



# Parallel tempering simulation of the three-dimensional Edwards–Anderson model with compact asynchronous multispin coding on GPU

Ye Fang<sup>a,b</sup>, Sheng Feng<sup>a,c,\*</sup>, Ka-Ming Tam<sup>a,c</sup>, Zhifeng Yun<sup>a,d</sup>, Juana Moreno<sup>a,c</sup>, J. Ramanujam<sup>a,b</sup>, Mark Jarrell<sup>a,c</sup>

<sup>a</sup> Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803, USA

<sup>b</sup> School of Electrical Engineering and Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA

<sup>c</sup> Department of Physics and Astronomy, Louisiana State University, Baton Rouge, LA 70803, USA

<sup>d</sup> Center for Advanced Computing and Data Systems, University of Houston, Houston, TX 77204, USA

## ARTICLE INFO

### Article history:

Received 17 February 2014

Received in revised form

20 May 2014

Accepted 25 May 2014

Available online xxxx

### Keywords:

Spin glass

Edwards–Anderson model

Ising model

Parallel tempering

Multispin coding

GPU

CUDA

## ABSTRACT

Monte Carlo simulations of the Ising model play an important role in the field of computational statistical physics, and they have revealed many properties of the model over the past few decades. However, the effect of frustration due to random disorder, in particular the possible spin glass phase, remains a crucial but poorly understood problem. One of the obstacles in the Monte Carlo simulation of random frustrated systems is their long relaxation time making an efficient parallel implementation on state-of-the-art computation platforms highly desirable. The Graphics Processing Unit (GPU) is such a platform that provides an opportunity to significantly enhance the computational performance and thus gain new insight into this problem. In this paper, we present optimization and tuning approaches for the CUDA implementation of the spin glass simulation on GPUs. We discuss the integration of various design alternatives, such as GPU kernel construction with minimal communication, memory tiling, and look-up tables. We present a binary data format, Compact Asynchronous Multispin Coding (CAMSC), which provides an additional 28.4% speedup compared with the traditionally used Asynchronous Multispin Coding (AMSC). Our overall design sustains a performance of 33.5 ps per spin flip attempt for simulating the three-dimensional Edwards–Anderson model with parallel tempering, which significantly improves the performance over existing GPU implementations.

© 2014 Published by Elsevier B.V.

## 1. Introduction

Stochastic or Monte Carlo (MC) simulation is one of the most important methods in the study of complex interacting systems. However, even with the huge success of Monte Carlo methods, many systems remain very difficult to simulate. The main obstacle very often is the long required simulation time, while the memory demands are quite modest. A prominent example is the Edwards–Anderson (EA) model, where the inherent randomness and frustration lead to very long relaxation times. Although the EA model has been intensively simulated over the past few decades,

including implementations using gate-level reconfigurable processors [1] and some dedicated computers designed specifically for solving this model [2–6], many aspects are still far from completely understood. Some prominent topics, such as the nature of the spin glass phase below the upper critical dimension, remain highly debated issues [7–18].

The Graphics Processing Unit (GPU) provides an opportunity to significantly improve the computational performance of Monte Carlo simulations of classical systems. Massive parallelism and acceleration can be achieved by implementing these algorithms on GPUs. In the past few years some GPU accelerated simple spin models have been proposed, including the two-dimensional Ising model by Hawick et al. [19] and Block et al. [20], and the Ising model in the cubic and network lattices by Preis et al. [21]. Weigel [22,23] studied the Ising and the Heisenberg models in both two- and three-dimensional lattices. These implementations focus predominately on unfrustrated systems with large lattice sizes. In

\* Corresponding author at: Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803, USA. Tel.: +1 2255787584.

E-mail address: [sfeng4@lsu.edu](mailto:sfeng4@lsu.edu) (S. Feng).

<http://dx.doi.org/10.1016/j.cpc.2014.05.020>

0010-4655/© 2014 Published by Elsevier B.V.

this study, we mainly focus in the simulation of a random frustrated Ising system in equilibrium. Due to its slow relaxation rate, a large number of Monte Carlo steps are required, at the same time the system sizes that can be simulated are relatively small, in most cases limited to only a few thousands sites. Precisely because of these characteristics, Monte Carlo simulations of random frustrated systems are a good match for the GPU computing architecture.

Our implementation targets cluster computers with NVIDIA Fermi GPUs. Using C/CUDA we control and tune details of the program. We expose the inherent parallelism of the algorithm to the GPU accelerator, including parallel computation on multiple sites, multiple temperature replicas and multiple disorder realizations. The memory requirements are efficiently handled through memory tiling. In addition, the computation is simplified and vectorized using table look-ups and the Compact Asynchronous Multispin Coding (CAMSC). We also substitute all floating point arithmetic with integer or bit string computations while preserve the same precision. Combining various tuning techniques, we achieve an average spin flip time of 33.5 ps. This is the fastest GPU implementation for the random frustrated Ising system on a  $16^3$  cubic lattice, and is comparable to that obtained with a field programmable gate array (FPGA) hardware [24] for small to intermediate system sizes. We note that a very recent paper reported a faster speed in a new FPGA system [25].

The paper is organized as follows. In Section 2, we discuss the algorithm. In Section 3, we present an outline of the code framework. The implementation and optimization methods are described in Section 4. Section 5 shows the experimental results. Conclusions and future directions are described in Section 6.

## 2. Theoretical background

### 2.1. Spin glass

The discovery of a plethora of unusual magnetic behaviors in disordered materials initiated the field of glassy systems [26]. Spin glasses are beyond the conventional description of long range magnetic ordering, e.g., ferromagnetic ordering. Some of their features, including their frequency-dependent susceptibilities and the discrepancy between zero-field and field cooling measurements, suggest that spin glasses have very slow dynamics. Notwithstanding most experimental spin glass systems, which exhibit glassy behavior, randomness and frustration seem to share some common properties. In real materials, dilution introduces randomness and directional or distance-dependent couplings, such as dipolar interactions in insulating systems and the Ruderman–Kittel–Kasuya–Yoshida coupling in metallic systems, introduce frustration.

The simplest model that captures the consequences of disorder is an Ising model with quenched randomly disordered couplings. This model was first proposed by Edwards and Anderson [27]. The mean field solution of the EA model for infinite dimensions was first attempted by Sherrington and Kirkpatrick [28]. However, the replica symmetric mean field solution was found to be unstable below the Almeida–Thouless line [29,30], a line in the temperature–field plane below which replica symmetry is broken. The difficulty of obtaining a stable solution was solved by Parisi with his replica symmetry breaking ansatz [31–36]. Although the mean field solution has been proven to provide the exact free energy for the spin glass phase in infinite dimensions [37,38], the spin glass physics in finite dimensions, which presumably is more relevant to experiments, is still not fully understood. Indeed, it had long been debated whether a spin glass phase at finite temperatures exists in three dimensions.

The EA model may be deceptively simple. Since it is a classical spin model, one may think that its numerical study can be simply

carried out by Monte Carlo methods on conventional hardware. One of the defining signatures of spin glass systems is their long relaxation time. For sufficiently low temperatures, the system becomes very sluggish and equilibration is prohibitively difficult even for modest systems sizes. Moreover, it has been shown that finding the ground state of the three dimensional EA model is an NP-hard problem [39]. Until recently, there has been no consensus on whether there is a finite spin glass critical temperature in the three dimensional EA model.

The breakthrough in the numerical study of spin glass systems came with the introduction of the parallel tempering method. It allowed the study of larger systems at lower temperatures than the simple single spin flip method [40–42]. Combined with improved schemes for finite size scaling, it is now widely believed that the thermodynamic finite-temperature spin glass phase does exist in the three dimensional EA model [43]. As the upper critical dimension of the model is six [44–46], a prominent remaining question is the nature of the spin glass phase below the upper critical dimension [47]. In particular, if the spin glass can still be described by the replica symmetry breaking scenario, there should be an Almeida–Thouless line below the upper critical dimension. A possible test of whether the Almeida–Thouless line exists is to determine whether a spin glass phase exists under an external magnetic field. Correlation length scaling analysis seems to suggest the absence of the spin glass phase in cubic lattices when a finite external field is applied [9]. On the other hand, a recent study in four-dimensional lattices suggests that by using a different quantity for the finite size scaling analysis, a spin glass phase can be revealed [15]. Given the relevance of spin glasses and the on-going controversy on the nature of the spin glass phase below the upper critical dimension, it is desirable to implement an efficient parallel tempering Monte Carlo algorithm using the graphics processing units to accelerate the simulations. In this work we show that using the multispin coding method [48], an efficient Monte Carlo algorithm can be implemented on the GPU.

