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

Monte Carlo Simulation of Ising Models by Multispin Coding on a Vector Computer

Stephan Wansleben,[1] John G. Zabolitzky,[1] and Claus Kalle[1]

Received December 7, 1983; revision received May 15, 1984

Rebbi's efficient multispin coding algorithm for Ising models is combined with the use of the vector computer CDC Cyber 205. A speed of 21.2 million updates per second is reached. This is comparable to that obtained by special-purpose computers.

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

1. INTRODUCTION

Monte Carlo simulation of the Ising model has been improved by various techniques during the last few years. The most efficient methods are the multispin coding technique for general-purpose computers[(1,2)] (such as the IBM 370/168 or CDC Cyber 176), the use of special-purpose computers,[(3)] and array processors.[(4)] The multispin coding technique is based on the bit logical operations of a general-purpose computer. Monte Carlo simulation of the three-dimensional Ising model using this technique has been performed with a speed of up to 1.6 million updates per second on a CDC Cyber 176.[(2,5)] Special-purpose machines realize the algorithm by an appropriate hardware structure. Their speed is up to 25 million updates per second, which is 16 times the speed of multispin coding on a scalar computer. Finally, the array processor is a set of parallely working microprocessors. These processors can simultaneously work on and store different parts of the lattice due to the locality of the Monte Carlo algorithm. Speeds up to 9.5 million updates per second are reached when applying this method.[(4)]

[1] Institut für Theoretische Physik der Universität zu Köln, Zülpicher Str. 77, 5000 Köln 41, West Germany.

822/37/3-4-1

In view of the speed reached by special-purpose machines, it is an exciting question whether it is possible to reach comparable speedups by using a faster general-purpose machine such as the vector computer CDC Cyber 205. A speedup factor of 13 above the CDC Cyber 176 program[2] and an absolute speed of 21.2 million updates per second on a two-pipe 500K CDC Cyber 205 of the state of Nordrhein–Westfalen located at Bochum University, West Germany, were reached using the multispin coding technique.

2. MULTISPIN CODING ON THE SCALAR COMPUTER CDC CYBER 176

The multispin coding technique is explained in detail in the literature.[1,2] A modified version of the standard program is given here to facilitate understanding of the vector algorithm presented later. This program runs on a CDC Cyber 176 with 60-bit words.

The configuration (up–down encoded by 1–0) of 20 spins is stored in one computer word, using three bits per spin. This allows the addition of the values of the logical difference (XOR) to all six neighbors for each of these spins simultaneously while calculating the interaction energy. This precludes that next neighbors are stored in different words. Thus the minimum system size of a cubic lattice is 40^3, where each row (for convenience, in 1-direction) is represented by two computer words. In this way, the lattice is divided into two sublattices (ISODD, ISEVEN) each containing the odd, resp. even, lattice sites within all rows.

The main parts of this program are quoted in Fig. 1. Helical boundary conditions[6] are employed in 2-direction. With these boundary conditions the left-most spins of each 1–2 plane are coupled to the rightmost spins of the plane above (for convenience, left means 2-direction, above 3-direction, and backwards 1-direction). So elements of the arrays ISODD and ISEVEN can be treated consecutively, without any conditionally executed code. Periodic boundary conditions in 1-direction are an effect of the circular shift performed by the function SHIFT. Periodic boundary conditions in 3-direction are achieved by placing copies of the first and the last plane of the lattice above and below the real lattice.[7] These copies are not treated in the course of the Monte Carlo procedure but are updated after a complete sweep through the lattice.

