Computer Physics Communications 183 (2012) 1155–1161

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

GPU-based Swendsen–Wang multi-cluster algorithm for the simulation of two-dimensional classical spin systems

Yukihiro Komura *, Yutaka Okabe

Department of Physics, Tokyo Metropolitan University, Hachioji, Tokyo 192-0397, Japan

ARTICLE INFO

Article history:
Received 20 July 2011
Received in revised form 20 December 2011
Accepted 26 January 2012
Available online 31 January 2012

Keywords:
Monte Carlo simulation
Cluster algorithm
Ising model
Parallel computing
GPU

ABSTRACT

We present the GPU calculation with the common unified device architecture (CUDA) for the Swendsen–Wang multi-cluster algorithm of two-dimensional classical spin systems. We adjust the two connected component labeling algorithms recently proposed with CUDA for the assignment of the cluster in the Swendsen–Wang algorithm. Starting with the q-state Potts model, we extend our implementation to the system of vector spins, the q-state clock model, with the idea of embedded cluster. We test the performance, and the calculation time on GTX580 is obtained as 2.51 nsec per a spin flip for the $q = 2$ Potts model (Ising model) and 2.42 nsec per a spin flip for the $q = 6$ clock model with the linear size $L = 4096$ at the critical temperature, respectively. The computational speed for the $q = 2$ Potts model on GTX580 is 12.4 times as fast as the calculation speed on a current CPU core. That for the $q = 6$ clock model on GTX580 is 35.6 times as fast as the calculation speed on a current CPU core.

1. Introduction

Computer simulation is an essential tool for studying physical properties of many-particle systems. The Metropolis-type Monte Carlo simulation [1] with a single spin flip has been a success as a standard method of simulation of many-particle systems. However, the single-spin-flip algorithm often suffers from the problem of slow dynamics or the critical slowing down; that is, the relaxation time diverges at the critical temperature. To overcome difficulty, a cluster flip algorithm was proposed by Swendsen and Wang [2]. They applied the Fortuin and Kasteleyn [3] representation to identify clusters of spins. The problem of the thermal phase transition is mapped onto the geometric percolation problem in the cluster formalism. In the cluster algorithm, spins in the cluster are updated at a time. In the Swendsen–Wang (SW) algorithm, all the spins are partitioned into clusters; thus, the SW algorithm is called the multi-cluster algorithm. Wolff [4] proposed another type of cluster algorithm, that is, a single-cluster algorithm, where only a single cluster is generated, and the spins of that cluster are updated. Although the cluster algorithm was originally formulated for the scalar order parameter, such as the Potts model, Wolff [4] introduced the idea of embedded cluster to deal with systems of vector spins, such as the classical XY model or the classical Heisenberg model.

Computational physics develops with the advance in computer technology. Recently the use of general purpose computing on graphics processing unit (GPU) is a hot topic in computer science. Drastic reduction of processing times can be realized in scientific computations. Using the common unified device architecture (CUDA) released by NVIDIA, it is now easy to implement algorithms on GPU using standard C or C++ language with CUDA specific extension.

Preis et al. [5] studied the two-dimensional (2D) and three-dimensional (3D) Ising models by using the Metropolis algorithm with CUDA. They used a variant of sublattice decomposition for a parallel computation on GPU. The spins on one sublattice do not interact with other spins on the same sublattice. Therefore one can update all spins on a sublattice in parallel when making the Metropolis simulation. As a result they were able to accelerate 60 times for the 2D Ising model and 35 times for the 3D Ising model compared to a current CPU core. Recently, the GPU acceleration of the multispin coding of the Ising model was discussed [6]. Moreover, many attempts for simulating lattice spin models on GPU using the Metropolis algorithm were reported [7–9].

Since the Metropolis algorithm has the problem of slow dynamics as mentioned above, and this problem becomes conspicuous with increasing the system size, it is highly desirable to apply the GPU-based calculation to cluster algorithms. Only limited trials have been reported so far. The present authors [10] have proposed the GPU-based calculation with CUDA for the Wolff single-cluster algorithm, where parallel computations are performed for the newly added spins in the growing cluster. Hawick et al. [11] have studied the CUDA implementation of the Wolff algorithm using a modified connected component labeling for the assignment

* Corresponding author.
E-mail addresses: y-komura@phys.se.tmu.ac.jp (Y. Komura), okabe@phys.se.tmu.ac.jp (Y. Okabe).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2012.01.017

of the cluster. They put more emphasis on the hybrid implementation of Metropolis and Wolff updates and the optimal choice of the ratio of both updates. Quite recently, Weigel [12] has studied parallelization of cluster labeling and cluster update algorithms for calculations with CUDA. He realized the SW multi-cluster algorithm by using the combination of self-labeling algorithm and label relaxation algorithm or hierarchical sewing algorithm.

In this paper, we present the GPU-based calculation with CUDA for the SW multi-cluster algorithm of 2D classical spin systems. We realize the SW cluster algorithm by using the connected component labeling algorithm for the assignment of clusters. The rest of the paper is organized as follows. In Section 2, we briefly describe the standard way of implementing the SW algorithm on CPU. In Section 3, we explain two types of connected component labeling which are used in the present calculation, and the idea of implementing the SW cluster algorithm on GPU. In Section 4, we compare the performance of GPU calculation with that of CPU calculation. The summary and discussion are given in Section 5.

