Computer Physics Communications 183 (2012) 1568–1573

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Massively parallel Wang–Landau sampling on multiple GPUs

Junqi Yin [a,b,*], D.P. Landau [a]

[a] Center for Simulational Physics, University of Georgia, Athens, GA 30602, USA
[b] Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA

ARTICLE INFO

Article history:
Received 4 February 2011
Received in revised form 23 February 2012
Accepted 24 February 2012
Available online 3 March 2012

Keywords:
Wang–Landau sampling
GPU computing
Water clusters
Phase transition

ABSTRACT

Wang–Landau sampling is implemented on the Graphics Processing Unit (GPU) with the Compute Unified Device Architecture (CUDA). Performances on three different GPU cards, including the new generation Fermi architecture card, are compared with that on a Central Processing Unit (CPU). The parameters for massively parallel Wang–Landau sampling are tuned in order to achieve fast convergence. For simulations of the water cluster systems, we obtain an average of over 50 times speedup for a given workload.

1. Introduction

Wang–Landau sampling [1] as a novel uniform sampling method has been drawing increasing interest in simulations [2] of both discrete and continuous systems with complex free energy landscapes. The advantage over traditional Monte Carlo methods, such as Metropolis sampling, lies in the fact that in Wang–Landau sampling the system behaves like a random walker in energy and/or other parameter space, instead of being constrained by the Boltzmann distribution. Therefore, a broad parameter space can be effectively sampled in a single simulation. A similar idea has been applied to other general ensemble sampling techniques, such as multi-canonical sampling [3]. And by directly estimating the density of states, one can obtain thermal quantities, such as the free energy, as continuous functions of the temperature and/or other conjugated field variables. Furthermore, the algorithm can be easily implemented on multiple processors and the parallelization is also straightforward. There are already parallel Wang–Landau applications reported in the literature [4], in which several, or even hundreds of random walkers, are sampling simultaneously with a common density of states. However, the convergence of such parallel scheme was not carefully tested before. In this paper, we find that the originally proposed parameters for the single Wang–Landau simulation would not ensure convergence to the correct density of states for the corresponding parallel algorithm, especially in the case of a massively parallel simulation, and we propose a remedy for this potential pitfall.

On the other hand, the GPU device provides an ideal platform for massively parallel applications. For Monte Carlo simulations, there are several studies reported for lattice magnetic models [5], since a regular lattice with short range interactions can be naturally divided into blocks and hence run simultaneously following a checkerboard scheme. For off-lattice models, the parallelization usually comes with the appropriate algorithm, such as the parallel tempering or the Wang–Landau method, and it does not depend on the specific system.

In this paper, we implement a parallel Wang–Landau algorithm on multiple GPUs for the simulation of water clusters. In the next section, we shall review some appropriate background for both the water model and the Wang–Landau method, and in Section 3 we present our multiple GPUs implementations using Message Passing Interface (MPI). In Section 4, we show results on convergence and performance. We summarize and conclude in Section 5.

2. Water cluster model and Wang–Landau method

Simulations of water clusters can be challenging due to large numbers of minima in the potential energy surface of the system [7]. The low temperature behavior is also of interest since it sheds light on inherent structures of water in non-bulk-like environments, and water models themselves as well. In our previous study [9], we compared structural and thermodynamic properties for several empirical water models, and found that Transferable Intermolecular Potential 4 Point (TIP4P), a widely used two-body rigid-monomer potential, provides good properties with relatively small computational cost. For more details, please refer to Ref. [9] and references therein.

There are four interaction sites in TIP4P model: one negative charge site q_M (not on the Oxygen site but extended by the

* Corresponding author at: Center for Simulational Physics, University of Georgia, Athens, GA 30602, USA.
E-mail address: yinj@ornl.gov (J. Yin).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2012.02.023

Table 1
Parameters for water potential models TIP4P.

r_{OH} (Å)	∠HOH (deg)	$A \times 10^{-3}$ (kcal Å^{12}/mol)	C (kcal Å^6/mol)	q_O	q_H	q_M	r_{OM} (Å)
0.9572	104.52	600	610	0.0	0.52	−1.04	0.15

distance r_{OM} along the bisector line of HOH angle), two positive charge sites q_H and the Oxygen site for the Lennard–Jones interaction. The potential energy ϵ_{mn} for interaction sites m and n of water molecule i and j, respectively, is given by [6]

$$\epsilon_{mn} = \frac{q_m q_n e^2}{r_{ij}} + \frac{A}{r_{OO}^{12}} - \frac{C}{r_{OO}^{6}}$$

where r_{ij} is the distance between two interaction sites, and the charge q and Lennard–Jones parameters (A and C) are listed in Table. 1.