3. MULTISPIN CODING ON THE CDC CYBER 205

A vector computer performs operations on a given set of data, termed a "vector," in an assembly-line fashion. The total execution time for a vector

```
      INTEGER COUNT
C      COUNT IS AN INTEGER FUNCTION TO COUNT THE BITS SET IN A
C      COMPUTER WORD SUPPLIED BY THE FORTRAN IV LIBRARY.
      DIMENSION IEX(7)
C      IEX CONTAINS THE FLIP PROBALITY IN AN UNNORMALIZED FLOATING
C      FORMAT, WHERE THE EXPONENT IS FORCED TO -47. THIS IS NEEDED
C      TO USE ONLY ONE MULTIPLY INSTRUCTION TO SUPPLY A RANDOM NUMBER
C      WHICH HAS ALWAYS THE EXPONENT -47 AND 48 BITS OF MANTISSA
C      SIGNIFICANCE.
             .
             .
             .
C      SET SYSTEM SIZE
      L=40
C      SOME USEFUL CONSTANTS
      LP1=L+1
      LSQ=L*L
      LSQPL=LSQ+L
             .
             .
             .
C      TREATMENT OF THE ODD SPINS
C      COMPUTE NUMBER OF ANTIPARALEL NEIGHBOURS
      DO 1 K=LP1,LSQPL
      IODD=ISODD(K)
      IEVEN=ISEVEN(K)
      IE=XOR(IODD,IEVEN)+XOR(IODD,SHIFT(IEVEN,57))
     &  +XOR(IODD,ISODD(K-1))+XOR(IODD,ISODD(K+1))
     &  +XOR(IODD,ISODD(K-L))+XOR(IODD,ISODD(K+L))
C
C      PREPARE LOOP OVER 20 SPINS IN ONE WORD
C      ICH    - FLIP DECISION ACCUMULATOR
C      KE     - MASK FOR EXTRACTING ONE SPINS ENERGY
C      KES    - SHIFTCOUNT TO RIGHT JUSTIFY MASKED ENERGY VALUE
C      KS     - SHIFTCOUNT TO MOVE SIGN BIT TO DESIRED POSITION
C      KSIGNM - MASK TO EXTRACT A NUMBERS SIGN IN 60 BIT OCTAL
C               REPRESENTATION
      ICH=0
      KE=7
      KES=0
      KS=1
      KSIGNM=40000000000000000000B
      DO 2 II=1,20
      ISCR=AND(IE,KE)
      ISCR=SHIFT(ISCR,KES)
      ISCR=IEX(ISCR)
      IRAND=IRAND*MULT
      ID1=IRAND-IEX(INDEX+1)
      ID1=AND(ID1,KSIGNM)
      ID1=SHIFT(ID1,KS)
      ICH=OR(ICH,ID1)
      KE=SHIFT(KE,3)
      KES=KES-3
      KS=KS+3
 2    CONTINUE
      ISODD(K)=XOR(IODD,ICH)
 1    CONTINUE
             .
             .
             .


             .
             .
             .
C      TREATMENT OF THE ODD SPINS
             .
             .
             .
C      CALCULATION OF THE MAGNETIZATION
      M=0
      DO 3 I3=LP1,LSQPL
      M=M+COUNT(ISODD(K))+COUNT(ISEVEN(K))
 3    CONTINUE
```

Fig. 1. Central parts of the modified scalar multispin coding program which the vectorization is based on. The lattice size is fixed at 40^3. AND, OR, and XOR are intrinsic functions supplied by CDC FORTRAN. They perform the specified boolean operation on their arguments. The intrinsic function SHIFT shifts a word left circular by the number of bits specified in the second argument.

instruction is composed of a fixed amount, called startup time, and a time proportional to the number of data elements or the vector length. For efficient algorithms, the startup time is comparatively small to the instruction execution time. The longer the stream of data the more efficient is the use of the vector feature.

The algorithm of a vector computer is similar to the scalar algorithm in which the elements of each row in 1-direction are scattered into two computer words (even/odd). In the vector algorithm, the whole lattice is divided into two vectors (ISRED and ISBLCK) consisting of multispin words. In the scalar algorithm, each word contains nonneighbored spins, whereas in the vector algorithm, each vector must contain nonneighbored spins because the entity treated by the machine is no longer a word but a vector.

The multispin coding technique relies on unsigned integer arithmetic instructions. These instructions use the 48 right-most bits of a 64 bit word on a CDC Cyber 205. Thus one word can accomodate only 16 spins each, using three bits, while the 16 left-most bits are always zero. The programming of boundary conditions in 1-direction can be carried out as a single shift operation by changing them from periodic to fixed: the backward neighbor of the most backward and the forward neighbor of the most forward spin in each row are fixed at zero, resulting from a shift of the 16 left-most bits of a machine word. The boundary conditions in the other directions do not present any difficulty.

