Parallelized Stochastic Cutoff Method for Long-Range Interacting Systems

Eishin Endo¹ *, Yuta Toga¹, and Munetaka Sasaki²

¹Department of Applied Physics, Tohoku University, Sendai 980-8579, Japan
²Faculty of Engineering, Kanagawa University, Yokohama 221-8686, Japan

We present a method to parallelize the stochastic cutoff (SCO) method, which is a Monte-Carlo method for long-range interacting systems. After interactions are eliminated by the SCO method, we subdivide the lattice into non-interacting interpenetrating sublattices. This subdivision enables us to parallelize Monte-Carlo calculation in the SCO method. Such subdivision is found by numerically solving the vertex coloring of a graph created by the SCO method. We use an algorithm proposed by Kuhn and Wattenhofer to solve the vertex coloring by parallel computation. The present method was applied to a two-dimensional magnetic dipolar system on an \( L \times L \) square lattice to examine its parallelization efficiency. The result showed that, in the case of \( L = 2304 \), the speed of computation increased about 102 times by parallel computation with 288 processors.

1. Introduction

It is widely recognized that recent trend in computational physics is parallel computing with a large number of computational resources. This recognition is supported by the fact that all of top 100 supercomputers released in November 2014 consists of more than ten thousand cores.¹ Moreover, computation with the graphics processing unit (GPU) is a hot topic in recent years²–¹⁰ because it enables us to perform massively parallel computing at a fraction of the cost. To fully utilize these parallel architectures, the development of efficient parallel algorithms is indispensable. Such parallel algorithms are required particularly in long-range interacting systems because of their high computational cost.

One example in which the parallelization of computation has been successfully achieved is the molecular dynamic (MD) method (see Refs. 11, 12 and references therein). If there are only short-range forces, parallel computations are performed by dividing the simulation box into cubic domains and assigning each domain to each processor.¹³ If the system involves long-range forces like the Coulomb ones, long-range forces for all the molecules are efficiently calculated with \( O(N \log N) \) or \( O(N) \) computational time (\( N \) is the number of molecules) by sophisticated methods such as the Barnes-Hut tree algorithm,¹⁴–¹⁶ the first multipole method,¹⁷¹⁸ the particle mesh Ewald method,¹⁹,²⁰ and so forth. Because these methods can be parallelised, it is possible to perform parallel computations in the MD methods even in the presence of long-range forces.

The main reason why parallel computations in the MD methods are relatively simple lies in their simultaneous feature. In the MD methods, forces acting on all the molecules are calculated at the beginning of each step, and new positions of molecules in the next time step are determined simultaneously by use of forces calculated in advance. In contrast, in a normal Monte-Carlo (MC) method, elements such as particles and spins are moved one at a time. This sequential feature of the MC method makes parallelization difficult. If the system involves only short-range interactions, it is still possible to perform parallel computations in the MC method. For example, MC simulations in lattice systems can be parallelised by a checkerboard decomposition.²¹ Even in off-lattice systems, parallel computations are still possible by employing a spatial decomposition method.²²–²⁵ The spatial-decomposition technique is also used in the kinetic (event-driven) MC method in short-range interacting systems to parallelize the computation.²⁶–²⁹ However, such spatial-decomposition method does not work in long-range interacting systems because all the elements interact with each other. Furthermore, the above-mentioned efficient algorithms used in the MD method to calculate long-range forces does not work in the MC method because these methods calculate long-range forces (and potentials) for all the elements at once. In the MC methods, long-range forces calculated for all the elements become invalid after a part of elements are updated because of the sequential feature of the MC method. Therefore, it is difficult for long-range interacting systems to perform parallel computations in a normal (and most widely applicable) single-update MC method.