To prevent the evaporation of water molecules, a "hard wall" constraining sphere is usually employed [7]. But for relatively large clusters ($N > 10$), it is difficult to choose a proper radius such that the constraining sphere is neither so small that it distorts the structures nor so big that evaporation is still problematic. Instead, in this study, we use "pressure" to prevent evaporation. Similar to the uniform sampling of volume V in the NPT ensemble [8], the biased distribution for the uniform sampling of energy E would be

$$p_{\text{bias}} = \frac{\exp(-\beta PV)}{g(E)} \frac{Q(NVT)}{Q(NPT)}$$

where Q is the partition function for the corresponding ensemble, g is the density of states and β is the inverse temperature. We defined V^* as V/σ^{*3} and P^* as $\beta P \sigma^{*3}$ ($\sigma^* = 3.154$ Å for TIP4P), and let T approach infinity (every state of the system has the same probability). Then, the ratio of the two partition functions is proportional to V^N. Therefore, we have the transition probability acc(b → a) from state "b" to state "a"

$$\text{acc}(\text{b} \to \text{a}) = \min\left[1, \frac{g(E_b)}{g(E_a)} \frac{V_a^{*N}}{V_b^{*N}} \exp\left(-P^*\left(V_a^* - V_b^*\right)\right)\right].$$

In addition to the translational and rotational moves of a single molecule, a volume change move is also introduced to maintain constant P^*. For each type of move, the trial step length is generated from a uniformly distributed random number in the range of $(-1, 1)$ multiplying the maximum step length which is chosen properly such that the average acceptance rates are about $20 \sim 40\%$. From the analysis of data obtained, the dimension of the containing box is the same within error bars for different temperatures, and it depends only on the value of P^*. For small clusters, there are suitable values for the constraining radius reported [7], and by comparing average box sizes for different values of P^*, we find 0.1[1] $|\sigma^{*3}|$ is a value that prevents evaporations and provides the same estimate of the specific heat within error bars [9].

During the Wang–Landau sampling, the modification factor is reduced to $\ln(f) \to 10^{-7}$, and for each iteration we adopt a flatness criterion that requires the minimum energy histogram entry to be no less than $60 \sim 80\%$ of the histogram mean. Each Monte Carlo step consists of a sweep of every energy bin with translational and rotational moves for every molecule and a volume change move, and we choose the bin size and range for the energy such that the total number of bins is 2000. During the simulation, configurations with 100 lowest energies are saved as well and are used as initial states for the optimization procedures [10]: direction set and conjugate gradient method, to find the local minima. Then, the overall minima would be our estimate of the ground states.

Monte Carlo simulations have been performed on $(H_2O)_N$, $N = 6 \sim 50$ with ground state searching. For each case, 3 to 5 independent runs are taken to calculate standard statistical error bars, and by comparisons with our previous study [9] for small clusters, it seems to be sufficient to provide a good estimate of the order of magnitude of the errors. In all the plots of data and analysis shown in following sections, if error bars are not shown they are always smaller than the size of the symbols.

3. Multiple GPU implementation

The first issue that needs to be attended to for a Monte Carlo simulation is the choice of a proper random number generator, and on GPUs, the implementation itself is not trivial. One natural candidate is the linear congruential generator, since it is simple to realize and takes only one or two integers to save the intermediate state on the device memory. For instance, the multiplicative, congruential random number generator RANECU (two integers for the state) has good properties and is suitable for small to medium applications [5]. However, in our simulations, thousands of replicas of the system run simultaneously, so the desired case would be each replica has an independent random number generator rather than a portion of the same sequence from a single random number generator. The Mersenne Twister (MT) [11] with dynamically generated parameters is an ideal match for such application, and the sample implementation can be found in CUDA software development kit. MT requires 21 integers to store the intermediate state for each thread, so it runs slightly slower than RANECU. But the MT code [12] we use provides a random number sequence with period 2^{607} and can accommodate up to 4096 independent random walkers with different MT parameters (essentially different random number generators).

