Journal of Computational Physics 228 (2009) 4468–4477

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model ☆

Tobias Preis [a,b,*], Peter Virnau [a], Wolfgang Paul [a], Johannes J. Schneider [a]

[a] Department of Physics, Mathematics and Computer Science, Johannes Gutenberg University of Mainz – Staudinger Weg 7, D-55099 Mainz, Germany
[b] Artemis Capital Asset Management GmbH – Gartenstr. 14, D-65558 Holzheim, Germany

ARTICLE INFO

Article history:
Received 9 December 2008
Received in revised form 11 March 2009
Accepted 12 March 2009
Available online 25 March 2009

MSC:
65Z05
65C05
82C20

Keywords:
Monte Carlo simulation
GPU computing
Ising model
Phase transition
Finite size scaling

ABSTRACT

The compute unified device architecture (CUDA) is a programming approach for performing scientific calculations on a graphics processing unit (GPU) as a data-parallel computing device. The programming interface allows to implement algorithms using extensions to standard C language. With continuously increased number of cores in combination with a high memory bandwidth, a recent GPU offers incredible resources for general purpose computing. First, we apply this new technology to Monte Carlo simulations of the two dimensional ferromagnetic square lattice Ising model. By implementing a variant of the checkerboard algorithm, results are obtained up to 60 times faster on the GPU than on a current CPU core. An implementation of the three dimensional ferromagnetic cubic lattice Ising model on a GPU is able to generate results up to 35 times faster than on a current CPU core. As proof of concept we calculate the critical temperature of the 2D and 3D Ising model using finite size scaling techniques. Theoretical results for the 2D Ising model and previous simulation results for the 3D Ising model can be reproduced.

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

The Ising model, which is named after Ising [1], is a standard model of statistical physics and provides a simplified microscopic description of ferromagnetism. It was introduced to explain the ferromagnetic phase transition from the paramagnetic phase at high temperatures to the ferromagnetic phase below the Curie temperature T_C. A large variety of techniques and methods in statistical physics have originally been formulated for the Ising model and were generalized and adapted to related models and problems [2]. Supported by his results for a one dimensional spin chain, in which no phase transition occurs, Ising initially proposed in his doctoral thesis that there is also no phase transition in higher dimensions which turned out to be a misinterpretation. The Ising model on a two dimensional square lattice with no magnetic field was then analytically solved by Onsager in 1944 [3]. The critical temperature at which a second order phase transition between an ordered and a disordered phase occurs can be determined analytically for the two dimensional model ($T_C = 2.269185$ [3]). Despite much effort, an analytic solution for the three dimensional Ising model still remains one of the great challenges in statistical physics. However, computer simulations in combination with finite size scaling techniques [4–7] are able to determine $T_C \sim 4.5115$ [2] and the rest of the phase diagram with good accuracy. Since 1944, the Ising

☆ The consumer graphics card GeForce GTX 280 from NVIDIA is used as graphics processing unit (GPU).

* Corresponding author. Address: Department of Physics, Mathematics and Computer Science, Johannes Gutenberg University of Mainz – Staudinger Weg 7, D-55099 Mainz, Germany.
E-mail address: preis@uni-mainz.de (T. Preis).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.03.018

model not only became popular in main stream physics but also in various interdisciplinary fields, i.e., in econophysics [8–10]. For example, a simple Ising model based simulation of an artificial stock market is presented in [11], in which specific properties of financial markets like consequences of the herding behavior can be reproduced.

Critical phenomena, scaling and universality properties of Ising models have been studied by Monte Carlo simulations since decades [2] with continuously improving accuracy, which benefited from increasing computational resources. Such computing requirements, necessary not only for Monte Carlo simulations but also for various other tasks in computational physics including, e.g., molecular dynamics simulations [12–15] or stochastic optimization [16], need a large amount of high performance computing resources. Such high performance computing resources include recent multi-core computing architectures based on shared memory, which are accessible by OpenMP [17] or MPI [18]. Distributed computing cluster solutions with homogeneous or heterogeneous node structure are available in order to parallelize applications. Additionally, supercomputing centers provide access to modern massively parallel computing architectures. A recent trend in computer science and related fields is the use of general purpose computing on graphics processing units (GPU). Impressive reduction of required processing times for tasks in computational physics can be found in [19,20,12,21]. In the beginning of general purpose computing on graphics processing units, GPU programs had to use C-like programming environments for kernel execution such as the OpenGL shading language [22] or C for graphics (Cg) [23]. The common unified device architecture (CUDA) [24] released by NVIDIA is a new programming approach which employs the unified shader design of recent GPUs provided by NVIDIA. The programming interface allows the programmer to implement an algorithm using standard C language with CUDA extensions without any knowledge about the native environment. A comparative concept "Close To the Metal" (CTM) [25] was introduced by Advanced Micro Devices Inc. for ATI graphics cards. Note that computational power of recent consumer graphics cards exceeds that of a central processing unit (CPU) by orders of magnitude. A conventional CPU provides a peak performance of around 20×10^9 floating-point operations per second (FLOPS) [12]. The current consumer graphics card NVIDIA GeForce GTX 280 on the other hand reaches a theoretical peak performance of 933×10^9 FLOPS. In this context, power supply requirements should also be considered. A GeForce GTX 280 graphics card exhibits a peak power consumption of 236 W [26], while a recent Intel Quad Core CPU consumes around 100 W.

We employ this general purpose graphics processing unit technology for Monte Carlo simulations of two dimensional square lattice and three dimensional cubic lattice Ising models. In [20] a GPU based version of the Ising model is already proposed. This implementation was able to accelerate the Ising model computation by a factor of three by migration to a GPU. In this paper, we will demonstrate that a much better acceleration factor can be obtained. We will demonstrate that our implementation works by calculating the critical temperatures of the phase transitions in the two and three dimensional Ising models.