To overcome the difficulty, we utilize the stochastic cutoff (SCO) method.²⁰–³² The SCO method is a Monte-Carlo method for long-range interacting lattice systems. The basic idea of the method is to switch long-range interactions \( V_{ij} \) stochastically to either zero or a pseudointeraction \( \tilde{V}_{ij} \) by using the Stochastic Potential Switching (SPS) algorithm.³³,³⁴ The SPS algorithm enables us to switch the potentials with the detailed balance condition strictly satisfied. Therefore, the SCO method does not involve any approximation. Fukui and Todo have developed an efficient MC method based on similar strategy by use of different pseudointeractions and different way of switching interactions.³⁵ Because most of the distant and weak interactions are eliminated by being switched to zero, the SCO method greatly reduces the number of interactions and computational time in long-range interacting systems. For example, in a two-dimensional magnetic dipolar system, to which we will apply our MC method later, the...
number of potentials per spin and the computational time for a single-spin update is reduced from \( O(N) \) to \( O(1) \).

This reduction of potentials also makes it possible to subdivide the lattice into non-interacting interpenetrating sublattices, i.e., so that the elements on a single sublattice do not interact with each other. This subdivision enables us to parallelize the computation. However, a problem is that there is no trivial way to find such subdivision. In the case of Ising models on a square lattice with nearest-neighbouring interactions, the checkerboard decomposition is responsible for it. In contrast, there is no trivial subdivision in the present case because interactions are stochastically switched. To resolve the problem, we numerically solve the vertex coloring on a graph created by the potential switching procedure. This computation is performed in a parallel fashion by using an algorithm proposed by Kuhn and Wattenhofer.\(^{36,37}\)

The organization of the paper is as follows: In \( \S 2 \), we briefly explain the SCO method, and describe the parallel computation of the vertex coloring, which is a key of the present method. In \( \S 3 \), we show the results obtained by applying the present method to a two-dimensional magnetic dipolar system. Section 4 is devoted to conclusions.

2. Methods

2.1 Stochastic Cutoff (SCO) Method

In this subsection, we briefly explain the SCO method. We consider a system with pairwise long-range interactions described by a Hamiltonian \( \mathcal{H} = \sum_{i<j} V_{ij}(S_i, S_j) \), where \( S_i \) is a variable associated with the \( i \)-th element of the system. In the SCO method, \( V_{ij} \) is stochastically switched to either 0 or a pseudointeraction \( \bar{V}_{ij} \) as

\[
V_{ij}(S_i, S_j) = \begin{cases} 
0 & \text{prob. : } P_{ij}(S_i, S_j), \\
\bar{V}_{ij}(S_i, S_j) & \text{prob. : } 1 - P_{ij}(S_i, S_j).
\end{cases}
\]

The probability \( P_{ij} \) and the pseudointeraction \( \bar{V}_{ij} \) are given by

\[
P_{ij}(S_i, S_j) = \exp[\beta(V_{ij}(S_i, S_j) - V_{ij}^{\text{max}})],
\]

\[
\bar{V}_{ij}(S_i, S_j) = V_{ij}(S_i, S_j) - \beta^{-1} \log[1 - P_{ij}(S_i, S_j)],
\]

where \( \beta \) is the inverse temperature and \( V_{ij}^{\text{max}} \) is a constant equal to (or greater than) the maximum value of \( V_{ij} \) over all \( S_i \) and \( S_j \). With this potential switching process, the algorithm proceeds as follows:

(A) Potentials \( V_{ij} \) are switched to either 0 or \( \bar{V}_{ij} \) with the probability of \( P_{ij} \) or \( 1 - P_{ij} \), respectively.

(B) Perform a standard MC simulation with the switched Hamiltonian

\[
\mathcal{H}' = \sum_{ij}' \bar{V}_{ij}(S_i, S_j),
\]

for \( n_{\text{sw}} \) MC steps, where \( \sum_{ij}' \) runs over all the potentials switched to \( \bar{V}_{ij} \) and one MC step is defined by one trial for each \( S_i \) to be updated.

(C) Return to (A).

In the SCO method, an efficient method is employed to reduce the computational time of the potential switching in step (A) (see Ref. 30 for details). As a result, the computational time in step (A) becomes comparable to that in step (B) per one MC step. For example, in the case of a two-dimensional magnetic dipolar system, both the computational times are reduced to \( O(N) \).

