MDSLB: A new static load balancing method for parallel molecular dynamics simulations*

Wu Yun-Long(武云龙), Xu Xin-Hai(徐新海)†, Yang Xue-Jun(杨学军), Zou Shun(邹 顺), and Ren Xiao-Guang(任小广)

State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha 410073, *China*

(Received 30 May 2013; revised manuscript received 4 September 2013; published online 20 December 2013)

Large-scale parallelization of molecular dynamics simulations is facing challenges which seriously affect the simulation efficiency, among which the load imbalance problem is the most critical. In this paper, we propose, a new molecular dynamics static load balancing method (MDSLB). By analyzing the characteristics of the short-range force of molecular dynamics programs running in parallel, we divide the short-range force into three kinds of force models, and then package the computations of each force model into many tiny computational units called "cell loads", which provide the basic data structures for our load balancing method. In MDSLB, the spatial region is separated into sub-regions called "local domains", and the cell loads of each local domain are allocated to every processor in turn. Compared with the dynamic load balancing method, MDSLB can guarantee load balance by executing the algorithm only once at program startup without migrating the loads dynamically. We implement MDSLB in OpenFOAM software and test it on TianHe-1A supercomputer with 16 to 512 processors. Experimental results show that MDSLB can save 34%–64% time for the load imbalanced cases.

Keywords: molecular dynamics, static load balancing, parallel computing

PACS: 89.20.–a, 89.20.Ff, 02.70.Ns, 31.15.–p **DOI:** 10.1088/1674-1056/23/2/028903

1. Introduction

As an important and powerful computer simulation tool, molecular dynamics (MD) simulations have evolved over half a century and been applied widely in chemical physics, biology, and materials science.[1–6] With this method, we can get the macroscopic properties of molecular systems by studying the interactions between particles.[7] Compared with the continuum method, MD simulations are more intuitive and general for studying the local microphysics phenomena.[8] Therefore, many researchers study molecular structures with MD simulations to explore their physical essences. However, as the scale of molecular systems increases, the computations of MD simulations grow in a quadratic rate,[9] which is a challenge for most large-scale MD applications. In recent years, the performance improvement of supercomputers has made it possible to perform large-scale MD simulations. Many researchers have started to use supercomputers, such as Blue Gene/L,[10] TianHe-1A,[11] Cray XK6,[12] Titan,[13] Cray XT3,[14] and Blue Gene/P,[15] to carry out MD simulations in parallel.

With the parallel scale increasing, MD simulations meet new challenges on supercomputers, such as load imbalance, communication costs, and memory overheads. Among those problems, load imbalance is the most serious. In the parallel MD simulations, spatial decomposition is the most popular decomposition method, which decomposes the spatial region into cells, with one or more cells associated with each processor. In addition, the load for a given processor depends on the number of particles in the cells it holds. During the simulation, it is hard to predict the motions and aggregations of particles, which may lead to the non-uniform distribution: some processors hold relatively more particles, while others hold fewer. As a result, the simulation is inefficient as all processors have to wait for the one with the heaviest load.[16] Considering that MD simulations always need to run millions of time steps, the impacts of load imbalance will affect the parallel efficiency seriously. For this reason, it is quite necessary to solve the load imbalance problem in the parallel MD simulations.

The current methods for solving load imbalance can be divided into two types: static and dynamic. The static method aims to allocate the loads to every processor evenly at program startup. Once-run is the merit of this method, but it can only ensure that the load is balanced for each processor for the time being (cannot predict the future load distribution). In order to remedy the static method, some researchers put forward the dynamic method. By monitoring the load distribution periodically, the dynamic method can find the load imbalanced processors, and then do load migrations between these processors. Compared with the static method, the dynamic method can guarantee load balance in the whole process of simulation. However, the dynamic load migration will incur extra overheads and affect the parallel efficiency. In addition, with particles moving around, the dynamic method must be repeated at intervals.

In this paper, to make up the shortcomings of the dy-

*Project supported by the National Natural Science Foundation of China (Grant Nos. 61303071 and 61120106005) and the Natural Science Fund from the Guangzhou Science and Information Technology Bureau (Grant No. 134200026).

†Corresponding author. E-mail: xuxinhai@nudt.edu.cn

© 2014 Chinese Physical Society and IOP Publishing Ltd

namic method, we propose a new molecular dynamics static load balancing (MDSLB) method based on spatial decomposition and implement it on OpenFOAM. Considering that the short-range force calculation occupies most of the whole computations, even in the simulations that calculate long-range electrostatics,[17,18] we mainly focus on balancing the short-range force calculation among processors. In MDSLB, each spatial region is divided into many small blocks called "local domains". The computations of each local domain can be represented as a number of fine-grained and independent computational tasks called "cell loads". The cell loads are evenly allocated to each processor to achieve such a goal — the cell loads of each local domain are "supported" by all processors. In this way, we can guarantee load balance in the process of MD simulations by executing our method only once at program startup. The contributions of this paper are summarized as follows.

(i) We propose MDSLB, a new MD static load balancing method, which can make the load balanced without migrating the loads among all processors.

(ii) We design three kinds of independent computational models for the short-range force, which are called: the internal force, the local force, and the exchange force. As the basic units of load migration, these models help to flexibly move the force calculations among processors.

(iii) We demonstrate the validations of MDSLB by experiments on TianHe-1A supercomputer. Experimental results illustrate that MDSLB can solve the load imbalance problem excellently. Compared with the original parallel MD simulation, the average step time in MDSLB can be reduced by 34%–64%.

The rest of this paper is organized as follows. Section 2 briefly reviews the related work about the MD parallelization strategies and load balancing methods. Section 3 introduces the basic idea of MDSLB by an example. In Section 4 we present how the force computing models are built and then describe the allocation process by MDSLB in detail. Section 5 demonstrates our evaluation methodology and the results tested on TianHe-1A supercomputer. Finally, Section 6 concludes the paper and raises the future work.

2. Related work

Since the 1960s, researchers have begun to develop parallel algorithms for MD simulations and have found numerous interesting results.[19] Verlet[20] first proposed the Verlet Neighbor Lists method which opened the gate for the parallel MD simulations. Quentrec and Brot[21] put forward the idea of dividing the spatial region into subdomains, which serves as the foundation of spatial decomposition. Hockney and Eastwood[22] combined the cell structure and the linked lists method to serve computer simulations. S. Plimpton[23] summarized the parallel MD algorithms and divided them into three categories: atom decomposition, force decomposition, and spatial decomposition. A brief introduction is listed below.

i) Atom decomposition: All atoms are evenly distributed among the processors. Each processor is responsible for the force calculations, the velocity and position updates of the atoms it holds. Although atom decomposition can ensure load balance during the simulation, each processor needs to communicate with all the other processors, which produces extra communication costs.

ii) Force decomposition: The force matrix blocks are distributed across all processors according to the interactions between atoms. Compared with atom decomposition, this method does not need to communicate with all the other processors, which reduces the communication costs. However, both of the above methods should consider the interactions of all atoms, so they are not suitable for large-scale simulations.

iii) Spatial decomposition: The whole spatial region is decomposed into numerous cells. Each processor is assigned with one or more cells and responsible for the atoms within its cells. The region that each processor is in charge of is called "processor domain". In order to facilitate the realization of the load balancing mechanism, each processor is allocated many cells, and the size of which is designed relatively small. This fine-grained partition supports better load migration. Compared with the above two methods, spatial decomposition can effectively reduce the communication costs, but it will emerge load imbalance.

To solve the load imbalance brought by spatial decomposition, researchers have come up with various load balancing methods. We will discuss the key ideas of these methods below.

Intuitively, the static load balancing method aims to allocate the cells to every processor evenly while reducing the communication costs. There are mainly two kinds of implementations: Geometry–Partition and Graph/Hypergraph–Partition. Geometry–Partition is a simple mesh (a set of all cells in spatial region) partition model which just depends on the cell coordinate information. By using this model we can finish partitioning the meshes in a relatively short time, but the disadvantage is that the partition result is always of poor quality. The corresponding partitioning algorithms include: Recursive Coordinate Bisection (RCB),[24] Recursive Inertia Bisection (RIB),[25] Space-Filling Curve,[26] etc. The Graph/Hypergraph–Partition will firstly create a corresponding graph or hypergraph by making an abstraction in the top level of meshes. Afterwards, we can use some graph/hypergraph tools to finish the mesh partition. Compared with Geometry–Partition, Graph/Hypergraph–Partition may take more time but it can get a better partition result

for both the structured regular shaped meshes and the non-structured irregular shaped meshes. The typical implementations are Multi-Level K-Way Partition[27] and Hypergraph Partition.[28] Although the static method can guarantee the current load distribution to be balanced, it also faces challenges that particles move among different cells, and it leads to the load imbalance of each processor. Therefore, it is necessary to schedule the loads dynamically.[29–31]

The key idea of the dynamic load balancing method is to move the loads from overloaded processors to lightly loaded processors dynamically when the load imbalance appears. Deng *et al.*[16] suggested dividing meshes into square blocks or units to achieve load migrations dynamically. Hayashi *et al.*[32] proposed a three-dimensional (3D) load balancing method for the parallel MD simulations, and introduced "permanent cells" to reduce communication costs. A simple method introduced by Zhakhovskii *et al.*[33] is repartitioning the spatial region according to the current load distribution. Fattebert *et al.*[34] presented an automatic parallel load balancing method based on Voronoi cells domain decompositions, which gets good performance for quasi-2D and 3D practical applications. To explore the bimolecular functions, Kalé *et al.*[17] proposed a dynamic method based on computational units. Using their method, the computations are packaged into many computational units called "computes" which are transferred from overloaded processors to lightly loaded processors when the load is imbalanced. Although the dynamic method can guarantee load balance, there is no doubt that the load migrations will produce a lot of extra overheads.

