Computer Physics Communications 235 (2019) 88–94

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Shift/collapse on neighbor list (SC-NBL): Fast evaluation of dynamic many-body potentials in molecular dynamics simulations

Manaschai Kunaseth [a,*], Supa Hannongbua [b], Aiichiro Nakano [c]

[a] National Nanotechnology Center (NANOTEC), National Science and Technology Development Agency (NSTDA), Pathum Thani, Thailand
[b] Department of Chemistry, Faculty of Science, Kasetsart University, Bangkok, Thailand
[c] Collaboratory for Advanced Computing and Simulations (CACS), University of Southern California, CA, USA

ARTICLE INFO

Article history:
Received 11 December 2017
Received in revised form 10 September 2018
Accepted 27 September 2018
Available online 10 October 2018

Keywords:
Dynamic many-body potentials
Shift–collapse algorithm
Performance optimization
Neighbor list
Parallel molecular dynamics simulation

ABSTRACT

Dynamic many-body potentials (DMBPs) significantly extend the applicability of molecular dynamics (MD) simulations to broad material properties and processes, including chemical reactions. Recently, we have designed a shift/collapse (SC) algorithm that accelerates DMBP-MD simulations over the conventional approach based on linked-list cell and neighbor list (NBL) methods. Here, we present an extension of the SC algorithm called SC-NBL, which outperforms the original SC algorithm and can be implemented seamlessly in conventional NBL-based DMBP-MD programs. Excellent performance is achieved by combining communication-lean SC and computation-lean NBL approaches. For small and large granularities, the SC-NBL approach achieves 1.33× and 3.29× speedup, respectively, over the original SC approach. In addition, SC-NBL performance surpasses the conventional NBL method for DMBP-MD at large granularity with running time speedup of 2.2 folds, which was the shortcoming of the original SC algorithm.

1. Introduction

Molecular dynamics (MD) has become a common investigation tool to study dynamics of atoms for broad applications in material science, chemistry, physics, and biology [1,2]. Recently, chemically reactive MD simulations using dynamic many-body potentials (DMBP) [3–11] have gained great attention. These DMBP-MD simulations allow dynamic breakage and formation of atomic bonds, thereby enabling the study of broader material properties and processes, including chemical reactions. However, the computational cost of DMBP-MD is significantly higher than those for non-bonded pairwise potentials or fixed-bond many-body potentials [12,13]. It is challenging, but with immense benefits, to develop efficient computation algorithms for DMBP-MD for encompassing larger spatio-temporal scale simulations for reactive systems than are currently possible [14,15].

Conventional DMBP computations employ linked-list cell and neighbor list (NBL) methods [1,2], as well as their hybrids [16]. Recently, we developed a highly scalable algorithm called shift/collapse (SC), which outperforms the conventional hybrid cell-NBL approach for DMBP-MD computation [17]. SC algorithm is a cell-based method that minimizes intra-node computation by eliminating redundant n-tuple evaluation (*i.e.*, $n = 2$ for pair, $n = 3$ for triplet, $n = 4$ for quadruplet, and so on), while reducing inter-node data communication for DMBP-MD simulations on parallel computers. Superior performance of SC algorithm in DMBP-MD, including theoretical proofs and performance benchmarks in large-scale parallel machines, was demonstrated extensively in our previous work [17]. Despite an established excellent parallel scalability performance, application of SC algorithm is rather limited due to the incompatibility of SC algorithm and the commonly used NBL method. NBL is a data structure storing information of nearest-neighbor atoms for each atom in the system. Using NBL can substantially improve performance of MD simulations by reducing number of atoms involved in the force computation [2]. This is especially beneficial for many-body force computation in DMBP-MD.

In this work, we have developed an SC algorithm that utilizes neighbor list (SC-NBL). SC-NBL algorithm for parallel DMBP-MD provides excellent scalability of SC algorithm, while tremendously reduces many-body force computation ($n \geq 3$) when compared to the original SC algorithm. This paper presents the precise description and correctness proof of the SC-NBL algorithm, including implementation detail. This paper is organized as follows. Section 2 provides the details of SC and SC-NBL algorithms. Section 3 presents performance benchmark and conclusions are drawn in Section 4.

