CrossMark

Parallelization and implementation of multi-spin Monte Carlo simulation of 2D square Ising model using MPI and C++

Dariush Hassani[1] · Shahnoosh Rafibakhsh[1] ⓘ

Abstract

In this paper, we present a parallel algorithm for Monte Carlo simulation of the 2D Ising Model to perform efficiently on a cluster computer using MPI. We use C++ programming language to implement the algorithm. In our algorithm, every process creates a sub-lattice and the energy is calculated after each Monte Carlo iteration. Each process communicates with its two neighbor processes during the job, and they exchange the boundary spin variables. Finally, the total energy of lattice is calculated by map-reduce method versus the temperature. We use multi-spin coding technique to reduce the inter-process communications. This algorithm has been designed in a way that an appropriate load-balancing and good scalability exist. It has been executed on the cluster computer of Plasma Physics Research Center which includes 9 nodes and each node consists of two quad-core CPUs. Our results show that this algorithm is more efficient for large lattices and more iterations.

Keywords Ising model · Monte Carlo method · Multi-spin coding · MPI

Introduction

The Ising model [1] gives a microscopic description of the ferromagnetism which is caused by the interaction between spins of the electrons in a crystal. The particles are assumed to be fixed on the sites of the lattice. Spin is considered as a scalar quantity which can achieve two values $+1$ and -1. The model is a simple statistical one which shows the phase transition between high-temperature paramagnetism phase and low-temperature ferromagnetic one at a specific temperature. In fact, the symmetry between up and down is spontaneously broken when the temperature goes below the critical temperature. However, the one-dimensional Ising model, which has been exactly solved, shows no phase transition. The two-dimensional Ising model has been solved analytically with zero [2] and nonzero [3] external field. In spite of a lot of attempts to solve 3D Ising model, one might say that this model has never been solved exactly. All the results for the three-dimensional Ising model have been used approximation approaches and Monte Carlo methods.

Monte Carlo methods or statistical simulation methods are widely used in different fields of science such as physics, chemistry, biology, computational finance and even new fields like econophysics [4–9]. The simulation can proceed by sampling from the Probability Density Function and generating random numbers uniformly. The simulation of the Ising model on big lattices increases the cost of simulation. One way to reduce the simulation cost is to design the algorithms which work faster. Swendsen-Wang and Wolff algorithms [10, 11] and multi-spin coding methods [12–14] are the examples of such methods. Another way is to parallelize and execute the model on GPUs, GPU clusters and cluster computers [15–27].

In this paper, we present a parallel algorithm to simulate the 2D Ising model using Monte Carlo Method. Then, we run the algorithm on a cluster computer using C++ programming language and MPI. Message Passing Interface (MPI) is a useful programming model in HPC systems [28–34] in which the processes communicate through message passing and was designed for distributed memory architectures. MPI provides functionalities which allow two specified processes to exchange data by sending and

✉ Shahnoosh Rafibakhsh
rafibakhsh@srbiau.ac.ir

Dariush Hassani
dariush.hassani@outlook.com

[1] Department of Physics, Science and Research Branch, Islamic Azad University, Tehran, Iran

Springer

receiving messages. To get high efficiency, it is necessary to have good load balancing and also to have minimum communications between processes.

In our algorithm, each individual process creates its own sub-lattice, initializes it, gets all Monte Carlo iterations done and calculates the energy of the sub-lattice for a specific temperature. Each process communicates with its two neighbor processes during the job and they exchange the boundary spin variables. Finally, the total energy of lattice is calculated by map-reduce method. Since in multi-spin coding technique each spin is stored by 3 bits, inter-process communications are reduced considerably. Because computational load of each sub-lattice is assigned to each process and size of all sub-lattices is equal, an appropriate load balancing exists. Since each process—independent of number of processes—only communicates with its two neighbor processes and the lattice is decomposed into sub-lattices, the algorithm benefits a good scalability.

This paper has been organized as follows. In "Metropolis algorithm and Ising model" section, Metropolis algorithm and the Ising model are studied briefly. In "Multi-spin coding method" section, we explain how to use Multi-spin coding method to calculate the interaction energy between a specific spin and its nearest neighbors. We also study the boundary conditions in the memory-word lattice.[1] Details of parallelization of the algorithm are discussed in "Parallelization" section and the method of implementation is given in "Implementation" section. Finally, the results are given in "Results" section.

Metropolis algorithm and Ising model

The Ising model consists of spins variables which take values $+1$ or -1 and are arranged in a one-, two- or three-dimensional lattice. Each spin interacts with its neighbors, and the interaction is given by the Hamiltonian:

$$\mathcal{H} = -J \sum_{\langle m,n \rangle} s_m s_n, \qquad (1)$$

where J is the coupling coefficient. The summation in Eq. (1) is taken over the nearest neighbor pairs $\langle m, n \rangle$. Periodic boundary conditions are used which state that spins on one edge of the lattice are neighbors with the spins on the opposite side. In this paper, we focus on simulation of the 2D square Ising model using Metropolis Monte Carlo algorithm [35]. The lattice is initialized randomly and is updated as the following:

1. Select a spin $(s_{i,j})$ randomly and calculate the interaction energy between this spin and its nearest neighbors (E).
2. Flip the spin $s_{i,j}$ to $s'_{i,j}$ and again calculate the interaction energy (E').
3. $\triangle E = E' - E$, if $\triangle E \leq 0$, $s'_{i,j}$ is accepted. Otherwise, $s'_{i,j}$ is accepted with the probability $e^{-\triangle E/KT}$ where K is Boltzmann constant and T is the temperature.
4. Repeat steps 1–3 till we are sure that every spin has been flipped.
5. Calculate the total energy of the lattice for ith iteration $\left(E^i_{\text{total}}\right)$.