3. Basic idea of MDSLB

3.1. An MD phenomenon and spatial decomposition

In MD simulations, there is a kind of phenomenon called "diffusion phenomenon" where particles move from the high concentration range area to the low concentration range area driven by the concentration difference, as shown in Fig. 1(a).

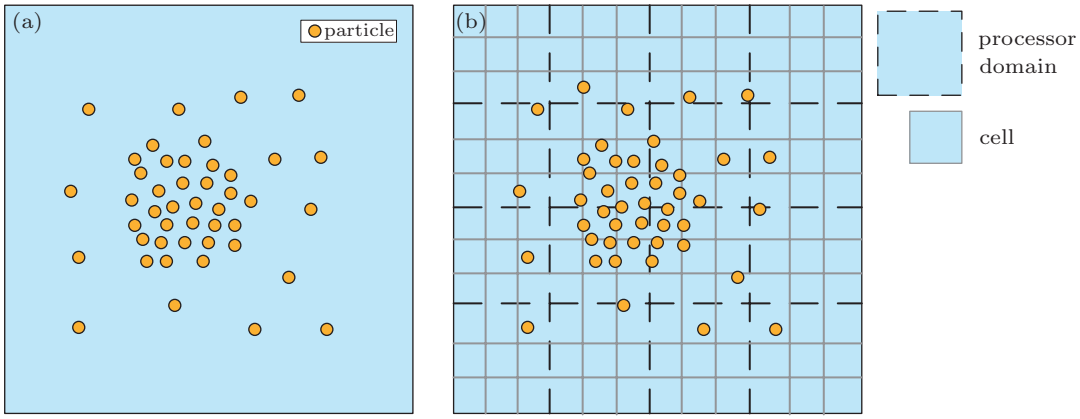


Fig. 1. (color online) Diffusion phenomenon. (a) The initial distribution of particles. (b) The spatial region is divided into 12×12 cells, every 9 cells are allocated to one processor.

The standard parallel molecular dynamics algorithm used for simulating this phenomenon is shown in **Algorithm 1**. In the parallel decomposition stage, there are many existing parallel decomposition algorithms (e.g., spatial decomposition) that can be used to distribute computing tasks to each processor. Afterwards the system parameters that specify the simulation conditions (e.g., initial temperature, number of particles, density) and initiate the properties of particles (e.g., positions and velocities) will be read in. When all conditions are ready, the MD loop can carry on with the step time iterating. In each time step, it mainly finishes two jobs: calculate the forces on all particles and integrate Newton's equations of motion. These two steps make up the core of the simulation which execute repeatedly until finishing the time evolution. After completion of the central loop, the averages of measured quantities we desired can be calculated. Therefore, it is not difficult to conclude that the time for getting the physical quantities primarily depends on the efficiency of each time step. The optimization for each time step can improve the computational efficiency of physical quantities. However, it is important to note that optimizing the computational efficiency of each time step will not change the physical process and the corresponding physical results.

In the parallel MD simulations, computations are always accompanied with communications; for example, when calculating the pairwise force between particles, each processor needs to get the remote particles back from other processors first, and then it can finish the force calculation. Therefore, each step time can be expressed as: $T_{\text{step}} = T_{\text{compt}} + T_{\text{comm}}$. T_{compt} is the time for calculating the force and integration consumed by the heaviest loaded processor. Obviously, when the load is imbalanced, it means that the heaviest loaded processor

will spend more T_{compt} to finish the computations. As shown in Fig. 1(b), according to the theory of spatial decomposition, the initial distribution of particles is non-uniform which will gain T_{compt}. By the way, T_{compt} will increase with the scales of molecular systems increasing, and decrease with the number of processors increasing. T_{comm} is the time brought by communication costs. When the scales of molecular systems are fixed, T_{comm} will increase as the number of processors increases.

Algorithm 1: Standard parallel molecular dynamics algorithm.

input: The properties of molecules and simulation conditions
output: Statistical averages (e.g., temperature and pressure)

```
Parallel decomposition;
Initialization;
Set t = 0;
for each time step do
    Calculate pairwise force (short-range force and long-range force);
    Calculate tether force;
    Calculate external force (e.g., gravity);
    Calculate the velocity v(t + Δt/2) = v(t − Δt/2) + Δt F(t)/m;
    Calculate the velocity v(t) = 1/2 [v(t − Δt/2) + v(t + Δt/2)];
    Calculate the position r(t + Δt) = r(t) + v(t + Δt/2) Δt;
    Set t = t + Δt;
    Calculate statistical averages;
end
Print the averages of measured quantities, and stop;
```

3.2. Dynamic load balancing method

We have mentioned in Section 1 that force calculations occupy most of the execution time in MD simulations. Based on this insight, the dynamic load balancing methods can be classified into two categories. For the first category, force calculations are migrated from overloaded processors to lightly loaded processors (the particles are still belonging to the overloaded processors), as shown in Fig. 2(a). The second category, as shown in Fig. 3, repartitions the meshes according to the current load (particles) distribution periodically. However, with the time step moving on, the particles will move among different processors, which may result in load imbalance. For example, the lightly loaded processors shift the loads to the overloaded processors, as shown in Fig. 2(b). Therefore, we need to schedule the loads dynamically, which will incur extra overheads.

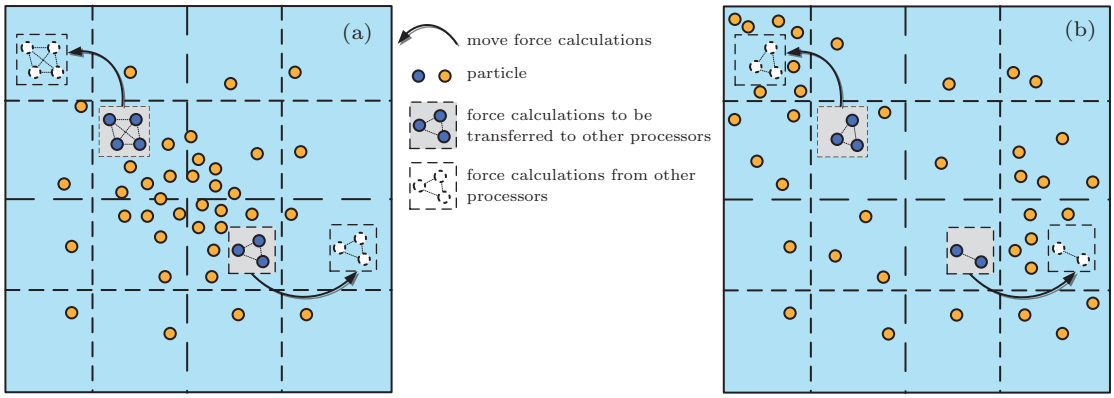


Fig. 2. (color online) Force calculation migration. (a) Move the force calculations from the overloaded processors to the lightly loaded processors according to the current distribution of particles. (b) The load is imbalanced so that the lightly loaded processors abnormally migrate the force calculations to the overloaded processors after a time.

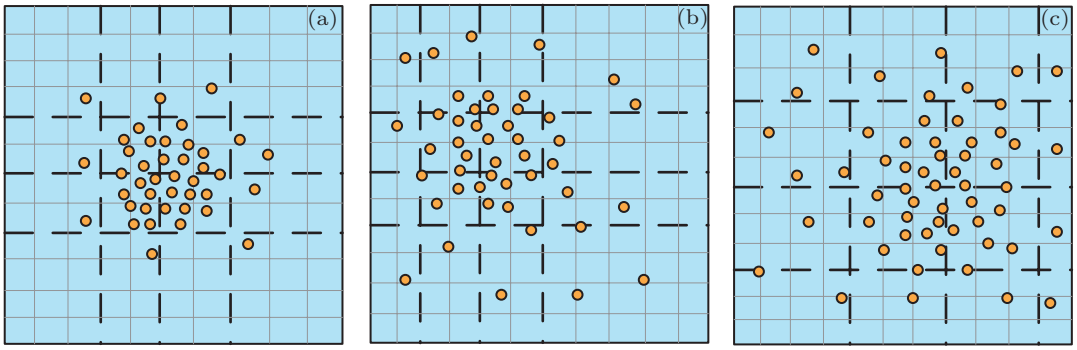


Fig. 3. (color online) Mesh repartition. (a) Particles are centralized in the center region. (b) Particles are centralized in the left region. (c) Particles are centralized in the right region.

3.3. The MDSLB method

In order to reduce the extra overheads, we propose our MDSLB, a new MD load balancing method where the cells in a spatial region are divided into a lot of sub-regions called "local domains". In two-dimensional space, there are 4 neighbor cells in each local domain (8 cells in three-dimensional space), as shown in Fig. 4, and the computations involving each cell is called "cell load". Each cell load is composed by not only the force calculation between two particles within one cell but also those from two neighbor cells, the details of force calculations will be described in Section 4. We iteratively allocate the cell loads of each local domain to every processor to realize the computations shared by all processors. Compared with the dynamic method, MDSLB can ensure load balance in the whole process of the simulation without load migrations.

