 64 data sets from the different lattices. (a) Shows the data without correction; (b) Shows the data after correcting for the shift in β and using eq. 10. (c) Shows the data corresponding to the same experiment but using a different random number stream for all 64 lattices. (a conventional, but slower, standard Metropolis simulation). The vertical line that crosses the graphs corresponds to the exact value. Note that the error bars are all of almost the same size.

```
c 3-d ising model program
c
c  this program performs the metropolis algorithm on cubic lattices
c  of size up to 50↑3 , it uses periodic boundary conditions.
c  the lattice size must be even.

      program ising ( input , output , tape6 , tape7 )

c define  parameters
c                l     =  lattice size  (must also be def.in sub. metro)
c                init  =  initialization parameter (0:ordered start,
c                         1:disordered start,6:reads lattice from tape6)
c                istor =  (1,0):(does,does not) write last conf. to tape7
c                iseed =  seed for the r.n.g. (if =0 ==> default seed.)
c                nb    =  # of beta (= 1/Temperature) values processed
c                beta  =  initial beta
c                dibe  =  succesive decrements in beta.
c                nsba  =  # of sweeps to thermalize
c                nsw   =  # of sweeps during averaging
c                nav   =  number of sweeps between measurements.
c                np,nq =  prim. trinomial for the shift register random
c                number generator (must also be defined in the sub. metro)
c                                                  ( nh > np > nq )
      parameter (l     =  20
     +          ,init  =  1
     +          ,istor=   0
     +          ,iseed=   7893789324783
     +          ,nb    =  1
     +          ,beta  =  .22165
     +          ,dibe  =  .0
     +          ,nsba  =  25000
     +          ,nsw   =  25000
     +          ,nav   =  100
     +          ,np    =  2281
     +          ,nq    =   715
     + , ns=l**3, lm=l-1 , nh=ns/2 , nhp=nh+1 , nt=nh+np)

c  VARIABLES IN THE PROGRAM
c  bx,by,bz: Indexing vectors to implement the periodic boundary conditions
c  xrand: vector for the shift register random number generator
c  xv,xv1,xv2,dn1v,dn2v: help vectors (to store intermediate results)
c  spin: vector of ising spin variables for the 64 lattices.
c  b1v,b2v,b3v: vectors used in the updating
c  d1v,d2v: demon vectors
c  nr,nrp,num,nr1,nr2: vectors used in demon permutations.
c  ran: vector for congruential random number generator.
c  tf,ti,tm,ta,timet,t64,tt: scalars  for timing.
c  i64,n64: vectors used to permute demons d1v,d2v.
c  am,s,r,abe,ntb: variables used in measurements.
c  cv,cv1,cv2: bit vectors used to update the demons

      integer bx,by,bz,xrand,xv,x1v,x2v,spin,b1v,b2v,b3v
     c  ,d1v,d2v,dn1v,dn2v
     c              ,dx,dy,dz,x,x1,x2,dxra,dsp1,dsp2,b1,b2,b3,d1,d2
     c  ,dn1,dn2,dnr,dnrp,dnr1,dnr2

      descriptor dx,dy,dz,x,x1,x2,dxra,dsp1,dsp2,b1,b2,b3,d1,d2
     c  ,dn1,dn2,dnr,dnrp,dnr1,dnr2,ia,ib,ic,isd

      bit cv,cv1,cv2

      common/blk/ spin(ns),bx(nh),by(nh),bz(nh),xv(nh),x1v(nh),x2v(nh)
     c  ,b1v(nh),b2v(nh),b3v(nh),d1v(nh),d2v(nh),dn1v(nh),dn2v(nh)
     c  ,nrp(nh),num(nh),nr2(nh),nr1(nh),nr(nh),ran(nh),tf,ti,tm,ta
     c  ,timet,t64,i64(64),n64(64),am(64),s(64),r(16,64),abe(3)
     c  ,ntb,xrand(nt),cv(2*nh),cv1(2*nh),cv2(2*nh)

      data tf,ti,tm,ta,timet,tt,t64,(abe(i),i=1,3),ntb /10*.0,0/

c ASSIGN DESCRIPTORS TO ARRAYS:

      data dsp1,dsp2 / spin(1;nh),spin(nhp;nh) /
      data dx,dy,dz / bx(1;nh),by(1;nh),bz(1;nh) /
      data b1,b2,b3 / b1v(1;nh),b2v(1;nh),b3v(1;nh) /
      data d1,d2 / d1v(1;nh),d2v(1;nh) /
      data dnr,dnrp,dnr1,dnr2 / nr(1;nh),nrp(1;nh),nr1(1;nh),nr2(1;nh)/
      data dn1,dn2,x,x1,x2 / dn1v(1;nh),dn2v(1;nh),xv(1;nh),x1v(1;nh)
     +                     , x2v(1;nh) /
```

Fig. 2. Listing of the code.

```
      assign ia   , xrand(    1 ; nh )
      assign ib   , xrand( nq+1 ; nh )
      assign ic   , xrand( nh+1 ; np )
      assign isd  , xrand(    1 ; np )
      assign dxra , xrand( np+1 ; nh )

      tt1=second()

      do 74 i=1,nh
 74   num(i)=i

c initialization of: random number seeds, the lattice, and the indexing
c vectors

      if(iseed.ne.0)call ranset(iseed)
      call raninit(xrand,np)
      call initlat(init,betao,nso,dsp1,dsp2,ran,x1,spin,nh,ns,l)
      call index(bx,by,bz,l)

c begin simulation

      do 1234 iib = 1,nb

      b = beta - dibe*iib
      if ( b.lt.0. ) b=-b

      call metro( nsba , 0    , b ,dsp1,dsp2,dx,dy,dz,x,x1,x2,b1,b2,b3
     +          ,d1,d2,dn1,dn2,dnrp,dnr1,dnr2,dnr,dxra,ia,ib,ic,isd)
      call metro( nsw  , nav  , b ,dsp1,dsp2,dx,dy,dz,x,x1,x2,b1,b2,b3
     +          ,d1,d2,dn1,dn2,dnrp,dnr1,dnr2,dnr,dxra,ia,ib,ic,isd)

c write the data to the output file

      if(nsw.ne.0)call data(b,nsw,nsba,nav,init,betao,nso,iib,ntb
     +                      ,abe,ns,r)

 1234 continue

c simulation ends. Stores the last lattice

      if(istor.eq.1)then
      nsn=nsba+nsw
      if(dibe.eq..0)then
      nsn=nb*nsn
      if(betao.eq.beta)nsn=nso+nsn
      endif
      write(7,*)l,beta,nsn
      do 77 i=1,ns
 77   write(7,177) spin(i)
 177  format(z16)
      endif

c timing information:

      tt=second()-tt1
      print*,'time spent in the main parts of the program :
      print*
      print565,' program  ising =   ',tt
      print565,' in updating    =   ',timet
      print565,' in measuring   =   ',ta
      print565,' sub. demetro   =   ',tf
      print565,' sub. initnr    =   ',ti
      print565,' sub. init64    =   ',t64
      print565,' sub. meas      =   ',tm
      print*
      print566,(64e-6*ns*nb*(nsba+nsw))/timet
 565  format(1x,a20,f10.2,'  seconds')
 566  format(' running at ',f5.1,' megaflips')
      print*
      stop
      end

      subroutine index(bx,by,bz,l)
      integer  bx(*) ,by(*), bz(*)

c defines the index vectors to implement periodic boundary conditions.

      lm = l - 1
```

