Parallel Markov chain Monte Carlo simulations

Ruichao Ren[a)] and G. Orkoulas[b)]
Department of Chemical and Biomolecular Engineering, University of California, Los Angeles, California 90095

(Received 2 April 2007; accepted 1 May 2007; published online 5 June 2007)

With strict detailed balance, parallel Monte Carlo simulation through domain decomposition cannot be validated with conventional Markov chain theory, which describes an intrinsically serial stochastic process. In this work, the parallel version of Markov chain theory and its role in accelerating Monte Carlo simulations via cluster computing is explored. It is shown that sequential updating is the key to improving efficiency in parallel simulations through domain decomposition. A parallel scheme is proposed to reduce interprocessor communication or synchronization, which slows down parallel simulation with increasing number of processors. Parallel simulation results for the two-dimensional lattice gas model show substantial reduction of simulation time for systems of moderate and large size. © *2007 American Institute of Physics*. [DOI: 10.1063/1.2743003]

I. INTRODUCTION

The increasing availability of computer clusters with distributed memory has made parallel computing an inevitable trend to simulate more realistic system sizes and longer time scales in computational science. Recently, new advancements have been made in both serial and parallel molecular dynamics simulations of biomolecules.[1–3] On the other hand, the parallelism of Monte Carlo simulations is not just a technical implementation challenge, but also an unsettled theoretical problem due to the serial nature of Markov chain theory.

Any massively parallel Monte Carlo simulation based on spatial decomposition inevitably involves simultaneous updates on multiple CPUs. Several attempts have been made to parallelize Monte Carlo algorithms through system decomposition into a number of noninteracting domains.[4,5] Updating a molecule with interaction range beyond its own domain needs the most up-to-date information of its neighbor molecules from logically adjacent computers. In parallel simulations on distributed computers/clusters, communication between processors is a much slower process due to the limit of network speed and data traffic, compared with ultrafast local cache or memory operation. As the number of processors grows, the speed of simulation may drop dramatically due to the extra communication overhead. Thus, it is crucial to synchronize information between processors at a relatively low frequency, which inevitably involves switching active regions periodically.

The most straightforward parallel Metropolis algorithm is to divide the simulation box into stripes or squares (domains). Then, each processor tries to update particles or spins within one domain. Once an update on the border of a domain is accepted, the processor in charge will send the updated information to its logical neighboring processors. A smarter variation of this method is to use the same random number sequence on all processors to avoid conflicts when two adjacent sites are selected at the same time by two neighboring processors.[4] However, both methods are only practical in supercomputers with shared memory architecture. In cluster computing, they both suffer from frequent communication, which usually leads to worse-than-serial performance.

A more efficient example of domain decomposition algorithm on four processors[5] is shown in Fig. 1. This case corresponds to a two-dimensional system with short-range intermolecular forces. Each of the four processors is in charge of a domain which comprises four regions A,B,C and D: see Fig. 1. At any given time, only molecules in the active (shaded) region are updated, and the active region changes periodically in the order of $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A$. Since the molecules inside the active regions are separated by distances longer than the interaction range, synchronization frequency is reduced dramatically. In this example, strict detailed balance[6,7] is not obeyed because only moves in the active regions are immediately reversible. The violation of strict detailed balance causes concerns about the precision of this type of parallel Monte Carlo simulation.[5]

Most Monte Carlo simulations ensure strict detailed balance via random updating. Recently, we proposed a Monte Carlo algorithm based on sequential updating moves with partial randomness that only satisfies the weaker balance condition.[8] The new random skipping sequential (RSS) Monte Carlo algorithm, identifies the correct equilibrium distribution of states, and converges much faster than the conventional Metropolis algorithm.[8] The improved efficiency of the sequential updating algorithm is attributed to the so-called “neighbor effect.” In the language of lattice models, successful moves trigger a higher probability of nearest-neighbor moves. The cascade behavior of the neighbor effect results in fast decorrelation and enhanced statistical quality of the generated samples.

In this work, we explore the parallel version of the Markov process for the nearest-neighbor lattice gas on the square lattice. We show that sequential updating is the key to reduc-

[a)]Electronic mail: ruichao@ucla.edu
[b)]Electronic mail: makis@seas.ucla.edu

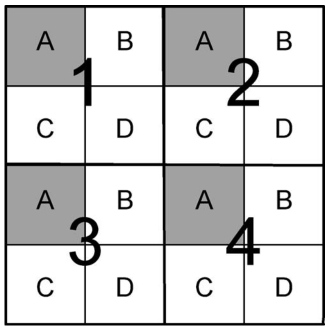

FIG. 1. An example of domain decomposition that combines random updating and periodic switching of active regions (shaded area). Active regions are separated by distances that are longer than the interaction range of particles or spins.

ing communication among processors and is ideally suitable for parallel implementation through domain decomposition without compromising precision.