### 2.2. Edwards–Anderson model

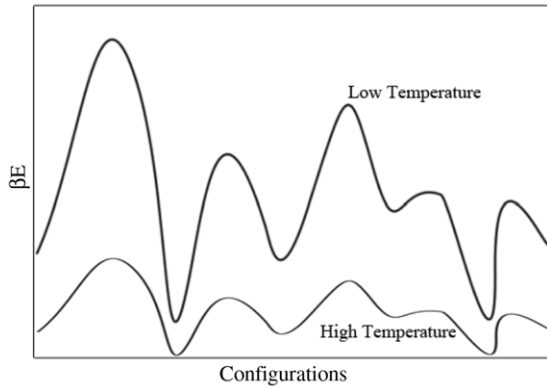
We consider the EA Model [27] on a simple cubic lattice. Spins on each lattice site have two states  $S_i = +1$  or  $-1$ . The couplings  $J_{ij}$  are between the nearest neighbors. In this study, we focus on a distribution of the couplings which is bimodal with a mean value of zero. That is, there are equal numbers of anti-ferromagnetic and ferromagnetic couplings. The effect of the distribution is certainly a non-trivial problem. We choose to focus on the bimodal distribution because it is best suited for multispin coding. In addition, a constant external field,  $h$ , is included in our implementation. The Hamiltonian is given by

$$H = - \sum_{(i,j)} S_{ij} J_{ij} S_j + h \sum_i S_i, \quad (1)$$

the summation is only over the nearest neighbors.

### 2.3. Single spin flip Metropolis algorithm

We implement the Metropolis algorithm as our sampling method. The spins are visited and tested for flipping according to the probability  $P = \exp(-\beta \Delta E)$ , where  $\beta$  is the temperature and  $\Delta E$  is the energy change associated with the proposed spin flip. As the algorithm satisfies detailed balance, the sampling will generate a distribution according to the partition function provided that the simulation is performed long enough. This type of Monte Carlo simulation is called a Markov process, because the evolution of the state only depends on the state at the current step, and not on its history.



**Fig. 1.** Schematic diagram of the energy landscape. The simulations at high temperatures allow shorter auto correlation times than those at low temperatures. Swapping the configurations at different temperatures facilitates the equilibrium simulations for low temperatures.

#### 2.4. Parallel tempering

For the simulation of glasses, the local single spin update algorithm is very slow in thermalizing the system. This problem is particularly severe when the temperature is close to the critical temperature for the second order transition. For certain spin glass models where random dilution is sufficiently large, some form of cluster algorithm can improve the rate of thermalization. Unfortunately, there is no efficient cluster methods for general spin glass systems. The possible exceptions are some random diluted systems or systems in low dimension [49–51]. Various other methods have been proposed in the past to improve the rate of thermalization including the umbrella sampling, the multi-canonical method, and rejection-free methods. It is now widely accepted that the parallel tempering method is one of the most efficient algorithms for improving the thermalization rate of general spin glass systems.

Parallel tempering uses several samples of the system within a range of temperatures (Fig. 1). The low temperature sample is more difficult to thermalize due to the larger barriers between low energy configurations [42,41]. Swaps between two samples  $i, j$  are accepted with probability:

$$P = \min(1, \exp[(\beta_i - \beta_j)(E_i - E_j)]) \quad (2)$$

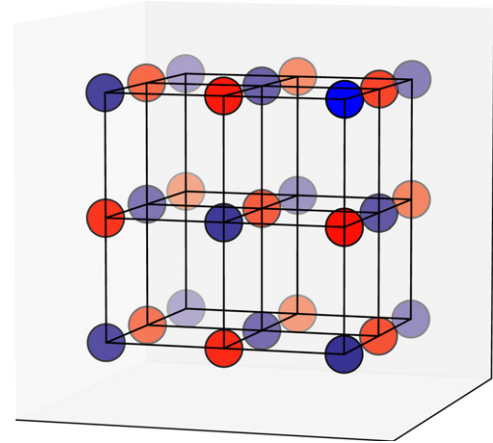
where  $\beta_k$  and  $E_k$  are the inverse temperature and the Hamiltonian of sample  $k$  respectively.

As the probability to swap the configuration between the high and the low temperature samples increases, the chance of the system to escape from a local minima in the low temperature sample also increases. The efficiency of such a parallel tempering move can be measured by the time it takes for a sample to perform a round trip along the temperature axis, that is from the lowest to the highest temperature and back to the lowest temperature. This largely depends on the system being simulated. Fine tuning the range of temperatures and the spacing between them is crucial to optimize the performance. Some recent proposals have been tested on the non-disorder Ising model [52–54]. Models with explicit disorder such as the EA lack an efficient general method. For a practical GPU implementation, one also needs to consider the effect of the number of replicas on the performance.

### 3. The framework

The GPU implementation is discussed in this and the following sections. In our replica exchange spin glass simulation we exploit three levels of parallelism:

1. Several tens of thousands, or more, of independent disorder realizations are required to obtain good statistics.



**Fig. 2.** A demonstration of the 3D checkerboard decomposition. The blue and red sites are on different sub-lattices. Since the sites in a sub-lattice never directly interact with each other, it is permissible to update different sites in parallel. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2. For each disorder realization, usually a few tens of systems at different temperatures are needed to study the physics, such as the possibility of a critical point. We denote these systems as temperature replicas. In the parallel tempering simulation, different temperature replicas communicate with each other only during the parallel tempering swap; these swaps are performed after every few Metropolis single spin sweeps of the lattice.
3. We are mainly interested in the systems on bipartite lattices. These are lattices that can be divided into two sub-lattices (A and B) with same sub-lattice spins that do not directly couple with each other. As a result, the update of the A sublattice is independent of the B sublattice.

These three levels of inherent parallelism allows an efficient GPU implementation. In this section we focus on the main structure of the code, which consists of three parts: (i) distributing the spin updates into different GPU threads; (ii) distributing different disorder realizations into different GPU blocks; and (iii) integrating and vectorizing the bit computations of many temperature replicas.

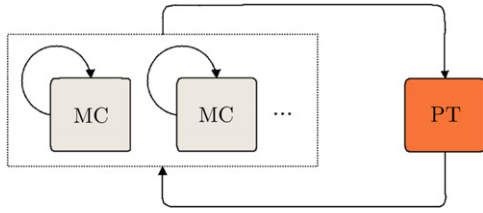
#### 3.1. Map lattice sites to GPU threads

The spin lattice is represented by a three dimensional primitive cubic system. To update the sites in the lattice, we follow the common practice of employing a checkerboard decomposition that splits the sites into two sub-lattices shown in blue and red in Fig. 2. Since a blue site is surrounded by red sites and never directly interacts with other blue sites and vice-versa, it is permissible to update each sub-lattice in parallel. We construct two consecutive stages concentrating independently on each of the sub-lattices for parallel computation. The combination of the two stages delivers a lattice sweep of Monte Carlo updates. The lattice is assigned to a GPU thread block, and sites are split across the threads. Details about the lattice site to thread mapping will be discussed in Section 4.2 where we discuss memory optimizations. The total available thread-level parallelism is half of the total lattice sites, and specifically, falls into the range between  $8^3/2 = 256$  and  $16^3/2 = 2048$  since our simulation targets lattices between  $8^3$  and  $16^3$  sites.

#### 3.2. Map temperatures replicas to bits

The parallel tempering technique facilitates the systems to achieve equilibrium. We choose the temperature as the tempering parameter and generate systems with the same couplings but





**Fig. 3.** Many temperature replicas can be simulated simultaneously, each using an independent Monte Carlo process. These replicas may be exchanged after a configurable steps of updates. A single GPU thread block is responsible for updating all the Monte Carlo processes and manipulating the parallel tempering exchange.

different temperatures, called temperature replicas. The temperature replicas are uncorrelated during the spin-flip process and can therefore be updated in parallel. However, they communicate and swap temperatures (Fig. 3) after a few Monte Carlo sweeps. To better utilize the parallelism of multiple temperature replicas and minimize the communication overhead we have developed the Compact Asynchronous Multispin Coding (CAMSC), where spins from different temperature replicas at the same position are encoded into an integer. This leads to sub-word vectorization and a significant reduction of memory transactions. Details of our multispin coding procedure can be found in Section 4.3.3. The number of temperature replicas depends on the system size and the temperature range. In our simulation we used 24 replicas for smaller systems, and 56 temperatures for bigger systems (for example,  $10^3$  and  $12^3$ ).

### 3.3. Map realizations to GPU blocks

Spin glass simulations usually require a larger number of disorder realizations ( $10^4$  or more) for reliable disorder averaging. A realization including all temperature replicas has been designated to a thread block. We launch numerous thread blocks across multiple GPUs of multiple hosts until we get the sufficient number of realizations for disorder averaging (Fig. 4). To distribute these jobs across multiple nodes, we employ a Pthreads/MPI wrapper for the job distribution.

### 3.4. Discussion

Some parallel processes are sequentialized for better memory locality. For example, although the temperature replicas could be fully parallelized as individual tasks or a lattice may be partitioned across multiple thread blocks, we avoid these forms of parallelism. The remaining parallelism is rich enough (with  $10^4$  or more thread blocks) to fully occupy the cluster.

To evaluate the performance, we employ a performance metric of average time (in ps) per proposed spin flip for a single GPU card:

$$t = T_{\text{total}} / N_{\text{MCS}} / (N_{\text{spins}} \times N_T \times N_{\text{samples}}), \quad (3)$$

where  $T_{\text{total}}$  is the total computation time of a simulation;  $N_{\text{MCS}}$  is the number of Monte Carlo sweeps;  $N_{\text{spins}}$  is the number of spins within a lattice;  $N_T$  is the number of temperature replicas;  $N_{\text{samples}}$  is the total number of disorder realizations on one GPU card. We develop and benchmark the code on a NVIDIA GeForce GTX 580 GPU. Detailed platform configurations can be found in Section 5.