Fig. 2 (*continued*)

```
      do 1 iz = 0,lm
      izm = mod( iz+lm , l )
      do 1 iy = 0,lm
      iym = mod( iy+lm , l )
      do 1 ix = 0,lm
      ixn = mod (ix+1  , l )
      if ( mod(iy+iz,2) .eq. 0 )ixn = mod( ix+lm , l )

      if(mod(ix+iy+iz,2).eq.0)then
      n     = (iz *l + iy )*(l/2) + (ix -mod(ix ,2) )/2 + 1
      bx(n) = (iz *l + iy )*(l/2) + (ixn-mod(ixn,2) )/2
      by(n) = (iz *l + iym)*(l/2) + (ix -mod(ix ,2) )/2
      bz(n) = (izm*l + iy )*(l/2) + (ix -mod(ix ,2) )/2
      endif

 1    continue
      return
      end

      subroutine initnr(nr,ti,nrp,num,ran,nh)

c creates a vector (nr) containing a random permutation of
c the integers 0 to nh-1

      dimension nr(*) ,nrp(*) ,num(*) ,ran(*)

      ti1=second()

      call vranf(ran,nh)
      nrp(1;nh)=num(1;nh)*ran(1;nh)+1
      call q8vrevv(x'00',,nrp(1;nh),,,,nr(1;nh))
      nrp(1;nh)=num(1;nh)-1

      do 1 i = 1,nh
      n=nr(i)
      nr(i) = nrp(n)
      nv=nh-i-n+1
      nrp(n;nh-i-n+1)=nrp(n+1;nh-i-n+1)
 1    continue

      ti=ti+second()-ti1
      return
      end

      subroutine metro( nsw  , nav , b ,dsp1,dsp2,dx,dy,dz,x,x1,x2,b1
     +        ,b2,b3,d1,d2,dn1,dn2,dnrp,dnr1,dnr2,dnr,dxra,ia,ib,ic,isd)
c metro( nsw , nav , b , ... ) does nsw sweeps measuring after each nav
c sweeps at beta = b. if nav = 0 no measurements are done. The other
c arguments are passive. They are descriptors needed by this routine.
      parameter (l=20,ns=l**3,lm=l-1,nh=ns/2,nhp=nh+1)
      parameter(np=2281,nq=715,nt=nh+np)
      integer bx,by,bz,xrand,xv,x1v,x2v,spin,b1v,b2v,b3v
     c  ,d1v,d2v,dn1v,dn2v
     c            ,dx,dy,dz,x,x1,x2,dxra,dsp1,dsp2,b1,b2,b3,d1,d2
     c  ,dn1,dn2,dnr,dnrp,dnr1,dnr2
      descriptor dx,dy,dz,x,x1,x2,dxra,dsp1,dsp2,b1,b2,b3,d1,d2
     c  ,dn1,dn2,dnr,dnrp,dnr1,dnr2,ia,ib,ic,isd
      bit cv,cv1,cv2
      common/blk/ spin(ns),bx(nh),by(nh),bz(nh),xv(nh),x1v(nh),x2v(nh)
     c  ,b1v(nh),b2v(nh),b3v(nh),d1v(nh),d2v(nh),dn1v(nh),dn2v(nh)
     c  ,nrp(nh),num(nh),nr2(nh),nr1(nh),nr(nh),ran(nh),tf,ti,tm,ta
     c  ,timet,t64,i64(64),n64(64),am(64),s(64),r(16,64),abe(3),ntb
     c  ,xrand(nt),cv(2*nh),cv1(2*nh),cv2(2*nh)

c set counters

      if( nav.ne.0 )then
      naver      =  0
      r(1,1;1024)=.0
      endif
      is=64
      isnr=63

c does nsw sweeps

      do 99999 niter = 1,nsw
      t2 = second()
```

