Accelerating the Simulations of the Ising Model by the GPU under the CUDA Environment

Xing Lu, Jing Cai, Peidong Cui, and Wei Zhang

***Abstract*—With the rapid development of the graphics processing unit (GPU), a recent GPU offers incredible resources for general purpose computing. We apply this technology to Monte Carlo simulations of the 2D and 3D lattice Ising models. By implementing the checkerboard algorithm, results are obtained up to 54, 62 and 68 times faster on the GPU than on a current CPU core for the honeycomb, square and triangular lattice respectively. For the 3D situation, however, the speedup is impacted greatly by the threads assignments on the GPU, the fastest one gives a speedup of 179, at the same time, the results of those simulations are consistent with the theoretical results for the 2D Ising model and previous results for the 3D Ising model.**

***Index Terms*—GPU, CUDA, Ising model, monte carlo**

I. Introduction

The Ising model [1], which is named after Ising, is a mathematical model of ferromagnetism in statistical mechanics. It was introduced to explain the phase transition of ferromagnetic material near the critical point K_c. According to Ising's research, no phase transition occurs in one dimensional spin chain. In the case of zero magnetic field, the Ising model on a square lattice was given a complete analytic description by Onsager in 1944 [2], the critical point $K_c = \ln(1+\sqrt{2})/2$. In the case of honeycomb and triangular lattices, the critical points are $\ln(2+\sqrt{3})/2$ and $\ln 3/4$ respectively [3]. However, there is no theoretical results for the 3D Ising model, the critical point K_c =0.2216544(3) [4] of the Ising model on a cubic lattice can be obtained by the computer simulations in combination with finite size scaling techniques. The Ising model, because of its rich content, became popular not only in physics but also in various interdisciplinary fields, i.e. biological systems [5], social sciences [6], biological physics [7], econophysics [8]. In the past 40 years, numerical simulations, represented by Monte Carlo (MC) method, have been used in the research of the Ising model, the performance of the MC method while dealing with large systems, however, is not satisfactory as lots of time is needed.

In the last few years, benefitting from the rapid development of the graphics processing unit (GPU), GPU computing has cought many scientists and engineers' eyes not only with its' incredible acceleration in various fields, but also with the improvement of GPU programming. In acceleration, GPU technology has gained a lot of high-performance computing applications and obtained encouraging results in many fields including molecular simulation [9], fluid dynamics [10], option pricing [11] and many other fields. In programming, GPU programming is much easier than ever before [12,13] as the emergence of new programming approach such as OpenCL [14], CUDA [15].

In this paper, some key facts of the GPU device architecture are briefly summarized in section II, the random number generation of this work is introduced in section III. The accelerations of GPU for Monte Carlo simulations of Ising model on various lattices are studied in section IV. The impact of thread assignment on the GPU performance is shown in section V.

II. GPU dEVice ARCHITECTURE

A GPU device consists of a set of multiprocessors and a large amount of global memory as illustrated in Fig. 1, each multiprocessor consists of a number of processors, a set of 32-bit registers, a shared memory, a read-only constant memory and texture memory and has a Single Instruction, Multiple Data architecture (SIMD): At any given clock cycle, each processor of the multiprocessor executes the same instruction, but operates on different data.

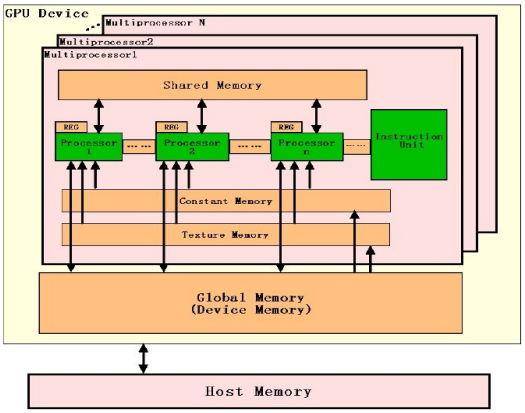


Fig.1 Device architecture of GPU.