## 4. Implementation

We discuss implementation details in this section, including the construction of the GPU kernel, memory optimization, and various techniques used to simplify the computation.

### 4.1. Kernel organization optimization

Our simulation starts with the Pthreads/MPI job dispatcher that forks many CPU processes across the cluster computer system. Each CPU process is responsible for initiating a lattice realization, which is offloaded to its attached GPU for simulation until the spin variables or thermal averaged results are retrieved from the GPU back to the CPU for analysis.

The GPU workload has three major components (Fig. 5):

1. **Metropolis moves:** The Metropolis steps for the single spin update for each temperature replica. This is done by calculating the local energy change and then comparing the acceptance ratio to a uniformly distributed random number.
2. **Parallel tempering moves:** Parallel tempering swaps are performed after a few complete Monte Carlo sweeps of the lattice. This step requires the calculation of total energy for all temperature replicas; we use this to evaluate the acceptance ratio of parallel tempering swaps.
3. **Measurements:** The spin configurations are dumped to the GPU global memory periodically to provide data for the measurements. In practice, we perform one measurement for every few thousands of Metropolis sweeps.

The measurement code has little overlap with the Monte Carlo and parallel tempering codes, and it is called much less frequently. We implement this part of the code as an separate GPU kernel.

Both Monte Carlo and parallel tempering functions compute spin local energies. Parallel tempering requires additional steps to sum the local energies. Since an efficient implementation of sum (a form of reduction) consumes a considerable amount of shared memory, it may be efficient to separate the parallel tempering as a dedicated GPU function apart from the Monte Carlo. We denote this scheme **MC-PT separated**. Alternatively, the **MC-PT integrated** scheme combines both the Monte Carlo and parallel tempering in a single GPU kernel. Benchmarks (Fig. 6) show that the **MC-PT separated** scheme always performs better regardless of the frequency of parallel tempering. However, we find that roughly 10 full Monte Carlo sweeps of the lattice between parallel tempering attempts is a speed/effectiveness sweet spot.

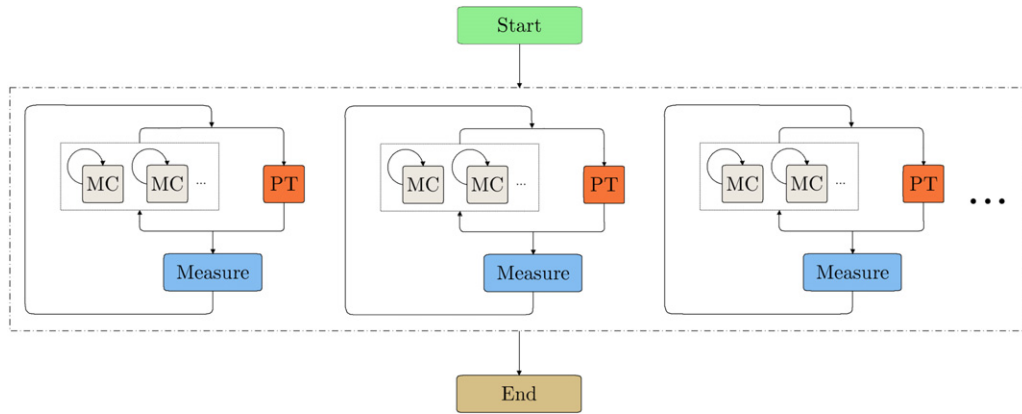
### 4.2. Memory optimization

Each spin interacts with its six nearest neighbors (Fig. 7) as a seven-point 3D stencil [55,56] with periodic boundary conditions. Unlike some stencil problems, e.g., the Jacobi finite difference solver for partial differential equations, in which the data for the new time step is completely based on the previous time step, the checkerboard decomposition allows the spin glass simulation to proceed with two consecutive update phases. Only half of the spins are updated in each of the phases. This unconventional stencil, associated with the checkerboard decomposition, leads to a stride-2 memory reference pattern and presents a more challenging memory optimization problem when compared to the stride-1 pattern of typical stencils problems.

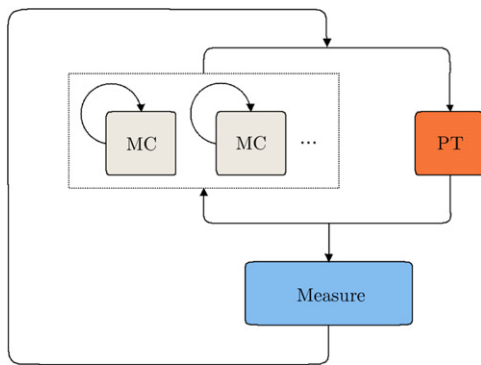
#### 4.2.1. Allocation

We propose three different schemes to address this problem.

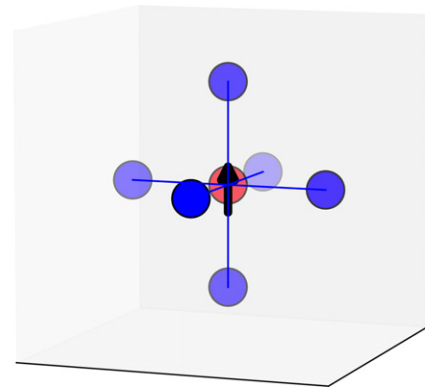
1. The **Unified** allocation (Fig. 8(a)) stores the checkerboard lattice in its native way as a single piece.
2. The **Separated** allocation (Fig. 8(b)) breaks the sub-lattices into two chunks stored separately.
3. The **Shuffled** allocation [57] (Fig. 9) mixes and integrates two temperature replicas, so that the memory access pattern is now identical to the conventional stencil. This is done by mixing the two temperature lattices in such a way that all the A sublattice



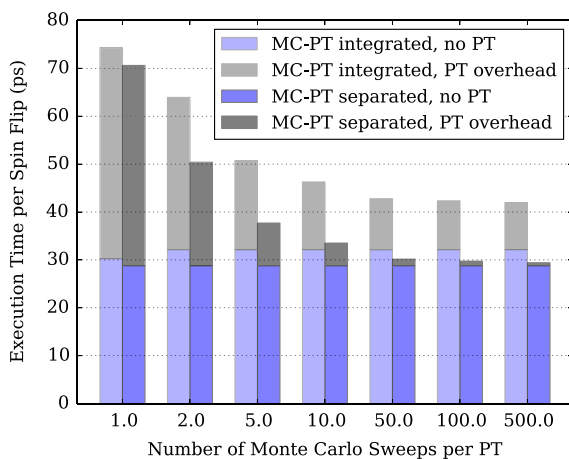
**Fig. 4.** The outline of the simulation application. Disorder realizations are completely independent and can run simultaneously. Each realization contains a unique Monte Carlo parallel tempering process as depicted in Fig. 3, and is assigned to a GPU thread block. This task level parallelism yields sufficient number of thread blocks and can fully occupy a parallel computer system.



**Fig. 5.** Three major components of the GPU program. One kernel calls Monte Carlo and parallel tempering, implemented as two device functions. Measurement is implemented as a separate GPU kernel.



**Fig. 7.** The memory access pattern for a single spin update where in addition to the state of the local spin, we also need the states of its 6 neighbors. Periodic boundary conditions are used.



**Fig. 6.** A comparison of the performance of the **MC-PT integrated** and the **MC-PT separated** schemes with different numbers of Monte Carlo sweeps between an attempted parallel tempering swap. The test is conducted with a  $16^3$  cubic lattice, shared memory probability table of integers, CURAND, and CAMSC.

spins from temperature 1 and the B sublattice spins from temperature 2 are packed together in the memory associated with one lattice. When the spins are being updated on this lattice, they are all independent of each other. They can be considered sequentially and continuously written into memory. Since there is no gap between each memory write, this should theoretically enhance the memory access speed.

**Table 1**

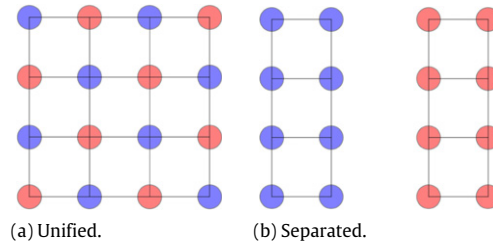
Performance comparison of the unified/separated/shuffled storage allocation schemes for a  $16^3$  lattice. The definition of a transaction is a sequence of 7 loads and a store that serve the spin update.