The processing of the neighbors in 1-direction is now discussed. When calculating the logical difference between a spin and its next neighbors in 1-direction, the latter must occupy the same bit position as the inspected spin. Since one neighbor is already in the correct bit position, the word containing the other neighbor must be shifted by three bit positions. The shift direction alternates its sign when passing a 1–2-plane boundary due to the sublattice structure. The correct shift count for every word of a sublattice vector is computed by the program into the vectors LOSC and NOSC (lines 27 to 32 in Fig. 2).

4. SPECIAL LANGUAGE ELEMENTS OF CDC CYBER 205 FORTRAN

To assist the understanding of the given program (Fig. 2), some introduction to the CDC Cyber 205 FORTRAN "dialect" is useful here. This is only an overview. A more detailed description can be found in the appropriate reference manuals.[8,9]

```
      PROGRAM ISING (OUTPUT,TAPE6=OUTPUT)
C PUT ALL DATA ON LARGE PAGES SO THAT ALL PAGES WILL
C FIT ASSOCIATIVE REGISTERS AND NO PAGE FAULTS OCCOUR
      COMMON /LP/
C ARRAYS FOR THE REGISTER SWAP INSTRUCTION
       IRSV(64), IEXL(64),
C THE TWO SUBLATTICE ARRAYS
       ISRED(1088), ISBLCK(1088),
C ARRAY HOLDING RANDOM NUMBERS
       ICDC(1274),
C ARRAY HOLDING ENERGY VALUES IN MULTISPIN CODING
       IE(1024),
C ARRAY USED TO ACCUMULATE FLIP DECISIONS
       ICH(1024),
C ARRAYS HOLDING SHIFT COUNTS (BOUNDARY CONDITION IN 1-DIRECTION)
       LOSC(1024), NOSC(1024),
C ARRAY USED FOR SCRATCH
       ISCR(1024),
C ARRAY AND INTEGER EQUIVALENT FOR BOLTZMANN PROBABILTIES
       EX(7), IEX(7)
C BIT ARRAYS EQUIVALENCED TO THE SUBLATTICES AND DESCRIPTOR NAMES
      BIT BRED1(34816),BRED2(34816),BBLCK1(34816),BBLCK2(34816),
         BRED1D,BRED2D,BBLCK1D,BBLCK2D
C DEFINE DESCRIPTOR NAMES
      DESCRIPTOR ISREDD, ISBLCKD, IED, ICDCD, ISCRD,
                ISOD, ISUD, ISRD, ISLD, LOSCD, NOSCD,
                BRED1D, BRED2D, BBLCK1D, BBLCK2D
      LOGICAL LD2
C EQUVALENCE BIT ARRAYS TO SUBLATTICES (USED TO COMPUTE MAGNETIZATION)
      EQUIVALENCE(BRED1(1),ISRED(1)),(BRED2(1),ISRED(545)),
                (BBLCK1(1),ISBLCK(1)),(BBLCK2(1),ISBLCK(545))
C
C PRESETS FOR LATTICE AND BOLTZMANN FACTORS
      DATA ISRED/1088*0/, ISBLCK/1088*0/, EX/7*.9999999999/
C
C SET TEMPERATURE
      T = .9/.221655
C SET SYSTEM SIZE AND RELATED CONSTANTS
      L = 32
      LP1 = L + 1
      LP1P1 = LP1 + 1
      LPL = L + L
      LSQ = L*L
      LSQPL = LSQ + L
      LSQP1 = LSQ + 1
      LCUBE = L*L*L
      DEN = 1./LCUBE
      LPLP1 = LPL + 1
      LSQPLP1 = LSQPL + 1
      K16 = 16
      K3 = 3
      K7 = 7
      KM47 = -47
C  INITIALIZE RANDOM NUMBER GENERATOR WITH SEED ICDCO
      ICDCO=0
      CALL RANINIT(ICDC,ICDCO)
C  PREPARE FOR SHIFTS
      LD2 = .FALSE.
      DO 99 I = 1,LSQ
      IF(I.NE.L*(I/L)) LD2 = .NOT.LD2
      LOSC(I) = -3
      IF(LD2) GOTO 99
      LOSC(I) = 3
99    NOSC(I) = - LOSC(I)
```