The steps above form a Monte Carlo iteration. We perform enough iterations (N times) and finally average on $\left(E^i_{\text{total}}\right)$ to obtain E_{total}:

$$E_{\text{total}} = \frac{1}{N} \sum_{i=1}^{N} E^i_{\text{total}}. \qquad (2)$$

Multi-spin coding method

Multi-spin coding refers to all techniques that store and process multiple spins in one memory word. In this paper, we apply the multi-spin coding technique to the 2D Ising model. In general, multi-spin coding technique results in a faster algorithm as a consequence of updating multiple spins simultaneously. However, we mainly employ this technique to reduce the inter-process communications.

We apply the multi-spin coding introduced in Ref. [12]. However, in our implementation, the size of a memory word is 64 bits, in contrast to Jacobs's 60-bit memory word. In addition, each spin is retained in three consecutive bits and the value of the 64th bit is always set to zero. 000 represents the spin down and the spin up is shown by 001. Since a memory word contains 21 spins, the size of the lattice is taken to be $21N \times 21N$, where N is an integer greater than one. Now, we need to convert the spin lattice (Fig. 1a) to the lattice of memory words (Fig. 1b). Therefore, the size of the memory-word lattice is considered as $N \times 21N$. Each column of the spin lattice is coded into the same column of the memory word in the memory-word lattice. So, $21N$ spins in one column of the spin lattice are arranged in N memory words of a column in the memory-word lattice as follows:

$$
\begin{aligned}
S(0,J): &\quad s_{0,j}, s_{N,j}, s_{2N,j}, \ldots, s_{19N,j}, s_{20N,j} \\
S(1,J): &\quad s_{1,j}, s_{N+1,j}, s_{2N+1,j}, \ldots, s_{19N+1,j}, s_{20N+1,j} \\
&\quad\vdots \\
S(N-1,J): &\quad s_{N-1,j}, s_{2N-1,j}, s_{3N-1,j}, \ldots, s_{20N-1,j}, s_{21N-1,j},
\end{aligned}
$$

where $S(I, J)$ represents the memory word at the row I and the column J, $0 \leq I \leq N - 1$, $0 \leq J \leq 21N - 1$. $s_{i,j}$ shows

---

[1] A word is a data unit of a defined length used as a piece of data by processors. The number of bits in a word, word size, is an important feature of any processor design.
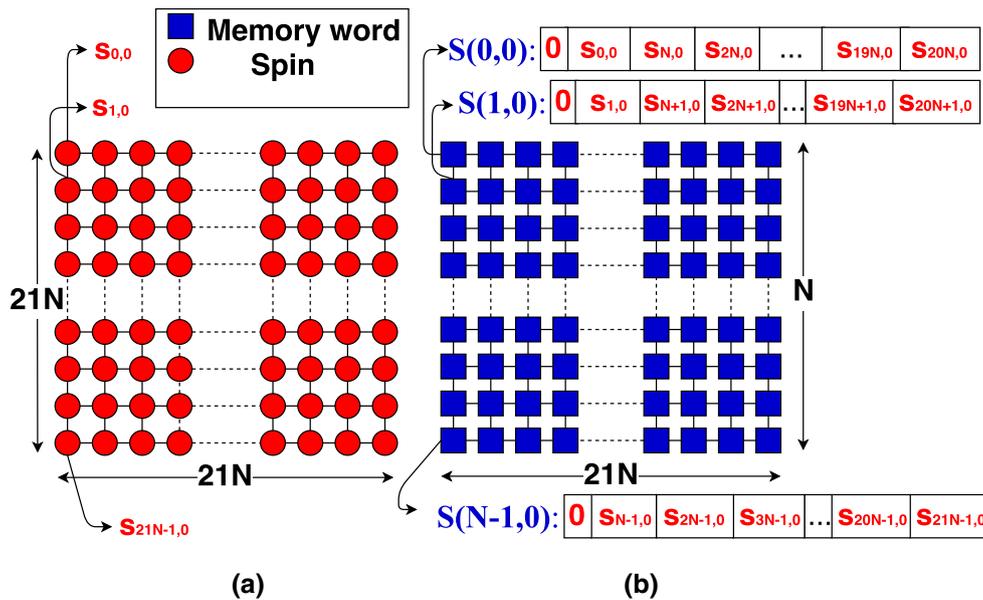
Fig. 1 Arranging spins in memory words. **a** Spin lattice, **b** memory-word lattice

the spin located at the row i and the column j where $j = J$. The advantage of this arrangement is that each spin is placed in the appropriate position related to its neighbors. Consider kth spin in a given memory word $S(I, J)$. The right/left/top/down neighbor of the kth in the spin lattice is exactly kth spin in the right/left/top/down neighbor of the memory word $S(I, J)$ in the memory-word lattice.