* Corresponding author.
E-mail address: manaschai@nanotec.or.th (M. Kunaseth).

https://doi.org/10.1016/j.cpc.2018.09.021
0010-4655/

2. Methods

2.1. Range-limited n-tuple MD computation

MD simulation follows time evolution of an N-atom system $R = \{\mathbf{r}_1, \ldots, \mathbf{r}_N\}$ by numerically integrating Newton's equations of motion

$$m_i \frac{d^2\mathbf{r}_i}{dt} = \frac{\partial \Phi(\mathbf{R})}{\partial \mathbf{r}_i}, \tag{1}$$

where $\Phi(\mathbf{R})$ denotes many-body interatomic potential-energy function. Here, $\Phi(\mathbf{R})$ is defined as a sum of n-body potential-energy function $\sum_{n=2}\Phi_n$. Each Φ_n term can be evaluated as a function of all atomic n-tuples in the system:

$$\frac{\partial \Phi_n}{\partial \mathbf{r}_i} = \sum_{j_1=1}^{N} \cdots \sum_{\substack{j_n=1 \\ j_1<j_n}}^{N} \frac{\partial \Phi(\mathbf{R})}{\partial \mathrm{G}(\mathbf{r}_{j_1}, \ldots, \mathbf{r}_{j_n})} \cdot \frac{\partial \mathrm{G}(\mathbf{r}_{j_1}, \ldots, \mathbf{r}_{j_n})}{\partial \mathbf{r}_i}, \tag{2}$$

where $\mathrm{G}(\mathbf{r}_{j_1}, \ldots, \mathbf{r}_{j_n})$ denoted n-tuple functions (*e.g.* interatomic distance for $n = 2$, bond angle for $n = 3$) and $j_i \neq j_k$ for $\forall\, i$ and $\forall k \in \{1,\ldots, n\}$. Based on Eq. (2), n-tuple computation can be accomplished by finding all combination of atomic n-tuples in $\mathbf{R}$. For range-limited n-tuple DMBP-MD, only atomic n-tuples within the cutoff radius ($r_{\mathrm{cut}-n}$) are included for force computation. Here, range-limited n-tuple is defined as

$$\left(\mathbf{r}_{j_1}, \mathbf{r}_{j_2}, \ldots, \mathbf{r}_{j_n}\right) | \forall k \in \{1, \ldots, n-1\} : r_{j_k j_{k+1}} \leq r_{\mathrm{cut}-n}, \tag{3}$$

where $r_{j_k j_{k+1}}$ is an interatomic distance between atoms $\mathbf{r}_k$ and $\mathbf{r}_{k+1}$. Since we focused on range-limited n-tuple, we denote $\chi = (j_1, j_2, \ldots, j_n)$ for $j_k \in \{1, \ldots, N\}$ as range-limited atomic n-tuple hereafter. Therefore, the main computation in range-limited n-tuple MD is to find all χ for a given set of atom positions $\mathbf{R}$.

Typically, finding all χ can be efficiently achieved by using linked-list cell method. Cell method divides simulation box into small cells of equal size with side length equal to or slightly larger than $r_{\mathrm{cut}-n}$. The number of cells in α direction, L_α, is calculated from $L_\alpha = \left\lfloor \frac{l_\alpha}{r_{\mathrm{cut}-n}} \right\rfloor$ for simulation box of side length l_α in $\alpha = x, y$, and z directions, respectively. After that, atoms are assigned to each linked-list cell based on their spatial coordinates. Specifically, an arbitrary atom $\mathbf{r}$ is assigned to cell $\mathbf{c} = (c_\mathrm{x}, c_\mathrm{y}, c_\mathrm{z})$ for $c_\alpha = \left\lfloor \frac{\mathbf{r}^\alpha}{r_{\mathrm{cut}-n}} \right\rfloor$, where $(c_\mathrm{x}, c_\mathrm{y}, c_\mathrm{z})$ denotes the c_α-th cell in $\alpha = x, y$, and z directions, respectively.

The computation steps of cell method can be described as follows. For an arbitrary atom in a particular cell $\mathbf{c}$, all of its n-tuples are found by traversing up to the $(n-1)$ th nearest neighbor cells of $\mathbf{c}$ (*e.g.* first nearest-neighbor cells for 2-body (pair) interaction, first and second nearest-neighbor cells for 3-body (triplet) interaction, and so on) [18]. This algorithm is described as a full-shell (FS) method based on its traversing pattern [2], which includes all nearest neighbor cells. This ensures that the n-tuples with interatomic distance less than $r_{\mathrm{cut}-n}$ are included in the force computation.

2.2. Computation pattern algebraic framework

The computation steps of cell method described in Section 2.1 are traditionally sufficient as it is straightforward to understand, implement, and verify. The correctness of cell method can be described using owner-compute rule, where each cell in the system is responsible to generate all n-tuples of atoms resided within it. However, the proof of correction is more sophisticated for methods that do not follow owner-compute rule such as zonal method [19], neutral territory [20,21], and SC algorithm [17]. In the previous work, we proposed a computation pattern algebraic framework (CPAF) as a generalized description of cell-based method MD [17]. CPAF describes cell-based MD computation by "convolution" of computation pattern onto every cell in the system to generate all χ. For the sake of clarity, we briefly described the CPAF in this section.