Fig. 2. Complete listing of the multispin coding program for a 32^3-lattice on a CDC Cyber 205. Special language elements of Cyber 200 FORTRAN are explained in the text.

```
C SET NONTRIVIAL BOLTZMANN FACTORS
      EX(1) = EXP(-12./T)
      EX(2) = EXP(-8./T)
      EX(3) = EXP(-4./T)
C NORMALIZE BOLTZMANN FACTORS TO (1,2**23-1) INTERVAL INTO ARRAY IEX
      DO 1 IND = 1,7
      I=(2.**47)*EX(IND)
 1    IEX(IND)=SHIFT(I,-24)
C  SETUP LOOKUP TABLE (IEXL) FOR VXTOV
      DO 101 II=1,7
      DO 101 I=1,7
      IEXL((II-1)*8+I)=OR(SHIFT(IEX(II),32),IEX(I))
 101  CONTINUE
C ASSIGN CONSTANT DESCRIPTORS TO CORRESPONDING VECTORS
      ASSIGN ISREDD,  ISRED(LP1;LSQ)
      ASSIGN ISBLCKD, ISBLCK(LP1;LSQ)
      ASSIGN IED,     IE(1;LSQ)
      ASSIGN LOSCD,   LOSC(1;LSQ)
      ASSIGN NOSCD,   NOSC(1;LSQ)
      ASSIGN ISCRD,   ISCR(1;LSQ)
C
C SWEEPS THROUGH LATTICE
C TOP OF LOOP FOR MONTE CARLO STEPS
      DO 6 ITIME = 1,30
      CALL SECOND(TO)
C TREATMENT OF THE RED-SPINS
C 1. ASSIGN LEFT - RIGHT - UPPER - LOWER NEIGHBOURS
      ASSIGN ISLD, ISBLCK(L;LSQ)
      ASSIGN ISRD, ISBLCK(LP1P1;LSQ)
      ASSIGN ISOD, ISBLCK(LPLP1;LSQ)
      ASSIGN ISUD, ISBLCK(1;LSQ)
C 2. COMPUTE NUMBER OF ANTIPARALLEL NEIGHBOURS
      CALL Q8XORV(O,,ISREDD,,ISBLCKD,,IED)
      CALL Q8XORV(O,,ISREDD,,ISLD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8XORV(O,,ISREDD,,ISRD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8XORV(O,,ISREDD,,ISOD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8XORV(O,,ISREDD,,ISUD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8SHIFTV(O,,ISBLCKD,,LOSCD,,ISCRD)
      CALL Q8XORV(O,,ISREDD,,ISCRD,,ISCRD)
      IED = IED + ISCRD
C 3. ATTEMPT TO FLIP THE RED SPINS
      CALL ISFLIP(IE,ISCR,ISRED(LP1),ICH,ICDC,IEXL,IRSV)
C TREATMENT OF THE BLACK-SPINS
      ASSIGN ISLD, ISRED(L;LSQ)
      ASSIGN ISRD, ISRED(LP1P1;LSQ)
      ASSIGN ISOD, ISRED(LPLP1;LSQ)
      ASSIGN ISUD, ISRED(1;LSQ)
      CALL Q8XORV(O,,ISBLCKD,,ISREDD,,IED)
      CALL Q8XORV(O,,ISBLCKD,,ISRD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8XORV(O,,ISBLCKD,,ISLD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8XORV(O,,ISBLCKD,,ISOD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8XORV(O,,ISBLCKD,,ISUD,,ISCRD)
      IED = IED + ISCRD
      CALL Q8SHIFTV(O,,ISREDD,,NOSCD,,ISCRD)
      CALL Q8XORV(O,,ISBLCKD,,ISCRD,,ISCRD)
      IED = IED + ISCRD
      CALL ISFLIP(IE,ISCR,ISBLCK(LP1),ICH,ICDC,IEXL,IRSV)
```