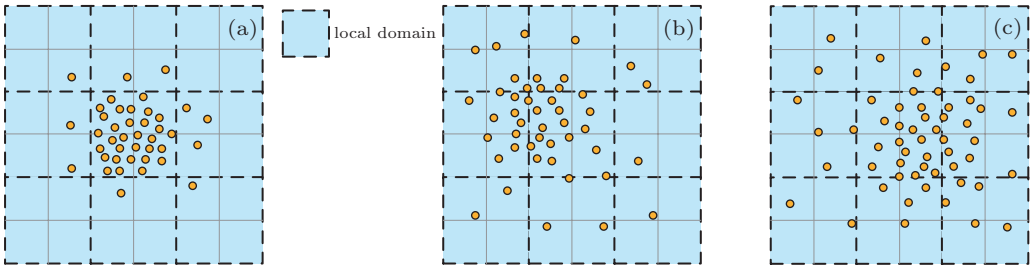


Fig. 4. (color online) Local domains. (a) Particles are centralized in the center region. (b) Particles are centralized in the left region. (c) Particles are centralized in the right region.

4. Force calculations and static allocation

Inspired by Kalé *et al.*,[17] we extract the force calculations out of the execution flow of the original parallel MD simulation (without load balancing strategies), and design the independent force models based on the results of the traditional static load balancing algorithms. Furthermore, we implement the corresponding allocation and force calculation schemes for each kind of force model.

4.1. Force classifications

After executing the traditional static load balancing methods, each processor is allocated with plenty of cells that can be divided into two types: exchange cells and local cells. As shown in Fig. 5, the exchange cells lie at the intersection of processor domains, and they need to exchange particle data with the neighbor processors when the interacting particles are not in the local processor. The local cells refer to the cells inside a processor domain, and they do not need to communicate with other processors (separated by at least one layer of cells).

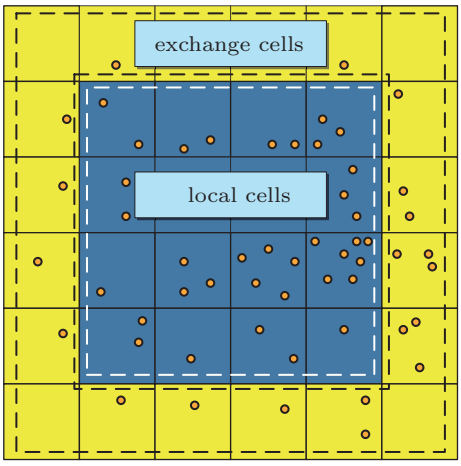


Fig. 5. (color online) Local cells and exchange cells in a processor domain.

Based on the two types of cell structures, we classify the short-range force into three kinds of force models: internal force, local force, and exchange force, as shown in Fig. 6.

(I) Internal force: The interaction between the two particles in the same cell is called the "internal force". Due to the fact that internal force just involves a single cell, we can package one cell as the basic unit of internal force.

(II) Local force: We regard the force involving the two particles in two neighbor local cells as the "local force", whose calculation does not need to communicate with other processors. Therefore, we can operate a pair of interacting local cells to handle local force.

(III) Exchange force: If the two interacting particles belong to two exchange cells from different processors, the force calculation will need to communicate with the neighbor processor to get the other particle data back. We call this type of force the "exchange force", which involves a pair of exchange cells.

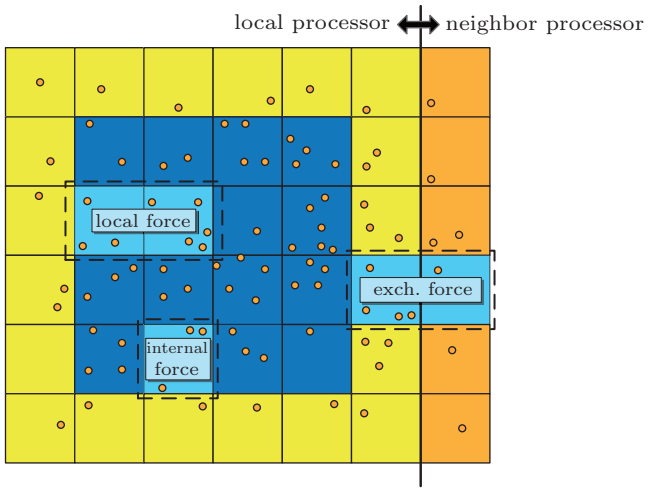


Fig. 6. (color online) Internal force, local force, and exchange (exch.) force.

4.2. MDSLB

In this subsection, we will introduce the MDSLB method in detail which aims to allocate the three categories of force calculations to every processor evenly. From the point of view of computers, the purpose is to specify the loads to every processor. When the allocation is finished, we also present the force calculation of each force model. In the previous work, we have found some useful concepts, such as processor domains, local domains, two types of cell structures, and three kinds of force models which provide the foundation for our load balancing method. Besides, the execution process of our method needs to traverse all local domains, so we design a suitable index scheme. As shown in Fig. 7, the local domains are indexed by the pattern of letter "Z" in X–Y plan. When finishing searching the local domains in one X–Y plan, we can continue getting to the next X–Y plan in the Z axis.

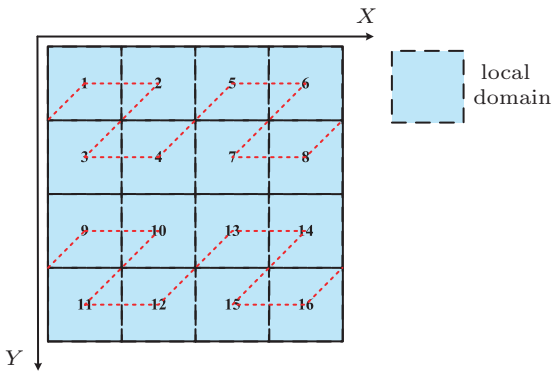


Fig. 7. (color online) The index scheme for local domains.

4.2.1. Internal force

The internal force calculation only involves one cell, therefore, its allocation is easiest to implement. We cyclically denote a number to each cell in local domains following the order of X, Y, and Z axes, and the maximum number is equal to the number of processors. The two-dimensional allocation with four processors is shown in Fig. 8. **Algorithm 2** introduces the details of the internal force allocation of each processor.

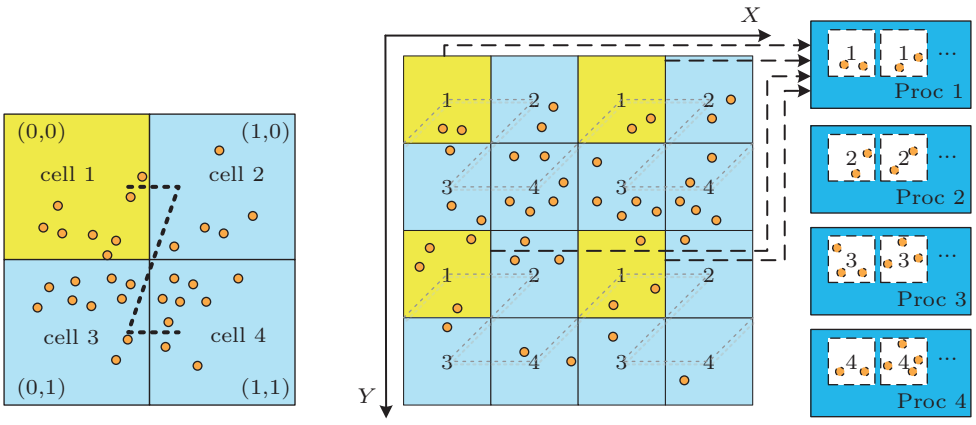


Fig. 8. (color online) Internal force allocation. In two-dimensional space, as the units of internal force, each cell in local domains is labeled with a processor number (1, 2, 3, or 4) and allocated the corresponding processor (Proc 1, Proc 2, Proc 3, or Proc 4).

In three-dimensional space, each local domain consists of 8 cells, which form a cubic local domain. The allocation strategy is similar to that of two-dimensional space, and adds "**for** $z = 0$ to $z = 1$" below the step 2 of **Algorithm 2** and changes $\boldsymbol{Cell}_{(x,y)}$ to $\boldsymbol{Cell}_{(x,y,z)}$.

The process of the internal force calculation is shown in Fig. 9. First of all, each processor needs to send particle data to other processors, which helps to calculate the internal forces, and at the same time each processor also receives the particle data from these processors. When the communication process is over, each processor will begin to calculate its own internal forces and those of other processors. After all the internal forces have been calculated, each processor must send the calculation results of the internal forces back to their original processors.

Algorithm 2: Internal force calculation (two-dimensional space).

input: The number of processors N
output: Internal force allocation

```
1  Set P = 0 ;                                   // P is the index of processor numbers
2  for every Local Domain in processor domain; // The index scheme is in Fig.7
3  do
4      for y = 0 to y = 1;                       // y is the index of Y axis
5      do
6          for x = 0 to x = 1;                   // x is the index of X axis
7          do
8              Allocate Cell_(x,y) to (P%N) processor;
9              Set P = P + 1;
10         end
11     end
12 end
```

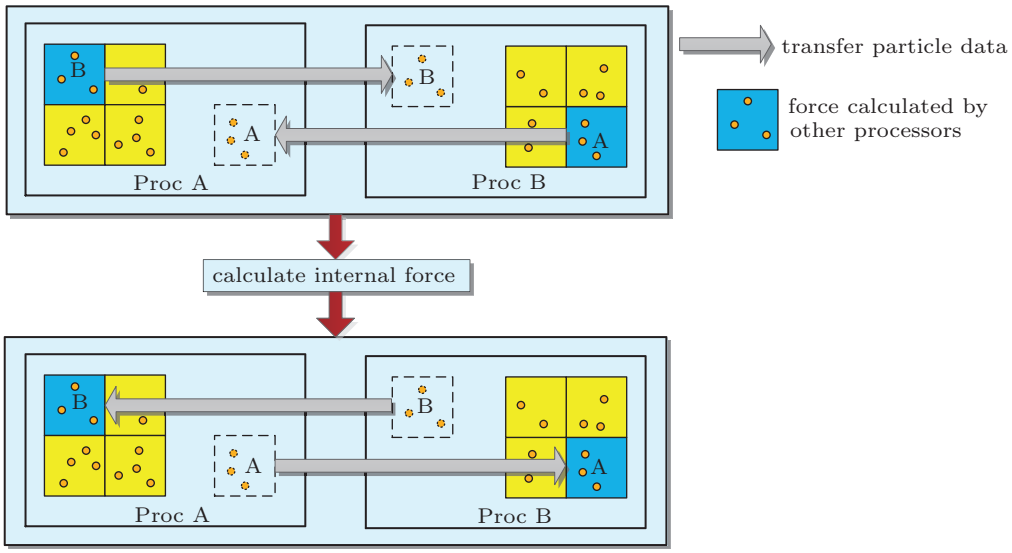


Fig. 9. (color online) Internal force calculation. We focus on two of the whole processors for example, processor A (Proc A) receives the cell with particle data from Proc B to help to calculate Proc B's internal forces, and so does the Proc B. After the force calculation, Proc A and Proc B should send the particle data received from other processors back for the following velocity and position updates.