2.2 Outline of parallel computations of the SCO method

Figure 1 shows a schematic illustration of parallel computations of the SCO method. A vertex denotes a variable \( S_i \) and an edge denotes a potential \( V_{ij} \) or \( \bar{V}_{ij} \). In step (1), each potential is switched to either 0 or \( \bar{V}_{ij} \). The edges whose potentials are switched to 0 are eliminated in the subsequent steps. In step (2), the edge denoted by a specific color is solved numerically in a parallel fashion. In step (3), variables with a specific color are updated simultaneously by a standard MC simulation. This procedure is done for all the colors.

2.3 Parallel computation of the vertex coloring

In this subsection, we briefly explain parallel computation of the vertex coloring. We refer the reader to the book in Ref. 37 for more details. By solving the vertex coloring in a parallel fashion, we can perform all the steps mentioned in the previous subsection by parallel computation. We hereafter call the vertex coloring by parallel computation distributed graph coloring.

The organization of this subsection is as follows: In \( \S 2.3.1 \),
we explain the basis of the distributed graph coloring. In §2.3.2, we explain a basic color reduction algorithm for the distributed graph coloring. This algorithm is used in an algorithm proposed by Kuhn and Wattenhofer, which is used in the present study. This algorithm is explained in §2.3.3.

2.3.1 Basis of the distributed graph coloring

We start with the introduction of several technical terms in the graph theory. The degree of a vertex is the number of edges which connect the vertex with other ones, and the maximum degree is the largest value of the degrees of a graph. In general, it is known that a graph with a maximum degree $\Delta$ can be colored with $\Delta + 1$ colors, while, in most cases, it is not the smallest number of colors needed to color the graph. The aim of the distributed graph coloring is to color a graph with $\Delta + 1$ colors by parallel computation.

In the distributed graph coloring, each vertex is initially colored by different colors, i.e., a graph is colored with $N$ colors, where $N$ is the number of vertices. The number of colors is gradually reduced from $N$ to $\Delta + 1$ by repeating synchronous communication and parallel computation. In the synchronous communication, the vertices communicate with each other to know the colors of their neighboring vertices. In the parallel computation, all the vertices simultaneously recolor themselves. The new color is locally calculated by using the information of the neighboring colors obtained in the preceding communication. The vertices do not communicate with each other in this parallel computation. The number of times of synchronous communications required to accomplish a ($\Delta + 1$)-coloring is called running time. We hereafter denote it by $t_R$. The main aim of the distributed graph coloring is to reduce $t_R$ as much as possible.

2.3.2 Basic color reduction algorithm

Basic color reduction (BCR) algorithm is one of the most fundamental algorithms for the distributed graph coloring. Figure 2 shows a graph and its coloring to which we apply the BCR algorithm. We suppose that the number of vertices and the maximum degree of the graph are $N$ and $\Delta$, respectively. The graph is initially colored with $\alpha$ colors ($\Delta + 1 < \alpha \leq N$) and the coloring is legal, i.e., no adjacent vertices share the same color. The color of a vertex is specified by an integer between 1 and $\alpha$. In the BCR algorithm, the number of colors is reduced from $\alpha$ to $\alpha - 1$ by the following steps:

1. Each vertex communicates with each other to obtain the colors of the neighboring vertices.

2. Each vertex recolors itself if its color is $\alpha$. The new color is chosen from a palette between 1 and $\Delta + 1$ by using the information obtained in step (1).

The steps (1) and (2) correspond to synchronous communication and parallel computation in the previous subsection, respectively. We can always choose a new color among $\Delta + 1$ colors because the maximum degree of the graph is $\Delta$. It is also important to notice that the vertices with the color $\alpha$ can not be adjacent to each other because the initial coloring is legal (see the vertices enclosed by a solid circle in Fig. 2). This means that the new coloring is also legal even if each vertex with the color $\alpha$ simultaneously changes its color according to the information of the neighboring colors. The BCR algorithm reduces the number of colors by one at a time by repeating these two steps. Therefore, the running time $t_R$ to accomplish a ($\Delta + 1$)-coloring from an initial $N$-coloring is $N - \Delta - 1$.

