Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Physics Communications 170 (2005) 175–185
Computer Physics Communications
www.elsevier.com/locate/cpc

Parallel implementation of molecular dynamics simulation for short-ranged interaction

Jong-Shinn Wu *, Yu-Lin Hsu, Yun-Min Lee

Department of Mechanical Engineering, National Chiao-Tung University, Hsinchu 30050, Taiwan

Received 13 January 2005; received in revised form 17 March 2005; accepted 24 March 2005

Available online 13 June 2005

Abstract

A parallel molecular dynamics simulation method, designed for large-scale problems, employing dynamic spatial domain decomposition for short-ranged molecular interactions is proposed. In this parallel cellular molecular dynamics (PCMD) simulation method, the link-cell data structure is used to reduce the searching time required for forming the cut-off neighbor list as well as for domain decomposition, which utilizes the multi-level graph-partitioning technique. A simple threshold scheme (STS), in which workload imbalance is monitored and compared with some threshold value during the runtime, is proposed to decide the proper time for repartitioning the domain. The simulation code is implemented and tested on the memory-distributed parallel machine, e.g., PC-cluster system. Parallel performance is studied using approximately one million L-J atoms in the condensed, vaporized and supercritical states. Results show that fairly good parallel efficiency at 49 processors can be obtained for the condensed and supercritical states (∼60%), while it is comparably lower for the vaporized state (∼40%).

Keywords: Parallel molecular dynamics; Short-ranged interaction; Dynamic domain decomposition; Parallel efficiency; Simple threshold scheme

1. Introduction

Classical molecular dynamics (MD) has been widely used to simulate properties of liquids, solids, and molecules in several research disciplines, including material science [1–4], biological technology [5–7], and heat and mass transfer [8–10] among others. It becomes increasingly important due to the burgeoning of nanoscience and nanotechnology, in which the size is often too small to characterize experimentally or to simulate properly by continuum methods. In applying the MD method to model these nanoscale phenomena, two main issues need to be resolved. The first is how to choose a physically correct potential model to truly represent the inter-particle interaction. The second is how to reduce the often time-consuming computation to an acceptable level. In this paper, we focus on resolving the second issue by developing an efficient parallel molecular dynamics method.

* Corresponding author. Tel.: +886-3-573-1693; fax: +886-3-572-0634.
E-mail address: chongsin@faculty.nctu.edu.tw (J.-S. Wu).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2005.03.110

Molecular dynamics simulates the N-particle (atoms or molecules) system treating each particle as a point mass, and Newton's equations are integrated for all particles to compute the motion. The physics of the modeled system is contained in the potential energy functional from which the Newton's equation for each particle is derived, assuming a conservative system. Being averaged from the trajectories of the particles, various useful static and dynamic properties can thus be derived [11]. As mentioned previously, MD simulation is very time-consuming due to the large number of time steps and possibly large number of atoms required to complete a meaningful simulation. In liquids and solids, MD simulation is required to resolve the vibration of the atoms, which limits the time step to be on the order of fentoseconds. Many hundreds of thousands or even millions of time steps are needed to simulate a nanosecond on the "real" time scale. In addition, hundreds of thousands or millions of atoms are needed in the MD simulation, even for a system size on the nanometer scale.

In the past, there has been considerable effort (e.g., [12]) that concentrated on parallelizing MD simulation on memory-distributed machines by taking the inherent parallelism [13,14]. Generally, parallel implementation of the MD method can be divided into three categories, including atom decomposition, force decomposition or domain decomposition among processors [12]. The atom decomposition method is generally suitable for small-scale problems. In the force decomposition method, it is based on a block-decomposition of the force matrix rather than a row-wise decomposition in the atom-decomposition method. It improves the $O(N)$ scaling to $O(N/\sqrt{P})$. It generally performs much better that the atom decomposition method; however, there exist some disadvantages. First, the number of processors has to be the square of an integer. Second, load imbalance may become an issue. From previous experience [12], it is suitable for small- and intermediate-size problems.

In the spatially static domain decomposition method, simulation domains are physically divided and distributed among processors. This method so far represents the best parallel algorithm for large-scale problems in MD simulation for short-ranged interactions [12]; however, it only works well for a system, in which the atoms move only a very short distance during simulation or possibly are distributed uniformly in space. MD simulation of solids represents one of the typical examples. In contrast, if the distribution of the atoms has much variation in configuration space, then the load imbalance among processors develops very quickly during simulation, which lowers the parallel performance. Thus, a parallel MD method capable of adaptive domain decomposition may represent a better solution for resolving this difficulty.

In the current study, we present a parallel algorithm for MD simulation, named parallel cellular molecular dynamics (PCMD), employing dynamic domain decomposition to address the issue of load imbalance among processors in the spatially static domain-decomposition method. This parallel algorithm is tested using one million L-J atoms in the condensed, vaporized and supercritical state, which all are initialized from a FCC-arranged atomic structure. Our goal is to develop a parallel MD method that is scalable for very large problems and large number of processors on memory-distributed parallel machines, such as a PC-cluster system, which is accessible to researchers in general.