4.2.2. Local force

Local force calculation involves two neighbor cells in the same processor domain, therefore, the allocation and force calculation are more complicated than those of internal force. According to the Cell Lists method,[9,22] each cell just interacts with the 8 neighbor cells (26 cells in three-dimensional space). In MD simulations, we generally use Newton's third law to re-

duce the repeated computations, as shown in Fig. 10.

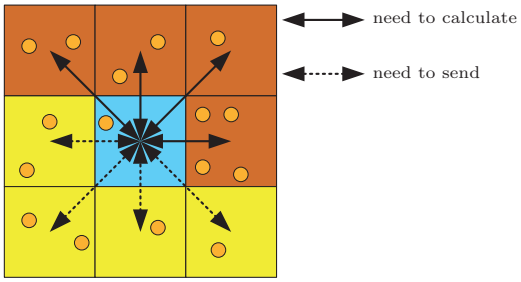


Fig. 10. (color online) Using Newton's third law to simplify the calculation of local force. When calculating the local force for the center cell (blue color), we can only calculate the forces with the neighbor cells (brown color) at the top right corner. The interactions with the neighbor cells (yellow color) at the lower left corner can receive from the yellow cells based on Newton's third law.

The local force allocation also uses the cyclic number method to iterate over all of the cells in each of the processor domains. The allocation process with four processors is shown in Fig. 11. The **Algorithm 3** presents the general case detailedly.

The process of local force calculation is also similar to internal force, but the transmission unit is a pair of cells instead of a single cell, as shown in the following figure (Fig. 12). When the calculations of the overall local forces are finished, each processor also needs to send to and receive from other processors the particle data.

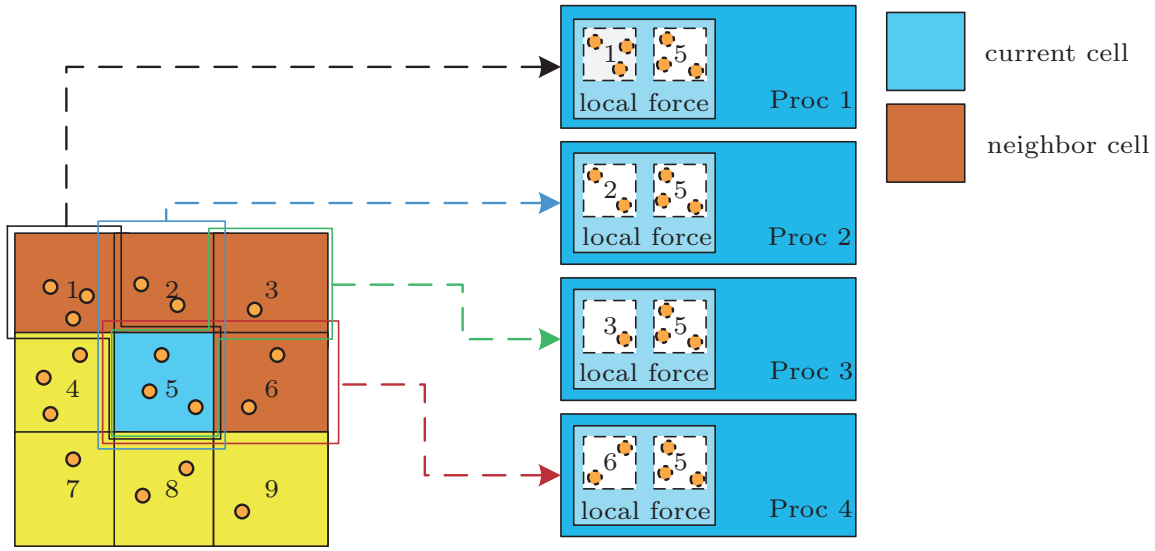


Fig. 11. (color online) Local force allocation (two-dimensional space). Cell 1, cell 2, cell 3, and cell 6 are composed local forces with cell 5. If 4 processors can be used, we can assign the pair of cell 1 and cell 5 to processor 1 (Proc 1), that of cell 2 and cell 5 to processor 2 (Proc 2), that of cell 3 and cell 5 to processor 3 (Proc 3), and that of cell 6 and cell 5 to processor 4 (Proc 4).

Algorithm 3: Local force allocation (two-dimensional space).

input: The number of processors N
output: Local force allocation

```
Set P = 0;
for every Local Domain in processor domain do
    for y = 0 to y = 1 do
        for x = 0 to x = 1 do
            for every Neighbor Cell of Cell_(x,y) do
                Allocate Pair (Cell_(x,y), Neighbor Cell_j) to (P%N) processor;
                Set P = P + 1;
            end
        end
    end
end
```

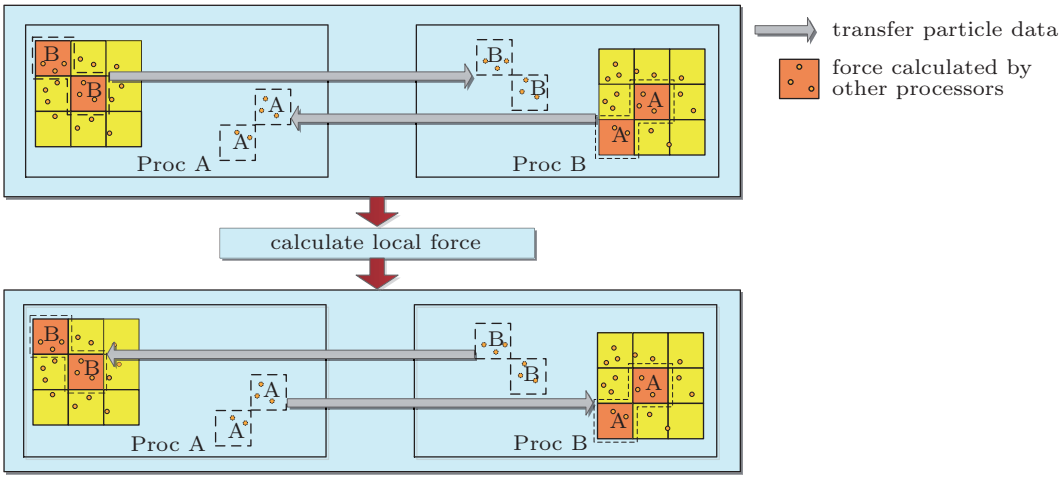


Fig. 12. (color online) Local force calculation. Proc A receives the cell pair (brown) with particle data from Proc B to help to calculate Proc B's local forces, and so does the Proc B. After the force calculation, Proc A and Proc B should send the particle data received from other processors back for the following velocity and position updates.

4.2.3. Exchange force

Compared with local force, the pair of cells involved in exchange force is from different processors. Therefore, before the allocation of exchange force, it is necessary to get the other cell back from the other processors. Afterwards, we can allocate the exchange forces, as shown in Fig. 13. **Algorithm 4** presents the allocating process of exchange force.

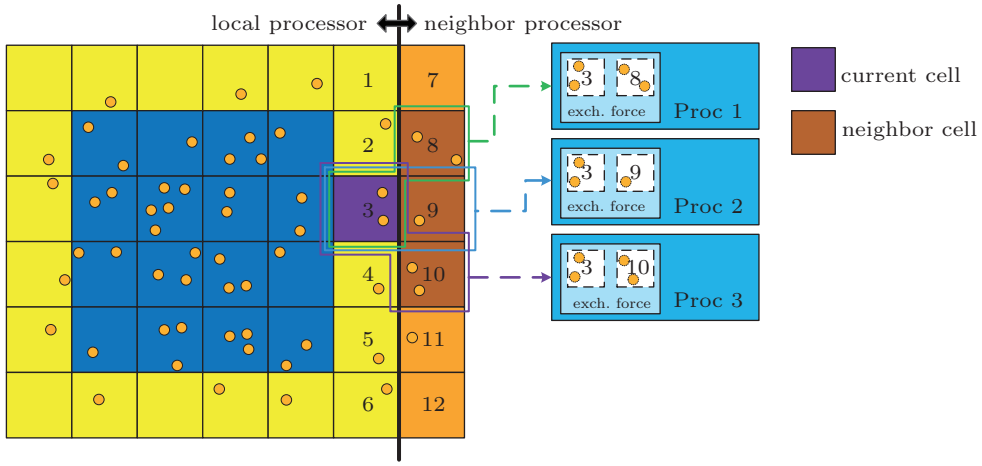


Fig. 13. (color online) Exchange (exch.) force allocation (two-dimensional space). Exchange cell 3 has three neighbor exchange cells from the neighbor processor, so exchange cell 3 can compose three exchange forces (three pairs of exchange cells) with cell 8, cell 9, and cell 10. If there are three processors, we can allocate the three exchange forces to all the three processors circularly.