Let computation pattern $\mathbf{P}$ be a set of cell interactions $\mathbf{p}$, $\mathbf{P} = \{\mathbf{p}\}$. Here, cell interaction $\mathbf{p}$ is defined as a tuple of length n for cell offset vectors, $\mathbf{p} = (\vec{\alpha}_1, \ldots, \vec{\alpha}_n)$, where cell offset vector $\vec{\alpha}_j$ is a 3-integer vector $(\alpha_{j,\mathrm{x}}, \alpha_{j,\mathrm{y}}, \alpha_{j,\mathrm{z}})$. Convolution of $\mathbf{P}$ onto an arbitrary cell $\mathbf{c}$ is defined as follows. For each $\mathbf{p} \in \mathbf{P}$, atomic n-tuples $(j_1, \ldots, j_n)$ are generated from all combination of n atoms j_1 to j_n selected from cells $\mathbf{c} + \vec{\alpha}_1, \mathbf{c} + \vec{\alpha}_2, \ldots, \mathbf{c} + \vec{\alpha}_n$, respectively (note that $\mathbf{c} + \vec{\alpha}_j = (c_\mathrm{x} + \alpha_{j,\mathrm{x}}, c_\mathrm{y} + \alpha_{j,\mathrm{y}}, c_\mathrm{z} + \alpha_{j,\mathrm{z}})$). To complete all force computation in MD, $\mathbf{P}$ is generally convoluted onto all cells in the system to generate all χ.

In term of correctness, a given computation pattern $\mathbf{P}$ is correct if and only if all χ is generated in the convolution step. If there are multiple generations of some χ throughout the convolution steps, the filter algorithm is required in order to assure that one and only one χ is counted for force computation. Otherwise, there will be force and energy double counting. One of the most commonly used computation patterns is full shell ($\mathbf{P}_{\mathrm{FS}}$) shown in Fig. 1(a), while Fig. 1(b) shows the system after convolution using $\mathbf{P}_{\mathrm{FS}}$ onto all cells (blue squares in Fig. 1(b)).

2.3. Original SC algorithm

Shift/collapse (SC) algorithm is an efficient computation algorithm based on cell method for general range-limited n-tuple MD computation. SC algorithm employs SC computation pattern ($\mathbf{P}_{\mathrm{SC}}$), which is an optimized computation pattern that eliminates redundant computation and minimize the stencil region (see Fig. 1(c)). SC algorithm to create $\mathbf{P}_{\mathrm{SC}}$ for arbitrary n contains three main phases. First, $\mathbf{P}_{\mathrm{FS}}$ is created by performing $(n-1)$-fold nested loops to iterate $(n-1)$-nearest neighbor cells. Second, $\mathbf{P}_{\mathrm{FS}}$ is transformed using octant-compression shift algorithm, which shifts all cell interactions $\mathbf{p} = (\vec{\alpha}_1, \ldots, \vec{\alpha}_n) \in \mathbf{P}_{\mathrm{FS}}$ toward the upper corner such that all cell offset vector $\vec{\alpha}_j > 0$. This step compacts the computation footprint to only single octant (see Figure S1 in supplementary data). In the last phase, reflective collapse algorithm is performed to find and remove redundant cell interactions that generate the same χ. The detailed description of SC algorithm can be found in [17]. As a result, SC algorithm reduces computation by eliminating redundant tuple evaluation. The reduction of stencil region (*i.e.*, striped region in Fig. 1(d) compared to Fig. 1(b)) eventually minimizes communication in parallel SC-MD and improves data-cache utilization during the force computation. The detail on data communication in parallel SC-MD is described in the next section.

2.4. Parallel implementation of SC-MD

Parallel SC-MD employs domain decomposition scheme, where atoms in the simulated system are distributed among compute nodes based on their spatial coordinates. The computation steps in each parallel SC-MD loop are shown in Fig. 2(a). Three steps are involved in the inter-node communication (shown in orange boxes in Fig. 2(a)), which are: (1) import atom data before force computation; (2) return forces after force computation; and (3) migrate atom data that moved out of the domain after updating coordinates. The communication steps can be described as follows.