This paper is organized as follows. The MD method is described next, followed by the description of the proposed parallel implementation in detail. Results of parallel performance for the different test cases is then presented and discussed. Finally, the paper is summarized with some important conclusions.

2. Numerical method

2.1. Molecular dynamics simulation

Molecular dynamics simulation is used to solve the dynamics of the N-particle (atom or molecule) system by integrating Newton's equation of motion, as shown in Eqs. (1) and (2), for each particle to obtain the position of the particle as a function of time,

$$m_i \frac{d\vec{v}_i}{dt} = \sum_j F_2(\vec{r}_i, \vec{r}_j) + \sum_j \sum_k F_3(\vec{r}_i, \vec{r}_j, \vec{r}_k) + \cdots, \tag{1}$$

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i, \tag{2}$$

where m_i is the mass of atom i, $\vec{r}_i$ and $\vec{v}_i$ are its position and velocity vectors, respectively. F_2 is a force

term describing pair-wise interactions between atoms, F_3 is a force term describes the three-body interactions, and many-body interactions can be added if needed. The force on each atom is the spatial derivative of potential energy that is generally written as a function of the position of the atom itself and other atoms.

In practice, only one or a few terms in Eq. (1) are kept to simplify the problem. They are constructed from either fitting some properties to experimental data or a quantum computation. Thus, classical MD is intrinsically cheaper in computation as compared with *ab initio* electronic structure calculations, which require solving Schrodinger's equation at each time step.

2.1.1. Interaction potential model

Potential energy can also be categorized as either a short- or long-ranged interaction in nature. The former (e.g., the L-J potential) only considers the interaction between the atoms geometrically nearby the interested atom, while the latter (e.g., Coulomb potential in ionic solids or biological systems) needs to consider the interaction far away from the atom under consideration. Because of the simplicity of the short-ranged interaction, it has been applied extensively in many MD simulations in the past [15], especially for liquids and solids. In contrast, long-ranged interaction models are not commonly used in classical MD simulations except for polarized molecules, such as water [16]. In this paper, we are only interested in dealing with classical MD using short-ranged interactions. The L-J potential will be used throughout the current study unless otherwise specified. The proposed method can also be applied to other potential interactions if they are short-ranged. Strategies employed for the short-ranged interaction in the current study are introduced next.

2.1.2. Concept of neighbor list

In principle, all potentials on each atom resulting from all other atoms have to be taken into account when computing the force—this scales as $O(N^2)$. In practice, each atom stores an array containing the list (or *neighbor list*) [11] of atoms located within some cutoff distance ($r_c = 2.5\sigma$). Note that σ is the zero-potential distance of L-J potential. Basically, each neighbor list has to be updated at each time step, which is very time-consuming. Thus, the radius of this neighbor list is often extended to be $r_c + \delta$, where $\delta = 0.3\sigma$, to reduce the frequency of updating. In contrast, we only have to update each neighbor list every 8–10 time steps using the above choice. This reduces greatly the frequency of neighbor atom search. However, building the neighbor list through searching all the atoms in the system is also very time-consuming.

2.1.3. Concept of link-cell + concept of neighbor list

Link-cell data structure provides a means of organizing information (position and velocities) of atom into a form that avoids most of the unnecessary work and thus reduces the computational cost to $O(N)$. The general idea of the link cell is to divide the simulation domain into a lattice of small cells having edge length exceeding r_c. In the current study, we take the cell-edge length as $r_c + \delta$, which coincides with the designated radius of neighbor list. Thus, atoms in a cell only interact with atoms either in the same cell or cells nearby. Details of implementation can be found in Rapaport [11] and only a brief description is provided here. During the update of each neighbor list for each atom, we only search the atoms in the other 26 neighboring cells and the cell where the atom itself is located, rather than search through all the atoms in the computational domain. This can also reduce dramatically the time required to build up the neighbor list. The price we have to pay is to add an array to store the atoms in each cell and to update this array each time step, which is straightforward in the cell structure having equal edge length for all cells. Past results show that this combination represents the most efficient technique nowadays in classical molecular dynamics simulation [11].

2.2. Parallel molecular dynamics simulation

In this current study, we focus on developing a parallel MD method using dynamic domain decomposition by taking advantage of the existing link-cells as mentioned earlier. In this proposed method, not only are the cells used to reduce the cost for building up the neighbor list, but also are used to serve as the basic partitioning units. A similar idea has been applied in the parallel implementation of the direct simulation Monte Carlo (DSMC) method (e.g., [19]), which is a particle simulation technique often used in rarefied gas dynamics. Note that in the following IPB stands for interprocessor boundary. General procedures (Fig. 1) in the sequence include:

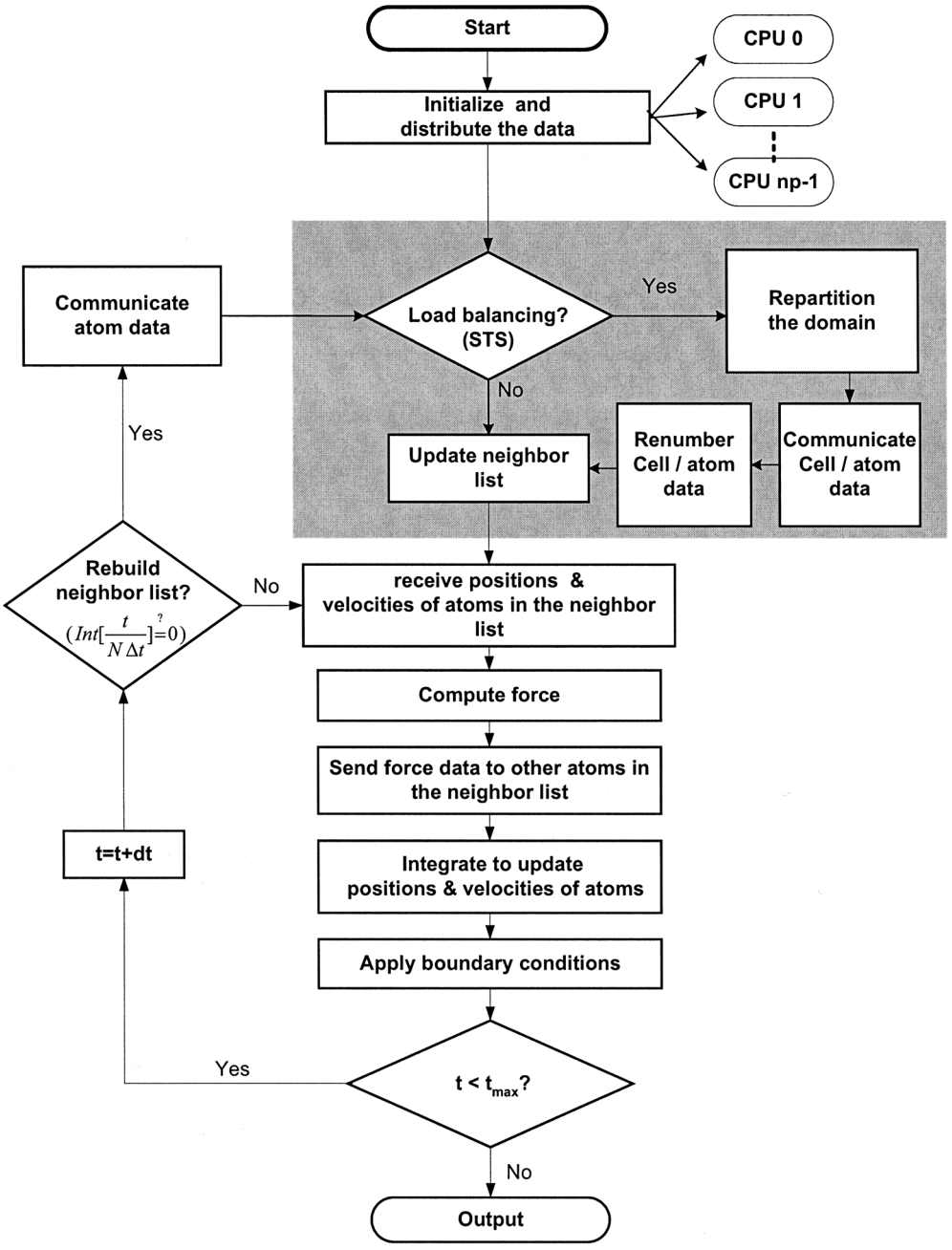

Fig. 1. Proposed flow chart for parallel molecular dynamics simulation using dynamic domain decomposition.

1. Initialize the positions and velocities of all atoms and equally distribute the atoms among processors;
2. Check if load balancing is required. If required, then first repartition the domain, followed by communicate cell/atom data between processors, renumber the local cell and atom numbers, and update the neighbor list for each atom due to the data migration;

3. Receive positions and velocities of other atoms in the neighbor list for all cells near the IPB;
4. Compute force for all atoms;
5. Send force data to other atoms in the neighbor list for all cells near the IPB;
6. Integrate the acceleration to update positions and velocities for all atoms;
7. Apply boundary conditions to correct the particle positions if necessary;
8. Check if preset total runtime is exceeded. If exceeded, then output the data and stop the simulation. If not, check if it is necessary to rebuild the neighbor list of all atoms using the most recent atom information;
9. If it is necessary to rebuild the neighbor list ($N = 8$ in the current study), then communicate atom data near the IPB and repeated the steps 2–8. If not necessary, then repeat steps 3–8.

Note that the subroutines included in the shaded region (Fig. 1) represent the load-balancing module. In the above procedures, in addition to the necessary data communication when atoms cross the IPB and atom/cell data near the IPB, there are two more important steps in the proposed parallel MD method as compared with the serial MD implementation. One is how to repartition the domain effectively and the other is the decision policy for repartitioning. These two steps are described next, respectively.