Fig. 2 (*continued*)

```
c creates vectors of demons and random permutations

      if( is.eq.64 )then
      is=0
      call demetro(b,nav,d1,d2,ia,ib,ic,isd,dxra,xrand,tf,ntb
     +                 ,d1v,d2v,abe,nh,np+1,cv,cv1,cv2)
      t641=second()
      call init64( i64 , n64 )
      t64=t64+second()-t641
      xv(1)=nr2(nh)
      xv(2;nh-1)=nr2(1;nh-1)
      dnr2=x
      call q8 vtovx (x' 00 ',,  dnr2 ,,  dnr1  ,,  x  )
      dnr1=x
      isnr=isnr+1
      if(isnr.eq.64)then
      call initnr( nr ,ti,nrp,num,ran,nh)
      call initnr( nr1,ti,nrp,num,ran,nh)
      call initnr( nr2,ti,nrp,num,ran,nh)
      isnr=0
      endif
      endif

c updates even sites
c adds the six products x to [b3,b2,b1] = [0,0,1]

      call q8 xor  v(x' 00 ',,  dsp1 ,,  dsp2 ,,  x   )
      call q8 vxtov (x' 00 ',,  dy   ,,  dsp2 ,,  x1  )
      call q8 xor  v(x' 00 ',,  x1   ,,  dsp1 ,,  x1  )
      call q8 and  v(x' 01 ',,  x1   ,,  x    ,,  b2  )
      call q8 xor  v(x' 00 ',,  x1   ,,  x    ,,  b1  )
      call q8 vxtov (x' 00 ',,  dz   ,,  dsp2 ,,  x   )
      call q8 xor  v(x' 00 ',,  x    ,,  dsp1 ,,  x   )
      call q8 vtovx (x' 00 ',,  dx   ,,  dsp2 ,,  x1  )
      call q8 xor  v(x' 00 ',,  x1   ,,  dsp1 ,,  x1  )
      call q8 and  v(x' 01 ',,  x1   ,,  x    ,,  dn2 )
      call q8 xor  v(x' 00 ',,  x1   ,,  x    ,,  dn1 )
      call q8 and  v(x' 01 ',,  dn2  ,,  b2   ,,  b3  )
      call q8 and  v(x' 01 ',,  dn1  ,,  b1   ,,  x   )
      call q8 xor  v(x' 00 ',,  dn2  ,,  b2   ,,  b2  )
      call q8 xor  v(x' 00 ',,  x    ,,  b2   ,,  b2  )
      call q8 xor  v(x' 00 ',,  dn1  ,,  b1   ,,  b1  )
      call q8 vtovx (x' 00 ',,  dy   ,,  dsp2 ,,  x   )
      call q8 xor  v(x' 00 ',,  x    ,,  dsp1 ,,  x   )
      call q8 vtovx (x' 00 ',,  dz   ,,  dsp2 ,,  x1  )
      call q8 xor  v(x' 00 ',,  x1   ,,  dsp1 ,,  x1  )
      call q8 or   v(x' 02 ',,  x1   ,,  x    ,,  dn2 )
      call q8 xorn v(x' 07 ',,  x1   ,,  x    ,,  dn1 )
      call q8 and  v(x' 01 ',,  dn2  ,,  b2   ,,  x1  )
      call q8 xor  v(x' 00 ',,  x1   ,,  b3   ,,  b3  )
      call q8 xor  v(x' 00 ',,  dn2  ,,  b2   ,,  b2  )
      call q8 and  v(x' 01 ',,  dn1  ,,  b1   ,,  x1  )
      call q8 and  v(x' 01 ',,  x1   ,,  b2   ,,  x2  )
      call q8 xor  v(x' 00 ',,  x2   ,,  b3   ,,  b3  )
      call q8 xor  v(x' 00 ',,  x1   ,,  b2   ,,  b2  )
      call q8 xor  v(x' 00 ',,  dn1  ,,  b1   ,,  b1  )

c keeps third demon ( now b3=1 if inc(s) < 0 )
c gets a new set of demons

      call q8 vtovx (x' 00 ',,  dnr1 ,,  dnr  ,,  dnrp )
      is  =  is + 1
      ishi=i64(is)
      call q8 vtovx (x' 00 ',,  dnrp ,,  d1   ,,  dn1 )
      call q8 shiftv(x' 08 ',,  dn1  ,,  ishi ,,  d1  )
      call q8 vtovx (x' 00 ',,  dnrp ,,  d2   ,,  dn2 )
      call q8 shiftv(x' 08 ',,  dn2  ,,  ishi ,,  d2  )

c now checks what happens when inc(s) > 0 (that is when b3=0 )
c adds demons [d2,d1] to [x=0,b2,b1]

      call q8 and  v(x' 01 ',,  d2   ,,  b2   ,,  x   )
      call q8 xor  v(x' 00 ',,  d2   ,,  b2   ,,  b2  )
      call q8 and  v(x' 01 ',,  d1   ,,  b1   ,,  x1  )
      call q8 and  v(x' 01 ',,  x1   ,,  b2   ,,  x2  )
      call q8 xor  v(x' 00 ',,  x2   ,,  x    ,,  x   )

c if now x = 1 also flips
```