	Unified	Separated	Shuffled
Bandwidth (GB/s)	645.1	279.0	832.6
Time per transaction (ps)	49.608	107.756	38.432
Spins per transaction	24	24	16
Time per spin (ps)	2.067	4.345	2.402

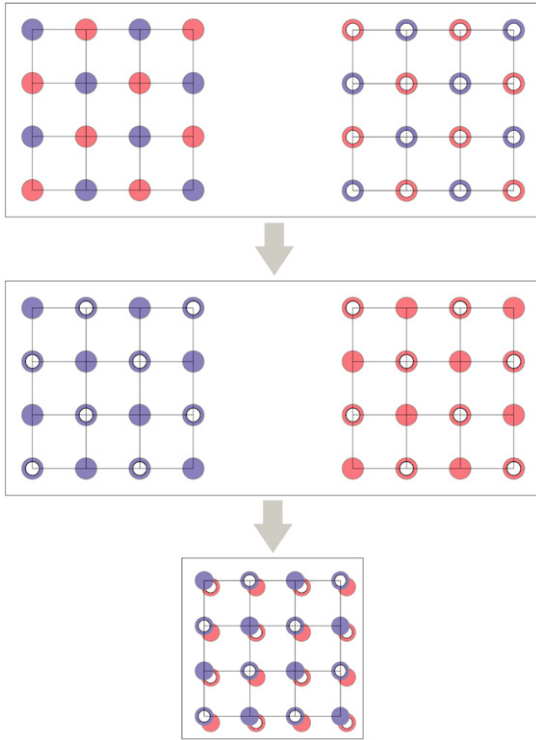
The performance comparison on Table 1 suggests that the separated allocation is inferior due to its significantly lower memory performance. This is because of the more complicated control flows in the code. Overall, the unified allocation provides the best memory performance in terms of time spent for each spin and is used in our implementation.

#### 4.2.2. Tiling for the multispin coding lattice

The basic idea of multispin coding (MSC) is to present many binaries or short vectors in a longer packed word. For example, Ising spins may be stored with a single bit per spin, with 0 being spin down and 1 being up. In our particular implementation, we also encode the 4 bit string of one site's spin-flip probability table's row index (Section 4.3.1) into an integer word. MSC [58,48] yields a more efficient way of calculating local energies ( $E$ ) and reduces the memory required for the spin configurations. This packing prevents the Arithmetic Logic Unit, which performs integer arithmetic and logical operations, and the memory bandwidth



**Fig. 8.** Unified and separated memory allocation schemes. The unified scheme stores the entire checkerboard lattice together. The separated scheme breaks the memory associated with each sublattice into separate continuous blocks of memory.



**Fig. 9.** The shuffled allocation mixes and integrates two lattices, shown on the top of the figure. The first transformation is taking the A sublattice on the left (blue dots) and the B sublattice on the right (blue circles) to construct an intermediate lattice (middle left figure of blue color). Another intermediate lattice of red color is similarly constructed. We then integrate these two intermediate lattices together, which occupy different bit positions under the compact multispin coding scheme (Section 4.3.3). By using one integer lattice instead of two, we avoid doubling the memory consumption. Also, the memory access pattern is identical to that of the 7-point 3D Jacobi stencil. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

from being under utilized. Also, a memory transaction (7 loads and 1 store) can serve the calculation of multiple spins, which helps improve the relative memory performance.

The usual practice for a single lattice MSC is integrating a line of spins into an integer. We denote this conventional method as Synchronous Multispin Coding (**SMSC**). For the simulation of spin glass models, the temperature replicas provide an alternative approach with a different memory layout. One can pack the spins at a specific site but at different temperature replicas into an integer; we call this the Asynchronous Multispin Coding (**AMSC**). The main idea of these two multispin coding schemes are:

- **SMSC:** A packed word stores the spins from a single replica, but different sites.
- **AMSC:** A packed word stores the spins belonging to different temperature replicas of the same site.

We find the AMSC scheme to be more efficient. Its storage consumption is small enough to fit in the GPU shared memory. Furthermore, AMSC's index system is more straightforward, thereby simplifying optimization. The performance of these different MSC schemes is described below. Here, we briefly discuss how the words associated with either scheme are organized into memory.

Three levels (Fig. 10) of the memory hierarchy are employed that reflect the GPU memory architecture of global memory, shared memory and registers:

- **Level 1:** The main data resides in the GPU global memory. Due to the limitation that a 32 bit integer represents at most 32 spins, we may need multiple integer cubes (with an integer cube including one integer per site on the cubic lattice) if there are more than 32 temperature replicas.
- **Level 2:** The shared memory scratchpad holds the working set of an entire integer cube (no larger than  $4 \times 16^3 = 16$  KB). The data transfer between global and shared memory is quite modest because we do not need to switch to another integer cube until the Monte Carlo and parallel tempering swaps within the temperature replicas contained within the current cube are exhausted.
- **Level 3:** The GPU threads scan the shared memory scratchpad for two consecutive sublattices and load the data into registers. The threads are organized as multiple layers of the 2D plates. We observe the optimal thread configurations are two or four layers ( $16^2/2 \times 2 = 256$  or  $16^2/2 \times 4 = 512$ ).

#### 4.3. Optimizing the computation

We may take advantage of the MSC mapping of the spins onto bits to dramatically reduce the number of floating point operations needed by the Monte Carlo parallel tempering calculations. For example, we may use a bitwise XOR ( $\oplus$ ) as opposed to multiplication to calculate the energy. In the equations below, we denote the variables in the original notation with a superscript  $^o$ , and variables without superscripts are used in the transformed notation. The variables  $S, J, e$  and  $E$  stand for spin, spin coupling, bound energy and local energy respectively. The summation is over the six nearest neighbors.

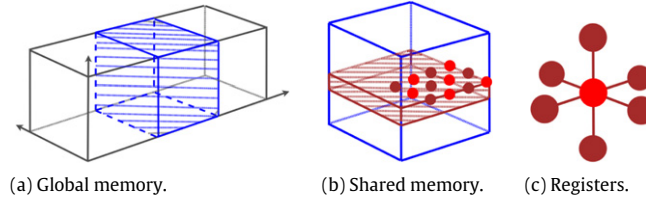
$$S^o \in \{-1, 1\}, \quad J^o \in \{-1, 1\}$$

$$E_i^o = \sum_j S_i^o \times J_{ij}^o \times S_j^o, \quad E_i^o \in \{-6, -4, -2, 0, 2, 4, 6\}.$$

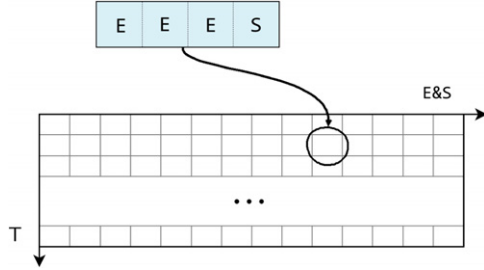
$$S \in \{0, 1\}, \quad J \in \{0, 1\}$$

$$E_i = \sum_j S_i \oplus J_{ij} \oplus S_j, \quad E_i \in \{0, 1, 2, 3, 4, 5, 6\}.$$

Note that the local energy  $E_i^o$ , the energy of a spin  $i$  in the field of its nearest neighbors, can only take one of the seven values as indicated.



**Fig. 10.** Memory tiling. The global memory may hold several integer cubes (including one integer per lattice site) if there are more than 32 temperature replicas. The shared memory scratchpad holds the working set of an entire integer cube (no larger than  $4 \times 16^3 = 16$  KB). The registers hold the data needed for local spin updates.



**Fig. 11.** The organization of the probability look-up table.

The computation is composed of four steps:

1. Energy: Compute the bound energy ( $e$ ) and the spin's local energy ( $E$ ).

$$e_{ij} = S_i \oplus J_{ij} \oplus S_j$$

$$E_i = \sum_j e_{ij}. \quad (4)$$

2. Probability: Compute the flip probability ( $P$ ) for the Metropolis Monte Carlo, where the temperature ( $T$ ) is an input parameter.

$$E^0 = 2 \times E - 6$$

$$S^0 = 2 \times S - 1$$

$$P = \min \left( \exp \left( 2 \times \frac{1}{T} \times (E^0 + h \times S^0) \right), 1 \right). \quad (5)$$

In the actual program the min function is not needed, since a probability larger than 1 will be automatically accepted.

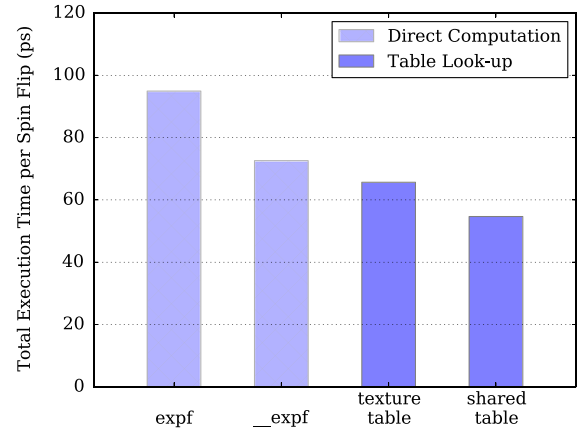
3. Rand: Generate a random number ( $R$ ) in the range of 0–1. The random number is always generated even when  $P > 1$ , since the SIMT (Single Instruction Multiple Threads) architecture and the elimination of branch statements yield better performance.
4. Compare: Compare and update spins.

$$S = (P < R) \oplus S. \quad (6)$$

#### 4.3.1. Probability look-up table

Eq. (5) expresses the straightforward yet expensive method to generate the spin flip probabilities. However, since the number of input/output values is finite (i.e., combinations of 7 possible local energies  $E$ , 2 spins  $S$ , and no more than 32 temperatures  $T$ ), a better solution is to deploy a pre-calculated look-up table. The table is a two-dimensional matrix (Fig. 11), with  $T$  as the row index and  $(E \times 2 + S)$  as the column index. The column index, as the combination of  $E$  and  $S$ , requires 4 bits for the address space. The maximum storage consumption of the table is 16 KB, assume that we have 32 rows times 14 columns times 4 bytes per entry (again, assume 32 temperature replicas). When a parallel tempering swap between two replicas at temperatures  $T_i$  and  $T_j$  is accepted, the two corresponding rows in the table are swapped.