Fig. 1 illustrates the scheme of a four GPU implementation. On the GPU level, each card possesses thousands of threads, with each thread representing a random walker. In the plot, these walkers share the same histogram $H(E)$ and the density of states $g(E)$, which can result in substantial reduction in the performance due to the memory access conflict. So, in practice, a "gather" pattern is employed where each thread has an independent memory space for $H(E)$ and $g(E)$. Every LEV1 Monte Carlo steps, a reduction procedure is invoked to synchronize the information across all threads. Hence the net effect would be all the walkers share the same $H(E)$ and $g(E)$. On the MPI level, each GPU works as a big random walker, and their accumulated $H(E)$ and $g(E)$ also need to be synchronized every LEV2 Monte Carlo steps. Note that the CUDA and MPI are two orthogonal components, which means GPUs can not communicate with each other directly through MPI and the synchronization is actually across CPUs by MPI. The choice of LEV1 and LEV2 depends on the tradeoff between the overhead in each level of synchronizing and the rate of convergence to the true density of states.

There are several other issues that need to be considered for applications on a GPU. For instance, since on the current generation GPU, a single floating point operation is much faster than that in double precision (on a Fermi architecture card, it is about two times faster), a mixture precision calculation is often used to improve the performance. This should be done with great caution. At least, the compensated summation [13] needs to be employed

[1] 0.1 is in the unit of atm/k_B/K, where k_B is the Boltzmann constant and 1 atm/k_B/K $\simeq \frac{1}{136.26}$ Å^{-3}.

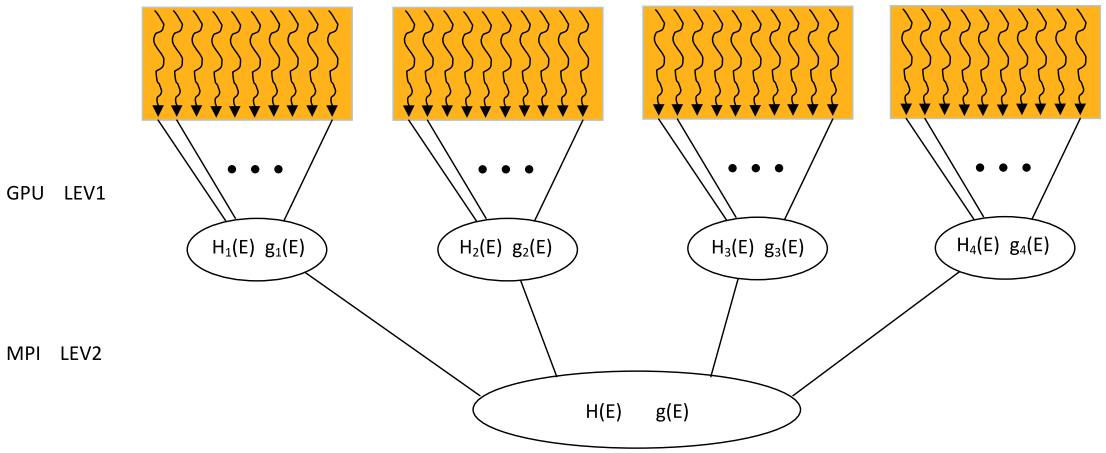

Fig. 1. Schematic plot of a four GPU implementation of the parallel Wang–Landau algorithm, where $H(E)$ is the energy histogram and $g(E)$ is the density of states. Each rectangle stands for a GPU card which contains thousands of threads.