Fig. 2 (*continued*)

```
c x is the acceptance criterion

      call q8 or   v(x' 02 ',, x    ,, b3   ,, x   )

c updates the spins

      call q8 xor  v(x' 00 ',, dsp1 ,, x    ,, dsp1)

c updates odd sites

      call q8 xor  v(x' 00 ',, dsp2 ,, dsp1 ,, x   )
      call q8 vxtov (x' 00 ',, dy   ,, dsp1 ,, x1  )
      call q8 xor  v(x' 00 ',, x1   ,, dsp2 ,, x1  )
      call q8 and  v(x' 01 ',, x1   ,, x    ,, b2  )
      call q8 xor  v(x' 00 ',, x1   ,, x    ,, b1  )
      call q8 vxtov (x' 00 ',, dz   ,, dsp1 ,, x   )
      call q8 xor  v(x' 00 ',, x    ,, dsp2 ,, x   )
      call q8 vxtov (x' 00 ',, dx   ,, dsp1 ,, x1  )
      call q8 xor  v(x' 00 ',, x1   ,, dsp2 ,, x1  )
      call q8 and  v(x' 01 ',, x1   ,, x    ,, dn2 )
      call q8 xor  v(x' 00 ',, x1   ,, x    ,, dn1 )
      call q8 and  v(x' 01 ',, dn2  ,, b2   ,, b3  )
      call q8 and  v(x' 01 ',, dn1  ,, b1   ,, x   )
      call q8 xor  v(x' 00 ',, dn2  ,, b2   ,, b2  )
      call q8 xor  v(x' 00 ',, x    ,, b2   ,, b2  )
      call q8 xor  v(x' 00 ',, dn1  ,, b1   ,, b1  )
      call q8 vtovx (x' 00 ',, dy   ,, dsp1 ,, x   )
      call q8 xor  v(x' 00 ',, x    ,, dsp2 ,, x   )
      call q8 vtovx (x' 00 ',, dz   ,, dsp1 ,, x1  )
      call q8 xor  v(x' 00 ',, x1   ,, dsp2 ,, x1  )
      call q8 or   v(x' 02 ',, x1   ,, x    ,, dn2 )
      call q8 xorn v(x' 07 ',, x1   ,, x    ,, dn1 )
      call q8 and  v(x' 01 ',, dn2  ,, b2   ,, x1  )
      call q8 xor  v(x' 00 ',, x1   ,, b3   ,, b3  )
      call q8 xor  v(x' 00 ',, dn2  ,, b2   ,, b2  )
      call q8 and  v(x' 01 ',, dn1  ,, b1   ,, x1  )
      call q8 and  v(x' 01 ',, x1   ,, b2   ,, x2  )
      call q8 xor  v(x' 00 ',, x2   ,, b3   ,, b3  )
      call q8 xor  v(x' 00 ',, x1   ,, b2   ,, b2  )
      call q8 xor  v(x' 00 ',, dn1  ,, b1   ,, b1  )
      call q8 vtovx (x' 00 ',, dnr1 ,, dnrp ,, dnr  )
      is=is+1
      ishi=i64(is)
      call q8 vtovx (x' 00 ',, dnr  ,, d1   ,, dn1 )
      call q8 shiftv(x' 08 ',, dn1  ,, ishi ,, d1  )
      call q8 vtovx (x' 00 ',, dnr  ,, d2   ,, dn2 )
      call q8 shiftv(x' 08 ',, dn2  ,, ishi ,, d2  )
      call q8 and  v(x' 01 ',, d2   ,, b2   ,, x   )
      call q8 xor  v(x' 00 ',, d2   ,, b2   ,, b2  )
      call q8 and  v(x' 01 ',, d1   ,, b1   ,, x1  )
      call q8 and  v(x' 01 ',, x1   ,, b2   ,, x2  )
      call q8 xor  v(x' 00 ',, x2   ,, x    ,, x   )
      call q8 or   v(x' 02 ',, x    ,, b3   ,, x   )
      call q8 xor  v(x' 00 ',, dsp2 ,, x    ,, dsp2)

c times the updating

      timet = timet + second()-t2

c takes averages

      if( nav.eq.0 )goto 99999
      if( mod(niter,nav).ne.0 )goto 99999
      ta3=second()
      call meas(dsp1,dsp2,dx,dy,dz,x,x1,x2,am,s,nh,tm)
      naver      = naver+1
      do 771 i=1,64
      ss=s(i)
      rm=abs(am(i))
      rm2=rm*rm
      rm4=rm2*rm2
      s2=ss*ss
      s4=s2*s2
      r( 1,i) = r( 1,i) + am(i)
      r( 2,i) = r( 2,i) + rm
      r( 3,i) = r( 3,i) + rm2
      r( 4,i) = r( 4,i) + rm4
      r( 5,i) = r( 5,i) + ss
```

Fig. 2 *(continued)*

```
      r( 6,i) = r( 6,i) + s2
      r( 7,i) = r( 7,i) + s4
      r( 8,i) = r( 8,i) + rm*ss
      r( 9,i) = r( 9,i) + rm2*ss
      r(10,i) = r(10,i) + rm4*ss
      r(11,i) = r(11,i) + ss*s2
 771  r(12,i) = r(12,i) + ss*s4

      ta=ta+second()-ta3

99999 continue

      if( nav.gt.nsw .or. nav.eq.0 )return
      r(1,1;1024) = r(1,1;1024) /naver
      return
      end

      subroutine demetro(b,nav,d1,d2,ia,ib,ic,isd,dxra,xrand,tf,ntb
     +                   ,d1v,d2v,abe,nh,npm,cv,cv1,cv2)

c sets the demons with the right probabilities

      integer xrand(*),d1v(*),d2v(*),dxra,d1,d2
      descriptor dxra,d1,d2,ia,ib,ic,isd
      bit cv(*),cv1(*),cv2(*)
      dimension abe(3),be(3)
      half precision ma(2),e1,e2,e3
      equivalence(mar,ma(1))

      tf1=second()
      nh2=2*nh
      d1 = 0
      d2 = 0
      be(1;3)=0.
      rrr=1./nh2

c gets normalized boltzman factors

      e1 = 2.**23 *exp(  -4.*b )
      e2 = 2.**23 *exp(  -8.*b )
      e3 = 2.**23 *exp( -12.*b )

c loops over the 32 bits of the halfwords (uses half precision)

      do 1 i=0,31
      mar=shift(1,i)

c gets random numbers using a shif register random number generator

      call q8 xor  v(x'00' ,, ia ,, ib ,, dxra )
      call q8 vto  v(x'00' ,, ic ,,,,   isd    )

c sets the demons and meassures the effective beta

      call q8cmplt(x'88',,xrand(npm;nh2),,e2    ,cv(1;nh2))
      call q8cmplt(x'88',,xrand(npm;nh2),,e1    ,cv1(1;nh2))
      call q8cmplt(x'88',,xrand(npm;nh2),,e3    ,cv2(1;nh2))

      if(nav.ne.0)then
      be(1)=be(1)+q8scnt(cv1(1;nh2))*rrr
      be(2)=be(2)+q8scnt(cv(1;nh2))*rrr
      be(3)=be(3)+q8scnt(cv2(1;nh2))*rrr
      endif

      call q8xor v(x'88',,d2v(1;nh2) ,,ma(2),cv(1;nh2),d2v(1;nh2))
      call q8andn(x'00',,cv1(1;nh2),,cv2(1;nh2),,cv1(1;nh2))
      call q8xor(x'00',,cv1(1;nh2),,cv(1;nh2),,cv(1;nh2))
      call q8xor v(x'88',,d1v(1;nh2) ,,ma(2),cv(1;nh2),d1v(1;nh2))

 1    continue

      if(nav.ne.0)then
      abe(1;3)=abe(1;3)+be(1;3)/32.
      ntb=ntb+1
      endif
      tf=tf+second()-tf1
      return
      end
```