We evaluate four different ways to calculate the probability in Eq. (5) (Fig. 12): (a) using the C standard library exponential function, (b) using the less accurate GPU intrinsic function,



**Fig. 12.** A comparison of the overall time consumed per spin flip using four different methods to compute the exponential probability in Eq. (5) as described in the main text. The experiment is done for a  $16^3$  lattice, fp32 CURAND and AMSC1. No parallel-tempering is performed.

(c) using the texture memory to store a table, and (d) using the shared memory table. The result shows that an optimal table look-up saves close to half of the total computation time when compared to the direct computation of the probabilities. In addition, the shared memory table outperforms the texture memory table. This is because the GPU threads are simultaneously computing on the replicas of the same temperature, and are randomly accessing multiple columns of the same row. This access pattern introduces no memory bank conflicts, and therefore fully exploit the high bandwidth and low latency performance potential of the shared memory.

#### 4.3.2. Random number generator

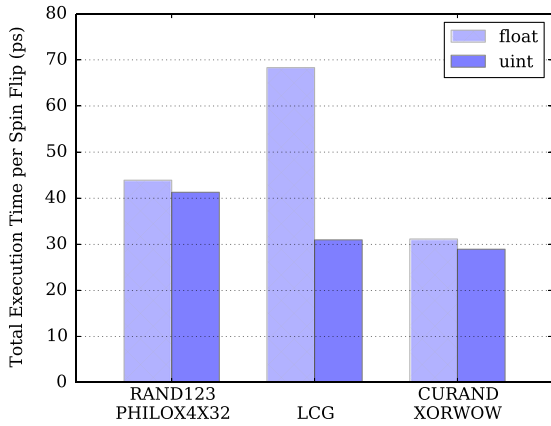
The simulation requires uniformly distributed random numbers between 0 and 1. However, due to the fact that pseudo random number generators (RNGs) manipulate integer values internally, directly using integer return values from the RNG provides higher performance and preserves identical precision. As a consequence, we convert the pre-generated probabilities from single precision floating point numbers to 32 bit unsigned integers.

We evaluated three random number generators: (a) NVIDIA CURAND library of XORWOW algorithm [59], (b) rand123 [60] philox4x32\_7 (version 1.06), and (c) our implementation of a multi-threaded 32 bit linear congruential generator (LCG). We decide to adopt CURAND due to its higher performance (Fig. 13) and quality [61].

#### 4.3.3. Multispin coding

We have briefly described the Multispin Coding (MSC). We have developed the Asynchronous Multispin coding (AMSC) as a more efficient alternative to the conventional Synchronous Multispin Coding (SMSC) for calculating the local energies ( $E$ ), generating the 4 bit string for the column index of the spin-flip probability table (Section 4.3.1), and optimizing the memory bandwidth utilization.





**Fig. 13.** A comparison of the overall time required per spin flip using different random number generators. The experiment used a  $16^3$  lattice, a shared memory probability table and CAMSC. No parallel-tempering is performed. The loop that consumes random numbers has been unrolled four times to match the four return values of rand123 philox4x32\_7.

In our particular GPU implementation, we use four byte unsigned integers, which hold up to 32 bits, as a packed word. Each spin, denoted as 0 or 1, takes only one bit of this packed word. Thus, the calculations in Eq. (4) can be vectorized via bit-wise operations. We integrate the  $J$  bits with the  $S$  bits in the same integer, so that we can fetch both the coupling and the spins in only one memory transaction. We then multiply the coupling with a bit-mask to match the pattern of  $S$ , and calculate the bond energy with bit-wise XOR operation. The next step is to add the six bond energies around a spin to obtain the local energy. To vectorize this process we need to reserve empty bits to avoid overflow, since the local energy takes 3 bits of storage. In this way, each spin, together with the empty bits reserved for calculation, constitute a virtual segment. We derived three variations of AMSC with different segment width of 1, 3 and 4, denoted as AMSC1, AMSC3 and AMSC4 respectively. In AMSC1 and AMSC3, some calculations are sequentialized to avoid

overflow. Figs. 14 and 15 demonstrate how the different variations of MSC parallelize the computations in Eqs. (4)–(6).

Fig. 16 illustrates that AMSC3 and AMSC4 are favored over AMSC1 due to improved overall performance. However, we also observe proportionally longer times for the memory transactions. This demonstrates the limitation of the AMSC scheme: there does not exist an optimal segment width that simultaneously provides the highest memory density, and the richest vectorization opportunities in computation.

To overcome the intrinsic limitation of AMSC, we propose a new scheme named Compact Asynchronous Multispin Coding (CAMSC). We dynamically change the segment width to match the data range. Longer width is adopted for larger data to qualify the vectorization of computing multiple segments. For small range data, we use shorter width to avoid blank bits reservations. For example, we allocate 1 bit per segment for  $S$  and  $e$ , and then expand to 4 bits when calculating  $E$ . The segment width expansion is implemented with shift and mask operations. Fig. 15 demonstrates the procedures of CAMSC and how it differs from traditional AMSCs. Our experiment (Fig. 16) shows 28.4% performance improvement when we switch from AMSC3 (46.8 ps/spin) to CAMSC (33.5 ps/spin).

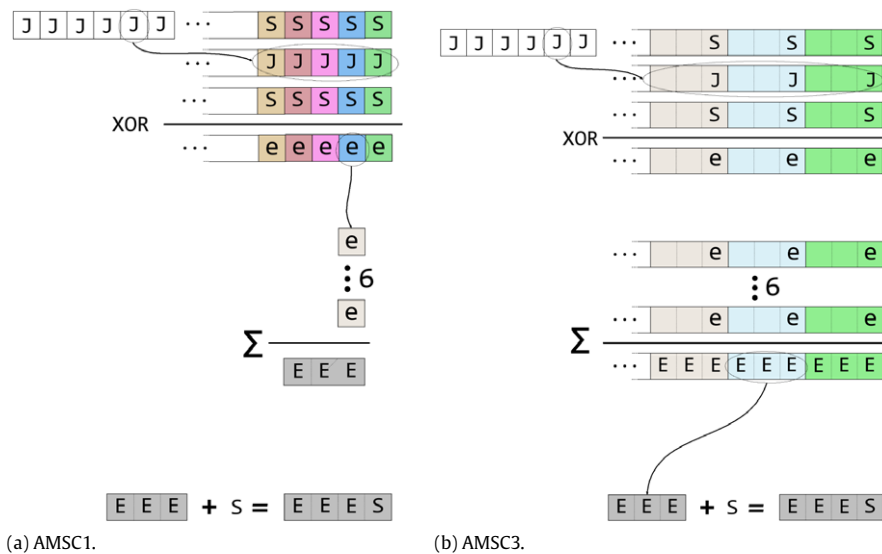
## 5. Experimental results

### 5.1. The platform settings

Our development and performance evaluations are carried out on a workstation with an Intel Core i7x990 CPU and an NVIDIA GeForce GTX 580 GPU card. The GeForce GTX 580 is equipped with a Fermi architecture GPU of 512 stream processors. We use Linux 2.6.32x86-64, CUDA toolkit version 4.1 and gcc 4.4.6, and optimization flag -O2. We always configure the GPU on-chip memory as 48 KB shared memory plus 16 KB L1 cache.