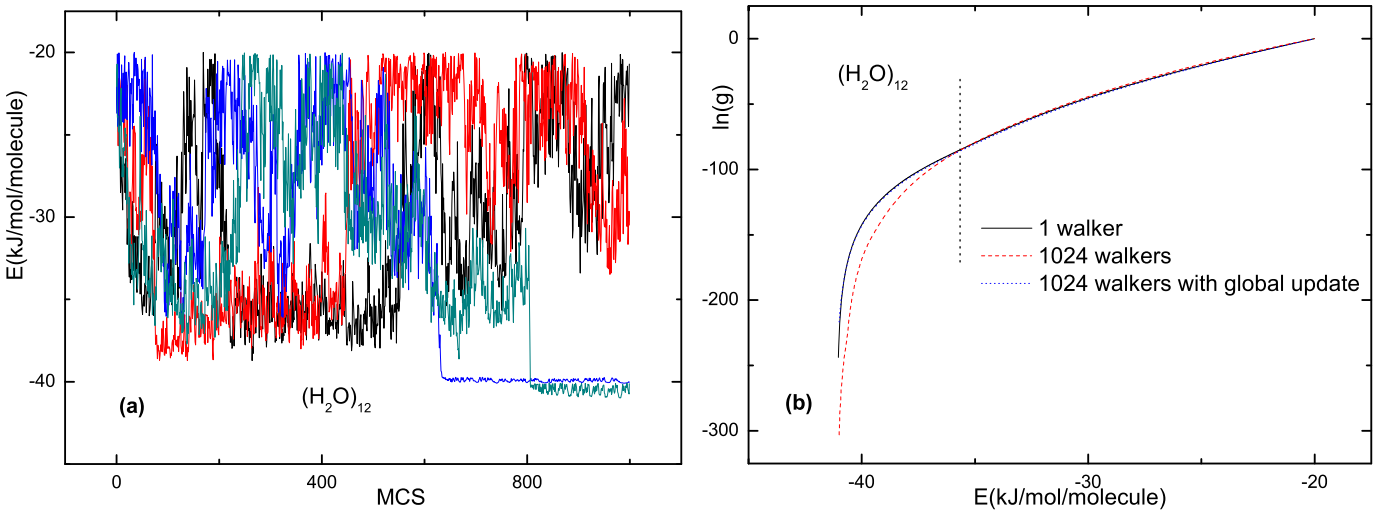

Fig. 2. (a) Energy time series of 4 typical random walkers (different color) in a simulation of totally 32 random walkers. (b) Density of states from simulations with the same Wang–Landau parameters (modification factor, flatness criterion and number of iterations) but different number of random walkers with or without global update.

in the calculation of the total energy of the continuous system to minimize the truncation error. Also, for large applications, issues like register overspilling, memory coalescing, etc., could greatly affect the performance.

4. Results

4.1. Convergence problem for regular Wang–Landau parameters

In fact, the parallel Wang–Landau sampling has already been used in several applications [4]. It seems that the originally proposed Wang–Landau parameters work fine in those specific cases; however, we find this is not generally true, especially for simulations with large numbers of parallel random walkers. In Fig. 2(a), we plot the time series of the energy for several walkers in the simulation of $(H_2O)_{12}$ clusters with 32 walkers in total. It shows that some walkers seem to get "stuck" at lower energy, which signals the underestimation of the density of states in the corresponding energy region, as shown in Fig. 2(b). The explanation is that although in Wang–Landau sampling every energy state is given the same probability, given a certain energy the ability for a walker to visit the neighboring energy state is not the same [14]. Usually, the lower energy region (in our case, the "solid" phase) is much more difficult to approach, as a result of which the density of states tends to be underestimated in the first few iterations. Therefore, once a walker gets in the region, it will stay there to rectify the underestimation. The problem is the regular Wang–Landau parameters cannot ensure convergence to the true density of states in parallel Wang–Landau simulations. In Wang–Landau simulation, the flatness of the energy histogram is usually defined as the number of visits of the least visited energy over average visits. In parallel Wang–Landau sampling, typically, each walker samples a portion of the entire energy range. If due to the roughness of the energy landscape, few walkers can access certain energy region, as shown in Fig. 2(a), then there could be a trend in the overall energy histogram. Although according to the aforementioned criterion it is "flat", the density of states does not converge. Of course one could make the flatness criterion more stringent, e.g., requiring that every walker visits the entire energy range. It may fix the problem, but it also cancels the efficiency brought by parallelism in the first place.

One remedy follows the idea of frontier sampling (also known as the global update) [2]. At the beginning of each iteration, an artificial distribution could be added to the density of states to force more walkers to sample towards the lower energy region. As shown in Fig. 2(b), it appears to fix the problem; but the choice of the additive distribution seems to be arbitrary and the system is very sensitive to it. If it is too small, then the lower energy region