2.2.1. Repartitioning scheme

Under the framework of graph theory, centers of each link-cell are considered as the *vertices* and the lines connecting them are considered as *edges*. Each vertex and edge can be assigned with a weight for the purpose of partitioning. Graph partitioning has been found very useful for an unstructured mesh in the computing community in the past. In the current study, a parallel multilevel graph-partitioning runtime library, PMETIS [18], is used as the repartitioning tool in our PCMD code. Thus, the data structure of the link-cell is reconfigured as unstructured for graph-partitioning purpose. The multilevel graph-partitioning scheme uses the multilevel implementation that matches and combines pairs of adjacent vertices to define a new graph and recursively iterate this procedure until the graph size falls under some threshold. The coarsest graph is then partitioned and the partition is successively refined on all the graphs starting with the coarsest and ending with the original. At evolution of levels, the final partition of the coarser graph is used to give the initial partition for the next finer level. A corresponding parallel version, PMETIS [18], uses an iterative optimization technique known as relative gain optimization, which both balances the workload and attempts to minimize the inter-processor communication overhead.

This parallel multilevel graph partition runs on the single program multiple data (SPMD) paradigm with message passing in the expectation that the underlying mesh will do the same. Each processor is assigned to a physical sub-domain and stores a double-linked list of the vertices within that sub-domain. However, each processor also maintains a "halo" of neighboring vertices in other sub-domains. For the serial version, the migration of vertices simply involves transferring data from one linked-list to another. In the parallel version, this process is far more complicated than just migrating vertices. The newly created halo vertices must be packed into messages as well, sent off to the destination processor, unpacked, and the pointer based data structure recreated there. This provides a possible solution to the problem of adaptive load balancing [18].

2.2.2. Decision policy for repartitioning

MD represents a typical dynamic (or adaptive) irregular problem, i.e. the workload distributions are known only at runtime, and can change dramatically as simulation proceeds, leading to a high degree of load imbalance among the processors. This load-changing situation is even obvious in simulating liquids or gases. Thus, the partitioning runtime library, PMETIS [18], described in the above is used to repartition the mesh based on some sort of decision policy. In the direct simulation Monte Carlo (DSMC) simulation [17], it has been shown that a decision policy based on stop at rise (SAR) [19] works well for improving the parallel performance. However, from our preliminary study, it does not work very well in the MD simulation since the domain repartition is too frequent and, thus, too costly in practice. In addition, the data locality of the MD simulation using link-cells is lower than that of the DSMC simulation, which only considers collisions (interaction) among atoms within the same cell. Instead, a simple threshold-like decision policy, termed "simple threshold scheme" (STS), is designed to de-

cide the proper time to repartition. This scheme simply asks for domain repartitioning if the workload in some processor is detected over the specified threshold (e.g., ±20% of the average workload). The number of atoms in each link-cell is used as the weighting for graph partitioning. Right after the repartition, the communication between geometrically neighboring processors is required to transfer the cell number and particle data to the destination processor, followed by renumbering of the cell and particle data into the local numbering in the destination processor. Since all processors know the geometrical information of all cells, renumbering of the received cells in some specific processor is done simply by adding up sequentially the local cell numbers for the new cells (as shown in the shaded region in Fig. 1). Similarly, the cells sent out to other processors are simply removed sequentially by copying the information of the final cell onto the memory of the sent cells. This decision policy for repartitioning the domain is inherently advantageous in which no prior knowledge of the evolution of the problem is necessary to determine the repartitioning interval, and the repartitioning can be expected to follow the dynamics of the problem.

The current PCMD code is implemented on a 64-bit Itanium PC-cluster system running Linux OS at the National Center for High-performance Computing in Taiwan (64-node, dual processor and 4 GB RAM per node). Standard message-passing interface (MPI) is used for data communication. It is thus expected that the current PCMD code should be highly portable among the memory-distributed parallel machines that is running with Linux (or equivalent) operating system.

3. Results and discussions

3.1. Problem descriptions and simulation conditions

Three test problems, including L-J (12, 6) atoms in condensed, vaporized and supercritical states, are chosen to test the current parallel implementation of the molecular dynamics simulation. Related simulation conditions are summarized in Table 1. Densities ($\rho^* = N\sigma^3/V$) are all taken to be 0.7 for convenience, while temperatures ($T^* = kT/\varepsilon$) are varied to represent different states. Note that the number of cells in

Table 1
Simulation conditions for three different cases (condensed, vaporized and supercritical states)

	Temp. (T^*)	Density (ρ^*)	No. of link-cells
Condensed	0.7	0.7	$75 \times 75 \times 75$
Vaporized	1.1	0.7	$75 \times 75 \times 75$
Supercritical	0.7	0.7	$39 \times 39 \times 39$

Table 1 represents the number of link-cells used for simulation. Note that the simulation volume is generally much larger (∼10 times) than the initial volume of FCC-arranged atomic structure near the system center, except the case of supercritical state, which simulation volume is approximately the same as the initial FCC volume. Each atom of an initially FCC-arranged atomic structure is given random velocities based on the desired temperature and starts to run for 10^5 time steps. Rescaling the kinetic energy of the system during runtime enforces the desired temperature of the simulation system for all three test cases. The simulation time step is 0.005 of the dimensionless MD time scale (or about 0.005×10 fs). Current test problems represent a more severe test for the parallel implementation of the MD method due to the rapidly changing workload among processors during the simulation. For example, it is rather difficult to parallel compute the condensed state efficiently if a conventional parallel paradigm such as static domain decomposition is used. All results presented below were obtained using 25 processors, unless otherwise specified. In addition, periodic boundary conditions are used to simplify the analysis in the current study.

