Journal of Statistical Physics, Vol. 60, Nos. 3/4, 1990

Solving the Ising Model Exactly on a $5 \times 5 \times 4$ Lattice Using the Connection Machine

Gyan Bhanot[1,2] **and Srikanth Sastry**[3]

Received December 13, 1989; final February 6, 1990

We implement a recently proposed exact method for solving distrete statistical models for the 3-dimensional Ising model with open boundary conditions. Our computations were done on the Connection machine because the problem maps very naturally onto massively parallel architectures. We explicitly calculate the number of states of the system at each energy for systems of size $5 \times 4 \times L_z$ for $L_z \leqslant 5$. On serial or vector computers, the time for the computation scales with the volume V like $V2^{L_x L_y}$. On the Connection Machine, the calculation can be spread across the processors. This decreases the computation requirements by a factor equal to the number of processors. We describe the method, its implementation on the Connection Machine both in PARIS and in FORTRAN, and our results. We also state the requirements for solving larger systems using this method.

KEY WORDS: Ising; exact; partition function; massively-parallel architecture; zeros; critical exponents.

1. INTRODUCTION

The number of states in an $L_x \times L_y \times L_z$ Ising model is 2^V, where $V = L_x L_y L_z$ is the lattice volume. Generating one state every nanosecond, it would take about 4×10^{13} years to generate all states for a cube of size $4 \times 5 \times 5$. Pearson[1] first described a method that made it possible to cut down the number of states generated to a square root of the total number and he used the method to solve a $4 \times 4 \times 4$ system. Recently, inspired by

[1] Thinking Machines Corp., Cambridge, Massachusetts 02142.

[2] On leave from Physics Department, San Francisco State University, San Francisco, California, and Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida.

[3] Physics Department, Boston University, Boston, Massachusetts 02215.

0022-4715/90/0800-0333$06.00/0

earlier work of Binder,[2] Bhanot[3] has described a method to solve the problem in a time that scales for serial and vector computers like $V2^{V/L_z}$. For massively parallel computers such as the Connection Machine, it is obvious that with 2^p processors, the computer time scales like $V2^{L_xL_y-p}$, since the calculation can be spread out across the processors. For the most direct implementation of the method, the storage required scales like $V'V2^{V/L_z}$, where $V'=3V-L_xL_y-L_yL_z-L_zL_x$ for open boundary conditions. V' is the number of bonds in the system. It is also the maximum possible value of the total energy. As a described below, the storage requirement can be reduced to

$$2m\left(V+\frac{V'}{m}\frac{\log(c)}{\log(2)}\right)2^{V/L_z}$$

where m and c are integers ($m\in[1, V'/2]$ and $c=1, 2, 3, 4,...$). In this case, however, one needs to do about $V'/2m$ separate runs at different values of c (and fixed m) and reconstruct the full solution from these runs. However, each of the runs needs less computations by just the correct factor to ensure that the total computational requirements remain the same.

2. THE METHOD

The method we will use is described in ref. 3. The present paper should be read in conjunction with ref. 3. Here we will explain the method in the explicit context of the 3D Ising model with open boundary conditions, spelling out all the details. Assume that the energy of a bond with opposite spins at its ends is unity and with same-sign spins is zero. The energy then takes integer values in $[0, V']$. Solving the model is equivalent to determining the number of states $P(E)$ of the system at each energy E. For open boundary conditions, a state with energy E can always be transformed into one with energy $V'-E$ by a change of variables. Thus, $P(E)=P(V'-E)$. This means that one only needs to find $P(E)$ for $E\in[0, V'/2]$. The partition function is given by

$$Z(u)=\sum_{E=0}^{V'}P(E)u^E \tag{1}$$

where $u=e^{-\beta}$. We will always work with u's which have a specific form: $u=c^{1/m}$ with $c\geqslant 0$ and $m\geqslant 1$.

One starts by enumerating all states of an L_xL_y Ising model ($2^{L_xL_y}$ states). Since the Ising variable is a bit variable, one can choose an up spin to be represented by the bit value 0 and a down spin by the bit value 1.

The binary bits corresponding to the $L_x L_y$ spins can be used to define an $L_x L_y$-bit integer S which labels the states.