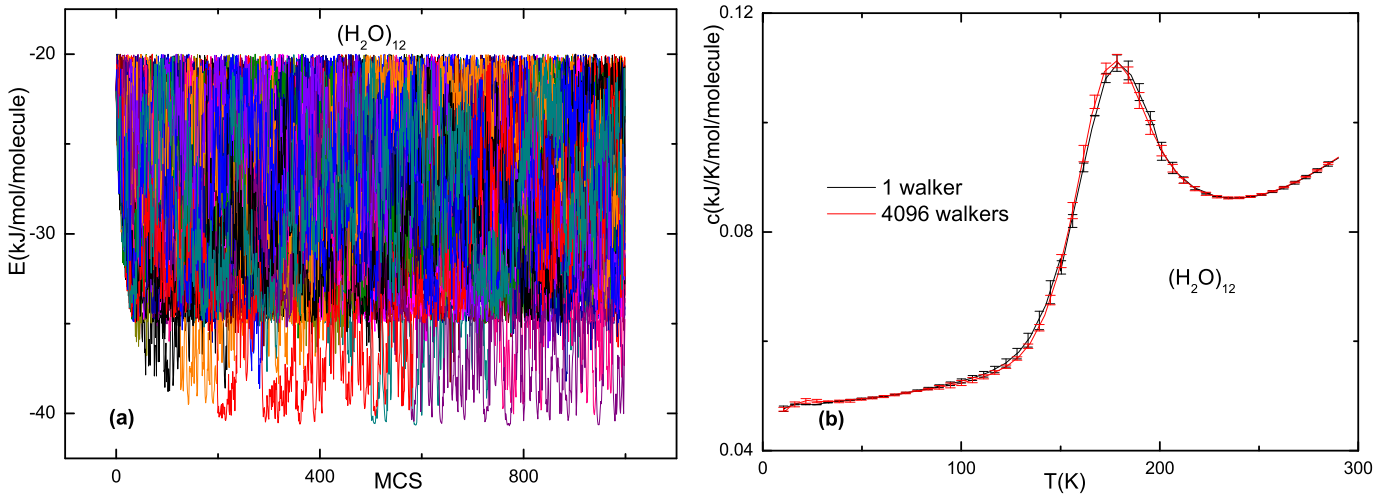

Fig. 3. $(H_2O)_{15}$: (a) Energy time series for all 32 random walkers (different color). (b) Specific heat curves for single Wang–Landau and parallel Wang–Landau with 4096 random walkers.

is still underestimated; If it is too big, then the density of states is distorted too much and a long time is needed to converge.

4.2. Parallel Wang–Landau with non-uniform modification factor

We propose to use a non-uniform modification factor f for the parallel Wang–Landau sampling. In the energy region where a walker is harder to approach, a bigger modification factor is added to the density of states for each visit, hence a faster update rate in the region. One natural choice is to use a step function for the modification factor, and we find it already works well for the water cluster system. In simulations, $\log(f)$ and $\log(g)$ are usually used and our iteration factor would be $(\kappa\Theta(E_0 - E) + 1)\log(f)$, where Θ is the Heaviside step function. Basically, $\log(g)$ is updated κ times faster where $E < E_0$. The determination of E_0 depends on the prior knowledge, such as the average energy for the interested transition temperature, or one can perform some preliminary analysis of the energy time series. Also, the value of κ need to be tested. A big value won't overshoot the density of states, just takes longer to converge. From the updating scheme $\log(g) \leftarrow \log(g) + \log(f)$, the final value of the density of states need to be corrected following $\log(g) \leftarrow \log(g) + \log(\kappa)\Theta(E_0 - E)$, and correspondingly, $H \leftarrow H/(1 + \kappa\Theta(E_0 - E))$ for histogram.

For $(H_2O)_{12}$ clusters, we use $E_0 = -35$ kJ/mol/molecule and $\kappa = 50$, and obtain the correct convergence efficiently. In Fig. 3, we plot the corrected energy time series, and the specific heat curve. The peak in the specific heat corresponds to a "melting transition" from "solid" to "liquid". For small size clusters, the transition temperatures are significantly smaller than those of bulk water, and generally different for different water potentials [9].

Note that one side effect of the big iteration factor is that the statistical error in the lower energy region is also magnified. If the low energy is of interest, extra iterations may be necessary. Luckily, it won't take much time for massively parallel Wang–Landau simulations. We also test the reverse scheme where instead of increasing f in the lower energy region, we decrease it in the higher energy region. For $(H_2O)_{12}$, we use $\kappa = 0.2$ for $E > -35$ kJ/mol/molecule, which also works well, and it seems the reverse scheme is even more robust.

4.3. Performance comparisons

In CUDA thread Hierarchy [15], threads are grouped into blocks, and blocks are further grouped into grids. Usually thousands of threads are needed for a GPU application to run efficiently, and deciding how to divide the threads into (grid, block) configuration actually affects the performance. In Fig. 4(a), we plot the computing time per MC step for different setups on a Tesla C2050 card. From the results, generally, using a big block is not bad, but for large applications there are often not enough resources to accommodate so many threads within a block, hence a comparable grid to block size gives the best performance for our code. For parallel Wang–Landau applications, the frequency of two consecutive synchronization of the information, such as the density of states, the energy histogram, etc., will also affect the efficiency. Basically, it is a tradeoff between the overhead of the reduction and the convergence rate, as shown in Fig. 4b. Synchronizing too often or too seldom will reduce the performance, and every several hundreds of MC steps seems to be the best frequency for our application.

In Fig. 5(a), we compare the performance of a single thread CPU (i7-930) code with the single GPU implementation on three different cards. GTX285 and Tesla S1070 have comparable performance, which is about 30 times faster than i7-930, and Tesla C2050 (Fermi architecture) outperforms the other two GPU cards by almost a factor of two. The main reason is that the Fermi card offers twice the amount of streaming processor cores and provides an order of magnitude faster peak performance in the double precision (we use the mixture precision). To show the scalability of our implementation, we plot the number of MC steps per second versus the number of GPU cards used, as shown in Fig. 5(b). Each card runs with 1024 threads, and the overall performance scales linearly with the number of cards.