The paper is organized as follows: In a brief overview in Section 2, key facts and further properties of the graphics processing unit architecture are provided in order to clarify implementation constraints for the following sections. In a preparation step for Monte Carlo simulations the generation of pseudo random numbers on a GPU device is described in Section 3. In Section 4 we will provide the implementation details of the two dimensional ferromagnetic square lattice Ising model including acceleration factors and determination of the critical temperature by finite size scaling. In Section 5 the Ising model implementation is expanded to three dimensions. Acceleration factors and critical temperatures will also be provided. Our conclusions are summarized in Section 6.

2. GPU device architecture

Some key facts of the GPU device architecture are briefly summarized in this section in order to provide and discuss information about implementation details on a GPU for Monte Carlo simulations. As already mentioned in the introduction, we utilize the compute unified device architecture (CUDA) released by NVIDIA. CUDA allows to implement algorithms using standard C language with CUDA specific extensions. Thus, CUDA issues and manages computations on a GPU device as a data-parallel computing device. The modern GPU generation uses a graphics card architecture, which is composed of a scalable array of so-called streaming multiprocessors [24]. One such multiprocessor contains amongst others eight scalar processor cores, a multi-threaded instruction unit, and shared memory. The NVIDIA GeForce GTX 200 series also provides one double-precision core per multiprocessor in order to support double-precision floating-point number operations. When a C program using CUDA extensions and running on the CPU core invokes a GPU kernel, which is a synonym for a GPU function, a large number of copies of this kernel, which are denoted as threads, are distributed to the available multiprocessors, where they are executed. For such a distribution, a kernel grid can be subdivided into blocks and each block is subdivided into various threads. Thread and block IDs are accessible in standard C language via built-in variables. All threads of a thread block are executed concurrently in the available multiprocessors. In order to manage the large number of threads a single-instruction multiple thread (SIMT) unit is used. A multiprocessor maps each thread to one scalar processor core and each scalar thread starts its execution independently. Threads are created, managed, scheduled, and executed by this SIMT unit in groups of 32 threads. A group of 32 threads forms a warp. Threads of one warp are handled on the same multiprocessor. If the threads of a given warp diverge by a data-induced conditional branch, each branch of the warp will be executed serially and the processing time of the given warp consists of the sum of the branches' processing times, which has to be avoided. As shown in Fig. 1, each multiprocessor contains local 32-bit registers per processor and shared memory, which is shared by all scalar processor cores of a multiprocessor. Furthermore, constant and texture cache are available for the programmer. In or-

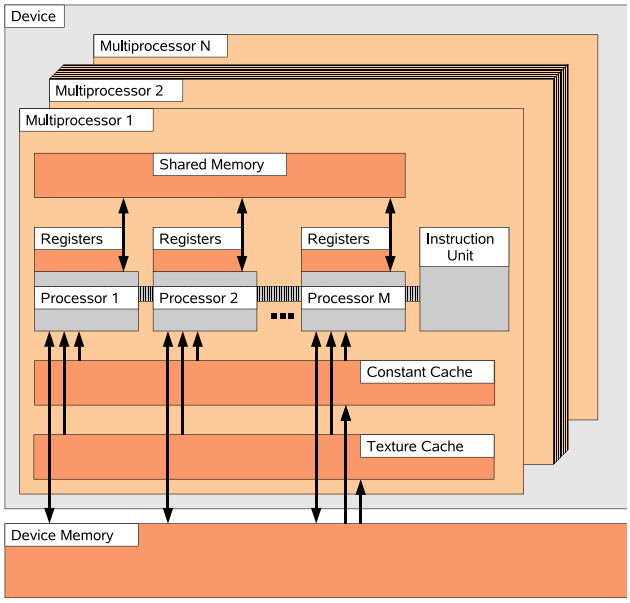

Fig. 1. Schematic visualization of a GPU device containing multiprocessors with on-chip shared memory and local registers [27].

Table 1
Key facts and properties of the consumer graphics card NVIDIA GeForce GTX 280.

	GeForce GTX 280
Global memory	1024 MB
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

der to combine calculation results of involved multiprocessors GPU global memory can be used, which is shared among all multiprocessors. It is also accessible by the main C function running on the CPU core. GPU's global memory is roughly 10 times faster than recent main memory of standard personal computers. Detailed facts of the applied consumer graphics cards NVIDIA GeForce GTX 280 can be found in Table 1. The workstation version of this card, which belongs to the NVIDIA Tesla series, offers up to 4 GB of GPU global memory, but is also considerably more expensive. It is worth noting that graphics cards older than the GeForce GTX 200 series only support single-precision floating-point operations. Furthermore the IEEE-754 standard is not realized completely [24]. Some deviations from IEEE standard numerics can be found especially for rounding operations. When calculations have to be performed with double-precision floating-point operations, one has to notice that each multiprocessor features only one double-precision processing core. Therefore, the theoretical peak performance is reduced by roughly one order of magnitude for such operations. Further information about the GPU device properties and how to implement CUDA can be found in [24]. We are using CUDA (version 2.0) in combination with NVIDIA graphics card driver (driver version 177.73). Early versions of the graphics card drivers for the NVIDIA Geforce GTX 200 series exhibit a bug if a large part of provided global memory is used. However, this problem was fixed recently. For the comparison between CPU and GPU implementation, we use an Intel Core 2 Quad CPU (Q6700) with 2.66 GHz and 4096 kB cache size, of which only one core is used. The standard C source code executed on the host is compiled with the gcc compiler with option –O3 (version 4.2.1). Other compilers in combination with more sophisticated compiler options can further improve processing times on the CPU.