3.2. Snapshots of MD evolution

Snapshots of the atomic distribution at the final time step (10^5 time steps) of the L-J atoms (∼1 million atoms) for the condensed, vaporized and supercritical states are illustrated, respectively, in Figs. 2(b)–(d). Note that Fig. 2(a) shows the initial FCC atomic structure and the simulation volume ($75 \times 75 \times 75$ cells) for both the condensed and vaporized states. For the supercritical state, the size of the initial FCC structure is the same as the other two cases, but with a much smaller simulation volume ($39 \times 39 \times 39$ cells). It is clear that the initial and final atomic structures differ to a large extent for all three cases considered. This renders the static spatial domain decompo-

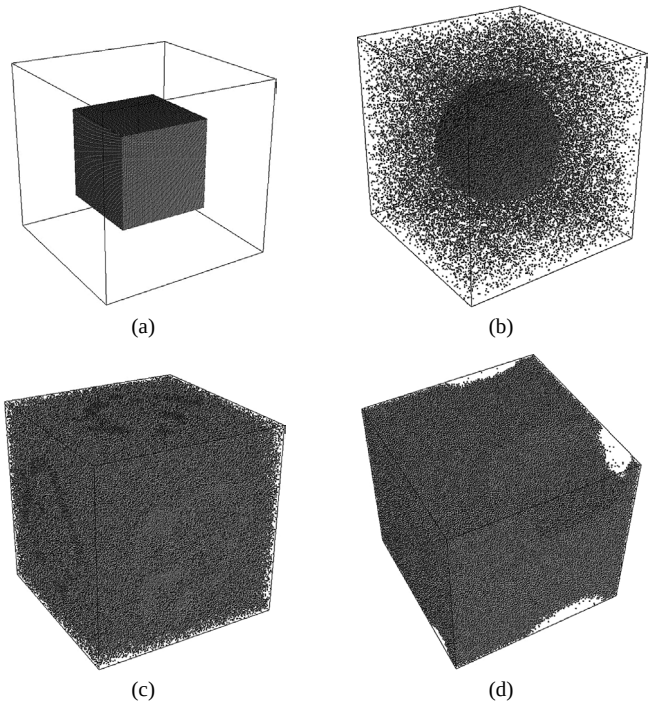

Fig. 2. Atomic structures of three test cases at 10^5 time steps: (a) initial FCC structure for the condensed and vaporized states; (b) final condensed state; (c) final vaporized state; (d) final supercritical state.

sition very inefficient or even useless, which is often used for the large-scale MD simulation. In the case of the condensed state (Fig. 2(b)), a spherical liquid droplet (∼25 nm in diameter) is formed at the center of the simulation system, around which comparatively few vaporized atoms spread, due to the lower preset temperature (0.7). In the case of the vaporized state (Fig. 2(c)), the atoms spread rather randomly uniform in the simulation domain due to the preset higher temperature (1.1). In the case of the supercritical state (Fig. 2(d)), several irregular cavities having vaporized atoms inside are present in the computational domain, in which the state is neither liquid nor gas in essence. Note that the spatial distribution of the cavities does not remain unchanged during the simulation.

3.3. Evolution of domain decomposition

Fig. 3(a)–(b), (c)–(d), and (e)–(f) show the domain decompositions ($500\Delta t$, and final) for the condensed, vaporized and supercritical states, respectively, on the system surface and on some special cross sections cutting through the system center. By comparing these figures, we can find that the domain decomposition changes to a large extent from the initial to the final state, except the case of the supercritical state, in which the initial FCC structure almost occupies the simulation volume. In Fig. 3(b) (condensed state), the domain size of the droplet near the center the system center is very small as compared with the domain size in other regions since the atoms are clustered near the system center. In contrast, the distribution of the domain size is relatively uniform in Fig. 3(d) (vaporized state) in the simulation volume since the atoms are spreading randomly in the computational domain. Fig. 3(f) shows the domain decomposition of the supercritical state, in which the distribution of domain is similar to the vaporized state (Fig. 3(d)) but with higher ordered structure. The above arguments about the evolution of domain decomposition in the test cases, along with the time-dependent distributions of the number of atoms in each processor, can explain the parallel performance obtained in the current study, which are introduced next.