To further illustrate the efficiency of the GPU implementation, we study a much bigger cluster $(H_2O)_{50}$. The required number of MC steps increases dramatically when the sampled lower energy bound approaches to the ground state. For instance, to obtain the low temperature thermodynamics of $(H_2O)_{30}$ cluster [9], it takes about one week to converge the density of states on an 8-CPU InfiniBand cluster. With the GPU implementation, we perform similar calculations for $(H_2O)_{50}$ in about two days on a Tesla C2050 card. The specific heat and two snapshots for corresponding temperatures and another snapshot for ground state structure (after quenching to the local minima by direction set and conjugate gradient methods), are plotted in Fig. 6. The "melting transition" seems to be very smooth, which is related to distributions of inherent structures near the transition temperatures [7]. In fact, as the size of water cluster increases, the transition behavior approaches to bulk water [9]. The ground state energy we find is -46.05 kJ/mol/molecule.

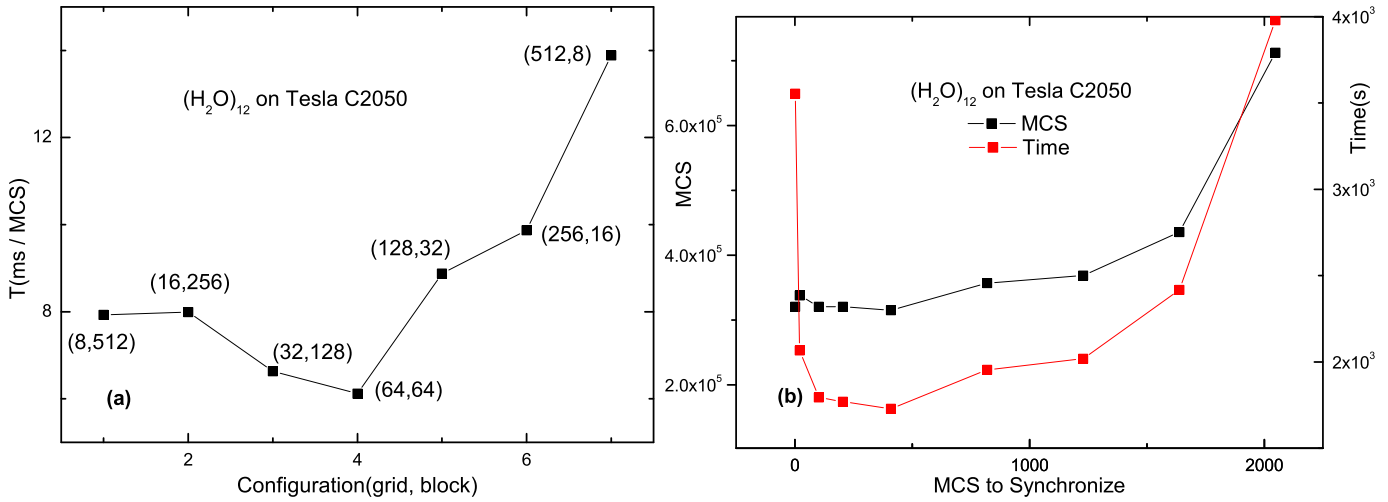

Fig. 4. (a) Computing time per MCS for different (grid, block) configurations on Tesla C2050. (b) Total number of MCS and computing time versus number of MCS between two consecutive synchronizations of the energy histogram and the density of states.