In the CUDA environment [15], a CPU is used as a host while a GPU treated as a coprocessor to the host CPU is viewed as a parallel computing device. The GPU device is able to execute a very high number of threads in parallel, a thread block is a batch of threads which cooperate together and can share data through shared memory and synchronize their execution to coordinate memory access, a grid of

Manuscript received November 28, 2011; revised December 30, 2011.

Xing Lu, Jing Cai and Peidong Cui are postgraduate students with the Department of Physics, Jinan University, Guangzhou, P. R. China (e-mail: landstar@126.com, caijing164@163.com).

Wei Zhang is working as Assistance Professor with the Department of Physics, Jinan University, Guangzhou, P. R. China(email: twzhang@jnu.edu.cn).

* Corresponding author (Wei Zhang, email: twzhang@jnu.edu.cn)

blocks is a group of blocks with the same dimensionality and size and executing the same kernel, the communication and data exchange between blocks can be realized through global memory. As a result, each thread has a set of label to define it's id. The label is organized as (*threadIdx.x, threadIdx.y, threadIdx.z*) and (*blockIdx.x, blockIdx.y, blockIdx.z*) defined as dim3 type under CUDA. For 2D Ising models, only *blockIdx.x* and *threadIdx.x* are used to map the y and x dimensions, while the 3D model uses *blockIdx.x*, *blockIdx.y* and *threadIdx.x* mapping the *x, y* and *z* dimensions. The thread assignments have important impacts on the GPU performance.

The device of this article is Tesla C1060, the key facts and properties of Tesla C1060 is listed in TABLE 1, the host CPU is Intel(R) Core(TM) i7 920 @ 2.67GHz.

TABLE 1 KEY FACTS AND PROPERTIES OF TESLA C1060

	Tesla C1060
Total amount of global memory	4 GB
Number of multiprocessors	30
Number of cores	240
Constant memory	64 KB
Shared memory per block	16 KB
Warp size	32
Clock rate	1.30 GHz

III. RANDOM NUMBER GENERATOR

As a large number of random numbers will be used in the Monte Carlo method, the efficiency of the random number generator has a great impact on the algorithm efficiency. In order to get a high-performance algorithm, an array of linear congruential random number generators (LCRNGs) is applied to generate pseudo random numbers [16]. A single random number generator provides the random numbers for thread j. A sequence of random numbers for the jth thread $x_{i,j}$ is generated by the recurrence relation

$$x_{i+1,j} = (a \cdot x_{i,j} + c) \bmod m \tag{1}$$

where *a*, *c* and *m* are integer coefficients. An appropriate choice of these coefficients is responsible for the quality of the produced random numbers. We use *a*=1664525 and *c*=1013904223 as suggested, e.g., in [17]. As by construction results on a 32-bit architecture are truncated to the endmost 32 bits, the parameter of the modulo operation *m* is set to 2^{32}. As a result, the LCRNG can be used to generate random numbers $x_{i,j}$ in the interval $(-2^{31}, 2^{31})$. As the linear congruential random number generators are used in parallel, each LCRNG j of this array is initialized by a random number obtained by a further LCRNG through

$$x_{0,j} = 16807 \cdot x_{0,j-1} \bmod m \tag{2}$$

with a initial value $x_{0,0}$. In order to achieve higher efficiency, the generation and access of random numbers are carried out in the shared memory.