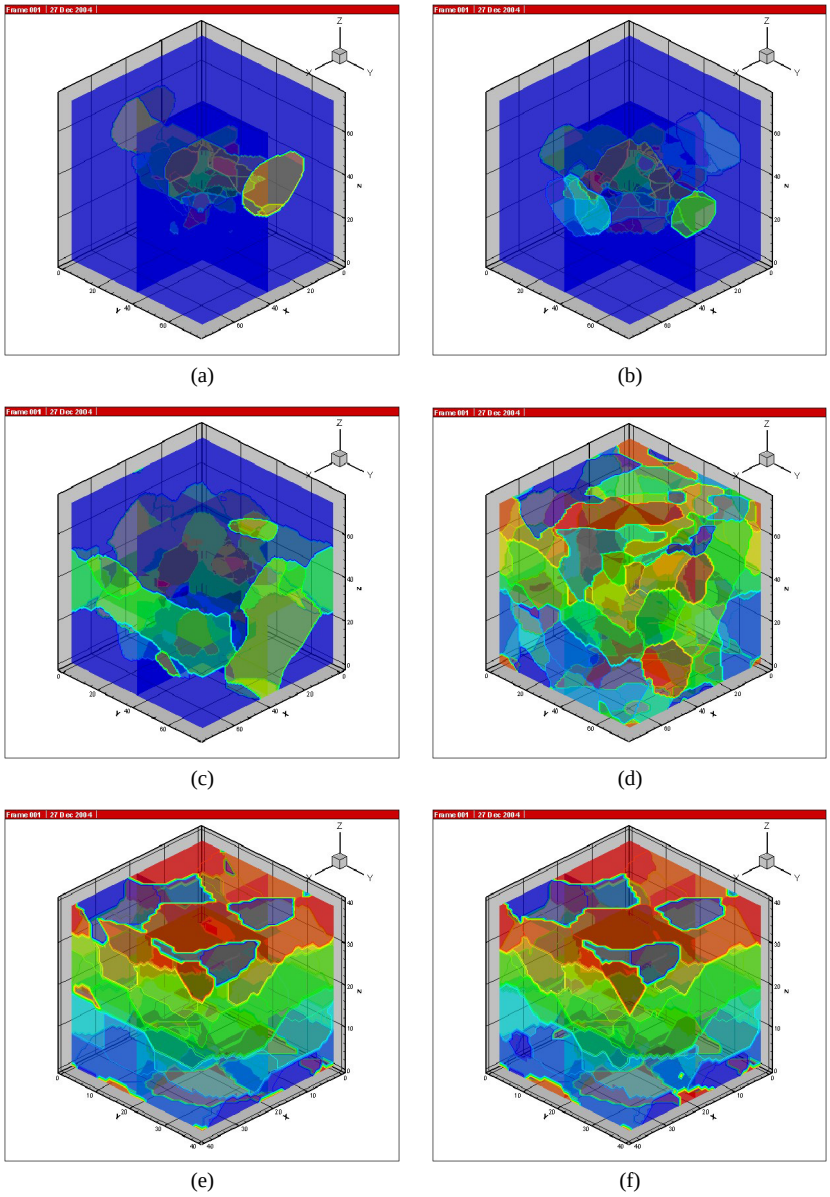

Fig. 3. Evolution of domain decomposition simulation domain on the system surface and special sections for the different atomic final states (25 processors): (a) at 500 steps (condensed); (b) final (condensed); (c) at 500 steps (vaporized); (d) final (vaporized); (e) at 500 steps (supercritical); (f) final (supercritical).

3.4. *Distribution of time history of atom numbers*

Fig. 4(a)–(c) show the distribution of the number of atoms in each processor as a function of the number of simulation time steps, respectively, for the condensed, vaporized and supercritical states using 25 processors, along with the upper/lower thresholds of the number of atoms (±20%, dotted lines, in the current study). In these figures, we do not intend to identify any specific processor used in the simulation. We are only interested in showing the general trend of effectiveness of the dynamic domain decomposition in a general