Before the force computation step, atoms near domain boundaries (within import radius $r_{\mathrm{import}} = \max_n[(n-1)\, r_{\mathrm{cut}-n}]$) are imported from the nearest compute nodes in x, y, z direction, respectively. The imported atoms from the prior import direction are also forwarded to the nodes in the subsequent directions, thereby reducing number of communications by omitting direct communication with the corner nodes. Therefore, only three import communications with three neighbor nodes are necessary

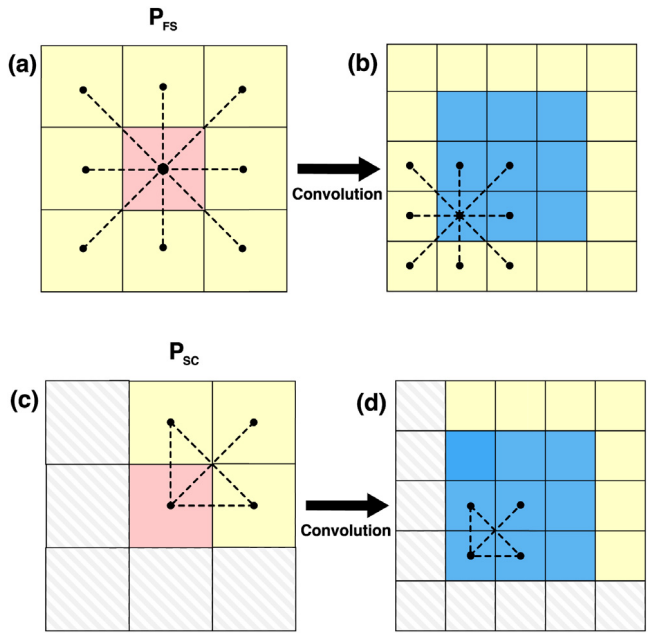

Fig. 1. (a) Full shell computation pattern ($\mathbf{P}_{\mathrm{FS}}$) for $n = 2$ (b) cells in the system after $\mathbf{P}_{\mathrm{FS}}$ convolution, where blue squares denoted domain cells and yellow squares denoted extended cells due to stencil of computation pattern. Extended cells are needed to be imported from nearest neighbor domain in parallel simulation. (c) Shift/collapse (SC) computation pattern ($\mathbf{P}_{\mathrm{SC}}$) for $n = 2$ and (d) system after convolution of $\mathbf{P}_{\mathrm{SC}}$. Note the stencil region and extended cells are reduced in $\mathbf{P}_{\mathrm{SC}}$ when compared to $\mathbf{P}_{\mathrm{FS}}$ (showed in striped squares). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

in this scheme before force computation (steps 1–3 in Fig. 2(b)). After the force computation is completed, reaction force acting on the imported atoms is returned and forwarded to the owner nodes in a reverse direction (z, y, x directions, respectively) (steps 6–8 in Fig. 2(b)). Finally, data of atoms that moved beyond the domain boundary are migrated to the corresponding neighbor nodes (step 10 in Fig. 2(a)).

2.5. Shift/collapse algorithm on neighbor list (SC-NBL)

The n-tuple MD computation using SC-NBL algorithm can be summarized in two steps: (1) NBL creation during SC algorithm for $n = 2$ and (2) n-tuple generation for $n > 2$ using NBL. Also, the convolution in SC-NBL algorithm is changed slightly when compared to original SC algorithm. The detail of SC-NBL algorithm is explained in the following subsections.

2.5.1. Neighbor list creation

In SC-NBL, NBL of atom i (NBL(i)) is defined as a set of all atom indices j such that $r_{ij} < r_{\text{cut-NBL}}$. Here $r_{\text{cut-NBL}}$ is the largest cutoff among n-tuple computation $r_{\text{cut-NBL}} = \max_{n\geq 2}(r_{\text{cut}-n})$. To save the NBL creation cost, NBL can be created simultaneously with the force computation for $n = 2$ in the convolution step. This approach should be common in most cases because $r_{\text{cut-2}}$ is usually larger than $r_{\text{cut}-n}$ for $n > 2$ in general. In addition, since the force computation for $n = 2$ is completed at the same time as NBL creation, NBL is only needed to store atom data for force computation of $n > 2$. Therefore, we consider $r_{\text{cut-NBL}} = \max_{n>2}(r_{\text{cut}-n})$ hereafter.