3. Random number generation

An efficient method to create random numbers is essential for Monte Carlo simulations of the two dimensional ferromagnetic square lattice and the three dimensional ferromagnetic cubic lattice Ising model on a GPU in Sections 4 and 5. For this purpose, we use an array of linear congruential random number generators (LCRNGs) applying one of the oldest and best-

known algorithms for generation of pseudo random numbers [16]. Starting at a seed value $x_{0,j}$, a sequence of random numbers $x_{i,j}$ with $i \in \mathbb{N}$ of the LCRNG j can be obtained 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 in Eq. (1) is responsible for the quality of the produced random numbers. We use $a = 1664525$ and $c = 1013904223$ as suggested, e.g., in [28]. In order to exploit the local 32-bit architecture provided by the GPU device the parameter of the modulo operation m is set to 2^{32} as by construction results on a 32-bit architecture are truncated to the endmost 32 bits. Thus, pseudo random numbers $x_{i,j} \in [-2^{31}; 2^{31} - 1]$ can be obtained, which have to be normalized according to $y_{i,j} = \mathrm{abs}(x_{i,j}/2^{31}) \sim 4.656612 \cdot 10^{-10}\mathrm{abs}(x_{i,j})$ in order to get uniformly distributed pseudo random numbers $y_{i,j}$ in the interval $[0; 1)$. As we are using a whole set of linear congruential random number generators in parallel, each LCRNG j of this array is initialized by a random number obtained by a further LCRNG through

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

with $x_{0,0} = 1$.

In the GPU implementation each thread of a thread block handles its own linear congruential random number generator. Denoting s as the number of involved thread blocks and denoting σ as the number of threads per block, in a GPU kernel values of $s \cdot \sigma$ LCRNGs are determined in parallel. Each LCRNG calculates S pseudo random numbers. Thus, a total of $S \cdot s \cdot \sigma$ pseudo random numbers are created. In Fig. 2, a comparison of processing times of the random number generation between calculation on a GPU device and on a CPU core is presented for $S = 10^4$ and $\sigma = 512$ in dependence of the number of involved blocks s. On the CPU the same $S \cdot s \cdot \sigma$ pseudo random numbers are determined for comparison. The acceleration factor β, which is shown as inset, is determined by the relationship

$$\beta = \frac{\text{Total processing time on CPU}}{\text{Total processing time on GPU}}. \tag{3}$$

A maximum acceleration factor $\beta \sim 130$ is obtained for $s = 30$ which corresponds to the number of multiprocessors of the NVIDIA GeForce GTX 280 consumer graphics card as indicated in Table 1. Up to 30 groups of $\sigma = 512$ single linear congruential random number generators can be executed concurrently at the same time on this GPU device. If s is larger than 30, the 30 available multiprocessors are able to execute only the first 30 blocks in parallel, such that the remaining groups of $\sigma = 512$ single linear congruential random number generators have to be handled in a second step. Thus, processing time on the GPU device is doubled when $s = 31$ is used instead of $s = 30$. Note that the computation on the GPU becomes inefficient if s is not a multiple of the number of multiprocessors on the GPU device because some multiprocessors are idle in the

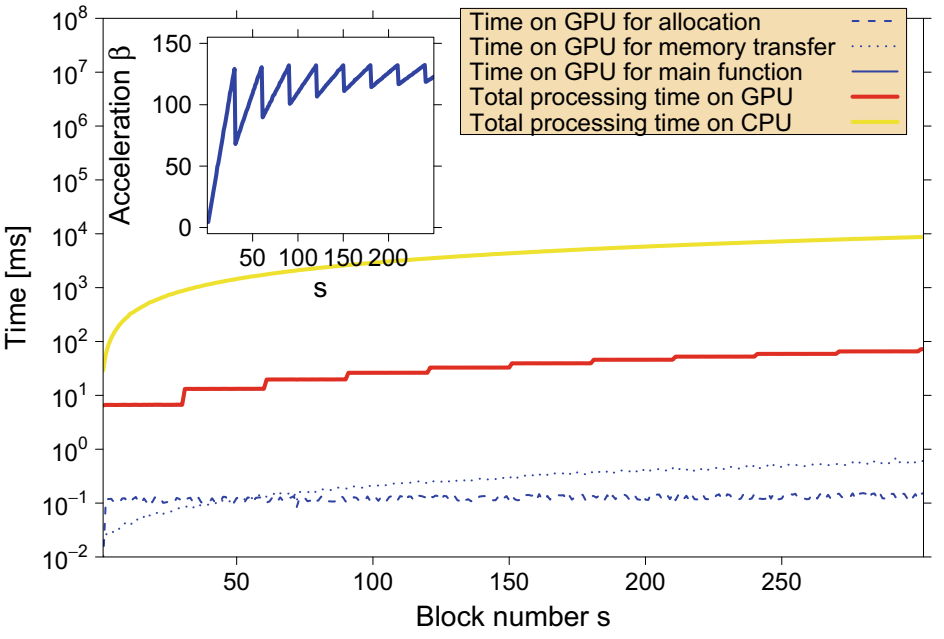

Fig. 2. Processing times for the calculation of $S \cdot s \cdot \sigma$ pseudo random numbers on GPU and CPU for $S = 10^4$ and 512 threads per block in dependence of the number of involved thread blocks s. The total processing time on GPU is divided into allocation time, time for memory transfer, and for main processing. The acceleration factor β is shown as inset. A maximum acceleration factor of roughly 130 can be obtained for $s = 30$, which corresponds to the number of multiprocessors on our GPU device. Note that the computation on the GPU becomes inefficient if s is not a multiple of the number of multiprocessors on the GPU device because some multiprocessors are idle in the end of the calculation (see inset).

end of the calculation (see inset of Fig. 2). The fraction of GPU processing time for allocation and memory transfer of the LCRNG seed values can be neglected. Please note that there are many other possibilities to realize an efficient creation of pseudo random numbers on a GPU device. In [12], e.g., an algorithm is presented in order to produce a set of pseudo random numbers in parallel on a GPU device with a single linear congruential random number generator which determines the set of pseudo random numbers according to a serial rule. A CUDA based version of the Mersenne Twister random number generator [29] is able to generate pseudo random numbers in parallel as well.