When we implemented the BCR algorithm in our simulation, we slightly modified the algorithm to improve its efficiency. To be specific, we modified the step (2) in the following way:

(2) Each vertex recolors itself if its color is locally maximum. The new color is chosen from a palette between 1 and $\Delta + 1$ by using the information obtained in step (1).

In Fig. 2, the vertices recolored in step (2)' and those recolored in step (2) are enclosed by dashed circles and solid ones, respectively. We see that the former involves the latter. This means that the running time is reduced by this modification. We also find in Fig. 2 that the vertices recolored in step (2)' are not adjacent to each other because they are locally maximum. Therefore, the new coloring is also legal by the same reason as before. A demerit of this modification lies in the computational cost to check whether the color of a vertex is locally maximum or not. However, this demerit was not significant in our simulations because the degrees of graphs were not so large.

2.3.3 KW algorithm

In this subsection, we explain an algorithm proposed by Kuhn and Wattenhofer. We hereafter call it the KW algorithm. The KW algorithm greatly reduces the running time $t_R$ by applying the BCR algorithm recursively. As mentioned above, we used this algorithm to numerically solve the vertex coloring.

Figure 3 shows a schematic illustration of the KW algorithm. For simplicity, we assume that the number of vertices $N$ and the maximum degree $\Delta$ are related by $N = (\Delta + 1) \times 2^M$, where $M$ is an integer. Generalization to other...
cases is straightforward. We suppose that all the vertices are initially colored by different colors. The color is specified by an integer between 1 and \( N \). The KW algorithm starts with partitioning all the vertices into \( N/(\Delta + 1) = 2^M \) groups according to their colors. We hereafter denote them by \( G_0(k) \) \((k = 1, 2, \cdots, 2^M)\), where the subscript ‘0’ represents the level of the grouping. In this partitioning, the vertices whose color is between 1 \((k-1)(\Delta + 1)\) and \(k(\Delta + 1)\) are assigned to the \(k\)-th group \( G_0(k) \). We next make groups at level 1 by integrating two adjacent groups at level 0 (see Fig. 3). We denote them by \( G_1(k) \) \((k = 1, 2, \cdots, 2^{M-1})\). Just after the integration, 2(\(\Delta + 1\)) vertices in a group \( G_1(k) \) is colored by 2(\(\Delta + 1\)) colors. We then apply the BCR algorithm to reduce the number of colors from 2(\(\Delta + 1\)) to \(\Delta + 1\).

We next consider the running time of the KW algorithm. At a level \( p \), there are \(2^{M-p}\) groups. In each group, the number of colors is reduced from 2(\(\Delta + 1\)) to \(\Delta + 1\) by the BCR algorithm. Now the point is that we can simultaneously perform BCR’s in all the \(2^{M-p}\) groups. To be specific, we simultaneously perform BCR’s in all the groups to reduce the number of colors by one, and perform synchronous communication just once for the next color reductions. By repeating this procedure \(\Delta + 1\) times, we can reduce the numbers of colors of all the groups from 2(\(\Delta + 1\)) to \(\Delta + 1\). The running time to achieve this color reduction at the level \( p \) is \(\Delta + 1\). Because the number of levels is \( M \), the total running time to reduce the number of colors from \( N \) to \(\Delta + 1\) is estimated to be

\[
t_T = (\Delta + 1) \times M = (\Delta + 1) \log_2 \left( \frac{N}{\Delta + 1} \right). \quad (5)
\]

where we have used the relation \( N = (\Delta + 1)2^M \). If \( N \gg \Delta \), this running time is much shorter than that of the BCR algorithm, which is, as mentioned above, of the order of \( N \).

3. Results

3.1 Model

To investigate the efficiency of parallel computation of the method developed in the present study, we apply the method to a two-dimensional magnetic dipolar system on an \( L \times L \) square lattice with open boundaries. The Hamiltonian of the system is described as

\[
\mathcal{H} = -J \sum_{\langle ij \rangle} S_i \cdot S_j + D \sum_{i,j} \left[ \frac{S_i \cdot S_j}{r_{ij}^3} - \frac{3}{r_{ij}^5} \frac{(S_i \cdot r_{ij})(S_j \cdot r_{ij})}{r_{ij}^5} \right], \quad (6)
\]