In order to apply periodic boundary conditions to the memory words in the first and last row (Fig. 2a), we need to make some changes to up and down neighbors in advance. In fact, the down (up) neighbor of $S(N - 1, J)$ ($S(0, J)$) is not exactly $S(0, J)$ ($S(N - 1, J)$). For a memory word in the first (last) row, its up (down) neighbor—which is the memory word in the last (first) row and in same column—has to be shifted 3 bits to the right (left). These two cases have been shown in the diagrams (b) and (c) of Fig. 2. We should recall that the 64th bit is always set to zero.

Calculation of energy

Now, using multi-spin coding method, we show how to calculate the energy difference (ΔE) between two configurations in the Ising model. At first, to better understand the process, we consider two 3 bit-spins s_1 and s_2. s_1 XOR s_2 produces 000 when the two spins are placed in the same direction and 001 is given when spins s_1 and s_2 are in the opposite directions.[2] Hence, for a given memory word $S(I, J)$, the expression

---

[2] Exclusive OR (XOR) is a logical operator that outputs true if exactly one (but not both) of the two inputs is true.

$$(S(I, J) \text{ XOR } S(I - 1, J)) + (S(I, J) \text{ XOR } S(I + 1, J))$$
$$+ (S(I, J) \text{ XOR } S(I, J - 1)) + (S(I, J) \text{ XOR } S(I, J + 1)),$$
$$(3)$$

generates a value in the range of [0, 4] for every 3bit-group given in the last column of Table 1. In the second and third columns, we have considered different cases that might occur between a selected spin and its four neighbors. The initial interaction energy E and the energy E' calculated after flipping the selected spin have been presented in the forth and fifth rows, respectively.

Parallelization

In a Monte Carlo Metropolis iteration, each memory word is updated at least once. The iterations must be performed enough times to yield accurate outcome energy. The given lattice could be vertically divided into N_p sub-lattices with equal sizes, where N_p is the number of processes. Computational load of each sub-lattice is assigned to the processes 0 to $N_p - 1$ from left to right. Each process creates a sub-lattice of the specific size, initializes the sub-lattice, performs all Monte Carlo iterations and calculates the energy of the sub-lattice using Eq. (2). When all individual processes calculate the energy of their own sub-lattice, the energies of the sub-lattices are added up, through a Map-Reduce operation, to calculate the total energy of the lattice. However, this approach results in two problems. As illustrated in Fig. 3, half of the neighbors of the memory words on the border, are placed in the sub-lattice of neighbor process. Therefore, to calculate the energy of the

Fig. 2 **a** Memory words in the first and last rows of a memory-word lattice, **b** up neighbor of $S(0, J)$ formed by shifting three bits of $S(N − 1, J)$ to the right, **c** down neighbor of $S(N − 1, J)$ formed by shifting three bits of $S(0, J)$ to the left
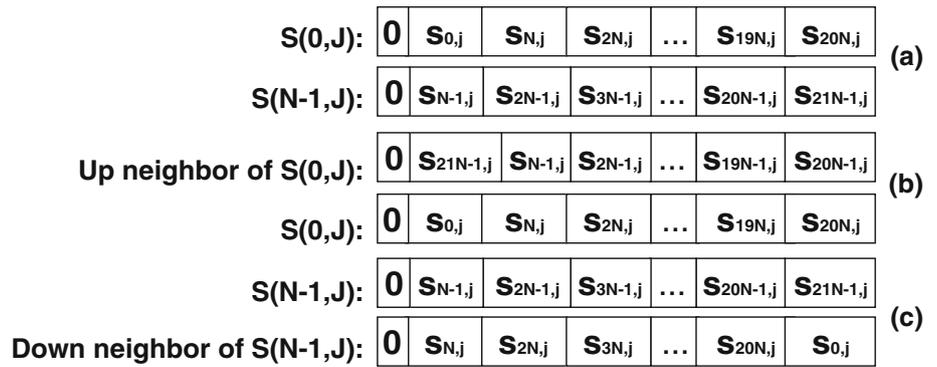
Table 1 Different configurations which might happen between a selected spin and its four nearest neighbors

Configuration	Selected spin	Nearest neighbors	E	E'	ΔE	Value of a 3-bit group
1	Up	4 Up–0 Down	− 4J	4J	8J	000
2	Down	0 Up–4 Down	− 4J	4J		
3	Up	3 Up–1 Down	− 2J	2J	4J	001
4	Down	1 Up–3 Down	− 2J	2J		
5	Up	2 Up–2 Down	0	0	0	010
6	Down	2 Up–2 Down	0	0		
7	Up	1 Up–3 Down	2J	− 2J	− 4J	011
8	Down	3 Up–1 Down	2J	− 2J		
9	Up	0 Up–4 Down	4J	− 4J	− 8J	100
10	Down	4 Up–0 Down	4J	− 4J		