Fig. 2 (*continued*)

```
C TAKE CARE OF PERIODIC BOUNDARY CONDITIONS
      ISBLCK(LSQPLP1;L) = ISBLCK(LP1;L)
      ISBLCK(1;L) = ISBLCK(LSQP1;L)
      ISRED(LSQPLP1;L) = ISRED(LP1;L)
      ISRED(1;L) = ISRED(LSQP1;L)
C COMPUTE CPU TIME USED
      CALL SECOND(T1)
      TTOT = T1 - TO
      TPS = TTOT/(L*L*L)
      FPS = 1.0E-6*L*L*L/TTOT
      WRITE(6,5) ITIME,TTOT,TPS,FPS
 5    FORMAT(I20,F20.6,F20.12,F20.6)
C BOTTOM OF LOOP FOR A MONTE STEP
 6    CONTINUE
C COMPUTE MAGNETIZATION USING VECTORIZED COUNT-COMMAND
      ASSIGN BRED1D,BRED1(2049;32768)
      ASSIGN BRED2D,BRED2(1;32768)
      ASSIGN BBLCK1D,BBLCK1(2049;32768)
      ASSIGN BBLCK2D,BBLCK2(1;32768)
      M = Q8SCNT(BRED1D)
      M = M + Q8SCNT(BRED2D)
      M = M + Q8SCNT(BBLCK1D)
      M = M + Q8SCNT(BBLCK2D)
      SM = (2*M - LCUBE)*DEN
C PRINT RESULT
      WRITE(6,7) SM
 7    FORMAT(/F9.6//)
      STOP
      END
      SUBROUTINE RANINIT(ICDC,ICDCO)
C SETUP RANDOM NUMBER SEED FOR SHIFT REGISTER RANDOM NUMBER GENERATOR
C INITIALIZE FIRST 250 WORDS OF ARRAY ICDC WITH RANDOM BITS
      DIMENSION ICDC(1274)
C
C SET SEED FOR CDC-SUPPLIED RANDOM NUMBER GENERATOR RANF
      CALL RANSET(ICDCO)
C LOOP OVER WORDS
      DO 200 IW=1,250
C ZERO ACCUMULATOR
      IC=0
C LOOP OVER HALFWORDS
      DO 100 IHW=1,2
C ACCOUNT FOR HALFWORD EXPONENT AND MANTISSA SIGN BIT
      IC=SHIFT(IC,9)
C LOOP OVER BITS IN A HALFWORD MANTISSA
      DO 100 IB=1,23
      IC=SHIFT(IC,1)
C EACH BIT IS SET USING A RANF DECISION
      IF(RANF(X).GE.0.5) IC=OR(IC,1)
 100  CONTINUE
C STORE A RANDOM SEED WORD
      ICDC(IW)=IC
 200  CONTINUE
      RETURN
      END
      SUBROUTINE ISFLIP(IE,ISCR,IS,ICH,ICDC,IEXL,IRSV)
C THIS ROUTINE DOES THE FLIP DECISIONS USING THE MONTE CARLO METHOD
C VARIABLE NAMES ARE THE SAME AS IN PROGRAM ISING
C
C DEFINE INTEGER NAMES FOR DESCRIPTORS USED FOR ARRAY ICDC
      INTEGER AD,BD,CD,SEED
C ARRAYS HAVE THE SAME DIMENSIONS AS IN PROGRAM ISING
      DIMENSION IE(1024), ISCR(1024), IS(1024), ICH(1024), ICDC(1274)
      DIMENSION IEXL(64), IRSV(64)
```