4. Two dimensional Ising model

In this section, we present an implementation of the two dimensional ferromagnetic square lattice Ising model on a GPU. The simple Ising model, which is one of the simplest lattice models, consists of an arrangement of spins with are located on the sites of the lattice and which exhibit only the values +1 and −1 [30]. These spins interact with their nearest neighbors on the lattice with interaction constant $J > 0$. The Hamiltonian $\mathcal{H}$ for this model is given by

$$\mathcal{H} = -J \sum_{\langle ij \rangle} S_i S_j - H \sum_i S_i, \tag{4}$$

where $S_i = \pm 1$ represents a spin at site i and H denotes the external magnetic field. We use a square lattice with n spins per row and column and periodic boundary conditions. The lattice contains $N = n^2$ spins. For the spin update the Metropolis criterion [31] is applied which was the first choice for a transition rate in statistical physics satisfying the 'detailed balance' principle, and which leads to the fastest dynamics for single spin flip simulations. In the expression $P_a(t)W_{a\to b} = P_b(t)W_{b\to a}$ for 'detailed balance', $P_a(t)$ denotes the probability of the system being in state a at time t and $W_{a\to b}$ the transition rate for $a \to b$. Denoting $\Delta\mathcal{H}$ the energy difference between the two states a and b, which is given through $\Delta\mathcal{H} = \mathcal{H}_b - \mathcal{H}_a$, the probability for the move $a \to b$ is given by $W_{a\to b} = \exp(-\Delta\mathcal{H}/k_B T)$ if $\Delta\mathcal{H} > 0$ and by $W_{a\to b} = 1$ if $\Delta\mathcal{H} \leqslant 0$. As a single spin flip dynamics is applied, two successive states differ only in a single spin, such that $\Delta\mathcal{H}$ can be calculated as a local energy difference.

As a first step of the GPU implementation, one has to allocate memory on the GPU device's global memory for the two dimensional spin field and for the seed values of the LCRNGs. After a random initialization of the spin field on the CPU and initialization of seed values as described in Section 3, spins and seeds are transferred to the GPU's global memory. The transition probability $W_{a\to b}$ depends on the temperature T which has to be passed to the GPU kernel function. The Boltzmann constant k_B is fixed to 1 in all simulations. We use a zero field ($H = 0$) and $J = 1$. One has to take into account that for Monte Carlo trial moves to be executed in parallel the system has to be partitioned into non-interacting domains. Thus, for the update process of the spin lattice, a checkerboard algorithm is applied, i.e., there is a regular scheme for updating the lattice spin by spin. First all spins on the 'white' squares of the checkerboard are updated and then all spins on the 'black' squares. Please note, that other methods for this updating process are also available, e.g. diverse cluster algorithms [32,33], which exhibit faster convergence. However, the systematic scheme of the checkerboard algorithm is most suitable for demonstrating the migration to the GPU architecture realizing non-interacting domains where the Monte Carlo moves are performed in parallel. In fact, this partitioning for "real life" problems is a challenge. On a GPU device, it is only possible to synchronize threads within a block and a native block synchronization does not exist. Therefore, only the termination of the GPU kernel function ensures that all blocks were executed. We divide the spin update process on the lattice into blocks on the GPU. In a single GPU kernel only a semi-lattice can be changed without creating conflicts.

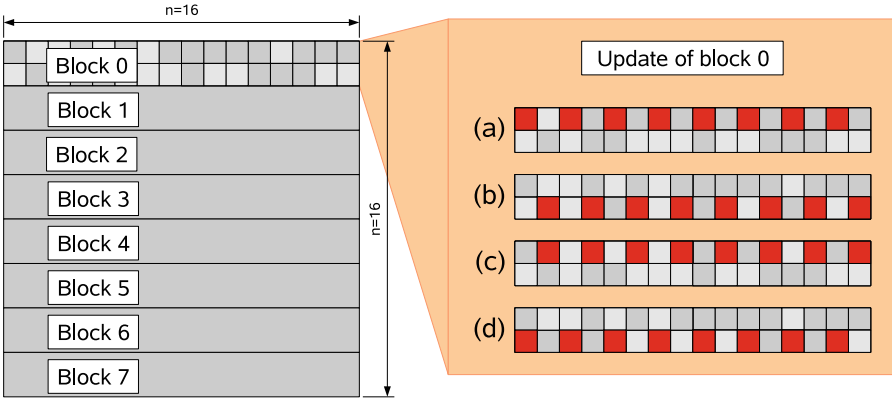

Fig. 3. Schematic visualization of the implementation of a two dimensional Ising model on a GPU for $n = 16$ spins per row. The 2D spin field is split in strips of 16×2 spins each treated by a single block on the GPU. In each block, we use $n/2 = 8$ threads, which are used for the spin update corresponding to the scheme presented on the right side and described in the main text.