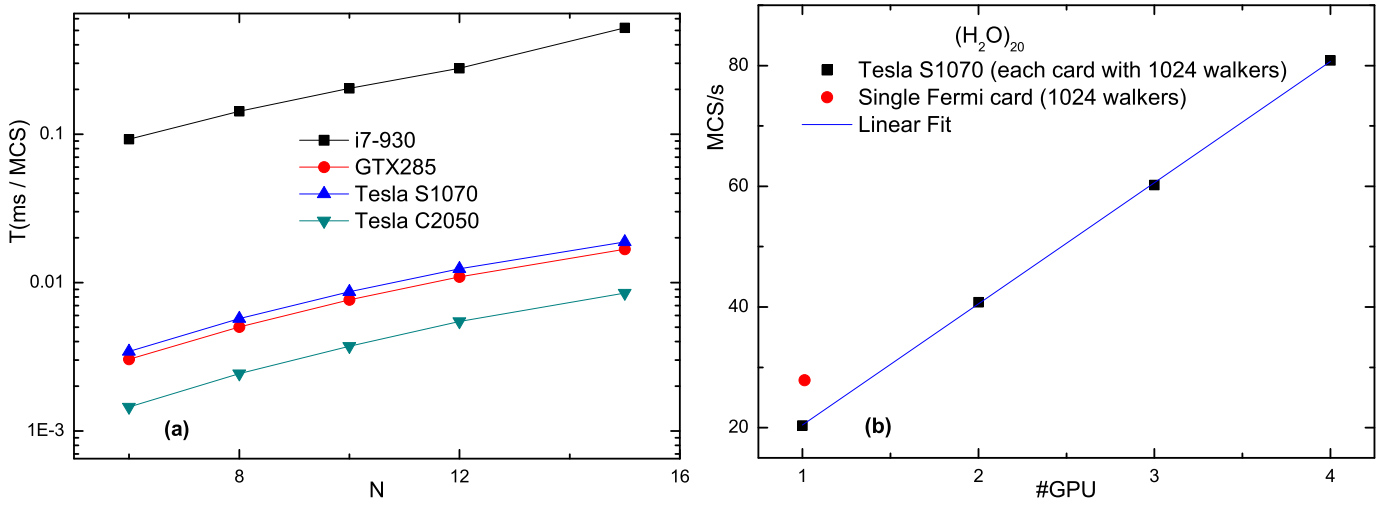

Fig. 5. (a) Computing time per MCS for i7-930 CPU versus three types of GPU. (b) MCS per seconds versus the number of Tesla S1070 GPU cards.

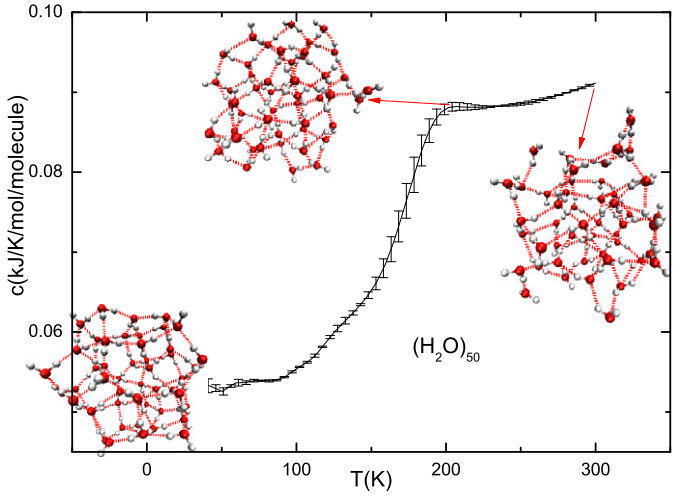

Fig. 6. Specific heat curve for $(H_2O)_{50}$. Two snapshots (with pointing arrow) are taken at the average energy for corresponding temperatures, respectively, and another snapshot (leftmost) is for the ground state structure after quenching. The red ball stands for an Oxygen atom, the white ball for a Hydrogen atom, and the dash line for the hydrogen bonding.

5. Conclusion

We presented a multiple GPU implementation of parallel Wang–Landau sampling. A potential pitfall for such applications is pointed out in that single Wang–Landau parameters would not ensure the convergence in a parallel Wang–Landau simulation. We propose using a non-uniform modification factor to accelerate the convergence. We should mention that the choice of parameters may vary for different systems, and the optimization of such process still requires further studies. By choosing an appropriate form of the modification factor, the convergence is restored and we obtain an average of over 50 times speedup for a given workload compared to a single thread code on an intel i7-930. We were able to obtain both the ground state structure and the thermodynamics of $(H_2O)_{50}$. Tesla C2050 with the new generation Fermi Architecture outperforms the old GT200 Architecture (GTX285, Tesla S1070) by about a factor of two. According to the test on Tesla S1070 four GPU clusters, our implementation scales linearly with the number of GPU's.

Acknowledgements

We thank D. Seaton for sharing codes, S. Schnabel for fruitful discussions and T. Vogel for critically reading the manuscript. Multiple GPU calculations were performed on a GPU cluster provided by the Research Computing Center (RCC) at the University of Georgia. We are grateful to S.-H. Tsai for her support at RCC. This work was supported by NSF grants Nos. OCI-0904685 and DMR-0810223.

References

[1] F.G. Wang, D.P. Landau, Phys. Rev. Lett. 86 (2001) 2050;
F.G. Wang, D.P. Landau, Phys. Rev. E 64 (2001) 056101;
D.P. Landau, S.-H. Tsai, M. Exler, Amer. J. Phys. 72 (2004) 1294.