Fig. 2 (*continued*)

```
C DEFINE DESCRIPTOR VARIABLES
      DESCRIPTOR IED, ISCRD, ISD, ICHD, ICDCD, AD,BD,CD,SEED, IEXD
      DESCRIPTOR IRSD
      DESCRIPTOR ICDCDH, ISCRDH
C DEFINE TWO DATA CONSTANTS
C  KONE IS A BIT MASK OF 001 REPEATED 16 TIMES, RIGHT JUSTIFIED IN HEX
C  NOTATION. KM29 IS A CONSTANT TO SHIFT RIGHT CIRCULAR BY 29 PLACES
      DATA KONE/X'0000249249249249'/, KM29/35/
C ASSIGN CONSTANT DESCRIPTORS
      ASSIGN IED,    IE(1;1024)
      ASSIGN ISCRD,  ISCR(1;1024)
      ASSIGN ISCRDH, ISCR(1;2048)
      ASSIGN ISD,    IS(1;1024)
      ASSIGN ICHD,   ICH(1;1024)
      ASSIGN ICDCD,  ICDC(251;1024)
      ASSIGN ICDCDH, ICDC(251;2048)
      ASSIGN AD,     ICDC(1;1024)
      ASSIGN BD,     ICDC(148;1024)
      ASSIGN CD,     ICDC(1025;250)
      ASSIGN SEED,   ICDC(1;250)
      ASSIGN IEXD,   IEXL(1;64)
      ASSIGN IRSD,   IRSV(1;64)
C DEFINE REGISTER NUMBER OF REGISTER SWAP (80 HEX)
      IREG=128
C DEFINE REGISTER BIT OFFSET USED BY THE VXTOV INSTRUCTION
      IREGB=IREG*64
C MOVE FLIP PROBABILITY LOOKUP TABLE TO REGISTER FILE FOR FAST ACCESS
C AT THE SAME TIME, THE OLD REGISTER CONTENTS ARE SAVED INTO ARRAY IRSV
      CALL Q8SWAP(IEXD,IREG,IRSD)
C CLEAR ARRAY RECEIVING FLIP DECISIONS
      ICHD=0
C SETUP A MASK FOR 2 SPINS (6 BIT)
      KE=63
C SETUP SHIFT COUNT TO RIGHT-JUSTIFY AN EXTRACTED ENERGY VALUE
      KES=0
C SETUP SHIFT COUNT TO POSITION RESULT OF SUBNV TO CORRECT BIT POSITION
      KS=-23
C ENTER HALFWORD REGISTER 10 (A HEX) WITH THE MANTISSA SIGN BIT CONSTANT
      CALL Q8EXH(10,X'800000')
C LOOP 8 TIMES TREATING 2 SPINS PER TRIP
      DO 3 II=1,8
C EXTRACT ENERGY (ANDV) AND RIGHT-JUSTIFY IT (SHIFTV)
      CALL Q8LINKV(X'10')
      CALL Q8ANDV(X'09',,IED,,KE,,ISCRD)
      CALL Q8SHIFTV(X'08',,ISCRD,,KES,,ISCRD)
C GET FLIP PROBABILITIES
      CALL Q8VXTOV(X'01',,ISCRD,,IREGB,,ISCRD)
C COMPUTE NEW SET OF RANDOM NUMBERS
      CALL Q8XORV(0,,AD,,BD,,ICDCD)
      CALL Q8VTOV(0,,CD,,,,SEED)
C SUBTRACT FLIP PROBABLITIES FROM RANDOM NUMBERS (SUBNV) AND EXTRACT
C SIGN BIT (ANDV). THIS IS DONE USING HALFWORD INSTRUCTIONS.
      CALL Q8LINKV(X'10')
      CALL Q8SUBNV(X'80',,ICDCDH,,ISCRDH,,ISCRDH)
      CALL Q8ANDV(X'89',,ISCRDH,,10,,ISCRDH)
C ADJUST POSITION OF SIGN BIT (SHIFTV) AND SAVE IT INTO ARRAY ICH (XORV)
      CALL Q8LINKV(X'10')
      CALL Q8SHIFTV(X'08',,ISCRD,,KS,,ISCRD)
      CALL Q8XORV(0,,ISCRD,,ICHD,,ICHD)
C UPDATE MASK AND SHIFT VARIABLES
      CALL Q8SHIFTI(KE,6,KE)
      KES=KES-6
      KS=KS+6
C BOTTOM OF LOOP 3
 3    CONTINUE
```