First, we make a digression to discuss a particular way we will store information in our calculation. The method relies on doing arithmetic in base $c^{1/m}$, so we will first discuss some aspects of such calculations. Any polynomial with integer coefficients in the variable $u = c^{1/m}$ can be written in terms of m integers $Q(j)$, $j = 0, 1, 2, \ldots, m-1$. Thus, it is easy to show that for any $u = c^{1/m}$, the partition function

$$Z(u) = Z(c, m) = \sum_{E=0}^{V'} P(E) c^{E/m} \tag{2}$$

can be written as

$$Z(c, m) = \sum_{k=0}^{m-1} Q(k) c^{k/m} \tag{3}$$

Moreover,

$$Q(k) = \sum_{j=0}^{\mathrm{Int}[(V'-k)/m]} P(k + jm) c^{j} \tag{4}$$

For more detail on these points, the reader is referred to ref. 3. This ends the digression.

Now, each state of $L_x L_y$ spins is labeled by a $L_x L_y$-bit integer S and has an associated energy $e(S)$ and a Boltzmann weight $u^{e(S)} = c^{e(S)/m}$. The energy function $e(S)$ is precalculated and placed in an array of length $2^{L_x L_y}$. In addition, we define two integer arrays $I^o(k, S)$ and $I^n(k, S)$, where k runs from 0 to $m-1$ and S from 0 to $2^{L_x L_y} - 1$. The number of bits of accuracy necessary in the I's will be specified later. The first index of the arrays labels the m integers 0, 1,..., $(m-1)$ for the I's and the second labels the states. The I^o are initialized as follows:

$$I^o(0, S) = 1, \qquad \forall S \tag{5a}$$

$$I^o(k, S) = 0, \qquad \forall S \quad \text{and} \quad k = 1, 2, \ldots, (m-1) \tag{5b}$$

The Boltzmann weights for the bonds in the $L_x L_y$ plane are then put in by multiplying I^o for the state S by $u^{e(S)}$. This is done by repeatedly multiplying I^o by u. Note that multiplication by u is equivalent to

$$(I(0, S), I(1, S), \ldots, I(m-1, S)) \to (cI(m-1, S), I(0, S), \ldots, I(m-2, S)) \tag{6}$$

As described in ref. 3, the layers in the $\hat{z}$ direction are now built up one by one be the following algorithm:

(a) Perform the operations

$$(I^n(0, S), I^n(1, S),..., I^n(m-1, S)) = (I^o(0, S), I^o(1, S),..., I^o(m-1, S)) + (cI^o(m-1, S'), I^o(0, S'),..., I^o(0, S'),..., I^o(m-2, S')) \quad (7a)$$

where S and S' differ in any one bit. Next, set

$$I^o(k, S) = I^n(k, S), \forall k, S \quad (7b)$$

This puts in one spin in the layer (see ref. 3).

Table I

```
1  ************************************************************************
2
3        program ising
4
5  C     Front End Variables
6        integer l1, l2, l3, m, cc, nproc
7        parameter(l1 = 4, l2 = 2, l3 = 5, m = 102, cc = 0, nproc = 2**(l1*l2))
8        integer i, j, k, maxconfig, ipar, jpar
9        integer z(0:m - 1, 0:1)
10
11 C     Connection Machine Variables
12       integer help(nproc), config(nproc), spin(0:l1*l2 - 1, nproc)
13       integer ep, energy, p, c, temp
14         common/cm/ep(nproc), energy(nproc), p(0:m - 1, 0:a, nproc), c(nproc),
15       !           temp(nproc), mask(nproc)
16         common/fe/ipar
17 C       CMF$LAYOUR explicitly declares the array indices to be serial
18 C       or news (i.e., processor) indces. For arrays of any rank, the
19 C       last index is the news index, by default.
20 CMF$LAYOUT p(:serial, :serial, :news)
21 CMF$LAYOUT spin(:serial, :news)
22 ************************************************************************
23 c There are many lines of code left out here. These do the initialization.
24 c In particular, the array config is initialized by config(i) = i.
25 c Thus, it contains a bit string that represents a 2D spin configuration.
26 c energy(i) contains the energy fo the spin configuration i.
27 ************************************************************************
28 c The core part of the code is shown below:
29
30         nener = 2*l1*l2 - l1 - l2
31 c nener is the maximum total energy for the bonds in the (12) plane.
```