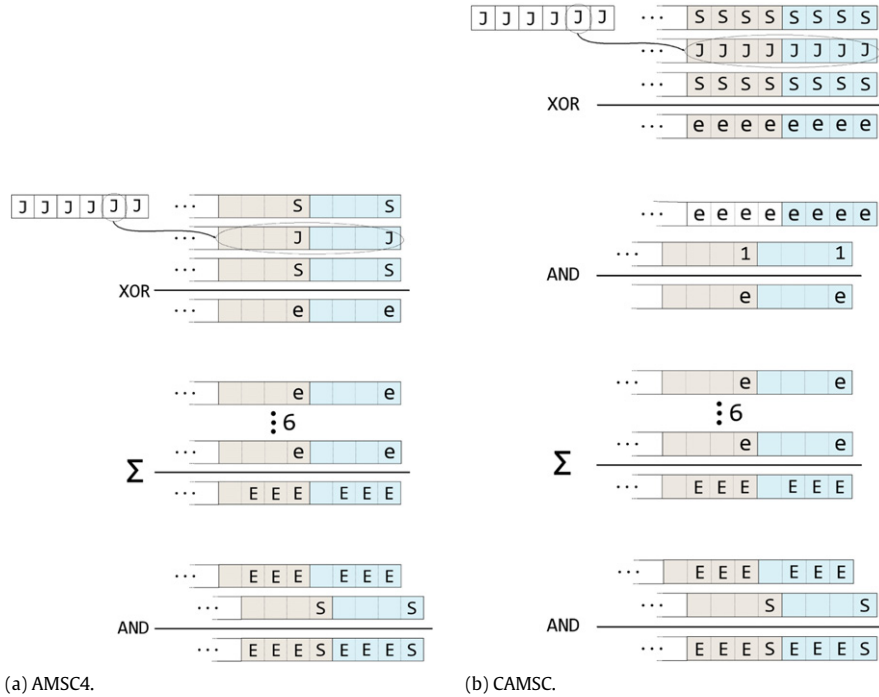
### 5.2. Performance evaluation

To evaluate the performance we use the time spent (in ps) per spin flip proposal, abbreviated as ps/spin (see Eq. (3)).

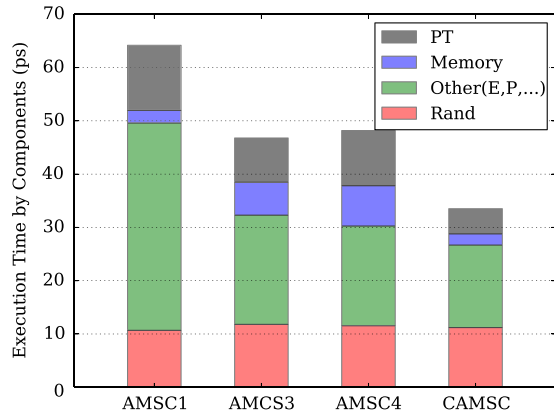


**Fig. 14.** This figure demonstrates the computation of  $(E \times 2 + S)$  for the purpose of accessing the probability look up table with the deployment of two variations of Asynchronous Multispin Coding (AMSC). Each line in the figure represents an integer, each box of a line represents a bit, and boxes of the same color represent a segment that hold a variable from one of the temperature replicas. We give the name AMSC1 and AMSC3 for these two AMSC schemes according to their segment width. Unlike the AMSC1, the AMSC3 scheme reserves three bits for each segment, and is a less dense storage format. For the calculation of the local energy, we need two spins ( $S$ ) and the coupling ( $J$ ) between them. The  $J$  bits and  $S$  bits are integrated in the same integer, so that we can fetch both the coupling and the spins using only one memory transaction. The local energy ( $e$ ) of each bond can be calculated by performing two XOR operations. The total local change of energy ( $EEE$ ) is the sum of the contributions from all the six nearest neighbors. Since  $EEE$  requires three bits for storage, the AMSC1 scheme computes each segment sequentially to avoid overflow, while the AMSC3 scheme can compute multiple segments in parallel. After we obtain  $EEE$  in three-bit format, we combine it with the spin state ( $S$ ) by doing string concatenation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)





**Fig. 15.** This figure demonstrates how the AMSC4 and CAMSC schemes help exploit bit-level parallelism in computing  $(E \times 2 + S)$ . Similar to that of the AMSC1 and AMSC3 (see the text and the caption of Fig. 14), the XOR operations and summation over the six nearest neighbors produce the total local energy ( $EEE$ ). However, since we reserve four bits for each segment, and is capable of holding one more bit over  $EEE$ , the string concatenation of  $EEE$  with  $S$  can now be vectorized. The difference between CAMSC and AMSC4 is that  $S$  and  $J$  are stored in a more compact format. With such a design, CAMSC avoids waste of space and provides much better parallelism in computing  $e$ .



**Fig. 16.** Comparing the performance using different multispin coding schemes. The experiment is done for a  $16^3$  lattice, a shared memory probability table with integers and CURAND. A parallel-tempering move is performed every 10 Metropolis single spin sweeps.

When we study the equilibrium properties of a spin glass, the system sizes that can be equilibrated within a reasonable time are not very large. Therefore, we used  $L = 16$ , or  $N_{\text{spins}} = 4096$  as the maximum system size. Meanwhile, to achieve efficient parallel tempering moves, we set the number of temperature replicas to  $N_T = 24$  or 56, and perform frequent parallel tempering moves (one parallel tempering move after every 5–10 Monte Carlo sweeps). The typical number of Monte Carlo steps required to equilibrate such a system is approximately  $10^7$ . Due to the huge sample-to-sample variation, a large number of disorder realizations ( $10^4$  or more) are usually required. However, since there is no correlation among different realizations, we can scatter the jobs to different GPU cards or nodes on a cluster. On each of the cards we only need 16–64 realizations to fully utilize all the multiprocessors.

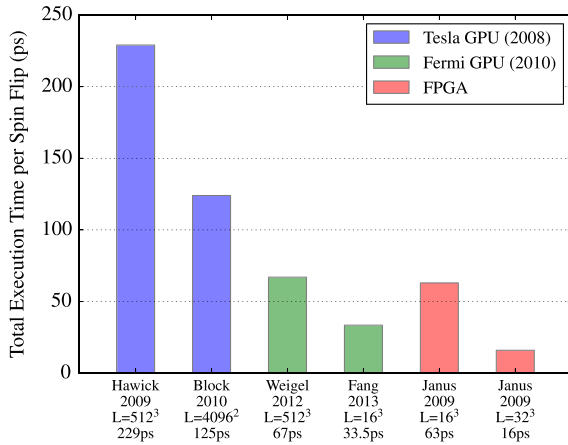
For benchmarking, we simulate 64 disorder realizations of the EA model on a  $16^3$  lattice with 24 temperature replicas, and

propose to swap adjacent temperatures every 10 Monte Carlo sweeps. We are able to complete  $10^7$  Monte Carlo sweeps in 40 min. This wall time consists of the spin flip Monte Carlo time, the parallel tempering swap time, and the measurement time. Discarding the measurements, the average computational speed is 33.5 ps/spin, for a single GPU device. If we simulate without parallel tempering and serve all temperature replicas with the same random number, we could obtain 17.6 ps/spin. Generating random numbers consumes about one third of the total simulations time, as shown in Fig. 16. We believe we are approaching the limit of performance optimization. For reference, our single thread CPU code (using AMSC4 without parallel tempering on a  $16^3$  cubic lattice) runs at the speed of 14 737 ps/spin; this represents a speed up of almost 440 for the GPU code over the CPU code.

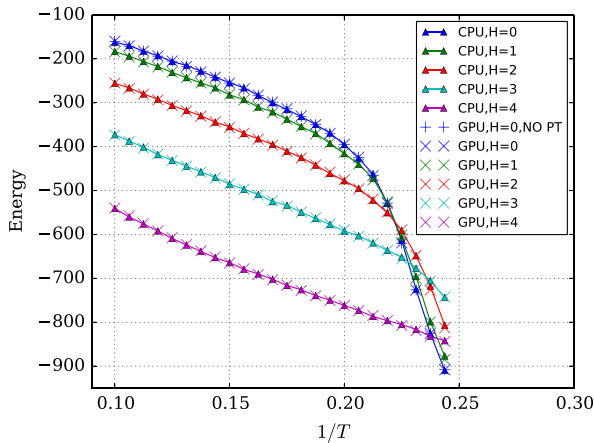
Fig. 17 compares our implementation with similar existing codes, where not all reference programs target at the random frustrated Ising systems, present the external magnetic field, and feature parallel tempering. Our program is substantially faster than any other GPU implementation [19,20,22,23] for small to intermediate system sizes. We are comparable to the performance achieved by special-purpose FPGA implementations[24].

### 5.3. Simulation results

We test the code by simulating both the simple ferromagnetic Ising and the EA spin glass models. In Fig. 18, our results from the GPU code are found to be consistent with the results from our CPU code for the ferromagnetic Ising model, at various external magnetic fields. We also compare the results with and without parallel tempering as a check to determine whether the parallel tempering swap is performed correctly. We find that the results with and without the parallel tempering swap are consistent with each other. In Fig. 19 we plot the correlation length for the ferromagnetic Ising model in three dimensions; here, the crossing point for the correlation length coincides with the known critical temperature for the ferromagnetic ordering [62]. For the



**Fig. 17.** Performance comparison with other heterogeneous Ising model simulation programs. Hawick et al. [19] reports 4360.1 million Monte Carlo hits per second, which equals to 229 ps/spin. Block et al. [20] reports 7977.4 spin flips per microsecond, which equals to 125 ps/spin.