The initial energy E and the energy E' after flipping the selected spin, have been shown in forth and fifth columns, respectively. The energy difference $\Delta E = E' − E$ has been given in the sixth column. the last column represents the value of a 3-bit group
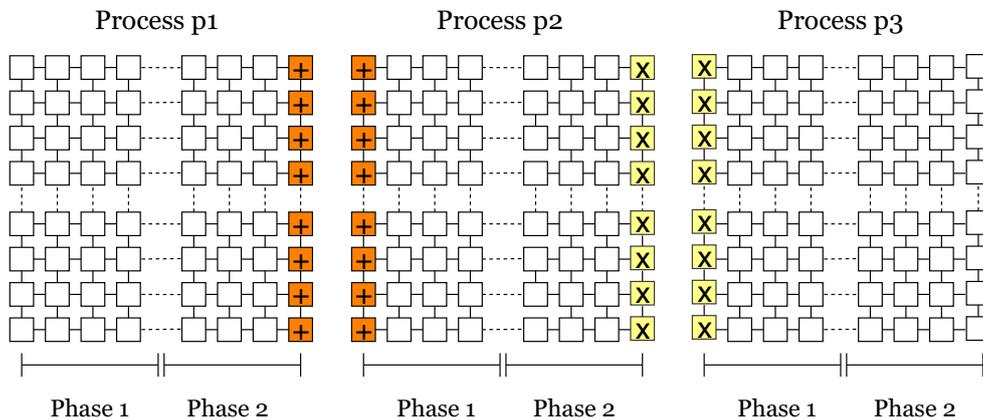

Fig. 3 Sub-lattices of the memory words that belong to the three consecutive processes. The memory words on the border of the two different sub-lattices which have interaction with each other, are marked the same. Each half of the memory-word lattice is updated in the phase 1 or 2. Therefore, the border memory words are not updated simultaneously

memory words on the border, some memory words of the side sub-lattice are needed. Therefore, these memory words have to be observed when needed. Moreover, we should note that neighbor memory words should not be updated simultaneously by different processes.

To deal with the second problem, we propose a method in which the memory words are updated in two phases (see Fig. 3). In each phase, half of the sub-lattice is updated while the other half stays unchanged, i.e., in phase 1 (2) the left (right) half of the sub-lattice is updated. After the phase 1 (2) is done, each process will pass to phase 2 (1) only if

Fig. 4 Function of our implemented program. Left and right arrows denote inter-process communications
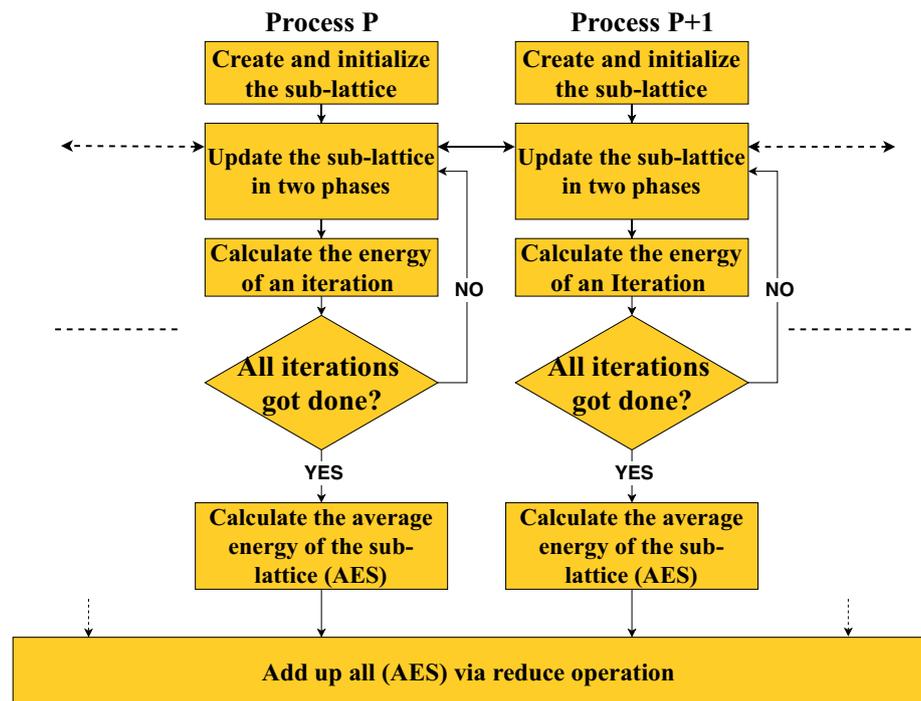
its right (left) process accomplishes the phase 1 (2). To better understand this process, we consider three consecutive processes in Fig. 3 which are updating the left half of their sub-lattices in phase 1. The process p2 has updated the left half of its sub-lattice and is going to start the phase 2 to update the right half. However, it is not able to reach the phase 2 until the process p3 accomplishes the phase 1 and finishes the update of the left half. So, the memory words on the borders of the processes p2 and p3, marked with × in Fig. 3, do not update simultaneously. In the same manner, the memory words on the borders of processes p1 and p2, marked with + in Fig. 3 do not update at the same time.

Now, we turn to the first problem. As mentioned before, in each phase half of a sub-lattice is updated. Before a process starts updating the half of the sub-lattice, it should receive the corresponding border memory words of the neighbor process. Suppose that the process p2 is going to update the left half of its sub-lattice in phase 1. It waits to receive the right-side border memory words of the process p1. The process p1 sends its right border memory words to the process p2 asynchronously just after it accomplishes the phase 2 of the last iteration. After p2 receives the border memory words from p1 synchronously, it starts updating the memory words in the phase 1. Just after finishing the phase 1, p2 sends its updated left-side border memory words to p1 asynchronously and goes to the phase 2. The similar procedure occurs for other processes as well. It should be mentioned that we use periodic boundary conditions thereby the left neighbor of the first process is

the last process, and likewise the right neighbor of the last process is the first process.