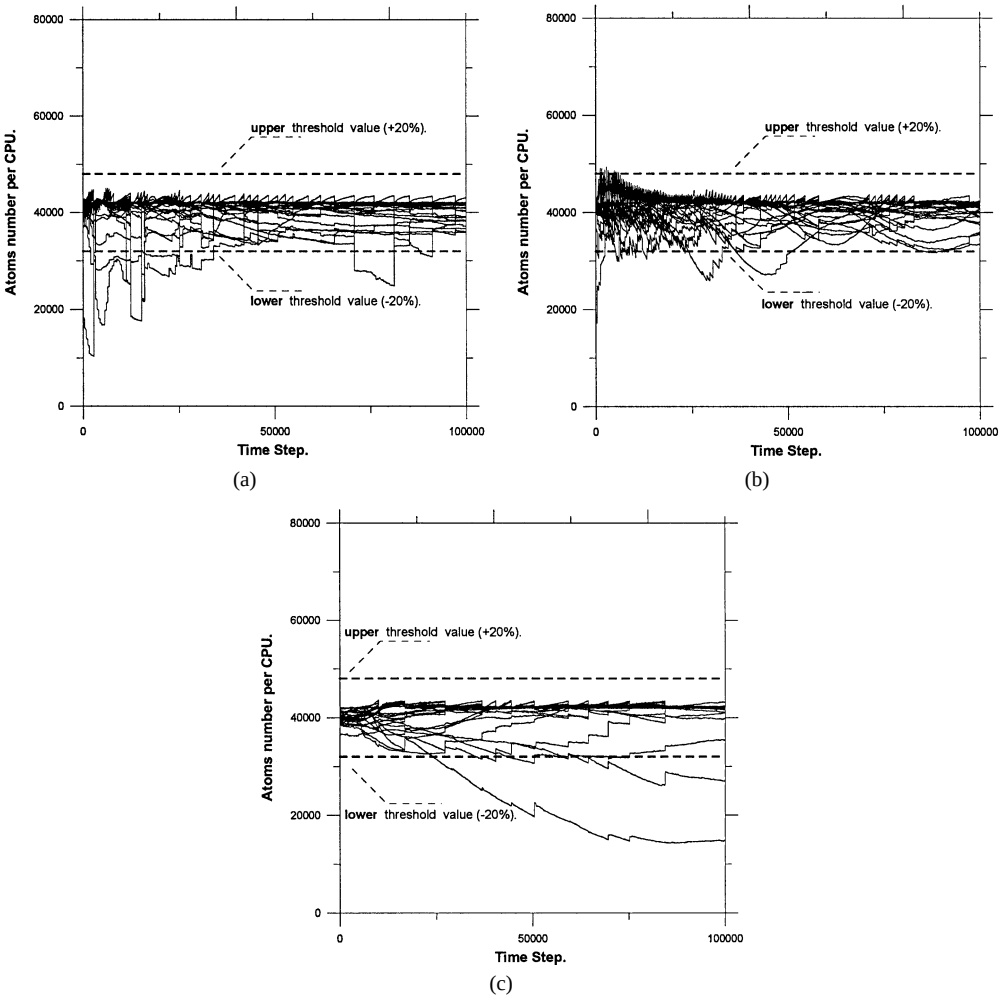

Fig. 4. Distribution of the number of atoms in each processor as a function of simulation time steps (25 processors): (a) condensed state; (b) vaporized state; (c) supercritical state.

sense. In addition, we can roughly identify how frequent the repartition is from these figures by observing the abrupt change of number of atoms. In general, the repartition in PCMD is rather effective in evenly redistributing the number of atoms in each processor, or approximately, the workload among processors. However, the load balancing in some (relatively few) processors is not so effective in bringing up the number of atoms to the averaged value due to too many atoms in a cell (e.g., Figs. 4(a) and (c)). The situation is much better for the vaporized-state case as shown in Fig. 4(b), where the repartition can keep the number of atoms roughly within the range of upper and lower threshold values. Another reason for this ineffective domain decomposition may be related to the multilevel graph-partitioning tool (PMETIS [18]) used in the current study, which assumes it optimizes the partition heuristically, rather than theoretically. Note that a more severe criterion of the threshold values can result in better load balance among processors, but possibly computational inefficiency can result from the frequent repartition of the domain. Another important observation from Figs. 4(a)–(c) is that it requires most frequent repartition during the simulation for the vaporized-state case (Fig. 4(b)) due to the fast moving atoms, while it requires much less repartition for

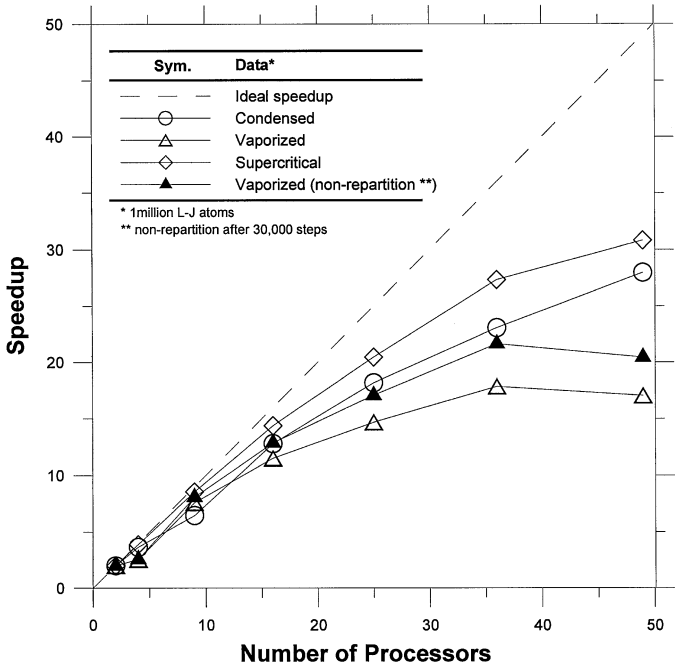

Fig. 5. Parallel speedup as a function of the number of processors for three different test cases (condensed, vaporized and supercritical states).

the supercritical-state case (Fig. 4(c)). Thus, it is not necessary that a better load balancing resulting from effective domain decomposition can lead to a better parallel speed-up since the repartition is computationally expensive. The above observation has a profound influence on the parallel speed-up, which is introduced next.