Next, we describe the convolution step for SC-NBL. Although $\mathbf{P}_{\mathrm{SC}}$ is still used in SC-NBL algorithm, the convolution step is slightly different than the original SC. Here, the $\mathbf{P}_{\mathrm{SC}}$ convolution is performed on both domain cells and imported cells rather than only the domain cells in original SC (see Fig. 3(a)). Particularly, $\mathbf{P}_{\mathrm{SC}}$ convolution on domain cells includes force computation for $n = 2$ and update the NBL, while $\mathbf{P}_{\mathrm{SC}}$ convolution on imported cells will only update the NBL. The $\mathbf{P}_{\mathrm{SC}}$ convolution on imported cells is required so that the information of NBL for atoms in the imported cell is completed. During SC-NBL convolution on the imported cells, cell interaction refers to the cell outside scope of the system (*i.e.* combined domain and imported cells) will be ignored. This creates "partial NBL" for atoms in the imported cells and near the boundary, which is required to generate all tuples of every atom (see Fig. 3(b)). Upon the completion of this procedure, the force contribution for $n = 2$ and NBL within the radius of $r_{\text{cut-NBL}}$ for every atom are obtained.

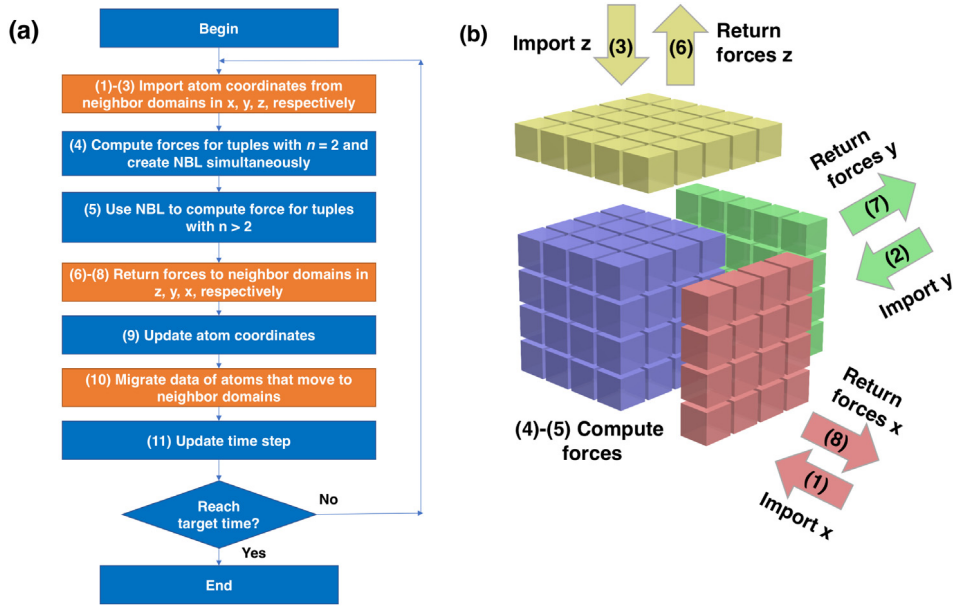

Fig. 2. (a) Computation steps in each MD loop of parallel SC-MD, where blue boxes denoted computation steps and orange boxes denoted communication steps. (b) Communication routine during atom import and force return. Blue cubes denoted domain cells, while red, green, and yellow cubes denoted imported cells in x, y, and z directions, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

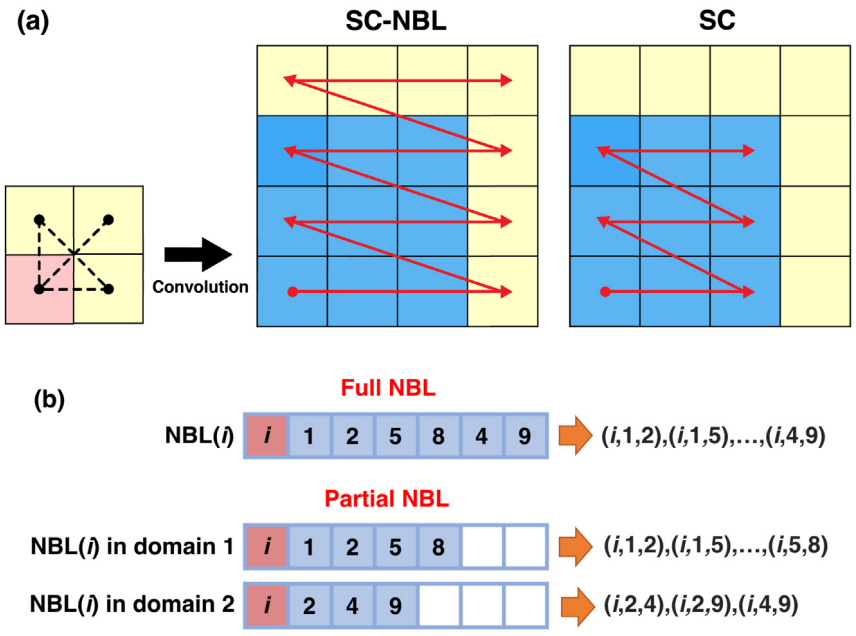

Fig. 3. (a) Comparison of $\mathbf{P}_{SC}$ convolution in SC-NBL and SC algorithms. Convolution in SC-NBL includes domain and imported cells (blue and yellow squares), while convolution of SC includes only domain cells (blue squares). Note that SC-NBL convolution at the imported cell performs only NBL update (no force computation). (b) Illustration of full NBL used in conventional NBL method and partial NBL used in SC-NBL. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.5.2. N-Tuple computation using NBL