Fig. 2 (*continued*)

```
      subroutine meas(dsp1,dsp2,dx,dy,dz,x,x1,x2,am,s,nh,tm)

c measures the magnetization and the action for the 64 lattices.

      integer    dx,dy,dz,x,x1,x2,dsp1,dsp2
      descriptor dx,dy,dz,x,x1,x2,dsp1,dsp2
      dimension am(*),s(*)

      tm1=second()

      do 2 i = 1,64
      ma = shift( 1 , i-1 )
      ls = mod( 65-i , 64 )

c measures the magnetization

      call q8 and   v(x' 09 ',,  dsp1 ,,  ma    ,,  x1  )
      call q8 shiftv(x' 08 ',,  x1    ,,  ls    ,,  x1  )
      am(i) = q8ssum( x1 )
      call q8 and   v(x' 09 ',,  dsp2 ,,  ma    ,,  x2  )
      call q8 shiftv(x' 08 ',,  x2    ,,  ls    ,,  x2  )
      am(i) = am(i) + q8ssum( x2 )

c measures the action

      call q8 xor   v(x' 00 ',,  x1    ,,  x2    ,,  x   )
      s(i) = q8ssum( x )
      call q8 vxtov (x' 00 ',,  dy    ,,  x2    ,,  x   )
      call q8 xor   v(x' 00 ',,  x     ,,  x1    ,,  x   )
      s(i) = s(i) + q8ssum( x )
      call q8 vxtov (x' 00 ',,  dz    ,,  x2    ,,  x   )
      call q8 xor   v(x' 00 ',,  x     ,,  x1    ,,  x   )
      s(i) = s(i) + q8ssum( x )
      call q8 vtovx (x' 00 ',,  dx    ,,  x2    ,,  x   )
      call q8 xor   v(x' 00 ',,  x     ,,  x1    ,,  x   )
      s(i) = s(i) + q8ssum( x )
      call q8 vtovx (x' 00 ',,  dy    ,,  x2    ,,  x   )
      call q8 xor   v(x' 00 ',,  x     ,,  x1    ,,  x   )
      s(i) = s(i) + q8ssum( x )
      call q8 vtovx (x' 00 ',,  dz    ,,  x2    ,,  x   )
      call q8 xor   v(x' 00 ',,  x     ,,  x1    ,,  x   )
      s(i) = s(i) + q8ssum( x )

 2    continue

      am(1;64) = 1. - am(1;64)/nh
      s(1;64)  = 1. - s(1;64) /(3.*nh)
      tm=tm+second()-tm1
      return
      end

      subroutine data(b,nsw,nsba,nav,init,betao,nso,iib,ntb,abe,ns,r)

      character a(130)
      dimension t(16),v(16),r(16,64),abe(3)
      data a/130*'-'/

c writes the header

      print*,a
      print * ,'beta = ',b
      if(iib.eq.1)then
      if(init.eq.0)print*,'initial lattice ordered'
      if(init.eq.1)print*,'initial lattice disordered'
      if(init.eq.6)then
      print*,'initial lattice read from tape 6, which was termalized'
     +      ,' during ',nso,' sweeps at beta = ',betao
      endif
      else
      print*,'initial lattice from previous run'
      endif
      print*,'thermalizing during ',nsba,' sweeps'
      print*,'taking averages after every ',nav,' sweeps during '
     +      ,nsw,' sweeps'
      print*,a

c calculates the data for every lattice
```

Fig. 2 (*continued*)

```
      t(1;16)  = .0
      v(1;16)  = .0
      do 1 i = 1,64
      r(13,i)    =  ns * (r(3,i) - r(2,i)*r(2,i))
      r(14,i)    =  r(3,i) * ns
      r(15,i)    =  r(4,i)/(r(3,i)*r(3,i))-3.
      r(16,i)    =  3.* ns * (r(6,i) - r(5,i)*r(5,i))
      t(1;16)    =  t(1;16)  + r(1,i;16)
      v(1;16)    =  v(1;16) + r(1,i;16)*r(1,i;16)
      print  8 ,i,r(1,i),r(2,i),r(5,i),r(13,i),r(14,i),r(16,i),r(15,i)
 8    format(' lat ',i2,'   m= ',f10.7,'   |m|= ',f10.7,'   s= ',f10.7
     + ,'   x= ',f11.6,'   xp= ',f11.6,'   c= ',f11.6,'   gr= ',f11.6)
 1    continue

c writes the data averaged from the 64 lattices

      do 53 i=1,16
      t(i)  = t(i)  /64.
 53   v(i)  = sqrt( v(i)/64. - t(i)*t(i) ) /8.
      print*,a
      print * ,'data from averaging the 64 lattices :
      nf=0
 888  print *
      print11,'  m = ',t(1),'|m| = ',t(2),'m↑2 = ',t(3),'m↑4 = ',t(4)
      print12,              '  +/- ',v(2),'  +/- ',v(3),'  +/- ',v(4)
      print *
      print12,'  s = ',t(5),'s↑2 = ',t(6),'s↑4 = ',t(7)
      print12,'  +/- ',v(5),'  +/- ',v(6),'  +/- ',v(7)
      print *
      print13,' xp =',t(14),'  x =',t(13),'  c =',t(16),' gr =',t(15)
      print13,'  +/-',v(14),'  +/-',v(13),'  +/-',v(16),'  +/-',v(15)
      if(nf.eq.1)goto 889
      print *
 11   format( 5x,4(10x,a6,f11.8))
 12   format(32x,3(10x,a6,f11.8))
 13   format(15x,2(a5,f12.5,10x),a5,f12.6,10x,a5,f12.7)

c correctes the data for shifted value of beta.

      abe(1;3)=abe(1;3)/ntb
      coe=(abe(1)+2.*abe(2)+3.*abe(3))/(1.+abe(1)+abe(2)+abe(3))
      z1=0.
      z2=1.
      do 75 i=1,100
      z=(z1+z2)*.5
      ed=(z+2.*z*z+3.*z*z*z)/(1.+z+z*z+z*z*z)
      if(ed.eq.coe)goto 777
      if(ed.lt.coe)then
      z1=z
      else
      z2=z
      endif
 75   continue

 777  ba=(-1./4.)*log(z)
      rlb=3.*ns*(b-ba)
      t6=t(6)
      t(2)  = t(2)   +(t(8) -t(2)*t(5)  )*rlb
      t(3)  = t(3)   +(t(9) -t(3)*t(5)  )*rlb
      t(4)  = t(4)   +(t(10)-t(4)*t(5)  )*rlb
      t(6)  = t(6)   +(t(11)-t(6)*t(5)  )*rlb
      t(7)  = t(7)   +(t(12)-t(7)*t(5)  )*rlb
      t(5)  = t(5)   +(t6   -t(5)*t(5)  )*rlb
      t(14) = t(3) * ns
      t(15) = t(4)/(t(3)*t(3))-3.
      t(13) = ns *(t(3) - t(2)*t(2))
      t(16) = 3.* ns *(t(6) - t(5)*t(5))

      print * ,'effective beta = ', ba
      print *
      print * ,'corrected data:'
      nf=1
      goto 888
 889  print*,a
      print*
      ntb=0
      abe(1;3)=0.
```