3.5. Parallel performance

Fig. 5 shows the corresponding parallel speed-up for all three test cases (up to 49 processors) in the current study along with the ideal speed-up (dotted line). Using 49 processors, the parallel speed-up is 32 (∼65% parallel efficiency) for the supercritical-state case and 28 (∼57% parallel efficiency) for the condensed-state case, while it is 17 (or 35% parallel efficiency) for the vaporized-state case. Note that the above parallel speed-up/efficiency are all computed assuming the value of speed-up as two for two processors. It is attributed to the too frequent repartition of the domain in the vaporized-state case, although slightly better load balancing is observed than in the other two cases (Fig. 4(b)). This could further degrade the speed-up due to the rapid increase of the frequent communication required for repartitioning the domain, in addition to the large number of processors, which in turn complicates and slows down the network communication. This can be shown by the increase of speed-up to 21 (49 processors) if we do not repartition the domain after the thermal equilibration period (∼30 000 time steps). Using the current parallel implementation of the MD code, approximately 70–80% of parallel efficiency can be achieved with 25 processors, which may be accessible for most researchers using a practical PC-cluster system.

4. Conclusions

In the current study, a parallel molecular dynamics simulation for short-ranged interactions using dynamic domain decomposition is developed for large-scale problems on the memory-distributed PC-cluster system, which uses MPI as the communication protocol. In the method, a multi-level graph-partitioning scheme is used to dynamically re-decompose the computational domain based on a simple threshold scheme

(STS), which is expected to keep the number of atoms in each processor within the range of the threshold values. Parallel performance of the current parallel MD method is studied using three different test cases, including the condensed, vaporized and supercritical states, using approximately one million L-J atoms, which all are initialized from a FCC atomic structure. Results show that fairly good parallel efficiency in the range of 40–65% using 49 processors can be achieved for the three test cases, compared with low efficiency if static domain decomposition or other methods are employed. Application of this parallel MD code to compute the dynamical processes involved with droplet evaporations and collisions is currently in progress and will be reported in the near future.

Acknowledgements

This investigation was supported by the National Science Council, Grant No. 93-2212-E-009-015. The authors also would like to express their sincere thanks to the computing resources provided by the National Center for High-speed Computing of National Science Council of Taiwan. Also, sincere thanks go to Prof. Karypis at University of Minnesota for generously providing the partitioning library, PMETIS.

References

[1] R. Komanduri, N. Chandrasekaran, L.M. Raff, Effect of tool geometry in nanometric cutting: a molecular dynamics simulation approach, Wear 219 (1998) 84–97.
[2] R. Komanduri, N. Chandrasekaran, L.M. Raff, MD simulation of indentation and scratching of single crystal aluminum, Wear 240 (2000) 113–143.
[3] T.H. Fang, C.I. Weng, Three-dimensional molecular dynamics analysis of processing using a pin tool on the atomic scale, Nanotechnology 11 (2000) 148–153.
[4] K. Cheng, X. Luo, R. Ward, et al., Modeling and simulation of the tool wear in nanometric cutting, Wear 255 (2003) 1427–1432.
[5] Y.M. Pan, T.J. Hou, M.J. Ji, et al., MD simulations of PTP 1B-inhibitor complex, Acta Chim. Sinica 62 (2004) 148–152.
[6] L.J. Smith, H.J.C. Berendsen, W.R. van Gunsteren, Computer simulation of urea–water mixtures: a test of force field parameters for use in biomolecular simulation, J. Phys. Chem. B 108 (2004) 1065–1071.
[7] P.J. Connolly, A.S. Stern, C.J. Turner, et al., Molecular dynamics of the long neurotoxin LSIII, Biochemistry-US 42 (2003) 14443–14451.
[8] L. Consolini, S.K. Aggarwal, S. Murad, A molecular dynamics simulation of droplet evaporation, Int. J. Heat Mass Transfer 46 (2003) 3179–3188.
[9] L.N. Long, M.M. Micci, B.C. Wong, Molecular dynamics simulations of droplet evaporation, Comput. Phys. Comm. 96 (1996) 167–172.
[10] J.H. Walther, P. Koumoutsakos, Molecular dynamics simulations of nanodroplet evaporation, J. Heat Transfer 123 (2001) 741–748.
[11] D.C. Rapaport, The Art of Molecular Dynamics Simulation, Cambridge University Press, UK, 1995.
[12] S. Plimpton, Fast parallel algorithms for short-range molecular dynamics, J. Comput. Phys. 117 (1995) 1–19.
[13] B.M. Boghosian, Computational physics on the connection machine, Comput. Phys. (1990).
[14] G.C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon, D.W. Walker, Solving Problems on Concurrent Processors, vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[15] S. Matsunaga, Structural study on liquid Au–Cs alloys by computer simulations, J. Phys. Soc. Japan 69 (2000) 1712–1716.
[16] D. DiCola, A. Deriu, M. Sampoli, et al., Proton dynamics in supercooled water by molecular dynamics simulations and quasielastic neutron scattering, J. Chem. Phys. 104 (11) (1996) 4223–4232.
[17] J.-S. Wu, K.-C. Tseng, Concurrent DSMC method using dynamic domain decomposition, in: Proc. 23rd Internat. Symp. on Rarefied Gas Dynamics, Whistler Conference Centre Whistler, British Columbia, July 2002, pp. 20–25.
[18] G. Karypis, K. Schloegel, V. Kumar, ParMetis, University of Minnesota, Department of Computer Science, September 1998.
[19] D.M. Nicol, J.H. Saltz, et al., Dynamic remapping of parallel computations with varying resource demands, IEEE Trans. Comput. 37 (1988) 1073–1087.