Algorithm 4: Exchange force allocation (two-dimensional space).

input: The number of processors N
output: Exchange force allocation

```
Send Exchange Cells to neighbor processors;
Receive Exchange Cells from neighbor processors;
Set P = 0;
for every Exchange Cell in processor domain do
    for every Neighbor Cell of Exchange Cell_i do
        Allocate Pair (Exchange Cell_i, Neighbor Cell_j) to (P%N) processor;
        Set P = P + 1;
    end
end
```

Notice that the exchange force calculation can be divided into two situations. One is that all the two cells are not located in the local processor, so the calculation process of which is similar to local force. The other is that one of the two cells is in the local processor, but the other cell is in the neighbor processor. On the second occasion, we just need to get the other cell with particle data back before calculating the forces, as shown in Fig. 14.

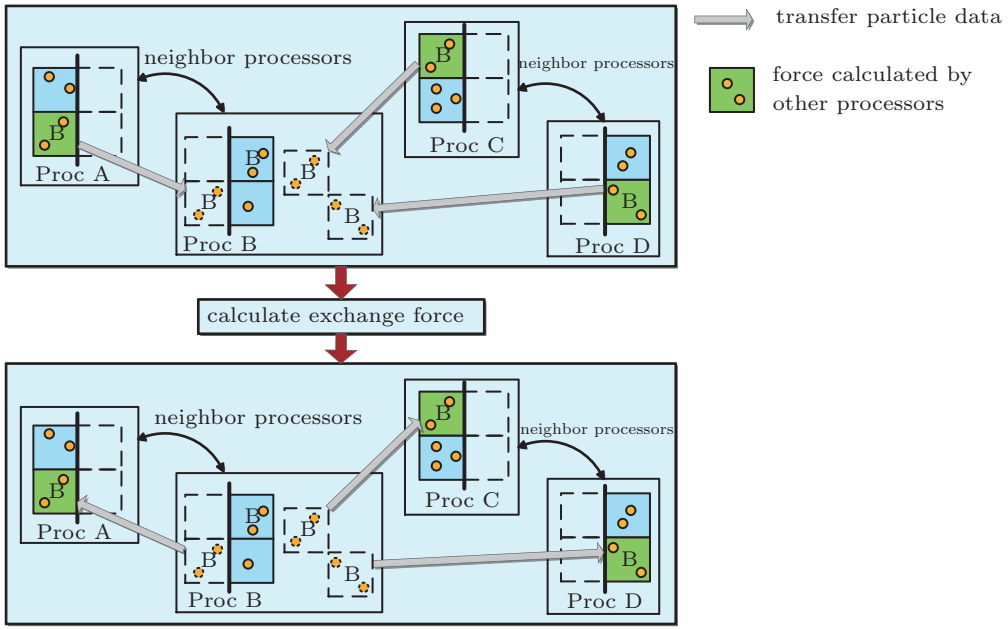


Fig. 14. (color online) Exchange force calculation. For the first situation, Proc C and Proc D are neighbor processors and transfer the cells with particle data to Proc B for calculating exchange forces. For the second situation, Proc B will receive the other cell with particle data from Proc A. After the force calculation, Proc B should send the particle data received from Proc A, Proc C, and Proc D back for the following velocity and position updates.

In summary, when using MDSLB to calculate the forces, each processor needs to pay some communication costs (transfer particle data), but when compared with the optimization of force calculations and the overheads of load migrations brought by the dynamic methods, the costs can be ignored. In the following section, we will explain this with the results of our experiments.

4.3. Analysis of computational complexity

4.3.1. Internal force

If the number of the whole cells is M and the number of processors is P, the complexity of the internal force allocation for each processor is $O(M/P)$. The overheads of internal force calculation include two parts. One part is that each processor needs to transfer about $M/P\times(P-1)/P\times\delta$ particles with other processors (N is the number of the whole particles, δ is the average number of particles in one cell, and $\delta\propto N/M$) in one time step, and its complexity is $O(N/P)$. The other part is the force calculation between particles: each processor is responsible for M/P cells, therefore, the overheads can be expressed as $M/P\times\delta(\delta-1)/2$, whose complexity is $O\left(N^2/PM\right)$.

4.3.2. Local force

For local force, each cell will combine with several neighbor cells. If ρ is the average number of cells adjacent to each local cell, then each processor will own $\rho M/P$ pairs of local cells. In three-dimensional space, when the shape of cells is a regular cube, each cell will have 26 neighbor cells ($\rho\approx 13$). Hence, the complexity of the local force allocation for each processor is $O(M/P)$. The overheads and the corresponding complexity of particle data transferring are $\rho M/P\times(P-1)/P\times\delta$ and $O\left(N^2/PM\right)$, respectively. For the local force calculation, the cost is $\rho M/P\times\delta\times\delta$, and the complexity is $O\left(N^2/PM\right)$. In MDSLB, the number of particles that each processor is in charge of is roughly equal, so the overheads of the local force calculation can be expressed as $N/P\times\delta\times\rho$, and the complexity is $O\left(N^2/PM\right)$.

4.3.3. Exchange force

The process of the exchange force allocation is similar to local force. The corresponding overheads and complexity are $(P-1)/P\times\eta\times\gamma$ and $O(M/P)$, γ is the average number of neighbor cells from other processors for each exchange cell, and η is the average number of exchange cells owned by each processor ($\eta\propto M/P$). Before force calculation, each processor needs to transfer $\eta\times\gamma\times\delta\times(P-1)/P$ particles with other processors whose complexity is $O(N/P)$. The overheads of the exchange force calculation are $\eta\times\gamma\times\delta\times\delta$, and the complexity is $O\left(N^2/PM\right)$.

With the above analysis, we have gotten the overheads of three kinds of force models: $M/P\times\delta(\delta-1)/2$ for internal

force, $\rho M/P \times \delta \times \delta$ for local force, and $M/P \times \gamma \times \delta \times \delta$ for exchange force. They are different with the factors: $(\delta - 1)/2$, $\rho\delta$, and $\gamma\delta$ (ρ and γ are constants), which are called internal factor, local factor, and exchange factor, respectively. In normal circumstances, ρ is the biggest of all, and γ is bigger than $1/2$. With the density of particles increasing, the difference among the three kinds of forces becomes larger.

Compared with force calculations, the overheads of integrations are quite small. Velocity and position updates for one particle take only constant time, hence the total complexity of integration is $O(N/P)$, which differs with a factor of N/M from the force calculation ($O\left(N^2/PM\right)$). If the density of particles is relatively big, the number of particles in one cell will increase, which also leads to the augment of N/M. As a result, the overheads of force calculations become much bigger than integrations. When the load is imbalanced, the densities of the cells are quite different, which directly results in the fact that the force calculations in one step takes much more time than the load balanced cases. In MDSLB, each processor is in charge of the same number of particles, thus the density is close to N/M, which guarantees load balance.

5. Experiments

To present the effectiveness of our method, we simulate the diffusion processes of several molecular systems with the standard TIP4P water model, which consists of a Lennard–Jones site for the oxygen atom and three charge sites. For each system, the water molecules are distributed in a fixed box that the molecules cannot pass through, and are move driven by the interacting forces from others. In addition, in order to show the performance results from all aspects, the molecular systems may differ about the box size and the distribution of molecules. Meanwhile, some settings and parameters are invariable in the whole testing, such as the initial temperature ($T = 298$ K), the interval of each time step ($\delta t = 0.001$ ps), and the size of each cell ($1.6 \times 1.6 \times 1.6 \times 10^{-27}$ m^3). In the initial stage of simulation, molecules are arranged in FCC configuration according to the mass density distribution which is specified in each test, and the velocities are randomly assigned from the Maxwell–Boltzmann distribution. The methods and aims of each test are introduced below.

In order to prove the conclusion of Subsection 4.3, we design a test that includes a molecular system with 1, 124, 864 water molecules to show the percentages of the three kinds of forces with 16 to 512 processors, as shown in Subsection 5.2. In Subsection 5.3, we use two processors to simulate the left–right diffusion of water molecules in a cubic box to demonstrate the effectiveness of our method. The tests in Subsection 5.4 are designed to present the scalability of MDSLB with 16 to 512 processors. Subsection 5.4.1 introduces a particle scalability test that keeps the number of processors fixed and changes the total number of molecules from 238, 328 to 1, 685, 159, which shows the scalability of our method for different molecular systems in a fixed parallel scale. Correspondingly, in Subsection 5.4.2, we design two kinds of tests: the left–right diffusion and the center diffusion. In the tests, we keep the molecular systems fixed and change the number of processors from 16 to 512. In addition, in Subsection 5.5, we construct a load balanced molecular system where each cell has the same number of molecules to test the extra overheads of the communications for answering the question proposed in Subsection 4.2.

5.1. Platforms

Our hardware platform is TianHe-1A supercomputer, a CPU/GPU heterogeneous computing system, which is the world's fastest supercomputer in the TOP500 list in 2011. In TianHe-1A, each computing node is equipped with two 2.93 G Intel Xeon X5670 CPUs and 24 GB RAM. Notice that although TianHe-1A consists of CPUs and GPUs, we just use the CPUs in our experiments. We implement MDSLB in OpenFOAM, which is an excellent piece of CFD open source software with C++ object-oriented class libraries and support for large-scale parallel computation. Meanwhile, we can adopt the outstanding static mesh partition methods from OpenFOAM to implement our load balancing strategies.