The spin update process is subdivided into threads and blocks as illustrated in Fig. 3 for $n = 16$. The spin field is divided into strips of width 2. Each stripe is handled by one block. Each thread is responsible for a square sub-cell of 2×2 spins in order to avoid idle threads. Thus, $n/2$ threads per block are necessary. A first GPU kernel handles the update process of the first semi-lattice, i.e., in each block $\sigma = n/2$ threads accomplish step (a) and (b) of Fig. 3. The termination of this GPU kernel ensures that all blocks were executed and a second kernel can start in order to accomplish steps (c) and (d). Each sub-cell, i.e. each thread in each block, has access to its own LCRNG. As each thread in one GPU kernel function needs up to two random numbers and as this array is also used after two semi-lattice updates for the reduction process of partial energies or magnetizations of the Ising lattice, the seed values or current random numbers are transferred at the beginning of a GPU kernel to a shared memory array in order to profit from the increased memory speed. After handling of the updating steps, i.e. after step (b) and after step (d), the current value is transferred back to global memory. As CUDA does not provide native reduction functions for treating partial results one has to manage this manually. For this purpose, the shared memory array is used after step (d). A binary tree structure realizes a fast reduction of the partial values within a block. These partial results of each block are stored at block-dependent positions in global memory and finally transferred back to host's main memory [27]. The final summation of the results of $n/2$ blocks is done by the CPU. In Fig. 4, the processing times of the Ising model implementation on the GPU are compared with an Ising model implementation on one CPU core. For the largest system, i.e. $n = 1024$, an acceleration factor of roughly 60 can be achieved. For very small systems, the acceleration factor is smaller than 1 because for small thread numbers per block the GPU device is not used efficiently.

In order to verify the GPU implementation, it is insufficient only to compare values of energy and magnetization in dependence of the temperature T for GPU and CPU version. A very sensitive test is given by the determination of the critical temperature of the Ising model. For this purpose, we use finite size scaling and calculate the Binder cumulant [4], which is given in zero field by

$$U_4(T) = 1 - \frac{\langle M(T)^4 \rangle}{3\langle M(T)^2 \rangle^2} \tag{5}$$

with M denoting the magnetization of a configuration at temperature T and $\langle \ldots \rangle$ being the thermal average. Near a critical point, finite size scaling theory predicts the free energy and derived quantities like the magnetization to be a function of linear dimension L over correlation length $\xi \simeq (T - T_c)^{-\nu}$. Therefore, moment ratios of the magnetization like, e.g. the Binder cumulant U_4, become independent of system size at the critical temperature. To test our implementation, we perform several simulations close to the critical point for different linear dimensions of the simulation box and determine U_4. As shown in Fig. 5, the curves of the Binder cumulants for various system sizes $N = n^2$ cross almost perfectly at the critical temperature derived by Onsager ($T_c \approx 2.269185$) [3], which is indicated by a dashed line. Note that single spin flips and parallelization

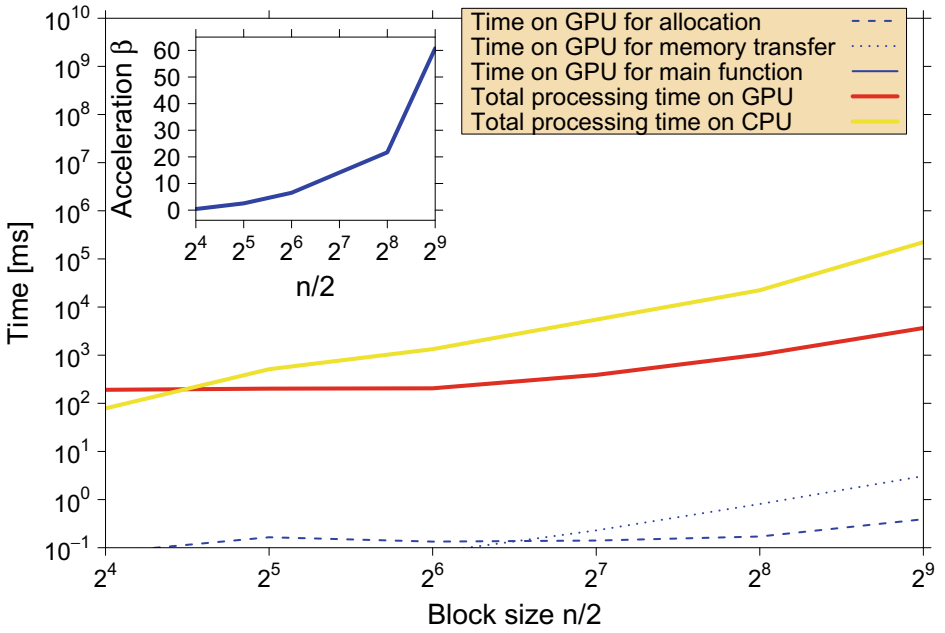

Fig. 4. Processing times for a two dimensional ferromagnetic square lattice Ising model for a cooling down process. The temperature $T \in [2.0; 3.0]$ is stepwise reduced by the factor 0.99. In each temperature step, 100 sweeps through the lattice are performed. The processing times are shown in dependence of the number of threads per block which is related to the system size by $\sigma = n/2$. The total processing time on GPU is divided into allocation time, time for memory transfer, and time for main processing. The acceleration factor β is shown as inset. A maximum acceleration factor of roughly 60 can be realized.