where \( S_i \) is a classical Heisenberg spin of \( |S_i| = 1 \), \( \langle ij \rangle \) runs over all the nearest-neighbouring pairs, \( r_{ij} \) is the vector spanned from a site \( i \) to \( j \) in the unit of the lattice constant \( a \), and \( r_{ij} = |r_{ij}| \). The first term describes short-range ferromagnetic exchange interactions and the second term describes long-range dipolar interactions, where \( J(> 0) \) is an exchange constant and \( D(> 0) \) is a constant which represents the strength of magnetic dipolar interactions. We hereafter consider the case that \( D/J = 0.1 \). We choose this model because it was used as a benchmark of the SCO method.\(^{30}\) It is established that the model undergoes a phase transition from a paramagnetic state to a circularly ordered state at \( T_c \approx 0.88 J \) as a consequence of the cooperation of exchange and dipolar interactions.\(^{39}\) We applied the SCO method only for magnetic dipolar interactions. The system was gradually cooled from an initial temperature \( T = 1.25 J \) to 0.05\( J \) in steps of \( 0.05 J \). The initial temperature was set to be well above the critical temperature. We set \( n_{sw} \) defined in §2.3 to be 100, i.e., potential switching and subsequent vertex coloring are
Table 1. Temperature dependence of the mean degree \( \langle k \rangle \) and the maximum degree \( \langle \Delta \rangle \). Graphs are created by the potential switching in the SCO method. The size \( L \) is 2304. The average is taken over 35 graphs.

| Temperature \( (J) \) | \( \langle k \rangle \) | \( \langle \Delta \rangle \) |
|----------------------|----------------|----------------|
| 1.25J                | 1.40           | 8.54           |
| 0.45J                | 3.36           | 12.9           |
| 0.05J                | 22.7           | 41.6           |

performed every 100 MC steps. It has been checked in Ref. 30 that this frequency of potential switching is enough for this model to obtain reliable results.

To check the correctness of our parallel computation, we performed MC simulation and measured the absolute value of the circular component of magnetization defined by

\[
M_\phi \equiv \left\langle \left( \frac{1}{N} \sum_{i=1}^{N} S_i \times \frac{r_i - r_c}{|r_i - r_c|} \right)_z \right\rangle,
\]

where \([ \cdots ]_z\) denotes the \( z \)-component of a vector, \( \langle \cdots \rangle \) denotes thermal average, and \( r_c \) is a vector describing the center of the lattice. In this measurement, the system was kept at each temperature for 100,000 MC steps. The first 50,000 MC steps are for equilibration and the following 50,000 MC steps are for measurement. We performed simulations for 10 different runs with different initial conditions and random sequences. The result is shown in Fig. 4. The squares and circles denote the result of single-thread computation with 1 processor and that of multi-thread computation with 8 processors, respectively. Both the data coincide with each other within statistical error. We also see that \( M_\phi \) rapidly increases around the critical temperature \( T_c \approx 0.88J \). From these results, we conclude that our parallel computation is performed correctly.

3.2 Properties of graphs and improvements to reduce communication traffic

In Table I, we show the mean degree \( \langle k \rangle \) and the maximum degree \( \langle \Delta \rangle \) of graphs at three temperatures. These are important quantities because the maximum degree determines the number of colors and the mean degree \( \langle k \rangle \) is proportional to the computational time per one MC step. The size \( L \) is 2304. As found in Ref. 30, these quantities hardly depend on the size of two-dimensional magnetic dipolar systems if the size is sufficiently large. As expected from Eq. (2), both \( \langle k \rangle \) and \( \langle \Delta \rangle \) increase with decreasing temperature. However, they are several tens at most. This means that most of interactions are cut off by the potential switching. It should be noted that both \( \langle k \rangle \) and \( \langle \Delta \rangle \) are \( N^{-1} \approx 5\times10^6 \) before potentials are switched. Figure 5 shows the distance dependence of the probability \( P_{\text{survive}} \) that a potential is survived by being switched to \( \bar{\mathcal{V}} \). The temperatures are the same as those in Table I. We see that the probability increases with decreasing temperature. The probability is close to one when \( T = 0.05J \) and \( r = 1 \). However, \( P_{\text{survive}} \) rapidly decreases with increasing \( r \) at any temperatures.