Table I *(continued)*

```
32  c Loop over the z direction adding l3 − 1 new layers
33        do i = 0, l3 − 2
34  c Loop over the spins in the xy plane
35           jj = 1
36           do 1 j = 0, l1*l2 − 1
37  c ipar and jpar do the switch between the old and new I's (see Eq. 7)
38              jpar = ipar
39              ipar = 1 − jpar
40  c construct an array help from config where the j th bit is switched.
41              jj = jj*2
42                 help(:) = (1 − 2*spin(j, :))*jj + config(:)
43  c the next lines implement Eq. (7) of the text [note that the p's here are the
44  c I's of Eq. (7)].
45           do 2 k = 0, m − 1
46  2           p(k, ipar, :) = p(k, jpar, help(:))
47
48              temp(:) = p(m − 1, ipar, :)
49              do 3 i = m − 1, 1, −1
50  3           p(i, ipar, :) = p(i − 1, ipar, :)
51              p(0, ipar, :) = temp(:)
52
53           do 4 k = 0, m − 1
54  4           p(k, ipar, :) = pp(k, ipar, :) + p(k, jpar, :)
55
56  1        continue
57
58  c Now put in the weights of the horizontal bonds.
59              ep(:) = energy(:)
60        do 5 i = 0, never − 1
61           mask(:) = ep(:) .ne. 0
62           where(mask) temp(:) = p(m − 1, ipar, :)
63              do 6 j = m − 1, 1, −1
64                 where(mask) p(j, ipar, :) = p(j − 1, ipar, :)
65  6           continue
66              where(mask)
67                 p(0, ipar, :) = temp(:)*c(:)
68                 ep(:) = ep(:) − 1
69              end where
70  5     continue
71  c now sum p over all configurations (all processors) to compute the partition
72  c function
73              z = sum(p, 3)
74           print 100, (z(i, ipar), i, i = 0, m − 1)
75        end do
76  100   format(20(2x, i10, 2x, i10, /))
77
78     stop
79     end
```

(b) Repeat step (a) once for each of the bits in S. This puts in all the spins in one layer.

(c) Multiply I^o for the state S by $u^{e(S)}$ [this involves applying Eq. (6) repeatedly as described before]. This operation puts in the Boltzmann weights of the bonds in the xy plane for the layer of spins just added.

After the required number of layers is done, compute

$$Q(k) = \sum_S I^o(k, S) \tag{8}$$

$Q(k)$ is related to the $P(k)$ according to Eq. (4).

Three important points should be noted from Eq. (4):

1. For any c, and $m = V' + 1$, $Q(k) = P(k)$, $k = 0, 1, \ldots, V'$. Thus, if there is sufficient storage available so that m can be made as large as V', one can generate all the P's in one run.

2. For $c = 0$ and any m, $Q(k) = P(k)$, $k = 0, \ldots, m-1$.

3. For any other case, each set of c, m values generates m relationships between the Q's and the P's according to Eq. (4). Using $c = 0, 1, 2, \ldots$ successively, a sufficient number of such relationships must be generated to solve for the (approximately) $V'/2$ independent values of P. In our simulation, we found that for a $4 \times 4 \times 10$ system, we were able to choose $m = 192$ and $c = 0$ to get the entire partition function in one run on a Connection Machine CM-2 with 2^{33} bits of memory. However, for the $5 \times 4 \times 5$ system, we had to use $m = 30$ and make independent runs for $c = 0, 1, 2, \ldots$ to generate the 118 independent equations necessary to compute the P's from the Q's.

Finally, a word about the accuracy necessary in the computation. From Eq. (4), one notes that the maximum number of bits in Q is bounded by the sum of the maximum number of bits in $c^{V'/m}$ plus the number of bits in $\sum_k P(k)$. The latter number is obviously V because the sum equals the total number of states in the system, which is 2^V. Hence, the number of bits N_{bits} of accuracy in I^o or I^n satisfies

$$N_{\text{bits}} \leqslant V + \frac{V' \log(c)}{m \log(2)} \tag{9}$$

Since we have two arrays I^o and I^n each of size $m \times 2^{L_x L_y}$, the storage ST (in bits) is

$$ST = 2m2^{L_x L_y} N_{\text{bits}} = 2m \left(V + \frac{V' \log(c)}{m \log(2)} \right) 2^{V/L_z} \tag{10}$$

as stated before.