IV. GPU PERFORMANCE ON THE ISING MODEL

The spins of the Ising model is located on the sites of the lattice and can only value +1 and -1 indicate spin up and spin down respectively. The Hamiltonian $\mathcal{H}$ of this model is given by

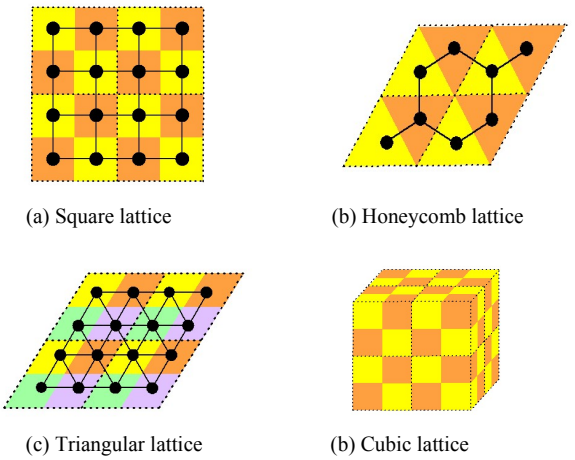
$$\mathcal{H} = -J \sum_{<i,j>} S_i S_j - H \sum_i S_i \tag{3}$$

where *J* indicates the interaction constant, <i,j> indicates the nearest spin pairs, $S_i = \pm 1$ represents a spin at site i and *H* denotes the external magnetic field. The Metropolis probablity [18] of flip of spin S_i in the case zero magnetic field is defined as

$$p(S_i \to -S_i) = \min[1, e^{-K \cdot \Delta \mathcal{H}, J}] \tag{4}$$

where $K=J/(k_B T)$ is the coupling constant, k_B is the Boltzmann constant, *J* and k_B set to 1 for facilitation, periodic boundary conditions are used. Such that the updating decision can be drawn solely upon examining the states of spin S_i and its nearest neighbors. Thus, the simulation of this model can be made local and highly parallel by dividing the spin field into many subcells and applying the checkerboard algorithm.

As it is not allowed to read and write a same memory unit simultaneously, simulating all the spins at a time is not allowed. As the GPU device is able to execute a very high number of threads in parallel, a certain amount of threads is implemented, each thread simulate one subcell, the efficiency of the algorithm depends on the number of threads executing in parallel, as a result, the division of the spin field is very important for the performance of the algorithm. The system size of the Ising model is denoted by *L* and the number of the subcells in each dimension is denoted by *Block size*.



(a) Square lattice (b) Honeycomb lattice

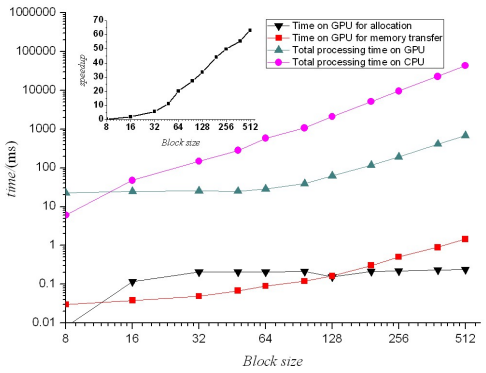
(c) Triangular lattice (b) Cubic lattice

Fig. 2 Schematic visualization of the implementations of the Ising model on various lattices on GPU: the areas enclosed by dotted lines are subcells simulated in parallel. The spins in the boxes with the same color are operated in parallel.

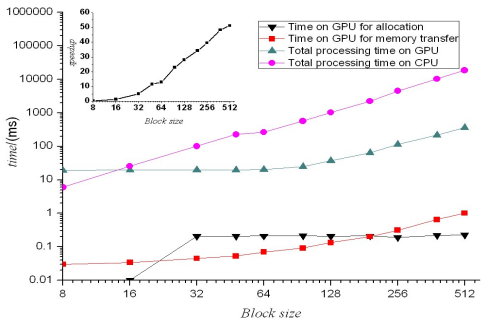
The implementations of the Ising model on various lattices on GPU are shown in Fig. 2. For a square lattice with system size *L*=4, as shown in Fig. 2(a), the spin field is divided into 2×2 square subcells with 2×2 spins in each subcell [19], *Block size*=*L*/2=2, GPU executes 2 thread blocks with 2 threads, each thread operates 2 spins on the diagonal of the subcell, as the synchronization is only possible between threads within a block, the only way to make sure all the block has finished the simulation is terminating the GPU kernel function, as a result, when the operation is finished, GPU needs to execute another 2×2 threads to operate the spins on the other diagonal. For a honeycomb lattice of *L*=2 with 2×4 sites, as shown in Fig. 2(b), the spin field is divided into 2×2 subcells with 2 spins