Implementation

In this section, we describe the implementation details of the algorithm presented in the previous section. Consider a memory-word lattice of size $N \times 21N$ where N is an arbitrary integer bigger than one. We execute the algorithm on N_p processes and each process is identified by an integer number, 0 to $N_p - 1$, called rank. Each process is responsible for N_c columns of the memory-word lattice where $N_c = \frac{21N}{N_p}$. Each individual process creates its own sub-lattice, initializes it, gets all Monte Carlo iterations done and calculates the energy of the sub-lattice for a specific temperature. Within each Monte Carlo iteration a sub-lattice is updated many times and the energy of iteration is calculated. Finally, the total energy of the memory-word lattice for a specific temperature is obtained via a reduce operation. This operation is illustrated in Fig. 4. Now, every step of the algorithm is studied in details.

Initialization

Now, we explain how a process creates its own sub-lattice and initializes it randomly. We use a 64-bit long integer as a memory word to store 21 spins. A 2D array of $N \times N_c$ long integers forms the sub-lattice of the process where

$N_c = \frac{21N}{N_p}$. However, we use an array with two additional columns $(N \times (N_c + 2))$ reserved for border memory words of the neighbor processes (see Fig. 5a). In this paper, we denote this array by S-lattice. Each spin in the memory words of the s-lattice is initialized with 0 or 1 randomly—except for the first and last columns—which represent spin down and up, respectively.

```
destination = (Rank == 0) ? NP-1: Rank-1;
source= (Rank==NP-1) ? 0:Rank+1;
```

Updating

As mentioned before, updating process is done in two phases. In each phase, one half of the sub-lattice is updated. In cases where N_c is odd, floor($N_c/2$) columns are updated in phase 1 and the rest of the columns is updated in phase 2. Before starting the update process in each phase, some inter-process communication should be carried out.

At first, each process sends its border memory words to its neighbor process asynchronously. Then, it waits to receive the border memory words from its neighbor processes. When the process receives the required border memory words, it can accomplish the phase by frequently updating the memory words that belong to the corresponding phase. In phase 1, in which the left half of the sub-lattice is updated, each process sends the rightmost column of its sub-lattice to its right neighbor (Fig. 5b). So, the destination of the sending memory words is determined by the following code:

```
destination = (Rank == NP-1) ? 0: Rank+1;
```

It means that, due to the periodic boundary conditions, the right neighbor of the process with the rank $N_p - 1$ is the process 0. Likewise, the source process from which the process receives the border memory words is determined by the following code:

```
source= (Rank==0) ? NP-1:Rank-1;
```

which means that the left neighbor of the process with the rank 0 is the process $N_p - 1$.

The received column of memory words is stored in the first column of the S-lattice which has been reserved for the border memory words of the neighbor process. When the border memory words are received, the left half of sub-lattice is updated (Fig. 5c). Likewise, the destination and the source, in phase 2, are determined by the following code (Fig. 5d):

The received column of the memory words is stored in the last column of the S-lattice which has been reserved for the border memory words of the neighbor process (Fig. 5e).

Calculating the Energy of a Monte Carlo Iteration

In order to obtain the total energy of the lattice, the energy of all nearest neighbor pairs must be considered. However, if we consider the interaction energy of the right and down neighbors of each memory word, the total energy is calculated. Each process calculates the energy of each memory word in the S-lattice except for the first and last columns. Notice that the last column of the S-lattice contains the copy of the border memory words of the right process (Fig. 5e). Since these border memory words are not used until they are sent, the copy of them is still valid. This copied column is used as the right neighbor of the last column of the sub-lattice. The code in Listing 1 shows how the interaction energy between a specific memory word and its right and down memory words is calculated: In the line 3, the outcome of the expression on the right side of the assignment operator, is a memory word which includes 21 3bit-groups. Every group contains a number between 0 and 2, i.e., 0 represents $-2J$, 1 represents 0 and 2 represents $+2J$. Each 3bit-group retains the sum of the interaction energy between a specific spin with its right and down

neighbors. The for loop iterates on the 3bit-groups of E. In each iteration, the energy of one 3bit-group is extracted and is added to rv where rv retains the sum of energies of the 3bit-groups. Therefore, rv contains the total energy of the 21 3bit-groups.

nodes is increased, efficiency goes down. Especially when one more node is exploited, the efficiency drops considerably. This fall is due to the fact that overhead of the communication between processes on different nodes is higher than overhead of the communication between processes on the same node.

Listing 1: Computing the energy of a memory word

```
1  inline double computeEnergy(long int memoryWord,long int right,
        long int down)
2  {
3      long int E = (memoryWord ^ right) + (memoryWord ^ down);
4      double rv=0;
5      for (int i = 1; i <= 21; i++)
6      {
7          switch (E & 7)
8          {
9          case 0:
10              rv-=2*J;
11              break;
12          case 2:
13              rv+=2*J;
14              break;
15          }
16          E >>= 3;
17      }
18      return rv;
19  }
```

Results

We have executed our program on a part of the computer cluster of Plasma Physics Research Center which includes 16 nodes networked by a switch. Each node is equipped with two Intel Xeon X5365 CPUs. We have used up to 9 nodes to test our program. Three different cases with different number of iterations have been considered in Table 2.