Taking these properties of the SCO method into consideration, we implemented our simulation in the following way: We first divide the lattice into \( N_{\text{proc}} \) square cells (\( N_{\text{proc}} \) is the number of processors) and assign each cell to each processor. We then list up the vertices whose information should be sent by inter-processor communication when we update spins with a certain color. For example, a vertex \( i \) is added to a list for red-spin update if it satisfies the following two conditions:

- The vertex \( i \) is connected with a red vertex \( j \).
- The two vertices \( i \) and \( j \) belong to different cells.

This list is made for each color just once when a new graph is created by the potential switching. When we update spins of a certain color, we perform inter-processor communication in advance according to the list. Although it requires some computational cost to make the lists, they enable us to reduce the communication traffic before parallel MC calculation as much as possible. Figure 6 shows the temperature dependence of the proportion of survived potentials that require inter-processor communication. The proportion increases with increasing the number of processors. It should be noted that the mesh size decreases as the number of processors increases. The proportion also increases with decreasing temperature. However, it is less than 20% in most cases, meaning that communication traffic is considerably reduced by the improvement.
3.3 Efficiency of parallel computation

In Fig. 7, we plot average computational time per MC step \( t_{\text{ave}} \) as a function of the number of processors \( N_{\text{proc}} \). The average time \( t_{\text{ave}} \) is defined by

\[
t_{\text{ave}} = \frac{1}{100} t_{\text{switch}} + \frac{1}{100} t_{\text{color}} + t_{\text{MC}},
\]

where \( t_{\text{switch}} \), \( t_{\text{color}} \), and \( t_{\text{MC}} \) are the computational times to switch potentials, to solve vertex coloring, and to perform MC simulation for one MC step, respectively. Recall that potential switching and the subsequent vertex coloring are performed every 100 MC steps. The average is taken over the temperatures between 0.05\( J \) and 1.25\( J \). The data for \( L = 2304 \) and those for \( L = 1152 \) are denoted by squares and circles, respectively. When \( L = 2304 \), the computations with 144 processors and those with 288 processors are about 85 and 102 times faster than that with one processor, respectively. In the case of \( L = 1152 \), the speedup by 144 processors and that by 288 processors are about 58 and 57, respectively.

In Fig. 8, \( t_{\text{switch}}/100 \), \( t_{\text{color}}/100 \), and \( t_{\text{MC}} \) are plotted as a function of the number of processors. The size \( L \) is 2304. The sum of the three computational time is equal to \( t_{\text{ave}} \) for \( L = 2304 \) shown in Fig. 7 (see Eq. (8)). The computation time of MC simulation \( t_{\text{MC}} \) is dominant due to the factor 1/100 in \( t_{\text{switch}} \) and \( t_{\text{color}} \). We also see that MC simulation is parallelized well, while potential switching and vertex coloring are not. To make the present method effective even for larger parallel computations, we need to improve the parallelization efficiencies of the two processes.

4. Conclusions

In the present study, we have developed a method to parallelize the SCO method, which is a MC method for long-range interacting systems. To parallelize MC calculation in the SCO method, we numerically solve the vertex coloring of a graph created by the SCO method. This computation is performed in parallel by using the KW algorithm.\(^{36,37}\) We applied the present method to a two-dimensional magnetic dipolar system on an \( L \times L \) square lattice to examine its parallelization efficiency. The result showed that, in the case of \( L = 2304 \), the speed of computation increased about 102 times by parallel computation with 288 processors.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 25400387. Part of the experimental results in this research were obtained using supercomputing resources at Cyberscience Center, Tohoku University.