in each subcell, that means *Block size*=*L*=2, GPU executes 2 thread blocks with 2 threads, each thread operates one spin of the same location in the subcell then GPU needs to execute another 2×2 threads to simulate the other spin in the subcells. For a Triangular lattices of *L*=4 with *L*×*L* sites, as shown in Fig. 2(c), the spin field is divided into 2×2 subcells with 2×2 spins, GPU executes 2 thread blocks with 2 threads, different from the square lattice, each thread can only simulate one spin of the subcell at one time and needs to run 4 times to finish the simulation. For a cubic lattice of *L*=4 with 4×4×4 sites, as shown in Fig. 2(d), the spin field is divided into 2×2×2 subcells [19], GPU executes a 2D grid with 2×2 thread blocks and each block contains 2 threads, then, each thread operates 4 spins in the subcell, after that, GPU executes another 2×2×2 threads to simulate the other spins of the subcell.

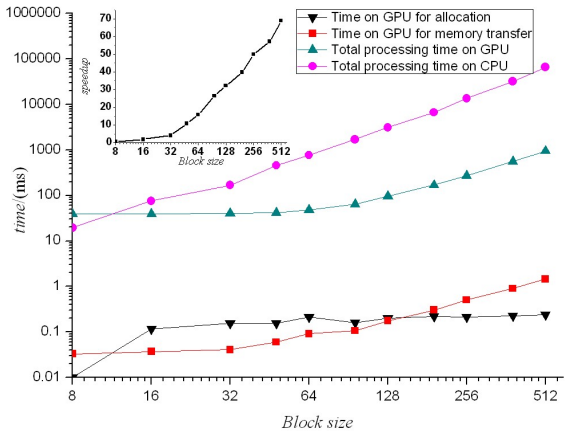
The speedups (acceleration factors [19]) of GPU for the Ising model on these lattices are shown in Fig.3. As shown in Fig. 3, the speedup increases with the *Block size*, as shown in Fig. 3(a), the speedup of a square lattice Ising model with *Block size*=512 is 63, at the same *Block size*, as shown in Fig. 3(b) and 3(c), the speedup of honeycomb and triangular lattice Ising model are 55 and 69 respectively, more strikingly, the speedup of 3D Ising model with *Block size* =384 can reach up to 142(details of 3D speedup are dicussed in section V).