5.2. Factor testing

In Subsection 4.3, we have proven that the three forces differ with factors of $(\delta - 1)/2$, $\rho\delta$, and $\gamma\delta$. When the density changes from 10 molecules/cell to 100 molecules/cell, the corresponding factors are shown in Fig. 15 where we can see that the differences of the three forces become increasingly larger. If $\delta = 100$, the local factor is 1, 300, the internal factor is 49.5, and the exchange factor is 500. The result shows that internal force is the smallest one, and the exchange force calculation consumes about 10 times more than internal force, for local force the multiple is 26.

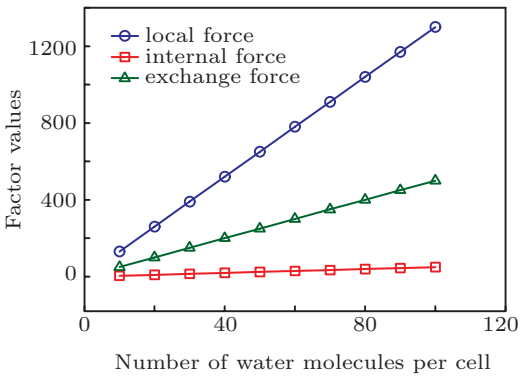


Fig. 15. (color online) The tendency of force factors.

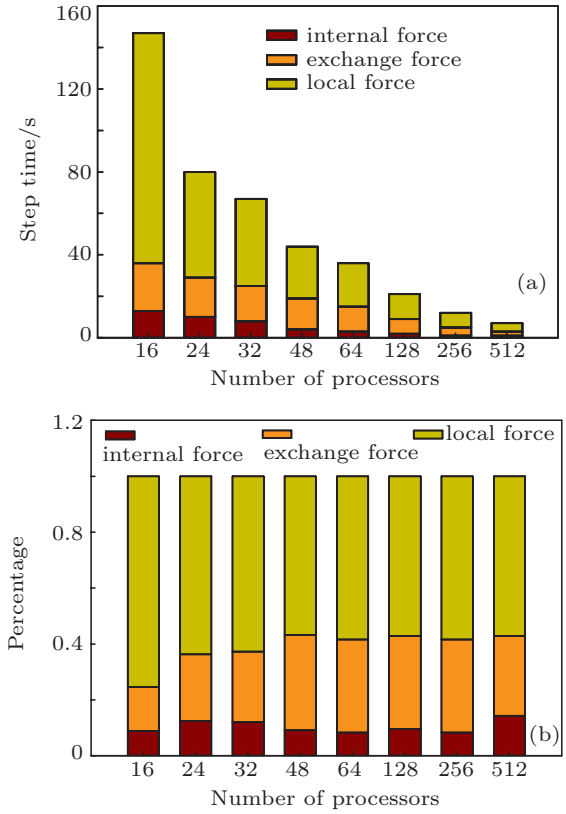


Fig. 16. (color online) The proportions (a) and percentages (b) of the three kinds of forces.

In the parallel MD simulations, the overheads of force calculations consist of not only the computations but also the extra communication costs. In addition, as the scale becomes larger, the proportion of communication costs also grows bigger, as shown in Fig. 16(a). Figure 16(b) shows the percentages of the three forces with 16 to 512 processors. When using a small number of processors, the exchange cells occupy a low proportion of the whole cells. But, as the number of processors increases, the percentage of exchange cells gets bigger, hence, the exchange force calculations will take more time than before. In this test, the proportions of internal force and local force calculations are relatively fixed.

5.3. Double-processor results

In order to demonstrate the effectiveness of our method, we use two processors to simulate the left–right diffusion that the mass density of the left region is higher than that of the right region. Driven by the density difference, water molecules will move from the left region to the right region, as shown in Fig. 17.

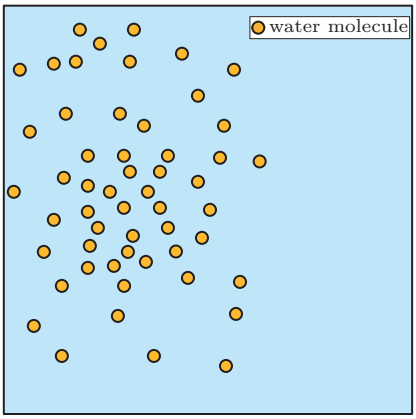


Fig. 17. (color online) The left–right diffusion.

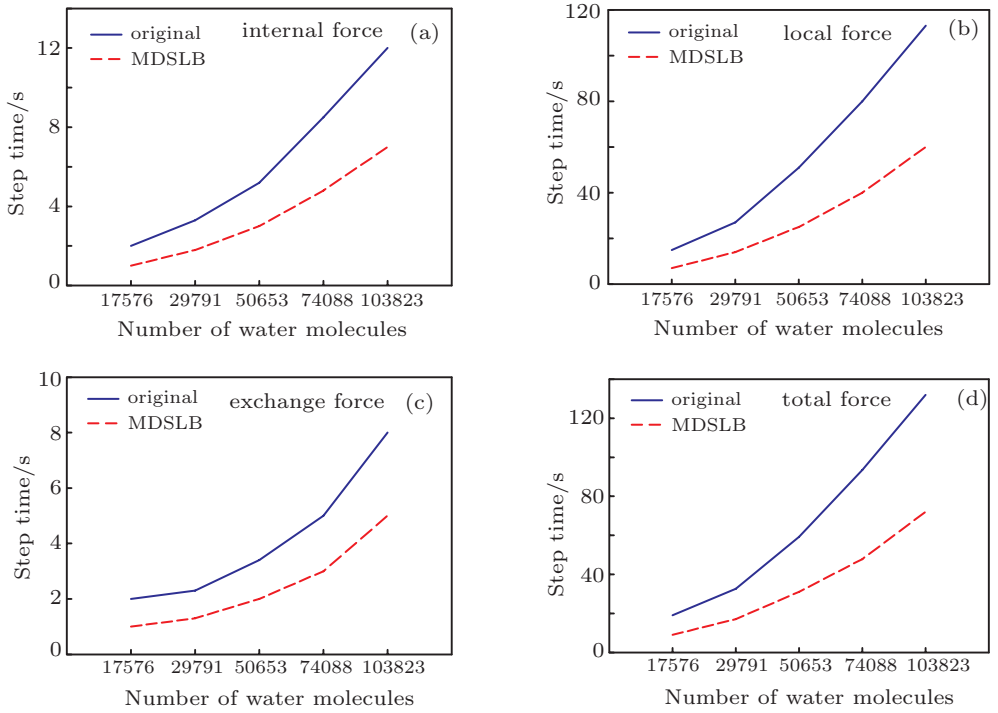


Fig. 18. (color online) The comparative results of two processors for internal force (a), local force (b), exchange force (c), and total force (d).

We keep the size of the spatial region fixed, and then change the number of water molecules from 17, 576 to 103, 823. Figure 18 shows the average step time consumed by the three kinds of force calculations with the original parallel MD simulation and MDSLB, respectively. In this test, the two processors are load-imbalanced; namely, the processor which is responsible for the left region owns more molecules than the other processor, so it will be in charge of more force calculations, which makes that processor overloaded. By just using two processors to test the communication costs are smaller than the computations so that T_{compt} is the decisive factor of T_{step}. In this test, T_{compt} of the original method will increase observably because the heaviest loaded processor owns most of the particles. But our method can achieve the load balance of each processor by allocating the cell loads of each local domain to both of the two processors, which apparently decrease T_{compt}. For internal force, MDSLB can save 42%–50% time than the original method (Fig. 18(a)). For local force, the saved time is 47%–53% (Fig. 18(b)). Overall, the computations of internal force and local force in the left region are always heavier than those in the right region, so for the two kinds of force models the optimization result is desirable. Meanwhile, due to the low proportion of exchange cells, it can save 37.5%–50% execution time for exchange force calculation (Fig. 18(c)). As a result, the sum of the three forces can get an acceleration from 45% to 53% (Fig. 18(d)).

5.4. Multi-processor results

5.4.1. Particle scalability

In this subsection, similarly, we design four molecular systems for simulating the left–right diffusion process. For each system, the initial parameters are the same, however, the only difference is that the scale of each box is different which leads they have different number of molecules (the four molecular systems consist of 238, 328, 474, 552, 804, 357, and 1, 685, 159 molecules respectively). We use 32, 64, 128, and 512 processors to test the simulation performance of the four molecular systems, the results are shown in Fig. 19.