3. THE CODE

The program we used was implemented in C-PARIS on the Connection Machine (PARIS stands for the Connection Machine *Parallel Instruction Set*[4]). The easiest way to program the problem for a massively parallel machine such as the Connection Machine is to use $2^{L_z L_y}$ processors. All arrays with an argument that runs over $2^{L_x L_y}$ values are spread over the processors. Note that this is possible even if the number of processors is less than $2^{L_x L_y}$. This is because, on the Connection Machine, one can define virtual processors. In this mode, each processor divides up its memory into several pieces, thus serving as many processors. Of course, since the number of computational units is still equal to the number of physical processors, the improvement in speed is bounded by a factor equal to the number of physical processors.

The inner loop of the code is the step of Eq. (7a). In a serial or vector computer, this loop would have to be done for each S separately and

Table II. Partition Function of the 3D Ising Model for $L_x=4$, $L_y=5$, and $L_z=1, 2, 3, 4, 5$[a]

E	$P(E)$, $L_z=1$	E	$P(E)$, $L_z=2$	E	$P(E)$, $L_z=2$
0	2	0	2	21	21190056
1	0	1	0	22	41936224
2	8	2	0	23	81833224
3	36	3	16	24	157229136
4	76	4	48	25	296873488
5	250	5	56	26	549817868
6	752	6	140	27	996960024
7	1820	7	456	28	1765855160
8	4344	8	1192	29	3046388248
9	10104	9	2272	30	5104139032
10	20602	10	4942	31	8281110400
11	38156	11	12176	32	12970188706
12	65364	12	27608	33	19549528080
13	98836	13	58080	34	28274100024
14	131080	14	124764	35	39137756280
15	152858	15	272968	36	51728944860
		16	582684	37	65155043624
		17	1214436	38	78085969284
		18	2520552	39	88935282376
		19	5201264	40	96185893070
		20	10571648	41	98738356640

[a] E denotes the possible values of the energy and $P(E)$ the number of states at that energy. Only about half the $P(E)$'s are shown. One can construct the $P(E)$'s for the remaining energies up to $V'=3L_xL_yL_z-L_xL_y-L_yL_z-L_zL_x$ using $P(E)=P(V'-E)$.

Table II *(continued)*

E	$P(E), L_z = 3$	E	$P(E), L_z = 3$
0	2	34	168960566408
1	0	35	319612192080
2	0	36	600782123300
3	16	37	1121636916756
4	48	38	2078736963248
5	92	39	3821963947148
6	100	40	6966602629582
7	512	41	12580034882528
8	1432	42	22486093152568
9	2720	43	39750215721880
10	5804	44	69431445259736
11	13392	45	119711727122326
12	33356	46	203531431305882
13	70548	47	340856249890966
14	145632	48	561653802726326
15	320390	49	909540211213592
16	687090	50	1445833872151524
17	1455844	51	2253406488205504
18	3007496	52	3439267569482740
19	6207780	53	5134323925278036
20	12863968	54	7488378337795196
21	26235196	55	10658418042690846
22	53173312	56	14788842275715094
23	107128560	57	19983699115701200
24	214696168	58	26273264465769916
25	428411660	59	33580011599608232
26	848900124	60	41691459968300196
27	1674147124	61	50248605577147974
28	3286412636	62	58757617081907958
29	6417955372	63	66628781943162882
30	12473871606	64	73240792047668192
31	24119137810	65	78021633313000432
32	46398487414	66	80531612307490760
33	88792359276		

would therefore take $2^{L_x L_y}$ computations. On the Connection Machine, the computations are done for all S values at once. Hence, theoretically, everything else being equal (CPU speed, I/O, code performance, etc.), these calculations done on the Connection Machine would be faster compared to a scalar or vector computer by a factor equal to the number of available processors.

The possibility of configuring the processors in different dimensional geometries on the CM is also a decided advantage for our problem. This