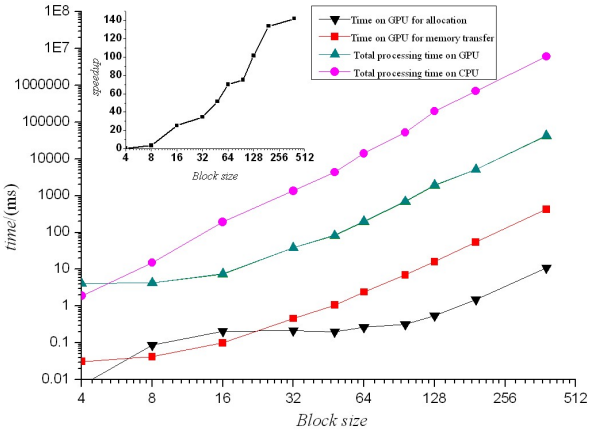
(a) Square lattice



(b) Honeycomb lattice



(c) Triangular lattice



(d) Cubic lattice

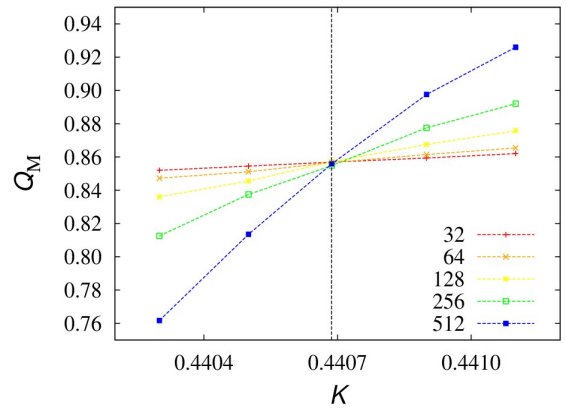
Fig. 3 Processing time and speedup of GPU compared with CPU: The processing time of CPU version and GPU version increases as the Block size increased, while the increase rate of GPU version is lower than CPU version, as a result, the speedup grows as the Block size increased.

In order to verify the GPU implementation, the critical point of Ising model is used. To obtain the critical point, we can use finite size scaling and Binder cumulant [20], which is given by

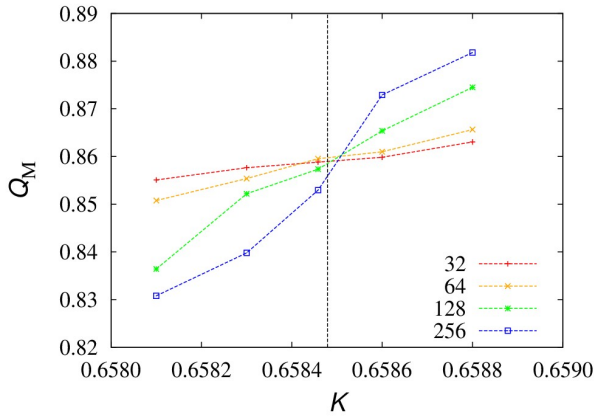
$$Q_{\mathrm{M}}(K,L)=\frac{<M(K,L)^2>^2}{<M(K,L)^4>} \tag{5}$$

where M denotes the magnetization of the Ising model and <…> means the average. According to finite size scaling theory, the Binder cumulant Q_{M} becomes independent of system size at the critical point, which means the crossover point of the Binder cumulants with various system sizes is the critical point. The Binder cumulant on these lattices is shown in Fig. 4.

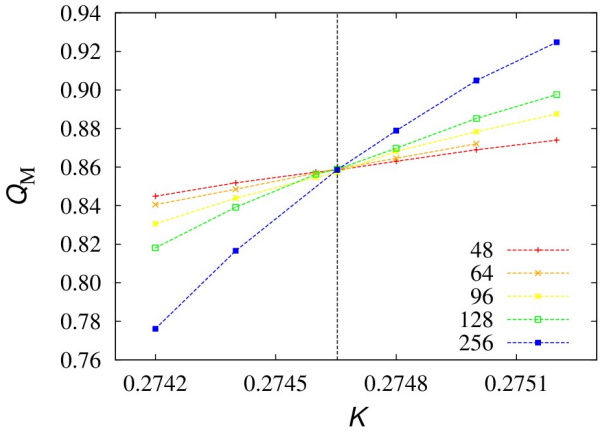
The critical point of the 2D and 3D Ising model on Fig. 4 is consistent with the theoretical results for the 2D Ising model and previous simulation results for the 3D Ising model mentioned in section 1.