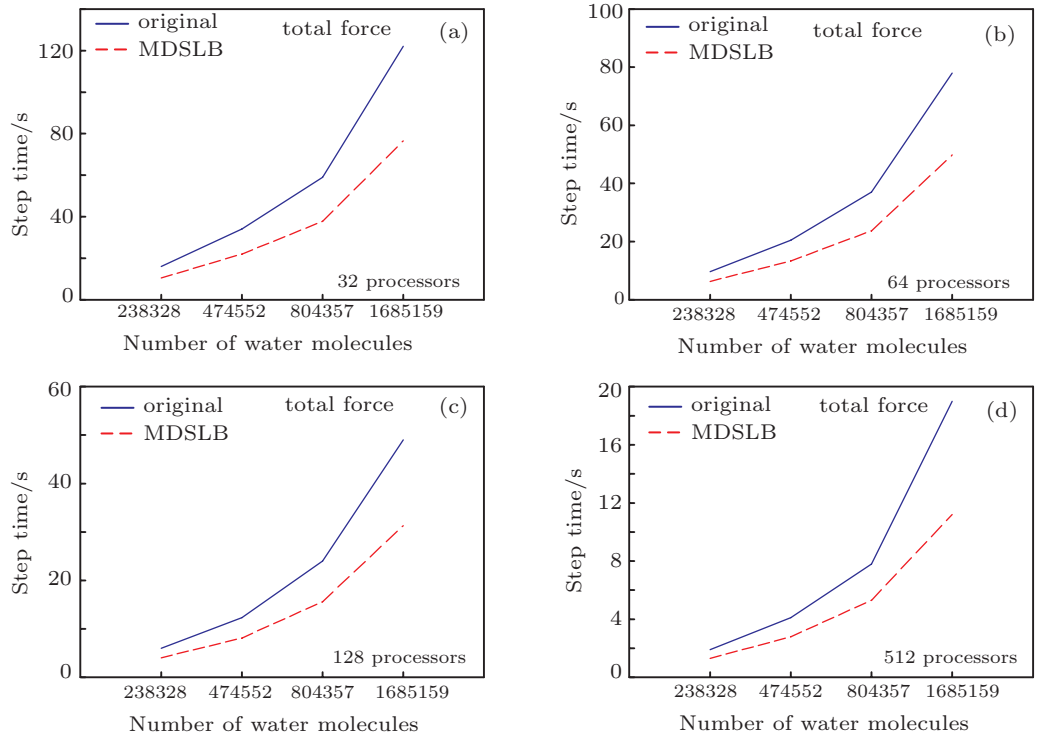


Fig. 19. (color online) The comparative result of left–right diffusion for total force in 32 processors (a), 64 processors (b), 128 processors (c), and 512 processors (d).

When the parallel scale is not very large (32 processors or 64 processors), compared with the original method, T_{compt} is also the decisive factor of T_{step}, so our method can get 34.5%–37% acceleration for 32 processors (Fig. 19(a)) and 34%–36% acceleration for 64 processors (Fig. 19(b)). As the number of processors increases, such as 128 processors or 512 processors, the communication costs T_{comm} will occupy larger proportions of T_{step} than it does in 32 processors or 64 processors. However, our method can get almost the same acceleration proportion: 33%–36% for 128 processors (Fig. 19(c)) and 31.5%–35% for 512 processors (Fig. 19(d)). One reason for this is that the optimization of our method to T_{compt} is more

than the increase of T_{comm}. The other is that, with the number of processors increasing, the overheads of communications between processors only gain a few proportions of T_{comm}, which we will introduce in Subsection 5.5.

5.4.2. Processor scalability

This test is used to prove the scalability of our method when the molecular systems are fixed. In order to show the performance from different aspects, we simulate two molecular systems with different processors: one is the left–right diffusion, and the other is called the center diffusion that water molecules are located in the center region and spread to all directions, as shown in Fig. 20.

For the left–right diffusion, we similarly create the box and configure the appropriate mass density for the left region and the right region, making the number of molecules add up to 1, 124, 864. In this test, the computations involved in local force and exchange force are relatively more than the communication costs, so the optimization of our method to T_{compt} can overlap the communication costs T_{comm}. As shown in Fig. 21, we can get the desirable results: 34%–50% acceleration for local force (Fig. 21(b)), and 22%–50% acceleration for exchange force (Fig. 21(c)). However, there are some ups and downs for internal force with 512 processors (Fig. 21(a)). The reason is that the internal force calculations only occupy a small part of the force calculations (discussed in Subsection 5.3), thus the impact of communication costs to T_{step} is relatively serious. The total force can save 34%–41% time compared with the original method, as shown in Fig. 21(d).

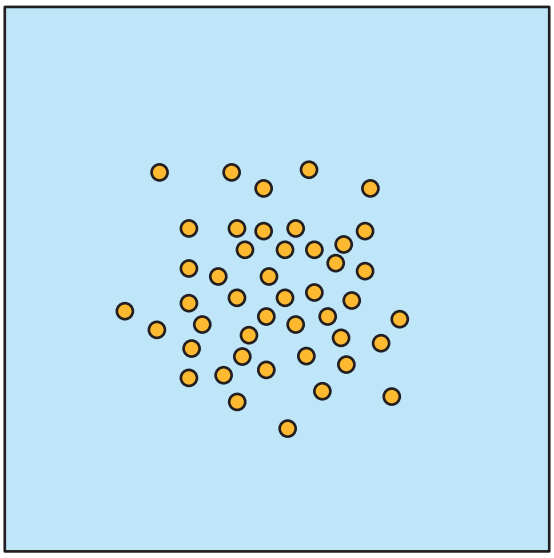


Fig. 20. (color online) The center diffusion.

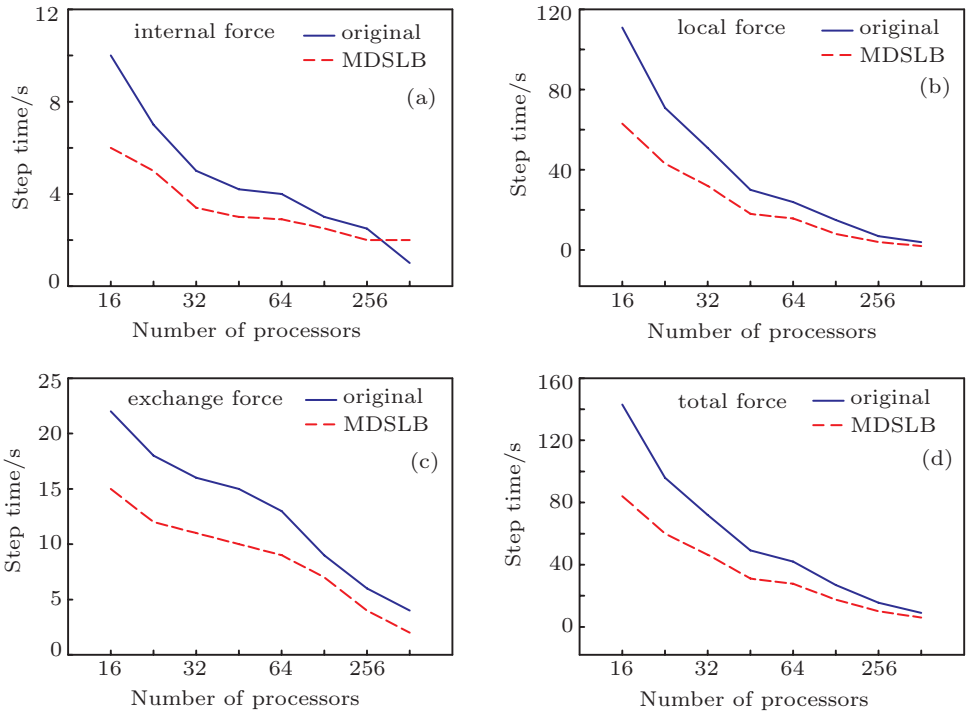


Fig. 21. (color online) The results of left–right diffusion with multiple processors for internal force (a), local force (b), exchange force (c), and total force (d).

Considering the simulation for the center diffusion, obviously, the initial load distribution is imbalanced since the processors which are in charge of the center region will take more loads (water molecules). With the step time iterating, the surrounding regions will be filled with molecules, and the load will be balanced. However, our method focuses on the early phase of diffusion, for the load balanced case, we can use the original method to solve.

This molecular system consists of 278, 512 molecules, and the simulation results for the center diffusion are shown in Fig. 22. The acceleration for each force model is: 22%–44% for internal force (Fig. 22(a)), 40%–77% for local force (Fig. 22(b)), and 29%–50% for exchange force (Fig. 22(c)). In summary, the total force can also save 45%–64% time than before, as shown in Fig. 22(d). We can see that the optimization for all the three forces is better than the left–right diffusion. In general, this is because the molecules are more concentrated, the heaviest loaded processor will be responsible for more particles which increases the T_{compt}. Therefore, the optimization of our method to T_{compt} can be more obvious. This also proves the effectiveness of the molecular dynamics static load balancing method from another aspect.

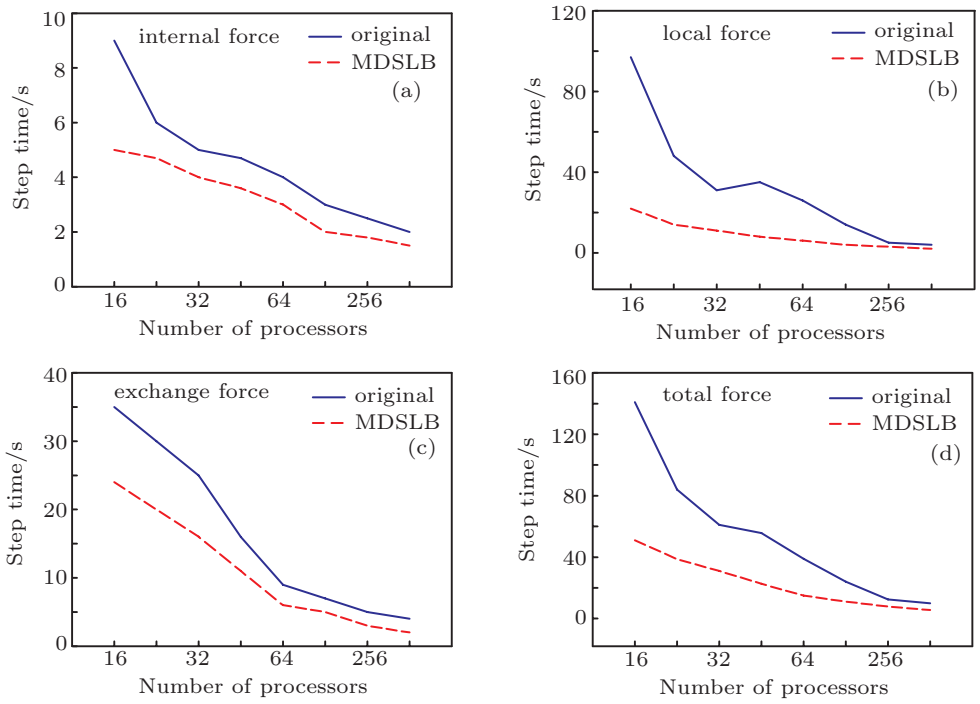


Fig. 22. (color online) The results of center diffusion with multiple processors for internal force (a), local force (b), exchange force (c), and total force (d).