Table II *(continued)*

E	$P(E)$, $L_z = 4$	E	$P(E)$, $L_z = 4$
0	2	47	1563511250675200
1	0	48	2889243636668444
2	0	49	5318811742264096
3	16	50	9752092052725712
4	56	51	17804338901616560
5	112	52	32357689030787028
6	112	53	58521965081542608
7	608	54	105294105546334176
8	1812	55	188397516161014016
9	3712	56	335091352548934300
10	7912	57	592227093744106128
11	18224	58	1039584650010233248
12	47428	59	1811662842032952976
13	105328	60	3132784073535606840
14	224896	61	5372784316131293616
15	498800	62	9133886266743986192
16	1095788	63	15383743389128191600
17	2413936	64	25655108576387177966
18	5132800	65	42338853191864887680
19	10811360	66	69103382878898694864
20	22868084	67	111478350198272356000
21	47883680	68	177641402891077968116
22	99515088	69	279439772203236133472
23	204435264	70	433658872628887404048
24	417729172	71	663512187574211432832
25	850371792	72	1000265594451155033100
26	1717248736	73	1484822655912270464832
27	3447167792	74	2168988596889183774456
28	6880851028	75	3116011911866416289744
29	13662938144	76	4399897459675202646588
30	26998590136	77	6102929083309597563056
31	53072093536	78	8310877267805936919472
32	103848067428	79	11105541705811090017984
33	202310582144	80	14554619025980130404078
34	392441899400	81	18699389865779547850592
35	758232552224	82	23541337015623652260648
36	1459256298208	83	29029433210633185007040
37	2798018995168	84	35050322703050240185260
38	5346036995848	85	41423795004781141644480
39	10179231529008	86	47905667521977206541944
40	19317593515850	87	54199388920730701695680
41	36540450951952	88	59976399877717496811932
42	68896352202848	89	64903733067748664039120
43	129488175256352	90	68675805358111933766200
44	242586293631232	91	71046207657110339443824
45	452984862584256	92	71854845789506804272616
46	843035430751760		

Table II *(continued)*

E	$P(E)$, $L_z = 5$	E	$P(E)$, $L_z = 5$
0	2	32	245570743258
1	0	33	489615658068
2	0	34	971671907300
3	16	35	1919588750576
4	64	36	3775153361772
5	132	37	7392726655148
6	132	38	14417558257408
7	688	39	28006100698820
8	2232	40	54193314806976
9	5012	41	104477937316974
10	10372	42	200702223335232
11	23868	43	384223003158216
12	65304	44	733109783815568
13	143300	45	1394315017278680
14	334972	46	2643666718296136
15	744808	47	4997496654574032
16	1694638	48	9419793266848240
17	3877236	49	17705673587591156
18	8477720	50	33189573318793390
19	18271848	51	62049816560886296
20	39576288	52	115706001027649668
21	85460904	53	215213616866306812
22	182793492	54	39929910564326748
23	385227164	55	739012858391124332
24	807454464	56	1364377978033611848
25	1687115266	57	2512717932778028966
26	3500595644	58	4616038752152925112
27	7211089480	59	8458531174306491880
28	14754704112	60	15459522240603432648
29	30038656980	61	28180012960496541868
30	60849001180	62	51226098210424957576
31	122562123172		

is because the step of getting data from S' to S in Eq. (7a), with S' different from S, in one bit can be done in one move (using NEWS on the CM[4]). This is done by configuring the geometry so that the $2^{L_x L_y}$ processors are on the vertices of an $L_x L_y$-dimensional hypercube with two sites in each dimension. Then S' is always a nearest-neighbor site to S along an axis and so the fetch from S' to S is very fast. Since the CM-2 can be configured as a hypercube of up to 31 dimensions, we were able to use it as a 20-dimensional hypercube for our $5 \times 4 \times 5$ lattice study. An important reason to use PARIS (for this particular problem) is that the

Table II *(continued)*

E	$P(E)$, $L_z = 5$	E	$P(E)$, $L_z = 5$
63	92853847640001354572	91	20338536703071839631648634 2
64	167808256532951181688	92	30439268061606847964638623 2
65	302322596606763325072	93	45020448395455483726581753 6
66	542879939360811490858	94	65777755335861724345158282 0
67	971489386562814514132	95	94902265874601076073397968 4
68	1732175624094562158696	96	13515740626910986292827067 40
69	3076645273437637029120	97	18993702494337428774861116 52
70	5442523949092898979060	98	26328773724620082295896526 80
71	9586507131640603631504	99	35987529698483852953924936 52
72	16809369419208639572836	100	48487304603990759955761342 92
73	2933344843218566930864	101	64375166331056592697205358 52
74	5092999084206787684901 2	102	84195463688976153219936623 80
75	8795587992481841925503 0	103	10844539650197267452578705 620
76	1510454950678029510081 92	104	13751957368810714602087124 348
77	2578515674174655877250 8	105	17164646394529323578290086 648
78	4374351676213142277870 24	106	21082180428038954999877558 464
79	7372213736422658287042 816	107	25474608682198235910342973 950
80	1233890523976485963942 596	108	30277477626405491950663227 908
81	2050218032815681882539 288	109	35389047528601978553123103 984
82	3380762924709505076582 434	110	40670547790868750567499276 652
83	5530492007367577105010 424	111	45950082889159072732537729 624
84	8971958395886235202798 363	112	51030420355704594119025295 656
85	1442854225609227696486 6248	113	55700401302105534320187883 732
86	2299345786003469135671 2956	114	59749179756651897941132252 092
87	3629657591333875297151 9676	115	62982007278965463962653620 400
88	5673328216568425746183 9120	116	65235926198099998753008434 080
89	8777104676551969132967 4804	117	66393595876006711062805752 772
90	1343492895647117239621 26008		