II. PARALLEL MARKOV CHAIN

We assume that it is possible to classify all elementary moves by independency: Moves of the same type are independent of each other, but moves of different type might be dependent. For example, in Fig. 2 we consider eight possible moves that can be categorized into two types (type1: A-D and type2: E-H). In serial Monte Carlo simulations, moves are usually randomly selected and thus mixed up. In parallel Monte Carlo simulations, however, different types of moves must be isolated so that moves of the same type can be updated concurrently. To render an ergodic transition matrix, different types of moves/updates are executed periodically. Referring to Fig. 2, at time t_1 only type1 moves are executed, while at time t_2 only type2 moves are executed, and so on.

Following this assumption, we must identify the form of transition matrix for parallel Markov chains in order to understand its convergence behavior. If S_m is the transition matrix[8] associated with updating site/cell m, the transition matrices at time t_1 and t_2 are defined as

$$T_1 = \prod_m S_m, \quad \forall\, m \in \text{type1}, \tag{1}$$

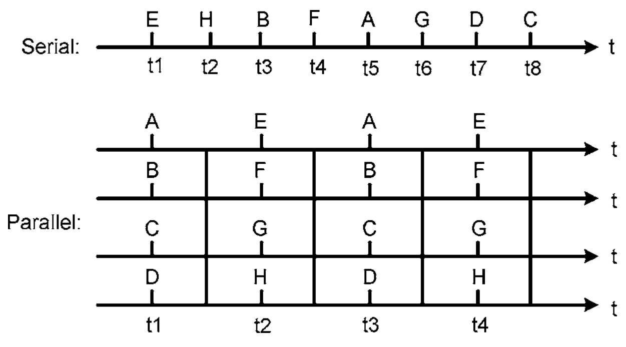

FIG. 2. In serial Monte Carlo simulations (top), all types of moves (A-H) are usually randomly selected and mixed up. But, in parallel Monte Carlo simulations (bottom), different types of moves (A-D and E-H) are isolated so that moves of the same type can be updated concurrently.

$$T_2 = \prod_m S_m, \quad \forall\, m \in \text{type2}. \tag{2}$$

The parallel transition kernel, R, can be related[8] with the type transition matrices T_n as

$$R = \prod_{n=1}^{k} T_n = T_1 T_2 T_3 \cdots, \tag{3}$$

where k is the number of types of independent moves. Note that a different sequence of T_n will generate a different parallel kernel because of the dependency between different types of moves.

Since each elementary move is accepted with Metropolis acceptance probabilities,[6,7,9] detailed balance is satisfied while updating on a single component

$$\pi_{\mathrm{i}}(S_m)_{\mathrm{ij}} = \pi_{\mathrm{j}}(S_m)_{\mathrm{ji}}, \tag{4}$$

which implies that the balance condition[8]

$$\boldsymbol{\pi}^{\top} \cdot S_m = \boldsymbol{\pi}^{\top}, \tag{5}$$

is also satisfied for updating an individual component, where $\boldsymbol{\pi}^{\top} = \{\pi_1, \pi_2, \ldots\}$ (row vector) is the equilibrium distribution of states. Following the analysis of Ref. 8, one can see that

$$\boldsymbol{\pi}^{\top} \cdot T_n = \boldsymbol{\pi}^{\top}, \tag{6}$$

and consequently that

$$\boldsymbol{\pi}^{\top} \cdot R = \boldsymbol{\pi}^{\top} \prod_{n=1}^{k} T_n = \boldsymbol{\pi}^{\top} \cdot T_2 \cdots T_k = \cdots = \boldsymbol{\pi}^{\top}. \tag{7}$$

Equation (7) indicates that the balance condition for the parallel transition kernel is satisfied. Since immediate reversal of a move is not possible within a sweep, detailed balance is not satisfied for the sweep transition kernel R, i.e.,

$$\pi_{\mathrm{i}} R_{\mathrm{ij}} \neq \pi_{\mathrm{j}} R_{\mathrm{ji}}. \tag{8}$$

Despite the absence of strict detailed balance, it has been shown[8] that sequential algorithms converge to the correct equilibrium distribution with the less strong balance condition, Eq. (7). Ergodicity is ensured by adding occasional random skipping.