[2] M.S. Shell, P.G. Debenedetti, A.Z. Panagiotopoulos, Phys. Rev. E 66 (2002) 056703;
Q. Yan, J.J. de Pablo, Phys. Rev. Lett. 90 (2003) 035701;
M. Troyer, S. Wessel, F. Alet, Phys. Rev. Lett. 90 (2003) 120201;
F. Rampf, W. Paul, K. Binder, Europhys. Lett. 70 (2005) 628;
D.F. Parsons, D.R.M. Williams, J. Chem. Phys. 124 (2006) 221103;
C.G. Zhou, T.C. Schulthess, S. Torbrugge, D.P. Landau, Phys. Rev. Lett. 96 (2006) 120201;
C.G. Zhou, T.C. Schulthess, D.P. Landau, J. Appl. Phys 99 (2006) 08H906;
T. Wust, D.P. Landau, Phys. Rev. Lett. 102 (2009) 178101;
C. Desgranges, J. Delhommelle, J. Chem. Phys. 120 (2009) 244109;
D.T. Seaton, T. Wuest, D.P. Landau, Phys. Rev. E 81 (2010) 011802;
F. Lou, P. Clote, Bioinformatics 26 (2010) i278;
T. Hoppe, J. Yuan, J. Phys. Chem. B 115 (2011) 2006;
M. Eisenbach, D.M. Nicholson, A. Rusanu, G. Brown, J. Appl. Phys. 109 (2011) 07E138.

[3] B.A. Berg, T. Neuhaus, Phys. Lett. B 267 (1991) 249;
B.A. Berg, T. Neuhaus, Phys. Rev. Lett. 68 (1992) 9;
B.A. Berg, Int. J. Mod. Phys. C 3 (1992) 1083.

[4] L. Zhan, Comput. Phys. Comm. 179 (2008) 339;
M. Eisenbach, C.-G. Zhou, D.M. Nicholson, G. Brown, J. Larkin, T.C. Schulthess, in: SC, Portland, Oregon, USA, November 14–20, 2009, ACM, New York.

[5] T. Preis, P. Virnau, W. Paul, J.J. Schneider, J. Comp. Phys. 228 (2009) 4468;
B. Block, P. Virnau, T. Preis, Comput. Phys. Comm. 181 (2010) 1549;
J. Yin, D.P. Landau, Phys. Rev. E 80 (2009) 051117;
J. Yin, D.P. Landau, Phys. Rev. E 81 (2010) 031121;
M. Bernaschi, G. Parisi, L. Parisi, Comput. Phys. Comm. 182 (2011) 1265;
M. Weigel, Comput. Phys. Comm. 182 (2011) 1833;
E.E. Ferrero, J.P.D. Francesco, N. Wolovick, S.A. Cannas, arXiv:1101.0876;
E.E. Ferrero, J.P.D. Francesco, N. Wolovick, S.A. Cannas, M. Weigel, J. Comp. Phys. 231 (2012) 3064.

[6] W.L. Jorgensen, J. Amer. Chem. Soc. 103 (1981) 335;
W.L. Jorgensen, J. Chem. Phys. 77 (1982) 4156.

[7] C.J. Tsai, K.D. Jordan, J. Chem. Phys. 95 (1991) 3850;
J.M. Pedulla, K.D. Jordan, Chem. Phys. 239 (1998) 593;
A.N. Tharrington, K.D. Jordan, J. Phys. Chem. A 107 (2003) 7380.

[8] G. Ganzenmüller, P.J. Camp, J. Chem. Phys. 127 (2007) 154504.

[9] J. Yin, D.P. Landau, J. Chem. Phys. 134 (2011) 074501.

[10] W. Press, S. Teukolsky, W. Vetterling, B. Flannery, Numerical Recipes in C, Cambridge University Press, NY, USA, 1992.

[11] M. Matsumoto, T. Nishimura, ACM Trans. Modeling Comput. Simulations 8 (1998) 3;
M. Matsumoto, T. Nishimura, Monte Carlo and Quasi-Monte Carlo Methods, Springer, 2000, 56 pp.

[12] V. Podlozhnyuk, Parallel Mersenne Twister, NVIDIA CUDA SDK 1.0, 2007.

[13] W. Kahan, Comm. ACM 8 (1965) 40.

[14] S. Trebst, D.A. Huse, M. Troyer, Phys. Rev. E 70 (2004) 046701.

[15] NVIDIA Corporation, NVIDIA CUDA Programming Guide, version 3.1.1, 2010.