word length can be made larger than 32 bits. As discussed above, I^o and I^n must have bit accuracy greater than 32 for large V. This is possible in a straightforward way with PARIS, whereas for a higher-level language like FORTRAN, one would have to do the many-bit precision arithmetic in software.

In addition to PARIS, we also programmed the problem in CM-FORTRAN for the Connection Machine. The interesting parts of the FORTRAN code are given in Table I. The CM-FORTRAN code for our problem is given in Table I. First, the size declarations are shown and after that the array dimensions that are to be spread across the processors are explicitly defined as *news* dimensions by the CMF$LAYOUT command. In the complete program, there are several lines of code after that (about 65) that do initialization. These are not shown in Table I. Instead, only the

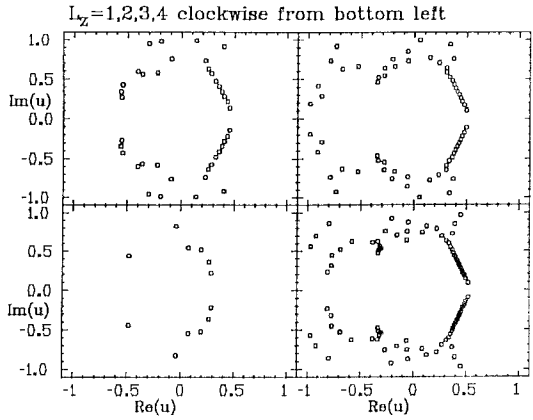

Fig. 1. The zeros of the 3D Ising model for $5 \times 4 \times L_z$ lattices for $L_z = 1, 2, 3, 4$.

part that implements Eq. (7), which is the core part of the code, is shown. Note the neat and simple format of the CM-FORTRAN parallel instructions. For instance, the command *help*(:) = ⋯ on line 42 will execute in parallel for all the processors. The compiler recognizes from the syntax of the statement that the statement is a CM-FORTRAN statement and hence should be executed on the Connection Machine. Also, it recognizes that *help* is an array that is defined across the Connection Machine processors and allocates memory accordingly. The comments in Table I are meant to explain the flow of the logic of the code.

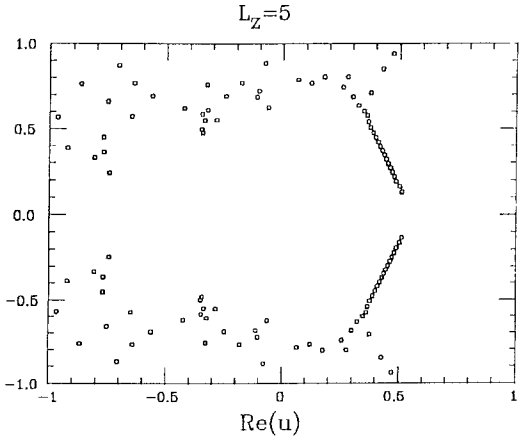

Fig. 2. The zeros of the 3D Ising model for a $5 \times 4 \times 5$ lattice.