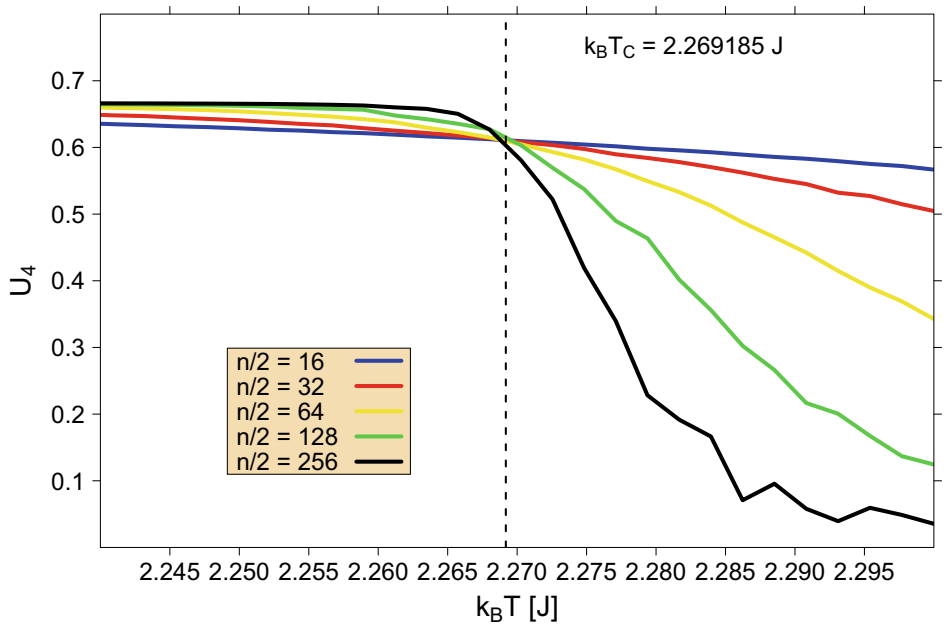

Fig. 5. Binder cumulant U_4 in dependence of k_BT for various numbers n of spins per row and column of the two dimensional square lattice Ising model. $n/2$ corresponds to the involved number of threads per block on the GPU implementation. The curves of the Binder cumulants for various system sizes $N = n^2$ cross almost perfectly at the critical temperature derived by Onsager [3], which is shown additionally as a dashed line. In each temperature step, the average was taken over 10^7 measurements.

schemes based upon them are not particularly well-suited for the determination of critical properties because of critical slowing down. Nevertheless, we were able to determine T_c with reasonable accuracy ($T_c = 2.2692 \pm 0.0002$).

5. Three dimensional Ising model

In this section, the GPU implementation of the two dimensional ferromagnetic square lattice Ising model will be expanded to a three dimensional cubic lattice model version on the GPU. In a first step, we allocate memory analogously to Section 4. In addition, the three dimensional spin field with $N = n^3$ spins and the random number seeds have to be transferred to the GPU device. In the three dimensional case, the spin update process is also subdivided into threads and blocks. Now, the update scheme as illustrated in Fig. 6 for $n = 16$ is applied. The 3D spin field is split in cuboids of $2 \times 2 \times 16$ spins each treated by a single block on the GPU. In each block, we use $n/2 = 8$ threads, which are used for the spin update

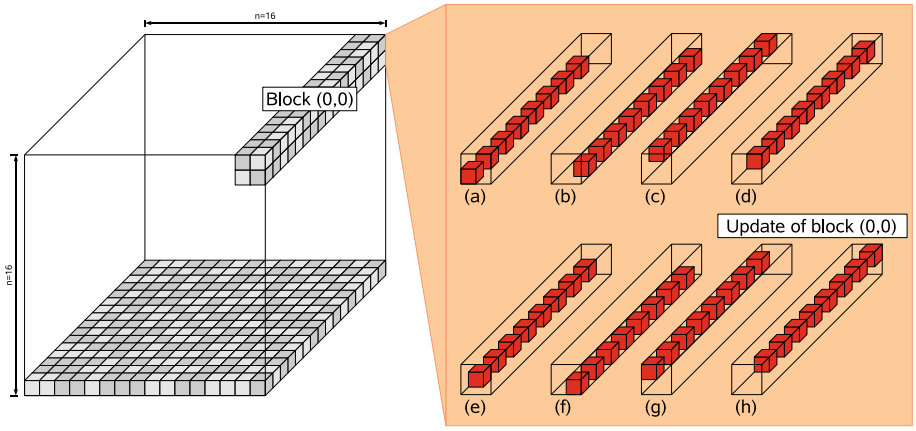

Fig. 6. Schematic visualization of the three dimensional ferromagnetic cubic lattice Ising model implementation on a GPU for $n = 16$. The 3D spin field is split in cuboids of $2 \times 2 \times 16$ spins each treated by a single block on the GPU. In each block, we use $n/2 = 8$ threads, which are used for the spin update corresponding to scheme presented on the right side and described in the main text.

corresponding to the scheme presented on the right side of Fig. 6. A first GPU kernel handles the update process of the first semi-lattice, i.e., in each block $\sigma = n/2$ threads accomplish steps (a), (b), (c), and (d) of Fig. 6. The termination of this GPU kernel ensures that all blocks were successfully executed and a second kernel can start in order to accomplish steps (e), (f), (g), and (h). Each sub-cell, i.e. each thread in each block, which consists in the three dimensional case of 8 spins has access