1) http://www.top500.org/
2) J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, Comput. Graph. Forum 26, 80 (2007).
3) J. A. Anderson, C. D. Lorenz, and A. Travesset, J. Comput. Phys. 227, 5342 (2008).
4) T. Preis, P. Virnau, W. Paul, and J. J. Schneider, J. Comput. Phys. 228, 4468 (2009).
5) M. Bernaschi, G. Parisi, and L. Parisi, Comput. Phys. Commun. 182, 1265 (2011).
6) Y. Komura and Y. Okabe, Comput. Phys. Commun. 183, 1155 (2012).
7) J. Micka, E. Hailath, V. Russob, K. Rushiahdah, L. Schwieberb, and J. Potoia, Comput. Phys. Commun. 184, 2662 (2013).
8) J. A. Anderson, E. Jankowski, T. L. Grubb, M. Engel, and S. C. Glotzer, J. Comput. Phys. 254, 27 (2013).
9) M. Bernaschia, M. Bissona, and F. Salvadordeb, Comput. Phys. Commun. 185, 2495 (2014).
10) M. B. Jesi, L. A. Fernández, V. M. Mayor, and J. M. Sanz, Phys. Rev. B 89, 014202 (2014).
11) G. S. Heffelfinger, Comput. Phys. Commun. 128, 219 (2000).
12) P. Larsson, B. Hess, and E. Lindahl, WIREs Comput. Mol. Sci. 1, 93 (2011).
13) D. Fincham, Mol. Sim. 1, 1 (1987).
14) A. W. Appel, SIAM J. Sci. Stat. Comput. 6, 85 (1985).
15) J. Barnes and P. Hut, Nature 324, 446 (1986).
16) J. Makino, J. Comput. Phys. 87, 148 (1990).
17) L. Greengard, The Rapid Evolution of Potential Fields in Particle Systems (MIT Press, Cambridge, MA, 1988).
18) J. Carrier, L. Greengard, and V. Rokhlin, SIAM J. Sci. Stat. Comput. 9, 669 (1988).
19) T. Darden, D. York, and L. Pedersen, J. Chem. Phys. 98, 10089 (1993).
20) U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, J. Chem. Phys. 103, 8577 (1995).
21) D. P. Landau and K. Binder, A Guide to Monte Carlo Simulations in Statistical Physics (Cambridge University Press, Cambridge, 2015) 4th ed., p. 76.
22) G. S. Heffelfinger, M. E. Lewitt, J. Comput. Chem. 17, 250 (1996).
23) A. Uhlherr, S. J. Leak, N. E. Adam, P. E. Nyberg, M. Doxastakis, V. G. Mavrantzas, and D. N. Theodorou, Comput. Phys. Commun. 144, 1 (2002).
24) R. Ren and G. Orkoulas, J. Chem. Phys. 126, 211102 (2007).
25) B. Sadigh, P. Erhart, A. Stukowski, A. Caro, E. Martinez, L. Z. Ruiz, Phys. Rev. B 85, 1 (2012).
26) B. D. Lubachevsky, Complex Systems 1, 1099 (1987).
27) G. Korniss, M. Novotny, and P. Rikvold, J. Comput. Phys. 153, 488 (1999).
28) E. Martinez, J. Marian, M. Kalos, and J. Perlsdo, J. Comput. Phys. 227, 3804 (2008).
29) G. Arampatzis, M. A. Katsoulakis, P. Plechat, M. Taifer, and L. Xu, J. Comput. Phys. 231, 7795 (2012).
30) M. Sasaki and F. Matsubara, J. Phys. Soc. Jpn. 77, 024004 (2008).
31) M. Sasaki, Phys. Rev. E 82, 031118 (2010).
32) p. 181 in Ref. 21.
33) C. H. Mak, J. Chem. Phys. 122, 214110 (2005).
34) C. H. Mak and A. K. Sharma, Phys. Rev. Lett. 98, 180602 (2007).
35) K. Fukui and S. Todo, J. Comp. Phys. 228, 2629 (2009).
36) F. Kuhn and R. Wattenhofer, Proc. of the 25th ACM Symp. on Principles of Distributed Computing, 2006, p. 7.
37) L. Barenboim and M. Elkin, Distributed Graph Coloring: Fundamentals and Recent Developments (Morgan & Claypool Publishers, San Rafael, 2013) p. 38.
38) p. 29 in Ref. 37.
39) J. Sasaki and F. Matsubara, J. Phys. Soc. Jpn. 66, 2138 (1996).