The measured speedup and efficiency versus the number of cores are illustrated in Figs. 6 and 7, respectively. As shown, for all three test cases, as number of cores and

Now, we are able to inspect the impact of the lattice dimension and the number of Monte Carlo iterations on the performance of our algorithm. Comparing the test cases 2 and 3, it is inferred that bigger lattice sizes get better speedup and efficiency. In addition, the comparison between the test cases 1 and 2, we can claim when the number of Monte Carlo iterations increases, better speedup and efficiency is deduced. Therefore, our algorithm has better performance for bigger lattice sizes and more Monte Carlo iterations.
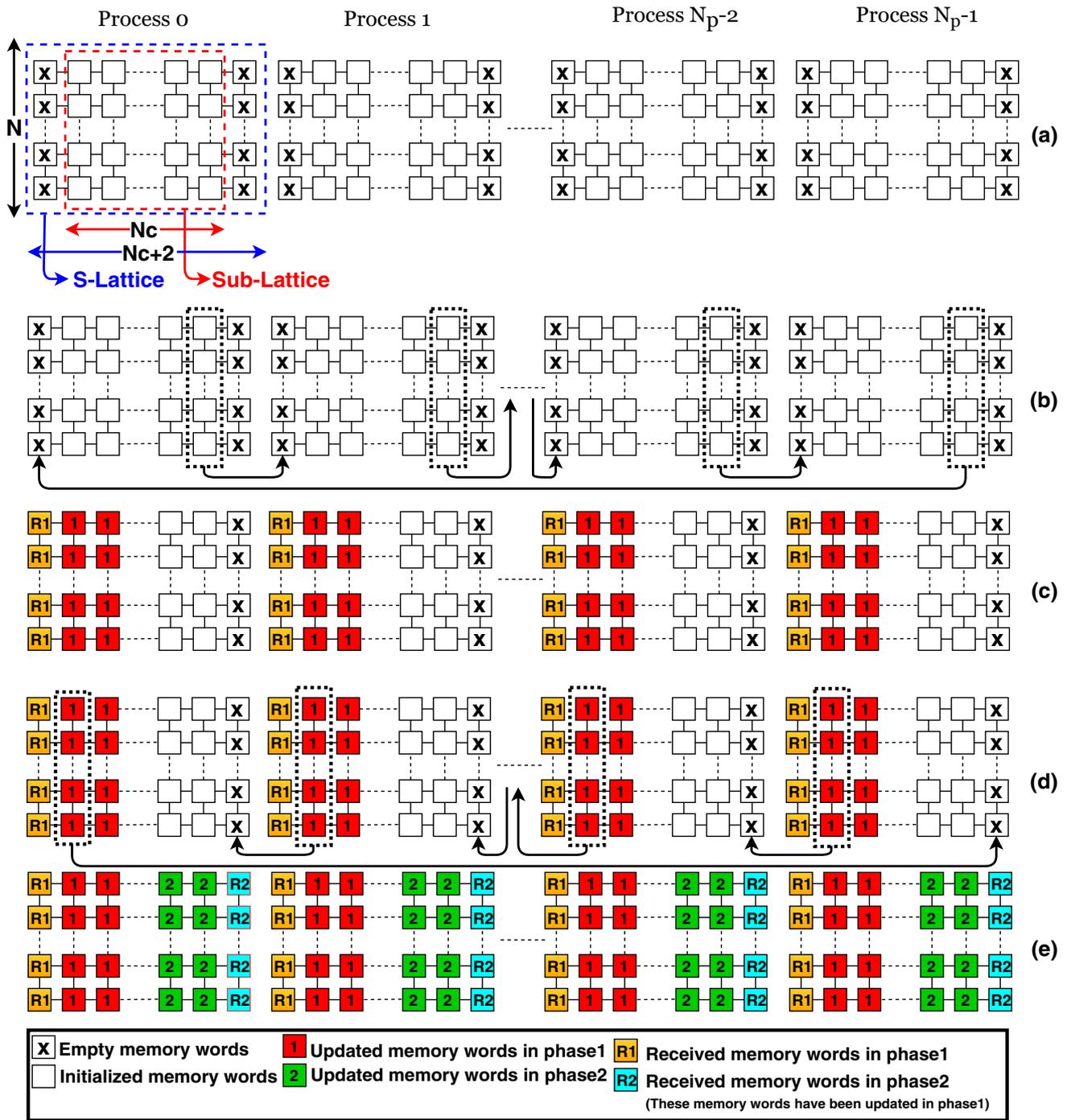
Fig. 5 **a** S-lattice of each process, **b** communication between processes before updating the memory words in phase 1, **c** updating memory words in phase 1, **d** communication between processes before updating memory words in phase 2, **e** updating the memory words in phase 2

**Table 2** Three different cases which have been examined in this paper

Test cases	Number of iterations	N	Average number of updates per spin in an iteration
1	5000	96	10
2	4500	96	10
3	4500	48	10

The number of iterations, N and the average number of updates per spin in one iteration have been presented in the second, third and forth columns, respectively