The parallel version of the random skipping sequential (RSS) Monte Carlo algorithm is shown in Fig. 3 for a two-dimensional system. The simulation box is divided into stripes and all sites in each row are updated sequentially before moving on to the next row; see Fig. 3. Parallelization does not affect the serial efficiency of the RSS algorithm, so that the parallel computing efficiency improvement is always on top of the statistical efficiency[8] improvement by sequential updating.

III. OPTIMAL NUMBER OF PROCESSORS

In parallel Monte Carlo simulations, increasing the number of processors does not necessarily reduce the simulation time due to the extra communication overhead introduced by additional processors. The competition between computation and communication results in an optimal number of processors for a given system size. Above the optimal number of processors, the parallel efficiency decreases with additional

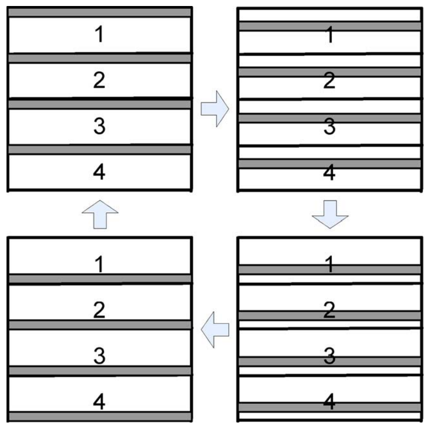

FIG. 3. Illustration of the parallel version of the random skipping sequential (RSS) Monte Carlo algorithm of Ref. 8. The simulation box is divided into four stripes, and each stripe is updated by one processor via sequential updating. There is no communication necessary until all CPUs finish updating the top and bottom line of their domains.

processors. An example of a parallel RSS Monte Carlo simulation is shown in Fig. 4. This case corresponds to the nearest-neighbor lattice gas on the square lattice.[10] As the number of processors increases, the simulation time decreases first; then, beyond a point (six CPUs in this case) it starts to increase.

To determine the optimal number of processors, we formulate the simulation time as an empirical function of the number of processors n_c and system size L,

$$t = \frac{\tau_c L^d}{n_c} + (\tau_o L^{d-1} + \tau_1) n_c. \tag{9}$$

In Eq. (9) d is the dimensionality of the system, τ_c is the unit computing time per cell, τ_o is the data transfer overhead, and τ_1 is communication initialization/finalization overhead.

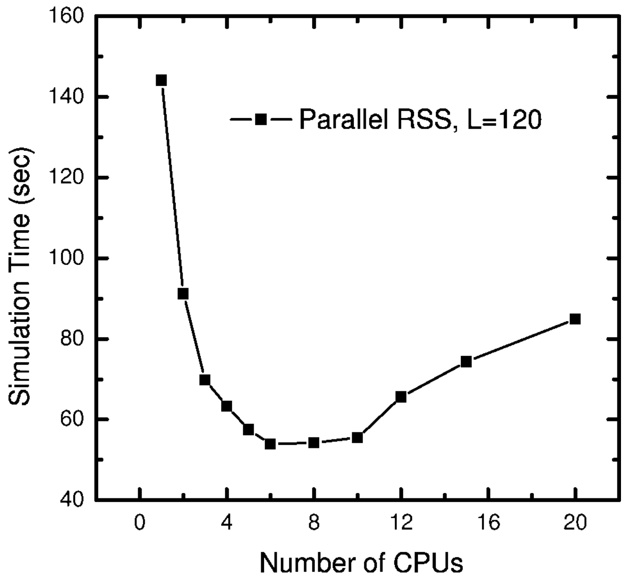

FIG. 4. Simulation time vs number of CPUs for a 120×120 lattice gas model on the square lattice. The simulation time corresponds to 10^5 lattice sweeps. The reduced temperature is $T^*=k_BT/\varepsilon=0.6$ and the value of the chemical potential corresponds to the symmetry (zero-field) axis for this model. The simulation time decreases first and eventually increases with the number of processors.