After the NBL is created from the first step, χ for $n > 2$ are generated as follows. Let $\chi(i)$ denote χ beginning with atom i, which can be generated by consecutively traverse through the chain of atoms in NBL starting from NBL(i):

$$\chi\ (i) = \left\{(i, j_2, \ldots, j_n)\middle|_{\forall j_k \in \mathrm{NBL}(j_{k-1})}^{\forall j_k \in \mathrm{NBL}(i)} \quad \begin{matrix} \text{for} k = 2 \\ \text{for} k > 2 \end{matrix}\right\}, \tag{4}$$

This n-tuple generation scheme may include duplicated χ from two scenarios: (1) inverse duplication $(i, \ldots, j_n)$ and $(j_n, \ldots, i)$; and (2) parallel duplication, in which duplicated n-tuple that may exist in the multiple computing domains in parallel implementation. To ensure that all generated n-tuples are unique among all nodes under this scheme, a filtering algorithm needs to be applied for all χ generated from NBL. Avoiding inverse duplication of $\chi(i)$ is straightforward by using filtering condition $i < j_n$. In case of parallel duplication, we have developed a Unique Tuple Test (UTT) to filter the non-unique tuples in parallel domains. The UTT is described as follows.

Let *import flag* (IPF), $\theta(i)$, be an import status of atom i. Here, $\theta(i) = (\theta_x(i), \theta_y(i), \theta_z(i))$ is a three-Boolean vector which specifies whether atom i is imported or forwarded in particular direction. Specifically, $\theta_m(i) \in$ {true, false}, where $m \in$ {x, y, z} denoted the directionality of imported domain. The value of $\theta_m(i)$ is assigned using the following condition:

$$\theta_m\ (i) = \begin{cases} \text{true} & \text{atom i is imported from node in m direction} \\ \text{true} & \text{atom i is forwarded from m direction in prior communication step} \\ \text{false} & \text{none above.} \end{cases} \tag{5}$$

Example of different scenarios for various IPFs is shown in Table 1. Then, the parallel duplication can be filtered using the UTT, which is defined as follows.

Unique Tuple Test Theorem. *Let us define Unique Tuple Test (UTT) as*

$$\begin{aligned} UTT\ (\chi = (j_1, \ldots, j_n)) &= \bigvee_{m=\{x,y,z\}} [\theta_m\ (j_1) \wedge \cdots \wedge \theta_m\ (j_n)] \\ &= [\theta_x\ (j_1) \wedge \cdots \wedge \theta_x\ (j_n)] \vee \left[\theta_y\ (j_1) \wedge \cdots \wedge \theta_y\ (j_n)\right] \\ &\vee [\theta_z\ (j_1) \wedge \cdots \wedge \theta_z\ (j_n)]\,, \end{aligned}$$

where $\chi = (j_1, \ldots j_n)$ be an arbitrary range-limited n-tuple, $\vee$ and $\wedge$ denoted logical OR and AND operators, respectively. For every χ in parallel SC-MD domains, there is a unique (one and only one) χ_0 among all nodes that satisfy UTT(χ_0) = false.

The proof of correctness for Unique Tuple Test Theorem is as follows. Assume that $\chi_p = (j_1, \ldots j_n)$ is a parallel duplicated n-tuple (UTT(χ_p) = true). There exists χ_0 in the neighbor domain in m^* direction that χ_p is imported or forwarded from (i.e. all IPFs from m^* direction, $\theta_{m*}(j_1)$ to $\theta_{m*}(j_n)$, must be true). Since χ_p is imported, χ_0 must be original. This proves that UTT(χ_0) = false when χ_0 is original. Therefore, UTT(χ) = true can be used as an exclusion criteria to avoid force double computing of parallel duplicated n-tuple.