Fig. 6 Parallel speedup versus the number of cores presented in Table 2
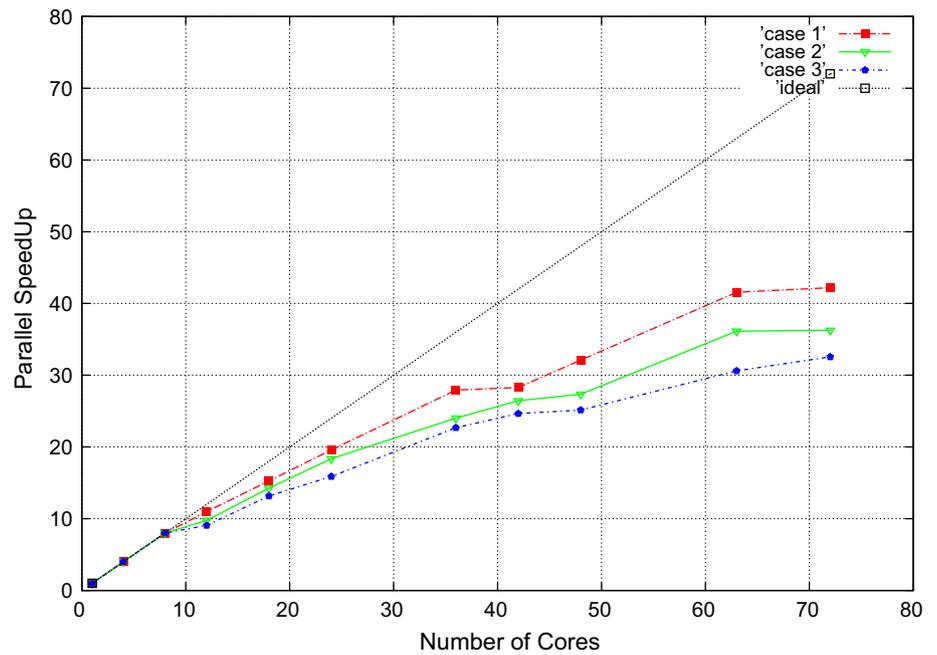
Fig. 7 Parallel efficiency versus the number of cores presented in Table 2
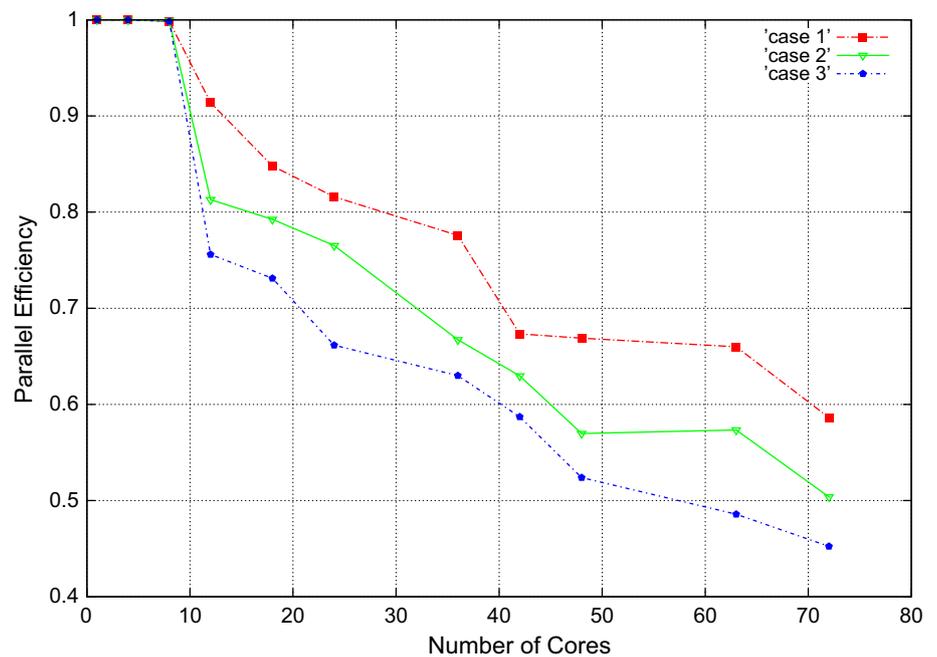
References

1. Ising, E.: Beitrag zur theorie des ferromagnetismus. Z. Phys. **31**(1), 253–258 (1925)
2. Onsager, L.: Crystal statistics. I. A two-dimensional model with an order-disorder transition. Phys. Rev. **65**, 117–149 (1944)

3. Baxter, R.J.: Exactly Solved Models in Statistical Mechanics. Courier Corporation, North Chelmsford (2013)

4. Deskins, W.R., Brown, G., Thompson, S.H., Rikvold, P.A.: Kinetic monte carlo simulations of a model for heat-assisted magnetization reversal in ultrathin films. Phys. Rev. B **84**, 094431 (2011)

5. Kozubski, R., Kozlowski, M., Wrobel, J., Wejrzanowski, T., Kurzydlowski, K.J., Goyhenex, C., Pierron-Bohnes, V., Rennhofer, M., Malinov, S.: Atomic ordering in nano-layered FePt: multiscale monte carlo simulation. Comput. Mater. Sci. **49**(1), 80–84 (2010)

6. Lyberatos, A., Parker, G.J.: Cluster monte carlo methods for the FePt hamiltonian. J. Magn. Magn. Mater. **400**, 266–270 (2016)

7. Masrour, R., Bahmad, L., Hamedoun, M., Benyoussef, A., Hlil, E.K.: The magnetic properties of a decorated ising nanotube examined by the use of the Monte Carlo simulations. Solid State Commun. **162**, 53–56 (2013)