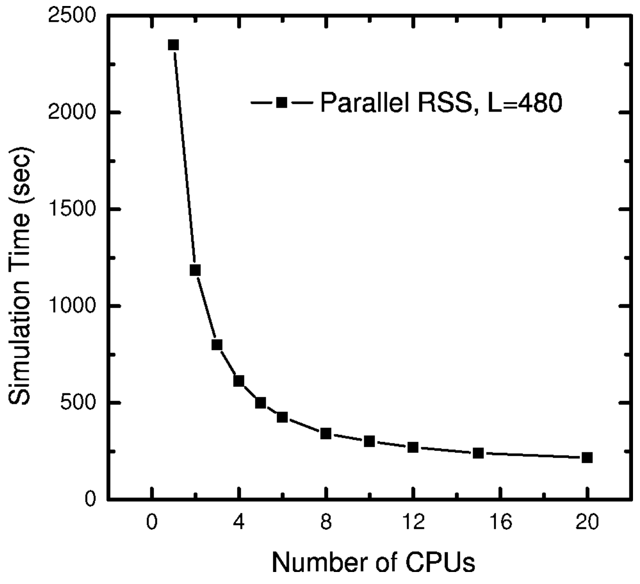

FIG. 5. Simulation time vs number of CPUs for a 480×480 lattice gas system. In this case, the simulation time decreases monotonically with the number of processors.

In systems of moderate size L, the initialization and finalization overhead τ_1 is significantly larger than the data transfer overhead τ_o. The simulation time can then be simplified as

$$t \simeq \frac{\tau_c L^d}{n_c} + \tau_1 n_c. \tag{10}$$

From Eq. (10) it follows that there is an optimal number of CPUs which renders the shortest simulation time. By taking the first derivative of the simulation time with respect to the number of CPUs,

$$\frac{\partial t}{\partial n_c} = -\frac{\tau_c L^d}{n_c^2} + \tau_1, \tag{11}$$

one can see that the optimal number of processors for system size L is

$$n_{\text{opt}} = \sqrt{\frac{\tau_c L^d}{\tau_1}}. \tag{12}$$

As the system size increases, the data transfer overhead, τ_o, becomes larger and more significant than the initialization/finalization overhead τ_1. The simulation time is

$$t \simeq \frac{\tau_c L^d}{n_c} + \tau_o L^{d-1} n_c. \tag{13}$$

In this case, the optimal number of processors,

$$n_{\text{opt}} = \sqrt{\frac{\tau_c L}{\tau_o}}, \tag{14}$$

is independent of the dimensionality d.

For very large systems, the communication overhead becomes negligible compared to the lengthy computing time. In this case, the simulation time is

$$t \simeq \frac{\tau_c L^d}{n_c}. \tag{15}$$

Thus, there is no optimal number of CPUs for very large systems, and the computation time decreases monotonically with the number of CPUs as shown in Fig. 5.

IV. CONCLUSIONS

Due to the absence of strict detailed balance, the sequential algorithm is ideal for parallel computing through domain decomposition. We introduce a parallel scheme that reduces the interprocessor communication which slows down parallel simulation with increasing number of processors. Parallel simulation results show substantial reduction of simulation time for systems of moderate and large size. The efficiency improvement is always on top of the statistical efficiency improvement by sequential updating. Future work in this direction is associated with devising sequential type of algorithms, both serial and parallel, for continuum fluid models.

ACKNOWLEDGMENTS

The authors are grateful to the Intel Higher Education Program, which provided the Linux cluster used in the simulations. The authors are also grateful to the developers of MPICH at Argonne National Laboratory, which made parallel programming of this work convenient and possible. Financial support from NSF, CBET-0652131, is gratefully acknowledged.

[1] X. Zhao, A. Striolo, and P. T. Cummings, Biophys. J. **89**, 3856 (2005).
[2] D. E. Shaw, J. Comput. Chem. **26**, 1318 (2005).
[3] K. J. Bowers, R. O. Dror, and D. E. Shaw, J. Chem. Phys. **124**, 184109 (2006).
[4] G. Barkema and T. Macfarland, Phys. Rev. E **50**, 1623 (1994).
[5] G. S. Heffelfinger, Comput. Phys. Commun. **128**, 219 (2000).
[6] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Clarendon, Oxford, 1987).
[7] D. Frenkel and B. Smit, *Understanding Molecular Simulation*, 2nd ed. (Academic, New York, 2002).
[8] R. Ren and G. Orkoulas, J. Chem. Phys. **124**, 064109 (2006).
[9] D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge University Press, Cambridge, 2000).
[10] R. Ren, C. J. O'Keeffe, and G. Orkoulas, Mol. Phys. **105**, 231 (2007).