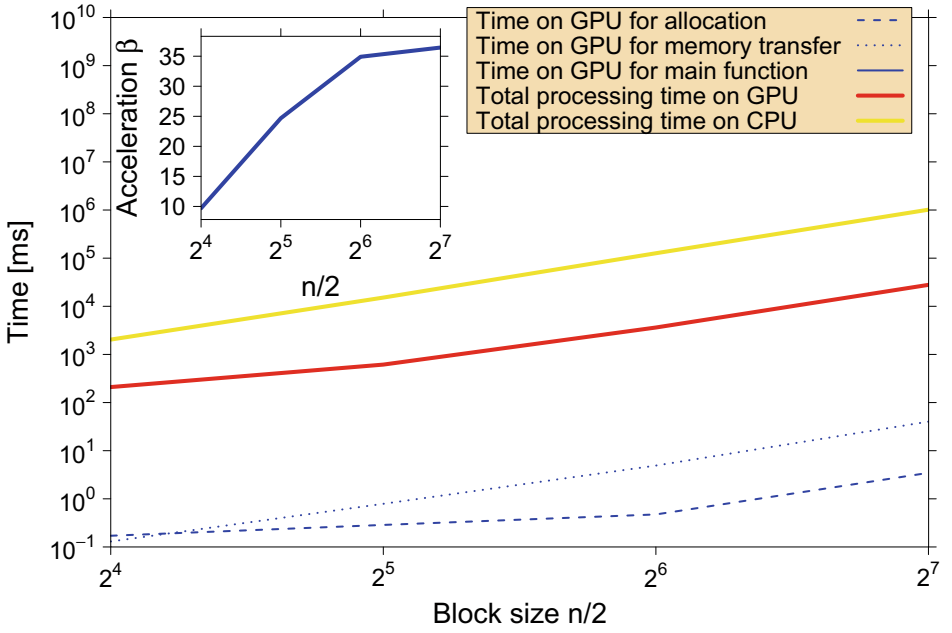

Fig. 7. Processing times for a three dimensional ferromagnetic cubic lattice Ising model for a cooling process. The temperature $T \in [4.0; 5.0]$ is stepwise reduced by the factor 0.99. In each temperature step, 100 sweeps through the lattice are performed. The processing times are shown in dependence of the number of threads per block which is related to the system size by $\sigma = n/2$. The total processing time on GPU is divided into allocation time, time for memory transfer, and time for main processing. The acceleration factor β is shown as inset. A maximum acceleration factor of roughly 35 can be achieved.

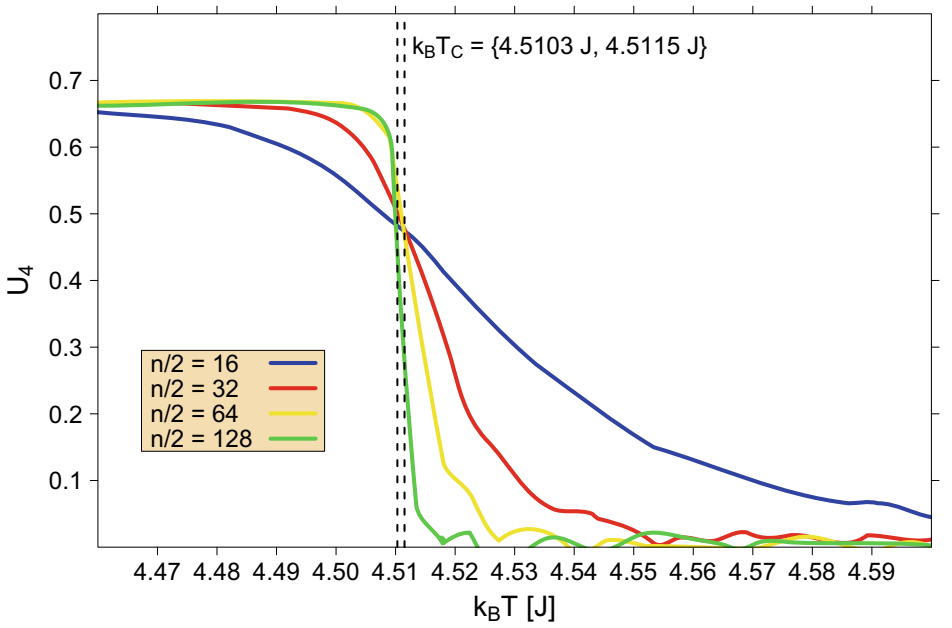

Fig. 8. Binder cumulant U_4 in dependence of k_BT for various numbers n of spins per row and column of the three dimensional Ising model on the simple cubic lattice. $n/2$ corresponds to the involved number of threads per block on the GPU implementation. The curves of the Binder cumulant for various system sizes $N = n^2$ cross almost perfectly at the critical temperature $T_c \approx 4.51$, which is in agreement with a selection of previous simulation results which are presented as dashed lines. For each temperature step, the average was taken over 10^6 measurements.

to its own LCRNG. In Fig. 7, the processing times of the three dimensional Ising model implementation on the GPU are compared with an appropriate Ising model implementation on a single CPU core. The largest system which can be realized on a GeForce GTX 280 in this way is $n = 256$. Global memory size limits the Ising system size to this value. Thus, a maximal acceleration factor of roughly 35 can be achieved.

In Fig. 8, the Binder cumulant for different system sizes $N = n^3$ is presented in dependence of temperature T. The crossing point at $T_c \sim 4.51$ is in good agreement with previous simulation results [2], according to which the critical temperature is located at 4.5115 [6] and 4.5103 [7]. These values are shown as dashed lines in Fig. 8.

6. Conclusion

We presented a GPU accelerated version of the two dimensional ferromagnetic square lattice Ising model and the three dimensional ferromagnetic cubic lattice Ising model. For the GPU implementation, the compute unified device architecture was employed which is a new programming approach for performing computations on a graphics processing unit (GPU) as a data-parallel computing device. The programming interface allows to implement algorithms using extensions to standard C language. With continuously increased number of cores in combination with a high memory bandwidth, a recent GPU offers incredible resources for general purpose computing. For our GPU based Monte Carlo simulations of the Ising model, we use a set of linear congruential random number generators on the GPU device. With the GPU implementation of a checkerboard algorithm of the two dimensional Ising model, results on the GPU can be obtained up to 60 times faster than on a current CPU core. An implementation of a three dimensional Ising model on a GPU is able to generate results up to 35 times faster than on a current CPU core. As proof of our conceptual method for the GPU implementation, the critical temperatures of the 2D and 3D Ising models were determined successfully by finite size scaling. Both the theoretical result for the 2D Ising model and previous simulation results for the 3D Ising model can be reproduced. The source codes of our implementations of the two and three dimensional Ising model for the GPU can be found on www.tobiaspreis.de. In future work, we will consider variations of the Ising model and will perform further Monte Carlo based simulations on a GPU device.