3. Results and discussion

3.1. Result validation

We have validated SC-NBL implementation using Vashishta's potential for silica glass [5]. The validation was performed by comparing the result of SC-NBL with the original SC using thermalization simulation of silica glass at 300 K in microcanonical (NVE) ensemble. The silica glass system was prepared using melt-quench techniques as follows. First, initial configuration of MD simulation was built from $8 \times 8 \times 8$ unit cells of β-crystobalite SiO_2 (total of 12,288 atoms). To validate the parallel implementation, we created three different initial configurations using different number of domain (*i.e.* number of MPI tasks to run), which are: (1) single domain (P1) for 12,288 atoms; (2) $2 \times 2 \times 2 = 8$ domains (P8) for 1536 atoms per domain on average; and (3) $4 \times 4 \times 4 = 64$ domains (P64) for 192 atoms per domain on average. Note that

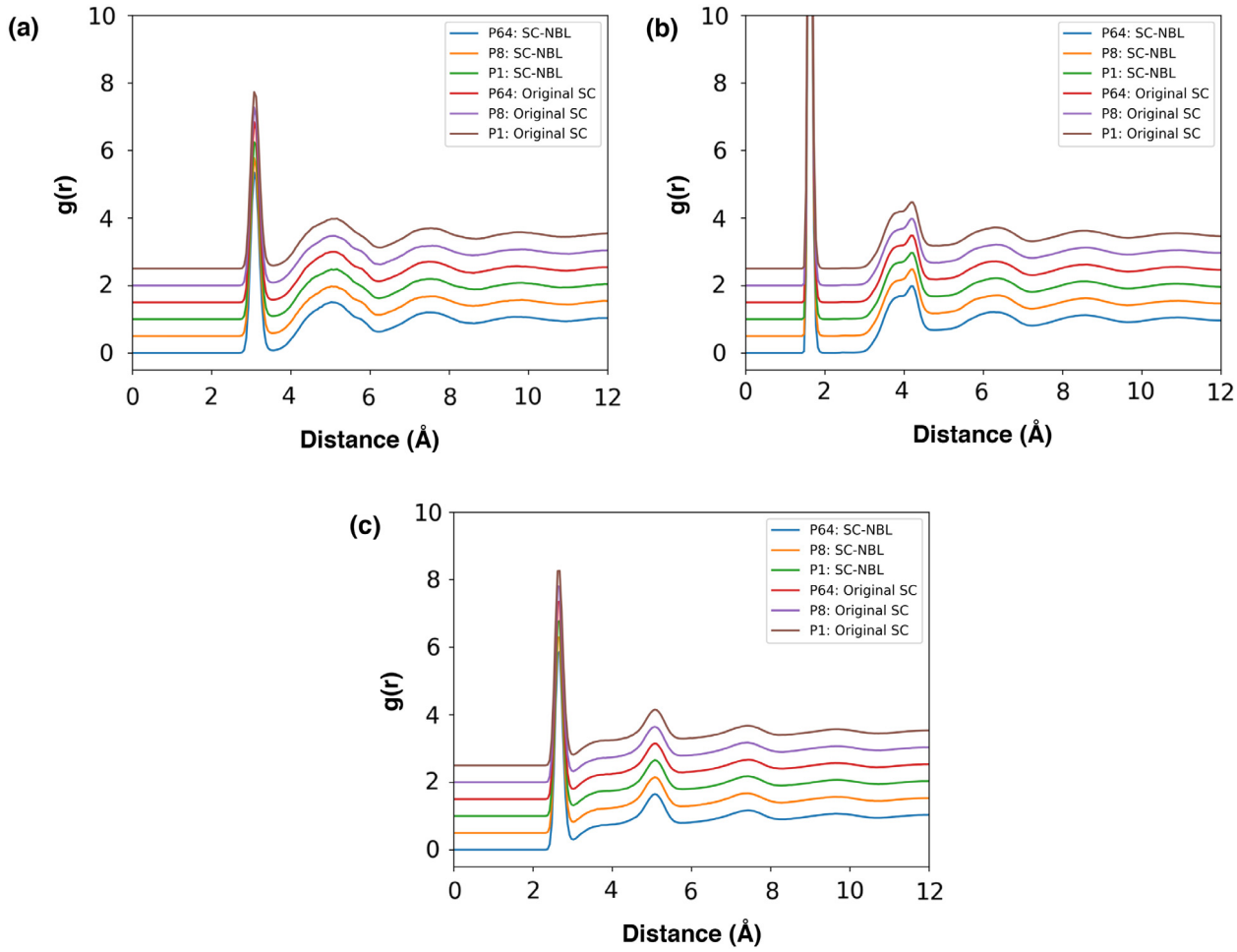

Fig. 4. Pair distribution function, g(r), of silica glass systems using different parallelizations (P1, P8, P64) based on SC-NBL and original SC algorithms (a) Si–Si (b) Si–O (c) O–O. Note that g(r) plots were shifted for comparison clarity.