Fig. 2 (*continued*)

```
      return
      end

      subroutine initlat(init,betao,nso,dsp1,dsp2,ran,x1,spin,nh,ns,l)

c initializes the lattices

      integer spin,x1,dsp1,dsp2
      descriptor x1,dsp1,dsp2
      dimension spin(*),ran(*)

      dsp1 = 0
      dsp2 = 0
      if ( init.eq. 1 )then

      do 1 i = 1,64
      ls=mod(65-i,64)
      x1=0
      call vranf(ran,nh)
      where (ran(1;nh).ge..5)x1=1
      call q8 shiftv(x' 08 ',,  x1   ,,  ls   ,,  x1  )
      call q8 xor  v(x' 00 ',,  x1   ,,  dsp1 ,,  dsp1)
      x1=0
      call vranf(ran,nh)
      where (ran(1;nh).ge..5)x1=1
      call q8 shiftv(x' 08 ',,  x1   ,,  ls   ,,  x1  )
      call q8 xor  v(x' 00 ',,  x1   ,,  dsp2 ,,  dsp2)
 1    continue

      else if(init.eq.6)then

      read(6,*)lo,betao,nso
      do 76 i=1,ns
 76   read(6,177) spin(i)
 177  format(z16)

      endif
      return
      end

      subroutine raninit(xrand,np)
      integer xrand(*)

c initializes the shift register random number generator.

      do 1 i=1,np
      ic=0
      do 2 j=1,55
      ic=shift(ic,1)
      if((j.le.23).or.(j.ge.33.and.j.le.55))then
      if(ranf().ge..5)ic=or(ic,1)
      endif
 2    continue
 1    xrand(i)=ic
      return
      end

      subroutine init64( i64 , n64 )

c sets i64 with integers in such a way as to satisfy that the serie
c [i64(1) , i64(1)+i64(2) , ... , i64(1)+i64(2)+...+i64(64) ; (modulo 64)]
c is a random permutation of the numbers 0 to 63.

      dimension i64(64),n64(64)

      n64(2;63)=0
      i64(1)=0
      n64(1)=1
      ns=0
      do 1 i=2,64
 7    n=int(63.*ranf())+1
      nso=mod(n+ns,64)
      if(n64(nso+1).eq.1)goto 7
      ns=nso
      n64(ns+1)=1
 1    i64(i)=n
      return
      end
```

Fig. 2 (*continued*)