Fig. 2 (*continued*)

```
C POSITION THOSE BITS RESULTING FROM UPPER HALFWORDS DURING LOOP 3
      CALL Q8LINKV(X'10')
      CALL Q8SHIFTV(X'08',,ICHD,,KM29,,ISCRD)
      CALL Q8XORV(0,,ISCRD,,ICHD,,ICHD)
C MASK OUT USEFUL BITS ONLY
      CALL Q8ANDV(X'09',,ICHD,,KONE,,ICHD)
C FLIP THOSE SPINS TO BE FLIPPED
      CALL Q8XORV(0,,ICHD,,ISD,,ISD)
C RESTORE REGISTER FILE FROM ARRAY IRSV
      CALL Q8SWAP(IRSD,IREG,)
      RETURN
      END
```

Fig. 2 (*continued*)

4.1. The DESCRIPTOR Statement

A vector is represented by descriptors. A descriptor consists of the bit address of the first element in bits 16–63 and the vector's length in bits 0–15. Bits are counted from left to the right starting with zero. All descriptors have to be declared as such and must be of the same type as the vectors which they are assigned to later on. The DESCRIPTOR statement is a nonexecutable statement, and explicit- or implicit-type declarations accomplish this.

4.2. The Vector ASSIGN Statement

The vector ASSIGN statement assigns a vector to a descriptor variable. A vector in this context means some contigious part of an array defined by the first element and the vector length denoted as VECTOR(IFIRST; LENGTH).

4.3. Coding of Vector Instructions

There are two ways of coding vector instructions. The first is to use descriptors or vectors in the above sense in the usual FORTRAN arithmetic assignment statements. This means that the expression on the right-hand side is evaluated for all vector elements by vector instructions. If a scalar appears in the expression its value is repeated for each vector element.

Not all vector hardware instructions are accessible by standard FORTRAN language elements. The remaining ones have to be coded by usage of special calls, which are in effect machine instructions. A special call for a vector instruction has the form

CALL Q8XXXXV(G-bits,,A,,B,,C)

where A, B, and C denote descriptors or scalar variables. The G-bits represent an 8-bit mask which further defines the operands and the instruction. The vector represented by C is computed using the operation XXXX on the operands A and B, which may be either a scalar or a descriptor as selected by G-bits 3 and 4.

In the presented program the following operations appear:

Q8XORV —a bit-wise exclusive OR,
Q8ANDV —a bit-wise AND,
Q8SHIFTV—a left circular shift A by B,
Q8SUBNV —subtract B from A giving normalized result C,
Q8VTOV —copy A to C,
Q8VXTOV—gather elements directed by vector A from list B to vector C, in effect similar to $C(I) = B(A(I) - 1)$ on a scalar machine,

Q8-calls using other syntax are:

Q8SHIFTI —shift first operand by number (second operand) left circular,
Q8EXH —enter halfword register (first argument) with value (second agument),
Q8SWAP —exchange part of register file to and from main memory,
Q8LINKV —combine the next two vector instructions to one combined instruction, effectively feeding the second instruction first operand with result of the first instruction.

4.4. Further Machine Dependencies

As on most scalar computers, the CDC Cyber 205 has the option of bit-wise logical operations. We use OR, a logical OR of the arguments, and SHIFT, a left circular shift by a positive second argument and a right sign extended, end off shift by a negative second argument. There is also the option to operate on "halfwords." They consist of 32 bits, and two of them can be regarded as one 64-bit word. The operating speed on halfwords is twice that for words. In the given program, vectors consisting of halfwords are represented by descriptors named ending with the letter H.

5. THE INNER-MOST LOOP

The inner-most loop is transfered into subroutine ISFLIP (Fig. 2) for technical reasons. Except for the random number generator code (line 35 and 36), this inner-most loop basically arises from the scalar code described above (Fig. 1) by straightforward vectorization neglecting for the moment halfwords and Q8LINKV instructions.