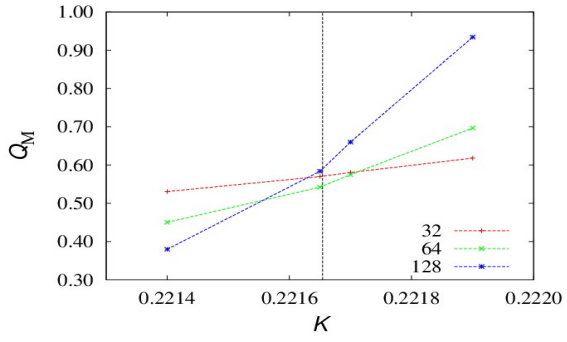
(a) Square lattice



(b) Honeycomb lattice



(c) Triangular lattice



(d) Cubic lattice

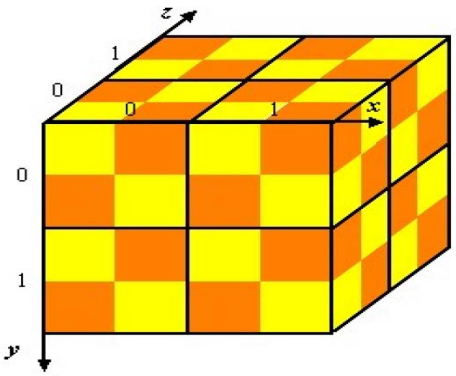
Fig. 4. QM vs K of various system sizes on various lattices. the intersections The critical points on various lattices are denoted by the black dotted lines around which are the intersections of the QM of various lattices.

V. Impact of Thread Assignment for 3D Ising Mdoel

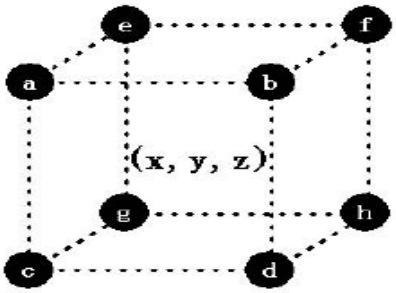
As mentioned in section II, thread assignment has important impact on the GPU's performance, some details will be discussed in this section.

The thread assignments have little impact on the 2D Ising models, while there are great differences on 3D situation between different assignments. As shown in Fig. 5. As shown in Fig. 1, the cuboid with $2\times2\times2$ spins is labeled by (x, y, z), we apply a 3D thread structure formed by a 2D grid and 1D block, for a system with $N=L\times L\times L$ spins, the size of block and grid in each dimension is defined as *Block size* = $L/2$. The location number of the 8 spins in the cuboid (x, y, z) are defined as follows:

$$\begin{aligned}
a:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x\\
b:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x+1\\
c:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x+2\cdot(Block\ size)\\
d:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x+2\cdot(Block\ size)+1\\
e:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x+4\cdot(Block\ size)^2\\
f:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x+4\cdot(Block\ size)^2+1\\
g:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x+4\cdot(Block\ size)^2+2\cdot(Block\ size)\\
h:&\ 8\cdot(Block\ size)^2\cdot z+4\cdot(Block\ size)\cdot y+2\cdot x+4\cdot(Block\ size)^2+2\cdot(Block\ size)+1
\end{aligned} \tag{6}$$



(a) Split of a 3D spin field



(b) a cuboid with 8 spins

Fig. 5. Schematic diagram of the 3D ferromagnetic cubic lattice Ising model implementation on a GPU for $L = 4$. The 3D spin field is divided into cuboids of $2\times2\times2$ spins assigned to 8 threads on the GPU, 2 threads form a block and the grid consists of 4 blocks with the structure of 2×2 square. The x, y, z axis used to describe the threads assignment on the GPU.

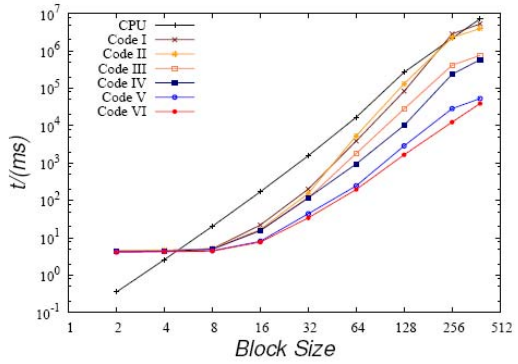
Each thread with a unique 3d label marked (*blockIdx.x, blockIdx.y, threadIdx.x*) under the CUDA environment operates a certain cuboid labeled (x, y, z), however, as each thread simulate a cuboid, the threads assignment is varied, six simplest assignments listed in TABLE 2 are studied in

this work.