THE OUTPUT

```
BETA = 0.2216500000000
INITIAL LATTICE DISORDERED
THERMALIZING DURING 25000 SWEEPS
TAKING AVERAGES AFTER EVERY 100 SWEEPS DURING 25000 SWEEPS

LAT  1  M= -0.0048340  |M|= 0.2362540  S= 0.3407013  X= 110.175884  XP= 556.703504  C= 13.820515  GR= -1.404986
LAT  2  M= -0.0060040  |M|= 0.2374780  S= 0.3424333  X= 100.785924  XP= 551.952328  C= 11.544944  GR= -1.439802
LAT  3  M=  0.0572650  |M|= 0.2364070  S= 0.3419873  X= 107.302749  XP= 554.408906  C= 11.325593  GR= -1.436963
LAT  4  M=  0.0091760  |M|= 0.2296100  S= 0.3406607  X=  96.555295  XP= 518.321312  C= 10.927884  GR= -1.448633
LAT  5  M= -0.0095580  |M|= 0.2367620  S= 0.3410167  X=  97.227219  XP= 545.677176  C= 12.959921  GR= -1.437544
LAT  6  M=  0.0282970  |M|= 0.2313130  S= 0.3396833  X= 106.297386  XP= 534.343018  C= 14.872295  GR= -1.339440
LAT  7  M= -0.0176570  |M|= 0.2341750  S= 0.3392287  X=  88.899693  XP= 527.603138  C= 11.822033  GR= -1.468217
LAT  8  M= -0.0533300  |M|= 0.2499920  S= 0.3417460  X= 103.686307  XP= 603.654308  C= 13.999393  GR= -1.479494
LAT  9  M=  0.0011660  |M|= 0.2428540  S= 0.3429007  X=  94.774681  XP= 566.599204  C= 12.160427  GR= -1.477912
LAT 10  M= -0.0645830  |M|= 0.2357510  S= 0.3415207  X= 105.472886  XP= 550.101158  C= 13.831400  GR= -1.449696
LAT 11  M= -0.0644610  |M|= 0.2312430  S= 0.3399600  X= 107.025518  XP= 534.812118  C= 12.742202  GR= -1.410785
LAT 12  M=  0.0571420  |M|= 0.2449060  S= 0.3432587  X= 102.932501  XP= 582.764092  C= 13.806341  GR= -1.459463
LAT 13  M=  0.0178130  |M|= 0.2385110  S= 0.3427120  X= 101.954661  XP= 557.054638  C= 12.487988  GR= -1.467532
LAT 14  M=  0.0222410  |M|= 0.2428410  S= 0.3416220  X= 105.205192  XP= 576.979202  C= 14.633021  GR= -1.401023
LAT 15  M=  0.0511210  |M|= 0.2472350  S= 0.3448967  X= 110.701104  XP= 599.702266  C= 13.894701  GR= -1.450949
LAT 16  M= -0.0228950  |M|= 0.2398090  S= 0.3416553  X= 101.638990  XP= 561.705842  C= 13.583701  GR= -1.449402
LAT 17  M= -0.0063080  |M|= 0.2367260  S= 0.3404867  X=  99.561595  XP= 547.875188  C= 12.919553  GR= -1.441420
LAT 18  M= -0.0140420  |M|= 0.2344560  S= 0.3404773  X=  99.605921  XP= 539.362848  C= 13.586372  GR= -1.411280
LAT 19  M= -0.0046000  |M|= 0.2374220  S= 0.3421447  X= 100.684839  XP= 551.638488  C= 13.391730  GR= -1.439817
LAT 20  M=  0.0655720  |M|= 0.2454060  S= 0.3426327  X= 102.462225  XP= 584.255064  C= 13.413788  GR= -1.480519
LAT 21  M= -0.0354740  |M|= 0.2415020  S= 0.3414160  X= 103.887932  XP= 570.473660  C= 14.753841  GR= -1.429721
LAT 22  M= -0.0398660  |M|= 0.2293420  S= 0.3415833  X=  99.504244  XP= 520.286268  C= 11.232732  GR= -1.395200
LAT 23  M=  0.0172160  |M|= 0.2342620  S= 0.3415793  X= 102.736039  XP= 541.765516  C= 11.589478  GR= -1.460724
LAT 24  M=  0.0144960  |M|= 0.2252000  S= 0.3397087  X= 108.871908  XP= 514.592228  C= 13.764910  GR= -1.327357
LAT 25  M=  0.0042670  |M|= 0.2266550  S= 0.3397967  X= 105.065110  XP= 516.045022  C= 13.711048  GR= -1.349436
LAT 26  M=  0.0410510  |M|= 0.2502710  S= 0.3440087  X= 112.049498  XP= 613.134086  C= 16.270822  GR= -1.447704
LAT 27  M= -0.0317170  |M|= 0.2323250  S= 0.3397793  X= 109.410365  XP= 541.209610  C= 12.744061  GR= -1.362801
LAT 28  M= -0.0167460  |M|= 0.2312400  S= 0.3402553  X= 101.122475  XP= 528.897976  C= 12.616406  GR= -1.428730
LAT 29  M= -0.0465300  |M|= 0.2261140  S= 0.3391940  X= 100.711656  XP= 509.731984  C= 11.692486  GR= -1.356412
LAT 30  M= -0.0068330  |M|= 0.2270630  S= 0.3397800  X= 110.964414  XP= 523.425262  C= 12.241286  GR= -1.355379
LAT 31  M=  0.0382730  |M|= 0.2247450  S= 0.3376960  X= 102.312602  XP= 506.395122  C= 14.611659  GR= -1.317529
LAT 32  M=  0.0475370  |M|= 0.2542070  S= 0.3428467  X= 100.382207  XP= 617.351798  C= 13.795692  GR= -1.523595
LAT 33  M= -0.0017660  |M|= 0.2405880  S= 0.3416613  X= 101.110378  XP= 564.171064  C= 12.900501  GR= -1.448031
LAT 34  M= -0.0330590  |M|= 0.2448170  S= 0.3432627  X= 109.150798  XP= 588.633706  C= 14.828275  GR= -1.449726
LAT 35  M=  0.0024570  |M|= 0.2357230  S= 0.3415580  X= 108.049696  XP= 552.572358  C= 14.470135  GR= -1.375503
LAT 36  M=  0.0074520  |M|= 0.2298380  S= 0.3402893  X= 106.734894  XP= 529.338944  C= 11.459671  GR= -1.417201
LAT 37  M= -0.0095780  |M|= 0.2363900  S= 0.3401787  X=  94.498939  XP= 541.540796  C= 13.412455  GR= -1.444748
LAT 38  M= -0.0225300  |M|= 0.2210240  S= 0.3375653  X= 103.354811  XP= 494.167680  C= 12.974650  GR= -1.349329
LAT 39  M=  0.0009450  |M|= 0.2466430  S= 0.3440880  X=  96.700954  XP= 583.363110  C= 13.392945  GR= -1.485523
LAT 40  M= -0.0502000  |M|= 0.2366260  S= 0.3413707  X= 105.732957  XP= 553.667868  C= 14.063455  GR= -1.409034
LAT 41  M=  0.0248120  |M|= 0.2258940  S= 0.3408440  X= 104.527774  XP= 512.752568  C= 11.817144  GR= -1.313046
LAT 42  M=  0.0417440  |M|= 0.2266040  S= 0.3393280  X= 103.064641  XP= 513.859624  C= 13.372458  GR= -1.343500
LAT 43  M=  0.0138510  |M|= 0.2352530  S= 0.3417393  X= 107.540090  XP= 550.291882  C= 11.980873  GR= -1.433784
LAT 44  M=  0.0119640  |M|= 0.2307960  S= 0.3392033  X= 102.290283  XP= 528.424632  C= 13.147762  GR= -1.372322
LAT 45  M= -0.0391620  |M|= 0.2252560  S= 0.3399860  X= 110.263588  XP= 516.185712  C= 14.517491  GR= -1.330908
LAT 46  M=  0.0187480  |M|= 0.2324940  S= 0.3403007  X=  99.446916  XP= 531.874596  C= 12.709390  GR= -1.440305
```

```
LAT 47   M=  0.0080730   |M|= 0.2440050   S= 0.3418740   X=  93.091318   XP= 569.398838   C= 12.516944   GR= -1.485784
LAT 48   M=  0.0870040   |M|= 0.2426860   S= 0.3434460   X=  96.809479   XP= 567.981436   C= 12.178103   GR= -1.476262
LAT 49   M=  0.0457270   |M|= 0.2388870   S= 0.3416993   X= 100.120920   XP= 556.656910   C= 12.530724   GR= -1.451987
LAT 50   M= -0.0133020   |M|= 0.2383760   S= 0.3425980   X=  95.285625   XP= 549.870564   C= 13.907340   GR= -1.460578
LAT 51   M=  0.0433750   |M|= 0.2370050   S= 0.3410700   X=  95.596766   XP= 544.967726   C= 11.677170   GR= -1.480204
LAT 52   M=  0.0024720   |M|= 0.2460040   S= 0.3447127   X= 111.659312   XP= 595.803056   C= 12.118731   GR= -1.453082
LAT 53   M=  0.0310740   |M|= 0.2358400   S= 0.3407327   X=  99.688343   XP= 544.652388   C= 13.143439   GR= -1.403298
LAT 54   M= -0.0165520   |M|= 0.2375160   S= 0.3416140   X= 102.624162   XP= 553.934964   C= 14.680107   GR= -1.424157
LAT 55   M=  0.0033870   |M|= 0.2414870   S= 0.3404900   X=  93.884013   XP= 560.411782   C= 14.944336   GR= -1.443537
LAT 56   M=  0.0870730   |M|= 0.2311990   S= 0.3402113   X= 101.265161   XP= 528.888982   C= 11.991000   GR= -1.437309
LAT 57   M=  0.0424970   |M|= 0.2380810   S= 0.3413160   X= 103.730126   XP= 557.190626   C= 14.273246   GR= -1.413908
LAT 58   M=  0.0267990   |M|= 0.2459770   S= 0.3418933   X= 102.415334   XP= 586.452810   C= 13.440079   GR= -1.470905
LAT 59   M=  0.0057930   |M|= 0.2307130   S= 0.3408100   X= 110.073627   XP= 535.901534   C= 14.677312   GR= -1.364311
LAT 60   M= -0.0756560   |M|= 0.2382820   S= 0.3410207   X=  93.147560   XP= 547.374052   C= 12.300062   GR= -1.438487
LAT 61   M=  0.0334920   |M|= 0.2270960   S= 0.3399493   X=  98.100586   XP= 510.681332   C= 11.204285   GR= -1.407199
LAT 62   M= -0.0207010   |M|= 0.2250570   S= 0.3370800   X=  94.834708   XP= 500.039934   C= 12.393889   GR= -1.427377
LAT 63   M= -0.0381700   |M|= 0.2480560   S= 0.3420673   X= 110.739259   XP= 602.993492   C= 14.359734   GR= -1.471558
LAT 64   M=  0.0069490   |M|= 0.2320050   S= 0.3407167   X= 107.225510   XP= 537.836070   C= 11.977857   GR= -1.378928
------------------------------------------------------------------------------------------------------------------------
DATA FROM AVERAGING THE 64 LATTICES :

          M =  0.00394073      |M| =  0.23622336      M↑2 =  0.06867325      M↑4 =  0.00744216
                               +/-    0.00091810      +/-    0.00044161      +/-    0.00007913

                                S  =  0.34115696      S↑2 =  0.11693744      S↑4 =  0.01393827
                               +/-    0.00019446      +/-    0.00013363      +/-    0.00003215

         XP =    549.38599      X  =   102.54262      C   =   13.126715      GR  =  -1.4226765
         +/-       3.53290     +/-       0.66730      +/-      0.144369      +/-     0.0059863

EFFECTIVE BETA = 0.2216629017657

CORRECTED DATA:

          M =  0.00394073      |M| =  0.23554887      M↑2 =  0.06834626      M↑4 =  0.00738606
                               +/-    0.00091810      +/-    0.00044161      +/-    0.00007913

                                S  =  0.34098685      S↑2 =  0.11681966      S↑4 =  0.01390967
                               +/-    0.00019446      +/-    0.00013363      +/-    0.00003215

         XP =    546.77009      X  =   102.90392      C   =   13.142999      GR  =  -1.4188134
         +/-       3.53290     +/-       0.66730      +/-      0.144369      +/-     0.0059863
------------------------------------------------------------------------------------------------------------------------

TIME SPENT IN THE MAIN PARTS OF THE PROGRAM :

 PROGRAM  ISING =       288.27  SECONDS
 IN UPDATING    =       260.21  SECONDS
 IN MEASURING   =        27.28  SECONDS
 SUB. DEMETRO   =        15.73  SECONDS
 SUB. INITNR    =         3.92  SECONDS
 SUB. INIT64    =         1.01  SECONDS
 SUB. MEAS      =        27.25  SECONDS

RUNNING AT  98.4 MEGAFLIPS
```