5.5. Worst cases

Through the above discussion, for the non-uniform distribution cases, our method can yield good effects. But if the molecules are distributed evenly among the whole spatial region, as shown in Fig. 23, our load balancing strategies will incur some communication overheads (particle data transmission) instead.

We test the worst cases with 32 processors by changing the number of molecules from 110, 592 to 1, 124, 864. According to the theory of our method, when each processor owns almost the same number of particles, T_{compt} of the original method and MDSLB is much the same. Meanwhile, the communications brought by our method will gain T_{comm} which leads to the increase of T_{step}. As shown in Fig. 24, the communication overheads account for 6%–8% in the total force calculations. As the number of water molecules increases, the proportion of the extra overheads becomes lower. In general, although the communication brings some overheads, compared with the acceleration of force calculation, the overheads are small and can be ignored.

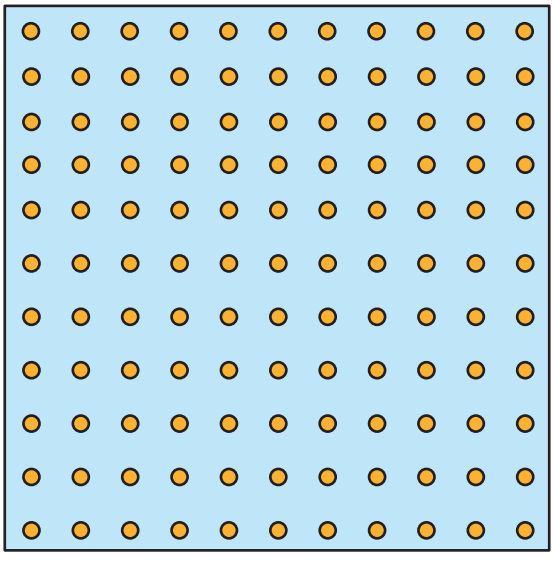


Fig. 23. (color online) Water molecules are distributed evenly in the spatial region.

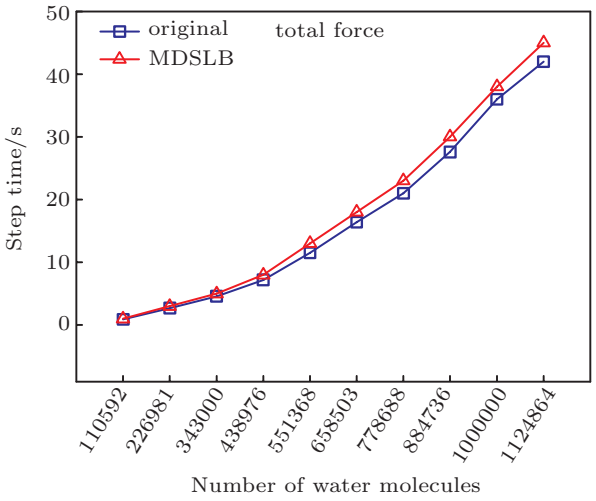


Fig. 24. (color online) The results of uniform distribution testing.

6. Conclusion and future work

With the MD method, we can get the motion details of particles, especially for the non-equilibrium systems, we can also calculate the statistical averages or simulate the physical processes directly over a period of time (always less than 200 ps). Considering that the MD method owns high parallelism, more and more researchers use this method to simulate large molecular systems in supercomputers. However, with the parallel scale increasing, for some non-equilibrium systems the distributions of particles are always non-uniform and change frequently, which will incur the load imbalance problem and decrease the parallel efficiency.

In this paper, we present a new static load balancing method (MDSLB) to solve the load imbalance problem that has happened in the parallel MD simulations. In this method, we classify the short-range force into three types: internal force, local force, and exchange force. Meanwhile, we have designed the corresponding static allocation and force calculation strategies. The features of MDSLB are as follows: it can guarantee load balance by executing the algorithm only once at program startup instead of performing the load migration dynamically. Our experimental results demonstrate that MDSLB can save 34%–64% time for the load imbalanced cases.

In our future work, we will use more processors to simulate the large-scale molecular systems to prove the scalability of our method. Considering that our method will produce some extra overheads, we plan to introduce load testing strategies that choose the suitable method flexibly according to the load distribution. When the load is imbalanced, we can use the MDSLB method, otherwise we adopt the original method.

References

[1] Rahman A 1964 *Phys. Rev.* **136** 405

[2] Zhu R Z and Yan H 2011 *Chin. Phys. B* **20** 016801

[3] Harvey M J and De Fabritiis G 2012 *Drug Discovery Today* **17** 1059

[4] Wu T Y, Lai W S and Fu B Q 2013 *Chin. Phys. B* **22** 076601

[5] Chai A H, Ran S Y, Zhang D, Jiang Y W, Yang G C and Zhang L X 2013 *Chin. Phys. B* **22** 098701

[6] Maginn E J and Elliott J R 2010 *Ind. Eng. Chem. Res.* **49** 3059

[7] Imran M, Hussain F, Rashid M and Ahmad S A 2012 *Chin. Phys. B* **21** 116201

[8] Mo Z Y, Zhang J L and Cai Q D 2002 *Int. J. Comput. Math.* **79** 165

[9] Rapaport D C 2004 *The Art of Molecular Dynamics Simulation* (New York: Cambridge University Press)

[10] Kumar S, Huang C, Zheng, G, Bohm E, Bhatelé A, Phillips J C, Yu H and Kalé L V 2008 *IBM J. Res. & Dev.* **52** 177

[11] Zhu X Q, Liu X, Jian X D, Meng X F and Feng J H 2012 *In Advances in Information Technology and Industry Applications* (Berlin: Springer-Verlag) p. 265

[12] Michael B W, Nguyen T D, Fuentes-Cabrera M, Fowlkes J D, Rack P D, Berger M and Bland A S 2012 *Proceedings of the International Conference on Computational Science*, June 4–6, 2012, Omaha, USA, p. 186

[13] Bland B, Wells J, Messer B, Hernandez O and R Jim 2012 https://cug.org/proceedings/attendee_program_cug2012/includes/files/pap138file2.pdf [2012-05-02]

[14] Alam S R, Vetter J S, Agarwal P K and Geist A 2006 *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, March 29–31 2006, New York, USA, p. 59

[15] Bhatelé A, Kalé L V and Kumar S 2009 *Proceedings of the* 23*rd International Conference on Supercomputing*, June 8–12, 2009, Yorktown Heights, USA, p. 110

[16] Deng Y F, Peierlsc R F and Riverad C 2000 *J. Comput. Phys.* **161** 250

[17] Kalé L V, Skeel R, Bhandarkar M, Brunner R, Gursoy A, Krawetz N, Phillips J, Shinozaki A, Varadarajan K and Schulten K 1999 *J. Comput. Phys.* **151** 283

[18] Brown W M, Wang P, Plimpton S J and Tharrington A N 2011 *Comput. Phys. Commun.* **182** 898

[19] Allen M P and Tildesley D J 1989 *Computer Simulation of Liquids* (Oxford: Clarendon Press)

[20] Verlet L 1967 *Phys. Rev.* **159** 98

[21] Quentrec B and Brot C 1973 *J. Comput. Phys.* **13** 430

[22] Hockney R W and Eastwood J W 1988 *Computer Simulation Using Particles* (Bristol: Taylor & Francis)

[23] Plimpton S 1995 *J. Comput. Phys.* **117** 1

[24] Chandy J A, Kim S, Ramkumar B, Parkes S and Banerjee P 1997 *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **16** 398

[25] Nour-Omid B, Raefsky A and Lyzenga G 1987 *Proceedings of the Symposium on Parallel Computations and Their Impact on Mechanics*, December 13–18, 1987, Boston, USA, p. 209

[26] Bially T 1969 *IEEE Trans. Inf. Theory* **15** 658

[27] Karypis G and Kumar V 1999 *Proceedings of the* 36*th Annual ACM/IEEE Design Automation Conference*, June 21–25, 1999, New Orleans, USA, p. 343

[28] Karypis G, Aggarwal R, Kumar V and Shekhar S 1997 *Proceedings of the* 34*th Annual Design Automation Conference*, June 9–13, 1997, Anaheim, USA, p. 526

[29] Kikuchi H, Karki B B and Saini S 2006 *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, June 26–29, 2006, Las Vegas, USA, p. 1083

[30] Hess B, Kutzner C, van der Spoel D and Lindahl E 2008 *J. Chem. Theory Comput.* **4** 435

[31] Larsson P, Hess B and Lindahl E 2011 *Wiley Interdisciplin. Rev. Comput. Mol. Sci.* **1** 93

[32] Hayashi R and Horiguchi S 2000 *Proceedings of the* 14*th International Parallel & Distributed Processing Symposium*, May 1–5, 2000, Cancun, Mexico, p. 85

[33] Zhakhovskii V, Nishihara K, Fukuda Y, Shimojo S, Akiyama T, Miyanaga S, Sone H, Kobayashi H, Ito E, Seo Y, Tamura M and Ueshima Y 2005 *Proceedings of the* 5*th International Symposium on Cluster Computing and the Grid*, May 9–12, 2005, Cardiff, UK, p. 848

[34] Fattebert J L, Richards D F and Glosli J N 2012 *Comput. Phys. Commun.* **183** 2608