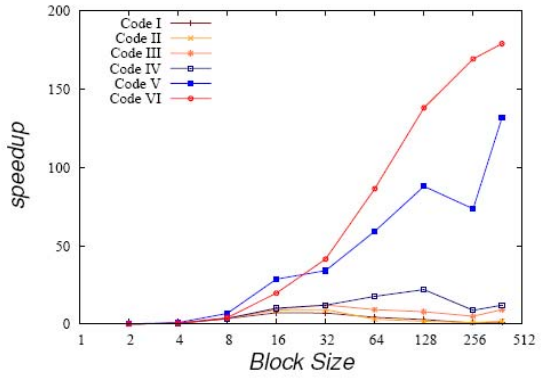
TABLE 2 THE ASSIGNMENTS OF THREADS

Code	*x*	*y*	*z*
I	*threadIdx.x*	*blockIdx.x*	*blockIdx.y*
II	*threadIdx.x*	*blockIdx.y*	*blockIdx.x*
III	*blockIdx.x*	*threadIdx.x*	*blockIdx.y*
IV	*blockIdx.y*	*threadIdx.x*	*blockIdx.x*
V	*blockIdx.x*	*blockIdx.y*	*threadIdx.x*
VI	*blockIdx.y*	*blockIdx.x*	*threadIdx.x*

The performances of these codes are expected to be same, however, they differs each other greatly as shown in Fig. 6.



(a) Processing time of different codes and CPU versus *Block Size.*



(b). The speedups of different codes versus *Block Size*

Fig. 6. Performance of different codes with various system sizes.

The efficiency of these codes are studied for various system size *L* ranging from 4 to 768, that is, *Block size* from 2 to 384. Larger systems are not simulated for the limit of the global memory on Tesla C1060. The processing time of different codes for 100 sweeps through the lattice is shown in Fig. 6(a). Further more, the speedups of these codes are shown in Fig. 6(b). When running these codes, the same number of threads are executing in parallel, the only difference between these codes is the digits multiplied by *threadIdx.x*, *blockIdx.x* and *blockIdx.y*. According to FIG. 6 and TABLE II, if *threadIdx.x* multiplied by the largest digit, that is, *z* assigned to *threadIdx.x*, the performance will be very slow, the assignment of *blockIdx.x* and *blockIdx.y* doesn't have much influence to the performance of the GPU (lines of code I and II), if we indicate *y* as *threadIdx.x*, *blockIdx.x* multiplied by a small digit will obtain a higher performance than multiplied with a large digit (lines of code III and IV), whatever, the performance of these codes is better than the form ones (I and II), for the fastest two codes V and VI, *threadIdx.x* multiplied by smallest digit 2, *blockIdx.x* multiplied by a small number will help us to gain a better performance, it is obvious that code VI has a better efficiency than code V. It can be guessed that the efficiency of the GPU is closely related with the digits multiplied by the thread label (*blockIdx.y, blockIdx.x, threadIdx.x*), smallest to *threadIdx.x*, later a larger number to *blockIdx.x*, the largest to *blockIdx.y* will behave best.