Fig. 3. Output obtained from running the code shown in Fig. 2. The parameters used are those given in the code. M stands for the magnetization per spin $\langle M\rangle/V$, XP for $\langle M^2\rangle/V$, X for $(\langle M^2\rangle - \langle |M|\rangle^2)/V$, GR for the renormalized coupling constant $(M^4)/(\langle M^2\rangle^2) - 3$, S for the nearest-neighbor spin–spin correlation function $\langle s_i s_{i+\hat{\mu}}\rangle$, and C for $dS/d\beta$. The data is corrected using eq. 10.

distribution. The corrected data is shown in Fig. 1(b). Note that each data point is corrected by a different amount because each corresponds to a different stream of random numbers and therefore, a different $\beta_{\rm eff}$. In Fig. 1(c) we plot a conventional Metropolis run done by using completely independent sets of random numbers for the 64 lattices. The vertical line corresponds to the exact value. We have repeated this experiment three more times for a total of 160 runs. In 106 and 103 of these runs, $\langle S \rangle$ and $\langle S^2 \rangle$ were within 1 sd of the exact value. This corresponds to 66.25% and 64.37%, respectively, and indicates that the errors are properly defined.

THE CODE

The code is written in standard Cyber Fortran 200 using the special *Q*8 calls which translate directly into machine code. We use descriptors to point to arrays in the standard way, and the motivated reader is directed to the Fortran 200 manual for inspiration. A listing of the code is included with this paper along with the output (Fig. 2 and 3).[3]

ACKNOWLEDGMENTS

The work of G.B. and R.S. was supported by the Department of Energy under cooperative agreement number DE-FC05-85ER25000. The work of D.D. was supported both by the previous agreement as well as the DOE grant number DE-AS05-76ER3509.

[3] Fortran 200 Version 1 Reference Manual (publication number 60485000) CDC Cuber 205 Hardware Reference Manual (publication number 60256020). These manuals are available from: Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, MN 55103.

REFERENCES

1. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. H. Teller, and E. Teller, *J. Chem. Phys.* **21**, 1087 (1953).
2. G. O. Williams and M. H. Kalos, *J. Stat. Phys.* **37**:283 (1984).
3. R. Zorn, H. J. Herrmann, and C. Rebbi, *Comp. Phys. Common.* **23**:337 (1981), and references therein.
4. S. F. Reddaway, D. M. Scott, and K. A. Smith, *Comp. Phys. Comm.* **37**:351 (1985).
5. G. Bhanot, D. Duke, and R. Salvador, *Phys. Rev. B.* **33**:7841 (1986).
6. M. Creutz, *Phys. Rev. Lett.* **50**:1411 (1983).
7. S. Wansleben, J. G. Zabolitzky, and C. Kalle, *J. Stat. Phys.* **37**:271 (1984).