8. Müller, M., Albe, K.: Lattice monte carlo simulations of FePt nanoparticles: influence of size, composition, and surface segregation on order-disorder phenomena. Phys. Rev. B **72**, 094203 (2005)

9. Yang, B., Asta, M., Mryasov, O.N., Klemmer, T.J., Chantrell, R.W.: Equilibrium Monte Carlo simulations of A1-L10 ordering in FePt nanoparticles. Scr. Mater. **53**(4), 417–422 (2005)

10. Swendsen, R.H., Wang, J.-S.: Nonuniversal critical dynamics in Monte Carlo simulations. Phys. Rev. Lett. **58**, 86–88 (1987)

11. Wolff, U.: Collective Monte Carlo updating for spin systems. Phys. Rev. Lett. **62**, 361 (1989)

12. Jacobs, L., Rebbi, C.: Multi-spin coding: a very efficient technique for Monte Carlo simulations of spin systems. J. Comput. Phys. **41**(1), 203–210 (1981)

13. Williams, G.O., Kalos, M.H.: A new multispin coding algorithm for Monte Carlo simulation of the Ising model. J. Stat. Phys. **37**(3), 283–299 (1984)

14. Zorn, R., Herrmann, H.J., Rebbi, C.: Tests of the multi-spin-coding technique in Monte Carlo simulations of statistical systems. Comput. Phys. Commun. **23**(4), 337–342 (1981)

15. Block, B., Virnau, P., Preis, T.: Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D ising model. Comput. Phys. Commun. **181**(9), 1549–1556 (2010)

16. Block, B.J., Preis, T.: Computer simulations of the ising model on graphics processing units. Eur. Phys. J. Special Top. **210**(1), 133–145 (2012)

17. Hawick, K.A., Leist, A., Playne, D.P.: Regular lattice and small-world spin model simulations using CUDA and GPUs. Int. J. Parallel Program. **39**(2), 183–201 (2011)

18. Komura, Y., Okabe, Y.: GPU-based swendsenwang multi-cluster algorithm for the simulation of two-dimensional classical spin systems. Comput. Phys. Commun. **183**(6), 1155–1161 (2012)

19. Komura, Y., Okabe, Y.: Gpu-based single-cluster algorithm for the simulation of the Ising model. J. Comput. Phys. **231**(4), 1209–1215 (2012)

20. Preis, T., Virnau, P., Paul, W., Schneider, J.J.: GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. J. Comput. Phys. **228**(12), 4468–4477 (2009)

21. Komura, Y., Okabe, Y.: CUDA programs for the GPU computing of the swendsenwang multi-cluster spin flip algorithm: 2D and 3D Ising, Potts, and XY models. Comput. Phys. Commun. **185**(3), 1038–1043 (2014)

22. Altevogt, P., Linke, A.: Parallelization of the two-dimensional Ising model on a cluster of IBM RISC system/6000 workstations. Parallel Comput. **19**(9), 1041–1052 (1993)

23. Ito, N.: Parallelization of the Ising simulation. Int. J. Mod. Phys. C **4**(6), 1131–1135 (1993)

24. Wansleben, S., Zabolitzky, J.G., Kalle, C.: Monte Carlo simulation of Ising models by multispin coding on a vector computer. J. Stat. Phys. **37**(3), 271–282 (1984)

25. Barkema, G.T., MacFarland, T.: Parallel simulation of the Ising model. Phys. Rev. E **50**, 1623–1628 (1994)

26. Kaupuzs, J., Rimsans, J., Melnik, R.V.N.: Parallelization of the wolff single-cluster algorithm. Phys. Rev. E **81**, 026701 (2010)

27. Weigel, M.: Simulating spin models on GPU. Comput. Phys. Commun. **182**(9), 1833–1836 (2011)

28. Petrov, G.M., Davis, J.: Parallelization of an implicit algorithm for multi-dimensional particle-in-cell simulations. Commun. Comput. Phys. **16**(3), 599–611 (2014)

29. Geng, W.: Parallel higher-order boundary integral electrostatics computation on molecular surfaces with curved triangulation. J. Comput. Phys. **241**, 253–265 (2013)

30. Keppens, R., Meliani, Z., van Marle, A.J., Delmont, P., Vlasis, A., van der Holst, B.: Parallel, grid-adaptive approaches for relativistic hydro and magnetohydrodynamics. J. Comput. Phys. **231**(3), 718–744 (2012)

31. Oger, G., Le Touz, D., Guibert, D., de Leffe, M., Biddiscombe, J., Soumagne, J., Piccinali, J.-G.: On distributed memory mpi-based parallelization of SPH codes in massive HPC context. Comput. Phys. Commun. **200**, 1–14 (2016)

32. Cheng, J., Liu, X., Liu, T., Luo, H.: A parallel, high-order direct discontinuous galerkin method for the Navier–Stokes equations on 3D hybrid grids. Commun. Comput. Phys. **21**(5), 1231–1257 (2017)

33. Leboeuf, J.-N.G., Decyk, V.K., Newman, D.E., Sanchez, R.: Implementation of 2D domain decomposition in the UCAN gyrokinetic particle-in-cell code and resulting performance of UCAN2. Commun. Comput. Phys. **19**(1), 205–225 (2016)

34. Wang, K., Liu, H., Chen, Z.: A scalable parallel black oil simulator on distributed memory parallel computers. J. Comput. Phys. **301**, 19–34 (2015)

35. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087–1092 (1953)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.