VI. SUMMARIES

Compaired with the CPU version algorithm, applying GPU as a data-parallel coprocessor of CPU has a great advantage on saving computing time. The speedup of GPU for Monte Carlo simulations of the Ising model is related to the system size(*Block size*), as the number of threads simulating is equal to the number of subcells, the larger the system is, the lager the speedup is. In the 2D situation, the speedups of a system with a *Block size* of 512 are 55, 63 and 69 for the honeycomb, square and triangular lattices respectively, which implicits that the speedup do not have much to do with the coordination numbers of lattices. While in the 3D case, it is more complex than 2D ones, the speedup of the GPU is also related to the thread assignment, different codes behave differently, the best code gain a speedup of 179 while the worst code does not show any speedup. As Metropolis algorithm does not efficient enough near the critical point as the critical slowing down happens, our further work may facous on the GPU based single-cluster algorithm and the hybrid algorithm consists of the GPU based Metropolis and CPU based wolff algorithm.

ACKNOWLEDGMENT

This paper is supported by "the National Natural Science Foundation of China", the project NO. 1047003 and NO. 11005048 and "the Fundamental Research Funds for the Central University", the project NO. 216113144. Thank H. C. Wong, U. H. Wong from Macao University of Science and Technology for valuable discussion.

REFERENCES

[1] E. Ising, Beitrag zur theorie des ferromagnetismus, Z. Phys., 31, 1925, PP: 253-258.
[2] L. Onsager, Crystal statistics. I: A two-dimensional model with an order-disorder transition, Phys. Rev., 65 (3-4), 1944, PP: 117-149.
[3] R. J. Baxter, Exactly Solved Models in Statistical Mechanics, Academic Press, London, 1982.
[4] K. Binder and E. Luijten, Monte carlo tests of renormalization-group predictions for critical phenomena in Ising models, Phys. Rep., 2001344, 2001, PP: 179-253.
[5] R. Schlicht and Y. Iwasa, Forest gap dynamics and the Ising model, J. Theor. Biol, 230 (1), 2004, PP: 65-75.
[6] D. O. Cajueiro, Enforcing social behavior in an Ising model with complex neighborhoods, Physica A, 390 (9), 2011, PP: 1695-1703.
[7] A. Imparato, A. Pelizzola, and M. Zamparo, Equilibrium Properties and Force-Driven Unfolding Pathways of RNA Molecules, Phys. Rev. Lett., 103, 2009, PP:188102-188105.
[8] A. Krawiecki, "Microscopic spin model for the stock market with attractor bubbling on scale-free networks," *Journal of Economic Interaction and Coordination*, 4 (2), 2009, PP: 213-220.
[9] J. E. Stone; J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, "Accelerating molecular modeling applications with graphics processors," *J. Comput. Chem.*, 28(16), 2007, PP: 2618-2640.
[10] J. Myre, S. D. C. Walsh, D. Lilja, and M. O. Saar, "Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters,"

Concurrency and Computation: Practice and Experience, 23 (4), 2011, PP: 332-350.
[11] B. W. Zhang and C. W. Oosterlee, "Option Pricing with COS method on Graphics Processing Units," *IEEE International Symposium on Parallel Distributed Processing*, IPDPS, 2009, PP: 1-8.
[12] R. J. Rost, "OpenGL Shading Language, second ed., Addison-Wesley, Longman," Amsterdam, 2006.
[13] R. Fernando and M. J. Kilgard, "The Cg Tutorial: The Definitive Guide to Programmable Real-time Graphics," Addison-Wesley, Longman, Amsterdam, 2003.
[14] NVIDIA Corporation, NVIDIA OpenCL JumpStart Guide, version 0.9, 2009.
[15] NVIDIA Corporation, NVIDIA CUDA C Programming Guide, version 3.1.1, 2010.
[16] J. J. Schneider and S. Kirkpatrick, Stochastic Optimization, Springer, Berlin, 2006.
[17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, Cambridge, 2007.
[18] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys*., 21, 1953, PP: 1087.
[19] T. Preis, P. Virnau, W. Paul, and J. J. Schneider, "GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model," *J. Comput. Phys.*, 228, 2009, PP: 4468-4477.
[20] K. Binder, "Finite size scaling analysis of Ising model block distribution functions," *Z. Phys. B*, 43, 1981, PP: 119-140.