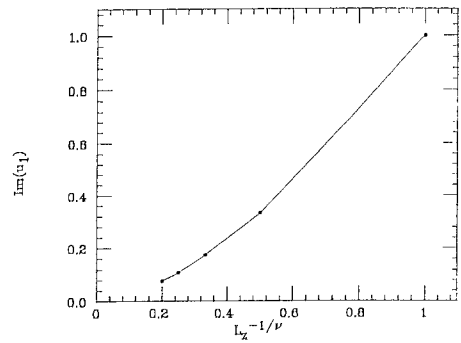

Fig. 3. The imaginary part of the zero closest to the Re(u) axis as a function of $L_z^{-1/\nu}$. We have used $\nu = 0.6295$.[6]

For this particular problem, FORTRAN is not the language of choice. The reason is that the word length in FORTRAN is fixed (to 32 bits in our case). This is an inherent limitation of FORTRAN and although one *can* do higher-accuracy arithmetic in FORTRAN, it must be done in software. The FORTRAN code of Table I can only handle situations where V is less than 33.

4. THE RESULTS

The partition functions for $5 \times 4 \times L$ for $L \in [1, 5]$ are given in Table II. We have checked the numbers in Tabled II by generating data for $Q(k)$ at values of c other than those used to generate Table II and checking that Eq. (4) is satisfied. The zero[5,6] of the partition function in the complex u plane obtained from the data are shown in Figs. 1 and 2. Note the accumulation of these zeros toward the real u axis. In Fig. 3, we plot

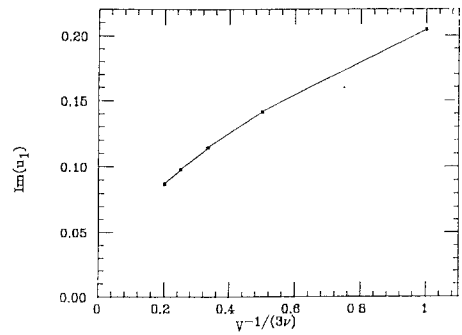

Fig. 4. The imaginary part of the zero closest to the Re(u) axis as a function of $V^{-1/(3\nu)}$, where $V = L_x L_y L_z$ is the volume.

the imaginary part of the zero closest to the real u axis as a function of $L_z^{-1/\nu}$ and in Fig. 4 as a function of $V^{-1/(3\nu)}$,(1,3,5,6) using $\nu = 0.6295(10)$.[6] If L_x and L_y were infinite, the scaling law that this quantity would satisfy is[7]

$$\mathrm{Im}(u_1(L)) \sim L_z^{-1/\nu} \tag{11}$$

It is clear that because of the finiteness of L_x and L_y, finite-size effects in our case are more subtle. We will present a detailed discussion on this issue as well as on other matters (such as estimating ν and β_c from our data) in a later publication.

It would also be interesting to extend these calculations to a $5 \times 5 \times 5$ lattice. However, this would require a minimum memory of 1.39×10^{11} bits. One could get this amount of storage on the Thinking Machines' Data Vault. This is currently being pursued.[8]

ACKNOWLEDGMENTS

The research of S.S. was supported by NATO, NSF, and the Office of Naval Research. S.S. thanks Gene Stanley for his encouragement and support. We thank Boston University for the use of their Connection Machine. G.B. thanks Denny Dahl, Roscoe Giles, Kyra Lowther, Jacek Myczkowski, Robert Putnam, and John Richardson for help with the code and many useful discussions and Profs. Claudio Rebbi and Gene Stanley for their generous and continuing hospitality at Boston University. G.B. also acknowledges the support of the Florida State University Supercomputer Computations Research Institute, which is partially funded by the U.S. Department of Energy through contract DE-FC05-85ER250000.

REFERENCES

1. R. Pearson, *Phys. Rev. B* **26**:6285 (1982).
2. K. Binder, *Physica* **62**:508 (1972).
3. G. Bhanot, CERN preprint CERN-TH-5474/89 (19898); *J. Stat. Phys.*, to appear.
4. Thinking Machines Corporation, Parallel Instruction Set Manual, Cambridge, Massachusetts.
5. C. N. Yang and T. D. Lee, *Phys. Rev.* **87**:101, 110 (1952); M. Fisher, in *Lectures in Theoretical Physics*, Vol. 12C (University of Colorado Press, Boulder, Colorado, 1965), p. 1.
6. G. Bhanot, R. Salvador, S. Black, P. Carter, and R. Toral, *Phys. Rev. Lett.* **59**:803 (1987).
7. C. Itzykson, R. Pearson, and J.-B. Zuber, *Nucl. Phys. B* **220**:415 (1983).
8. G. Bhanot, J. Richardson, and S. Sastry, work in progress.