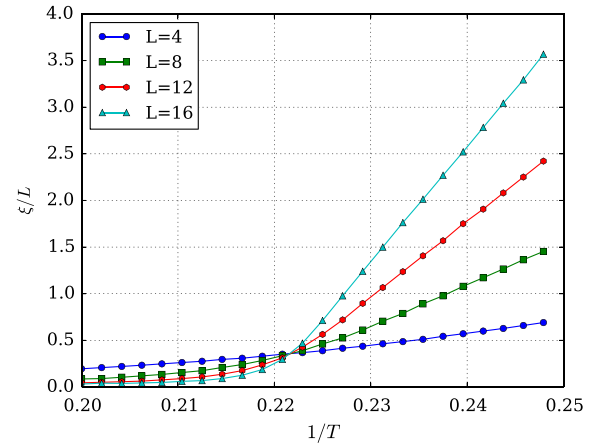


**Fig. 18.** Comparing the total energy of the  $16^3$  sites Ising model with the nearest neighbors coupling  $J = -1$ , to CPU generated results. At each value of the external field, the GPU results are nearly identical to the CPU results.

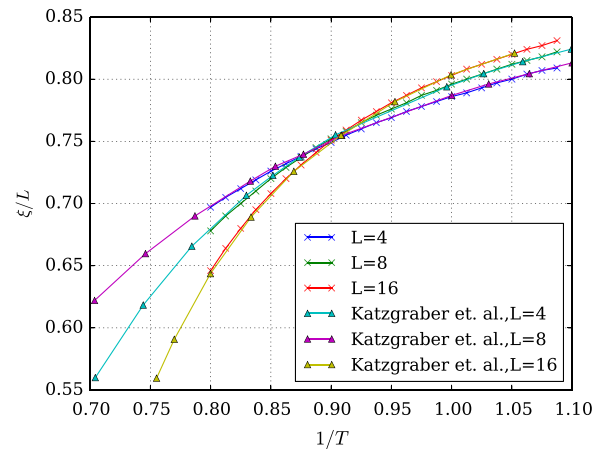
EA model we calculate the Binder ratio of the system at zero external magnetic field as shown in Fig. 20. The results match reasonably well with the published data [63]. Fig. 21 demonstrates the effectiveness of parallel tempering for the EA spin glass. The parallel-tempering simulation reaches equilibrium after  $10^5$  Monte Carlo sweeps, while without parallel tempering, the system did not reach equilibrium even with 100 times more iterations. This further supports that we have implemented the parallel tempering swapping correctly.

## 6. Conclusion and future works

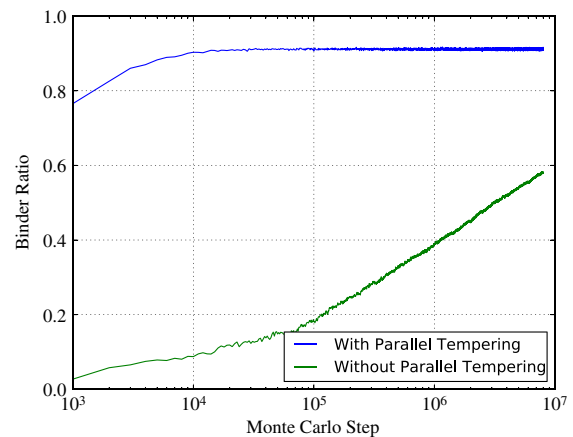
We design and implement a CUDA code for simulating the random frustrated three-dimensional Edwards–Anderson Ising model on GPUs. For small to intermediate system sizes, our code runs faster than other GPU implementations, and its speed is close to that of the specially built FPGA computer. We note that a very recent paper has reported an improvement in FPGA system [25]. Our performance tuning strategies include constructing three levels (tasks, threads, bits) of parallel workloads for the GPU; optimizing the memory access via a proper data layout and tiling; speeding up the computation by translating time consuming floating point operations to integer point operations and table look-ups; and finally, vectorizing bit computations with our binary format, the Compact Asynchronous Multispin coding.



**Fig. 19.** Correlation length vs. inverse temperature for the Ising model. The lines from different system sizes cross close to  $1/T = 0.2217$ , which is in agreement with the published result for the critical temperature [62].



**Fig. 20.** Binder ratio for the 3D Edwards–Anderson model. The data generated by our GPU code is compared with the data extracted from the paper by Katzgraber et al. [63].



**Fig. 21.** The convergence of the Binder ratio vs. number of Monte Carlo steps for the Edwards–Anderson model in a system with  $8^3$  sites, with and without parallel tempering for  $1/T = 2.0$ . Parallel tempering dramatically improves the convergence to equilibrium.

Our program can be extended for other models such as the Potts model and models with different random coupling distributions. The structure of our code may adapt well to the upcoming GPUs and future massive parallel accelerators.

## Acknowledgments

This work is sponsored by the NSF EPSCoR LA-SiGMA project under award number EPS-1003897. Portions of this research were conducted with high performance computational resources provided by Louisiana State University (<http://www.hpc.lsu.edu>). Part of this work was done on the Oakley system at the Ohio Supercomputer Center. We thank Helmut Katzgraber and Karen Tomko for useful discussions. We thank the following collaborators: Bhupender Thakur, Ariane Papke, Sean Hall, and Cade Thomasson. We thank Samuel Kellar for his careful reading of the manuscript.