The loop is executed only eight times rather than 16 times as expected for 16 spins per word. The reason for this is the simultaneous treatment of two spins during one loop trip. In lines 32 and 33 we extract the energy values for two spins at a time using a mask of six bits resulting in an index between zero and 62. This index is used (line 34) to retrieve a word from a list of Boltzmann factors, which at that time is located in the register file for fast access. The list is specially arranged (see main program, lines 33 to 41) such that the left-most part of a word contains the flip probability for the left of the two spins and vice versa. The next two statements produce a random vector ICDC as explained below. Looking at the vectors ICDC and ISCR as halfword vectors having twice the length, the next two lines get clear as they arise from straightforward vectorization. Now the flip decision, decoded from the sign bits of the halfword vector ISCR, is shifted to a correct position and saved into vector ICH. Before the spin flips can be carried out (line 51), some manipulations are needed to adjust the bit positions within the vector ICH (lines 47 to 50).

One of the most important parts of the algorithm is the random number generator. As the program requires 23-bit random numbers with large period, the CDC-supplied function RANF, which generates 47-bit equally distributed numbers cannot be used (and leads to problems[12]). A shift-register sequence random number generator introduced by Tausworthe[10,11] is employed. It can be viewed as 64 parallely working 1-bit random number generators each with a period of 2^{250}. The details of this implementation are of general interest and will be published separately.[12] Since this random number generator produces integers in the interval $[1, 2^{23}-1]$, the Boltzmann factors are normalized to this interval (main program, lines 33 to 38).

In using the Q8SWAP special call, the instruction is valid only if the following conditions are taken care of: (1) the length of the array which is being swapped to or from the register file must be an even number, (2) its first element must have an even word address, and (3) the register number must be an even number too. Usable registers can be found by inspecting the register allocation map generated by the FORTRAN compiler. In our case, those marked FR_nn turned out to be not in use by any FORTRAN-generated code.

6. DISCUSSION

In this paper we present a program which is useful to show basic methods to vectorize the multispin coding algorithm and to check out the power of general-purpose computers compared to existing special-purpose computers. We have shown that the speed of this program (21.2 million updates per second or 47 nsec per update) is comparable to those obtained on existing special-purpose machines. For a specific application, it might be necessary to treat systems of arbitrary size and lattices with periodic boundary conditions. This can be done at the same speed by enlarging the number of sublattices and more intelligent treatment of boundary conditions.[13] Moreover, for larger systems, larger vector lengths can be used to diminish the slackening effect of startup times.

Increasing the speed of this algorithm on a CDC Cyber 205 by further orders of magnitude seems to be impossible. M. Creutz, P. Mitra, and K. J. M. Moriarty, however, have shown that it might be possible when the algorithm is changed.[14] They reach a speed of 24 million updates per second on a CDC Cyber 176 using a microcanonical Monte Carlo procedure.[15] We cannot judge whether this method allows Monte Carlo simulations of specific statistical systems in shorter times compared to the conventional canonical method since real times for simulation are not yet published.

ACKNOWLEDGMENTS

We thank D. Stauffer and W. Welke for helpful discussions, and H. Schaefer for help and support using SWAP/VXTOV instructions.

REFERENCES

1. R. Zorn, H. J. Herrmann, and C. Rebbi, *Comp. Phys. Comm.* **23**:337 (1981).
2. C. Kalle and V. Winkelmann, *J. Stat. Phys.* **28**:639 (1982).
3. R. B. Pearson, J. L. Richardson, and D. Toussaint, *J. Comp. Phys.* **51**:241 (1983).
4. G. S. Pawley, D. J. Wallace, R. J. Swendson, and K. G. Wilson, *Phys. Rev. B* **29**:4030 (1984).
5. D. Stauffer, private communication.
6. W. Selke, private communication.
7. W. Oed, Angewandte Informatik 7/82, 358 (1982).
8. CDC Cyber 200 Fortran Version 3 Reference Manuel.
9. CDC Cyber 205 Computer Hardware Reference Manual.
10. R. C. Tausworthe, *Math. Comp.* **19**:201 (1965).
11. S. Kirkpatrick and E. Stoll, *J. Comp. Phys.* **40**:517 (1981).
12. C. Kalle and S. Wansleben, *Comp. Phys. Comm.* **33** (1984), in press.
13. C. Kalle, Diplom Thesis, Cologne University, unpublished.
14. M. Creutz, M. Mitra, and K. J. M. Moriarty, preprint for *Comp. Phys. Comm.*
15. M. Creutz, *Phys. Rev. Lett.* **50**:411 (1983).