Acknowledgments

Tobias Preis thanks Kurt Binder for very fruitful discussions. This work was financially supported by the German Research Foundation (DFG) and benefited from the Forschungsfond of the Materialwissenschaftliches Forschungszentrum (MWFZ) of the Johannes Gutenberg University of Mainz.

References

[1] E. Ising, Beitrag zur theorie des ferromagnetismus, Z. Phys. 31 (1925) 253–258.
[2] K. Binder, E. Luijten, Monte carlo tests of renormalization-group predictions for critical phenomena in Ising models, Phys. Rep. 344 (2001) 179–253, doi:10.1016/S0370-157(00)00127-7.
[3] L. Onsager, Crystal statistics. I: A two-dimensional model with an order–disorder transition, Phys. Rev. 65 (3–4) (1944) 117–149, doi:10.1103/PhysRev.65.117.
[4] K. Binder, Finite size scaling analysis of Ising model block distribution functions, Z. Phys. B 43 (1981) 119–140, doi:10.1007/BF01293604.
[5] B. Fierro, F. Bachmann, E.E. Vogel, Phase transition in 2d and 3d Ising model by time-series analysis, Physica B 384 (2006) 215–217, doi:10.1016/j.physb.2006.05.265.
[6] H.-O. Heuer, Critical crossover phenomena in disordered Ising systems, J. Phys. A: Math. Gen. 26 (1993) L333–L339, doi:10.1088/0305-4470/26/6/007.
[7] M.E. Fisher, The theory of equilibrium critical phenomena, Rep. Prog. Phys. 30 (1967) 615–730, doi:10.1088/0034-4885/30/2/306.
[8] R.N. Mantegna, H.E. Stanley, An Introduction to Econophysics – Correlations and Complexity in Finance, Cambridge University Press, Cambridge, 2000.
[9] W. Paul, J. Baschnagel, Stochastic Processes: From Physics to Finance, Springer, Heidelberg, 2000.
[10] T. Preis, W. Paul, J.J. Schneider, Fluctuation patterns in high-frequency financial asset returns, Europhys. Lett. 82 (2008) 68005, doi:10.1209/0295-5075/82/68005.
[11] W.-X. Zhou, D. Sornette, Self-organizing Ising model of financial markets, Eur. Phys. J. B 55 (2007) 175–181, doi:10.1140/epjb/e2006-00391-6.
[12] J.A. van Meel, A. Arnold, D. Frenkel, S.P. Zwart, R.G. Belleman, Harvesting graphics power for md simulations, Mol. Simul. 34 (2008) 259–266, doi:10.1080/08927020701744295.
[13] H. Köstler, R. Schmid, U. Rüde, C. Scheit, A parallel multigrid accelerated poisson solver for ab initio molecular dynamics applications, Comput. Vis. Sci. 11 (2008) 115–122, doi:10.1007/s00791-007-0062-0.
[14] W. Liu, B. Schmidt, G. Voss, W. Müller-Wittig, Harvesting graphics power for md simulations, Comput. Phys. Comm. 179 (2008) 634–641, doi:10.1016/j.cpc.2008.05.008.
[15] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, K. Schulten, Accelerating molecular modeling applications with graphics processors, J. Comput. Chem. 28 (16) (2007) 2618–2640, doi:10.1002/jcc.20829.
[16] J.J. Schneider, S. Kirkpatrick, Stochastic Optimization, Springer, Berlin, 2006.
[17] L. Dagum, R. Menon, Openmp: an industry-standard api for shared-memory programming, IEEE Comput. Sci. Eng. 5 (1) (1998) 46–55, doi:10.1109/99.660313.
[18] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.D.J. Daniel, R.L. Graham, T.S. Woodall, Open MPI: goals, concept, and design of a next generation MPI implementation, in: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, 2004, pp. 97–104.
[19] J. Yang, Y. Wang, Y. Chen, GPU accelerated molecular dynamics simulation of thermal conductivities, J. Comput. Phys. 221 (2007) 799–804, doi:10.1016/j.jcp.2006.06.039.
[20] S. Tomov, M. McGuigan, R. Bennett, G. Smith, J. Spiletic, Benchmarking and implementation of probability-based simulations on programmable graphics cards, Comput. Graph. 29 (2005) 71–80, doi:10.1016/j.cag.2004.11.008.
[21] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, J. Comput. Phys. 227 (2008) 5342–5359, doi:10.1016/j.jcp.2008.01.047.

[22] R.J. Rost, OpenGL Shading Language, second ed., Addison-Wesley, Longman, Amsterdam, 2006.
[23] R. Fernando, M.J. Kilgard, The Cg Tutorial: The Definitive Guide to Programmable Real-time Graphics, Addison-Wesley, Longman, Amsterdam, 2003.
[24] NVIDIA Corporation, NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, version 2.0, 2008.
[25] Advanced Micro Devices, Inc., ATI CTM Guide, Technical Reference Manual, version 1.01, 2006.
[26] NVIDIA Corporation, NVIDIA GeForce GTX 280 Specifications, 2008.
[27] T. Preis, P. Virnau, W. Paul, J.J. Schneider, Preprint.
[28] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes: The Art of Scientific Computing, Cambridge University Press, Cambridge, 2007.
[29] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. (TOMACS) 8 (1998) 3–30.
[30] D. Landau, K. Binder, A Guide to Monte Carlo Simulations in Statistical Physics, second ed., Cambridge University Press, Cambridge, 2005.
[31] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092, doi:10.1063/1.1699114.
[32] R.H. Swendsen, J.-S. Wang, Nonuniversal critical dynamics in Monte Carlo simulations, Phys. Rev. Lett. 58 (2) (1987) 86–88, doi:10.1103/PhysRevLett.58.86.
[33] U. Wolff, Collective monte carlo updating for spin systems, Phys. Rev. Lett. 62 (4) (1989) 361–364, doi:10.1103/PhysRevLett.62.361.