2. Swendsen–Wang cluster algorithm

We start with the Potts model whose Hamiltonian is given by

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} (\delta_{S_i,S_j} - 1), \quad S_i = 1, 2, \ldots, q, \tag{1}$$

and for $q = 2$ this corresponds to the Ising model. Here, J is the coupling and S_i is the Potts spin on the lattice site i. The summation is taken over the nearest neighbor pairs $\langle i, j \rangle$. Periodic boundary conditions are employed.

Swendsen and Wang proposed a Monte Carlo algorithm of multi-cluster flip [2]. There are three main steps in the SW algorithm: (1) Construct a bond lattice of active or non-active bonds. (2) The active bonds partition the spins into clusters which are identified and labeled using a cluster-labeling algorithm. (3) All spins in each cluster are set randomly to one of q. The cluster identification problem is a variant of connected component labeling, which is an algorithmic application of graph theory. For an efficient cluster-labeling algorithm, the Hoshen–Kopelman algorithm [13], which was first introduced in context of cluster percolation, is often used. The Hoshen–Kopelman algorithm is a special version of the class of union-and-find algorithms [14], and has an advantage over other methods in low computer memory usage and short computational time.

The actual spin-update process of the SW cluster algorithm on a CPU can be formulated as follows [15,16]:

(i) Choose a site i.
(ii) Look at each of the nearest neighbors j. If S_j is equal to S_i, generate bond between site i and j with probability $p = 1 - e^{-\beta}$, where β is the inverse temperature J/T.
(iii) Choose the next spin and go to (i) until all sites are checked.
(iv) Apply the Hoshen–Kopelman algorithm [13] to identify all clusters.
(v) Choose a cluster.
(vi) Assign the spins S_i in the cluster to one of q with probability $1/q$.
(vii) Choose another cluster and go to (vi) until all clusters are checked.
(viii) Go to (i).

The procedures from (i) to (iii) correspond to the step of active bond generation. The procedure (iv) corresponds to the step of cluster labeling. Those from (v) to (vii) correspond to the step of spin flip.

In the Hoshen–Kopelman cluster-labeling algorithm, integer labels are assigned to each spin in a cluster. Each cluster has its own distinct set of labels. The proper label of a cluster, which is defined to be the smallest label of any spin in the cluster, is found by the following function. The array `label` is used, and if `label` is a label belonging to a cluster, the `label[label]` is the index of another label in the same cluster which has a smaller value if such a smaller value exists. The proper label for the cluster is found by evaluating `label[label]` repeatedly.

3. GPU calculation of the Swendsen–Wang cluster algorithm

Since the calculations of the step of active bond generation and the step of spin flip are done independently on each site, these steps are well suited for parallel computation on GPU. On the other hand, in the step of cluster labeling the assignment of label of cluster is done on each site piece by piece sequentially; thus the cluster-labeling algorithm such as the Hoshen–Kopelman algorithm cannot be directly applied to the parallel computation on GPU.

Recently, Hawick et al. [17] studied the cluster-labeling algorithm efficient for GPU calculation. Checking four implementations of multi-pass labeling method, they proposed the labeling method of "Label Equivalence", which is the most efficient among four proposals. The procedure of their algorithm is explained in Fig. 1. Their algorithm consists of three kernel functions, that is, scanning function, analysis function and labeling function, and two variables for labeling; one is a variable for saving the label, "label" in Fig. 1, and the other is a temporal variable for updated label, "R" in Fig. 1. The scanning function compares the label of each site with that of the nearest-neighbor sites when the bond between each site and the nearest-neighbor site is active. If the label of the nearest-neighbor site is smaller than the label of that site, the temporal variable with the label number, R[label[index]] in Fig. 1, is updated to the smallest one. For the update of the temporal variable on the scanning function, the atomic operation `atomicMin()` is used. Atomic operations provided by CUDA are performed without interference from any other threads. The analysis function resolves the equivalence chain of "R" obtained in the scanning function; the temporal variable `R[index]` is updated from the starting site to the new site, which is similar to the method of the Hoshen–Kopelman algorithm. Each thread checks the temporal variable and the label on each site. When the label number, "label", is equal to the thread number, "index", each thread tracks back the temporal variable until the temporal variable, "R", remains unchanged. Since each thread executes this operation concurrently, the final value is reached quickly. The labeling function updates the label for saving by `label[index]` ← `R[label[index]]`. In the cluster-labeling algorithm due to Hawick et al., the loop including three functions is iterated up to the point when the information of the labeling needs no more process of scanning function. A small number of iterations are needed; 4096×4096 systems with free boundary conditions were labeled in 9 or less iterations [17].

More recently, Kalentev et al. [18] reported the refinement of the algorithm due to Hawick et al. The procedure of their algorithm is shown in Fig. 2. First, they used only one variable for labeling instead of two because there is no need for a temporal reference; the implementation was improved in terms of memory consumption. It means that the number of kernel functions are reduced from three to two because the process of the labeling function is no more needed. Second, they changed the execution condition on the analysis function from "when `label[index]` is equal to `index`" to "when `label[label]` is not equal to `label`". Finally, they eliminated the atomic operation. The update of labeling is executed up to the point when the labeling needs no more process of the scanning function; thus even if collision between threads happens because of the absence of the atomic operations, it will be resolved during the next iterative step. With the