## References

- [1] S. Monaghan, A gate-level reconfigurable Monte Carlo processor, *J. VLSI Signal Process. Syst. Signal, Image and Video technol.* 6 (1993) 139.
- [2] A.T. Ogielski, I. Morgenstern, Critical behavior of three-dimensional Ising spin-glass model, *Phys. Rev. Lett.* 54 (1985) 928.
- [3] A.T. Ogielski, Dynamics of three-dimensional Ising spin glasses in thermal equilibrium, *Phys. Rev. B* 32 (1985) 7384.
- [4] A. Cruz, J. Pech, A. Tarancón, P. Téllez, C. Ullod, C. Ungil, SUE: a special purpose computer for spin glass models, *Comput. Phys. Comm.* 133 (2001) 165.
- [5] J.H. Condon, A.T. Ogielski, Fast special purpose computer for Monte Carlo simulations in statistical physics, *Rev. Sci. Instrum.* 56 (1985) 1691.
- [6] M. Taiji, N. Ito, M. Suzuki, Special purpose computer system for Ising models, *Rev. Sci. Instrum.* 59 (1988) 2483.
- [7] T. Jörg, H.G. Katzgraber, F. Krzakala, Behavior of Ising spin glasses in a magnetic field, *Phys. Rev. Lett.* 100 (2008) 197202.
- [8] M.A. Moore, The stability of the replica-symmetric state in finite-dimensional spin glasses, *J. Phys. A: Math. Gen.* 38 (2005) L783.
- [9] A.P. Young, H.G. Katzgraber, Absence of an Almeida–Thouless line in three-dimensional spin glasses, *Phys. Rev. Lett.* 93 (2004) 207203.
- [10] T. Temesvári, Almeida–Thouless transition below six dimensions, *Phys. Rev. B* 78 (2008) 220401.
- [11] H.G. Katzgraber, Spin glasses and algorithm benchmarks: a one-dimensional view, *J. Phys. Conf. Ser.* 95 (2008) 012004.
- [12] M. Sasaki, K. Hukushima, H. Yoshino, H. Takayama, Absence of spin glass phase in the Edwards–Anderson Ising spin glass in magnetic field, *J. Magn. Magn. Mater.* 310 (2007) 1514. Proceedings of the 17th International Conference on Magnetism.
- [13] M. Sasaki, K. Hukushima, H. Yoshino, H. Takayama, Scaling analysis of domain-wall free energy in the Edwards–Anderson Ising spin glass in a magnetic field, *Phys. Rev. Lett.* 99 (2007) 137202.
- [14] D. Larson, H.G. Katzgraber, M.A. Moore, A.P. Young, Spin glasses in a field: three and four dimensions as seen from one space dimension, *Phys. Rev. B* 87 (2013) 024414.
- [15] R.A. Baños, A. Cruz, L.A. Fernandez, J.M. Gil-Narvion, A. Gordillo-Guerrero, M. Guidetti, D. Iiguez, A. Maiorano, E. Marinari, V. Martin-Mayor, J. Monforte-Garcia, A. Muñoz Sodupe, D. Navarro, G. Parisi, S. Perez-Gaviro, J.J. Ruiz-Lorenzo, S.F. Schifano, B. Seoane, A. Tarancón, P. Tellez, R. Tripiccone, D. Yllanes, Thermodynamic glass transition in a spin glass without time-reversal symmetry, *Proc. Natl. Acad. Sci.* 109 (2012) 6452.
- [16] H.G. Katzgraber, T. Jörg, F. Krzakala, A.K. Hartmann, Ultrametric probe of the spin-glass state in a field, *Phys. Rev. B* 86 (2012) 184405.
- [17] H.G. Katzgraber, D. Larson, A.P. Young, Study of the de Almeida–Thouless line using power-law diluted one-dimensional Ising spin glasses, *Phys. Rev. Lett.* 102 (2009) 177205.
- [18] L. Leuzzi, G. Parisi, F. Ricci-Tersenghi, J.J. Ruiz-Lorenzo, Ising spin-glass transition in a magnetic field outside the limit of validity of mean-field theory, *Phys. Rev. Lett.* 103 (2009) 267201.
- [19] K.A. Hawick, A. Leist, D.P. Playne, Regular lattice and small-world spin model simulations using CUDA and GPUs, *Int. J. Parallel Prog.* 39 (2011) 183.
- [20] B. Block, P. Virnau, T. Preis, Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D Ising model, *Comput. Phys. Comm.* 181 (2010) 1549.
- [21] T. Preis, P. Virnau, W. Paul, J.J. Schneider, GPU accelerated Monte Carlo simulation of the 2d and 3d Ising model, *J. Comput. Phys.* 228 (2009) 4468.
- [22] M. Weigel, Simulating spin models on GPU: a tour, *nt. J. Mod. Phys. C* 23 (2012) 1240002.
- [23] M. Weigel, Performance potential for simulating spin models on GPU, *J. Comput. Phys.* 231 (2012) 3064.
- [24] M. Baity-Jesi, R.A. Baños, A. Cruz, L. Fernandez, J. Gil-Narvion, A. Gordillo-Guerrero, M. Guidetti, D. Iiguez, A. Maiorano, F. Mantovani, E. Marinari, V. Martin-Mayor, J. Monforte-Garcia, A. Muñoz Sodupe, D. Navarro, G. Parisi, M. Pivanti, S. Perez-Gaviro, F. Ricci-Tersenghi, J. Ruiz-Lorenzo, S. Schifano, B. Seoane, A. Tarancón, P. Tellez, R. Tripiccone, D. Yllanes, Reconfigurable computing for Monte Carlo simulations: results and prospects of the Janus project, *Eur. Phys. J. Spec. Top.* 210 (2012) 33.
- [25] M. Baity-Jesi, R.A. Baños, A. Cruz, L. Fernandez, J. Gil-Narvion, A. Gordillo-Guerrero, D. Iiguez, A. Maiorano, F. Mantovani, E. Marinari, V. Martin-Mayor, J. Monforte-Garcia, A.M. Sodupe, D. Navarro, G. Parisi, S. Perez-Gaviro, M. Pivanti, F. Ricci-Tersenghi, J. Ruiz-Lorenzo, S. Schifano, B. Seoane, A. Tarancón, R. Tripiccone, D. Yllanes, Janus II: a new generation application-driven computer for spin-system simulations, *Comput. Phys. Comm.* 185 (2014) 550.
- [26] K. Binder, A.P. Young, Spin glasses: experimental facts, theoretical concepts, and open questions, *Rev. Modern Phys.* 58 (1986) 801.
- [27] S.F. Edwards, P.W. Anderson, Theory of spin glasses, *J. Phys. F* 5 (1975) 965.
- [28] S. Kirkpatrick, D. Sherrington, Infinite-ranged models of spin-glasses, *Phys. Rev. B* 17 (1978) 4384.
- [29] J.R.L. de Almeida, D.J. Thouless, Stability of the Sherrington–Kirkpatrick solution of a spin glass model, *J. Phys. A: Math. Gen.* 11 (1978) 983.
- [30] A.J. Bray, M.A. Moore, Replica-symmetry breaking in spin-glass theories, *Phys. Rev. Lett.* 41 (1978) 1068.
- [31] G. Parisi, Infinite number of order parameters for spin-glasses, *Phys. Rev. Lett.* 43 (1979) 1754.
- [32] M. Mézard, G. Parisi, N. Sourlas, G. Toulouse, M. Virasoro, Replica symmetry breaking and the nature of the spin glass phase, *J. Phys. France* 45 (1984) 843.
- [33] G. Parisi, A sequence of approximated solutions to the S–K model for spin glasses, *J. Phys. A: Math. Gen.* 13 (1980) L115.
- [34] G. Parisi, The order parameter for spin glasses: a function on the interval 0–1, *J. Phys. A: Math. Gen.* 13 (1980) 1101.
- [35] G. Parisi, Magnetic properties of spin glasses in a new mean field theory, *J. Phys. A: Math. Gen.* 13 (1980) 1887.
- [36] G. Parisi, The physical Meaning of Replica Symmetry Breaking, 2002, eprint arXiv:cond-mat/0205387.
- [37] M. Talagrand, The paris formula, *Ann. of Math.* 163 (2006) 221.
- [38] F. Guerra, Broken replica symmetry bounds in the mean field spin glass model, *Commun. Math. Phys.* 233 (2003) 1.
- [39] F. Barahona, On the computational complexity of Ising spin glass models, *J. Phys. A: Math. Gen.* 15 (1982) 3241.
- [40] R.H. Swendsen, J.-S. Wang, Replica Monte Carlo simulation of spin-glasses, *Phys. Rev. Lett.* 57 (1986) 2607.
- [41] K. Hukushima, K. Nemoto, Exchange Monte Carlo method and application to spin glass simulations, *J. Phys. Soc. Japan* 65 (1996) 1604.
- [42] E. Marinari, G. Parisi, Simulated Tempering: a new Monte Carlo scheme, *Europhys. Lett.* 19 (1992) 451.
- [43] H.G. Ballesteros, A. Cruz, L.A. Fernández, V. Martín-Mayor, J. Pech, J.J. Ruiz-Lorenzo, A. Tarancón, P. Téllez, C.L. Ullod, C. Ungil, Critical behavior of the three-dimensional Ising spin glass, *Phys. Rev. B* 62 (2000) 14237.
- [44] A.B. Harris, T.C. Lubensky, J.-H. Chen, Critical properties of spin-glasses, *Phys. Rev. Lett.* 36 (1976) 415.
- [45] H. Tasaki, On the upper critical dimensions of random spin systems, *J. Stat. Phys.* 54 (1989) 163.
- [46] J.E. Green, M.A. Moore, A.J. Bray, Upper critical dimension for the de Almeida–Thouless instability in spin glasses, *J. Phys. C: Solid State Phys.* 16 (1983) L815.
- [47] A.P. Young, H.G. Katzgraber, Absence of an Almeida–Thouless line in three-dimensional spin glasses, *Phys. Rev. Lett.* 93 (2004) 207203.
- [48] R. Zorn, H. Herrmann, C. Rebbi, Tests of the multi-spin-coding technique in Monte Carlo simulations of statistical systems, *Comput. Phys. Comm.* 23 (1981) 337.
- [49] J. Houdayer, A cluster monte carlo algorithm for 2-dimensional spin glasses, *Eur. Phys. J. B* 22 (2001) 479.
- [50] S. Liang, Application of cluster algorithms to spin glasses, *Phys. Rev. Lett.* 69 (1992) 2145.
- [51] T. Jörg, Cluster monte carlo algorithms for diluted spin glasses, *Progr. Theoret. Phys. Suppl.* 157 (2005) 349.
- [52] E. Bittner, A. Nußbaumer, W. Janke, Make life simple: unleash the full power of the parallel tempering algorithm, *Phys. Rev. Lett.* 101 (2008) 130603.
- [53] H.G. Katzgraber, S. Trebst, D.A. Huse, M. Troyer, Feedback-optimized parallel tempering monte carlo, *J. Stat. Mech. Theory Exp.* 2006 (2006) P03018.
- [54] S. Trebst, M. Troyer, U. H. E. Hansmann, Optimized parallel tempering simulations of proteins, *J. Chem. Phys.* 124 (2006) 174903.
- [55] A. Nguyen, N. Satish, J. Chhugani, C. Kim, P. Dubey, 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'10, IEEE Computer Society, Washington, DC, USA, 2010, p. 1.
- [56] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, K. Yelick, Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC'08, IEEE Press, Piscataway, NJ, USA, 2008, p. 4:1.
- [57] F. Belletti, M. Cotallo, A. Cruz, L.A. Fernández, A. Gordillo, A. Maiorano, F. Mantovani, E. Marinari, V. Martín-Mayor, A. Muñoz-Sodupe, D. Navarro, S. Pérez-Gaviro, J. J. Ruiz-Lorenzo, S.F. Schifano, D. Sciretti, A. Tarancón, R. Tripiccone, J.L. Velasco, Simulating spin systems on JANUS, an FPGA-based computer, *Comput. Phys. Comm.* 178 (2008) 208.
- [58] M. Creutz, L. Jacobs, C. Rebbi, Experiments with a gauge-invariant Ising system, *Phys. Rev. Lett.* 42 (1979) 1390.

- [59] NVIDIA, CURAND guide, Web page, documentation, 2012, [http://docs.nvidia.com/cuda/pdf/CURAND\\_Library.pdf](http://docs.nvidia.com/cuda/pdf/CURAND_Library.pdf).
- [60] J.K. Salmon, M.A. Moraes, R.O. Dror, D.E. Shaw, Parallel random numbers: as easy as 1, 2, 3, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC'11, ACM, New York, NY, USA, 2011, p. 16:1.
- [61] M. Manssen, M. Weigel, A. Hartmann, Random number generators for massively parallel simulations on gpu, Eur. Phys. J. Spec. Top. 210 (2012) 53.
- [62] A.M. Ferrenberg, D.P. Landau, Critical behavior of the three-dimensional Ising model: a high-resolution Monte Carlo study, Phys. Rev. B 44 (1991) 5081.
- [63] H.G. Katzgraber, M. Körner, A.P. Young, Universality in three-dimensional Ising spin glasses: a Monte Carlo study, Phys. Rev. B 73 (2006) 224432.