different random number seeds were used in those three systems to ensure the non-bias comparison. Next, velocity scaling was used to increase temperature of the initial configuration to 2,500 K, which is well-above T_{melt} of SiO_2 [5]. The temperature scaling was carried out based on Boltzmann distribution to 2,500 K every 23000 MD steps for 50,000 MD steps. After that, the systems were thermalized without velocity scaling for 50,000 MD steps. Then, the systems were slowly quenched by scaling velocity to 23000 K, 1,600 K, 300 K, using 50,000 MD steps with scaling period of 10,000 MD steps for each scaling temperature, respectively. Finally, the systems continued to thermalize without velocity scaling at 300 K for 50,000 MD steps before the final 50,000 MD steps were run for P1, P8, and P64 using both SC-NBL and original SC for collecting validation result.

In the first verification, the number of pairs and triplets that were evaluated for force in each MD step for P1, P8, and P64 systems using original SC and SC-NBL algorithms were compared in Table 2. Here, Table 2 shows that there are no significant differences (*i.e.* within the standard deviation) in terms of pair and triplet counts between SC and SC-NBL in all P1, P8, and P64 systems. In addition, considerable number of filtered triplets in all SC-NBL runs (last row of Table 2) indicated that the filtering algorithms used in SC-NBL essentially avoid double counting of triplets.

Next, to validate SC-NBL results, we compared pair distribution function, g(r) [1,2], of SC-NBL against that of original SC. Fig. 4 shows g(r) comparison for Si–Si (Fig. 4(a)), Si–O (Fig. 4(b)), and O–O (Fig. 4(c)), in which all of the results are almost identical for P1, P8, and P64 systems in both SC-NBL and original SC. Additionally, atomic positions, atomic forces, and total energies of SC-NBL and SC were validated. The results show that maximum atomic-force difference, maximum atomic-position difference, and total energies between SC-NBL and SC runs in each MD step are less than 0.1 pN, 10^{-4} Å, and 10^{-7} eV/atom, respectively (see details in Section S2 of supplementary data). This negligible degree of divergence for atomic forces, positions, and total energies is likely to be the numerical origin, and thus acceptable for validating SC-NBL implementation.

3.2. Performance benchmark

In this section, we compared computing performance of SC-NBL and the original SC algorithm. The benchmark was performed using 96 MPI tasks on 8 compute nodes of dual hexcore Intel Xeon 2.6 GHz (12 compute cores per node, total 96 compute cores) with 64 GB memory. Running time per MD steps was collected from thermalization simulation of amorphous silica glass at 300 K using NVE ensemble. Here, the numbers of atoms in the benchmark systems were varied from 2304 atoms (24 atoms per compute core) to 2,304,000 atoms (24,000 atoms per compute core). Fig. 5 shows average running time per MD steps for systems with different average number of atoms per compute core. Benchmark results show significant performance improvement of SC-NBL over the original SC algorithm in all of the system sizes, with 1.33× and 3.29× running time speedups on system with 24 and 24,000 atoms per compute core, respectively (see Fig. 5).

Table 1
Different scenarios with various corresponding IPFs.

Status of atom i	IPF $(\theta_x(i), \theta_y(i), \theta_z(i))$
• Owned by the current compute node (resident atom)	(false, false, false)
• Import from node in x direction • Not a forwarded atom	(true, false, false)
• Import from node in y direction • Not a forwarded atom	(false, true, false)
• Import from node in y direction • Forwarded from x direction (from prior communication)	(true, true, false)
• Import from node in z direction • Forwarded from x direction (from prior communication)	(true, false, true)
• Import from node in z direction • Forwarded from x and y directions (from first- and second-prior communications, respectively)	(true, true, true)

Table 2
Average number of pairs, triplets, and filtered duplicated triplets for SC and NBL-SC algorithms in P1, P8, and P64 systems, respectively. Note that numerals after ± denoted standard deviation.

Algorithm	P1 (1 MPI task)		P8 (8 MPI tasks)		P64 (64 MPI tasks)	
	SC	SC-NBL	SC	SC-NBL	SC	SC-NBL
Number of pairs	281,889.2 ± 105.6	281,876.5 ± 105.8	281,893.6 ± 106.0	281,902.5 ± 105.2	281,971.6 ± 107.1	281,949.0 ± 104.5
Number of triplets	31,186.5 ± 23.9	31,191 ± 25.2	31,235.6 ± 24.3	31,237.3 ± 24.2	31,141.9 ± 27.8	31,138.1 ± 26.7
Filtered Triplets	N/A	54,314.3 ± 331.2	N/A	28,928.5 ± 210.0	N/A	182.14.7 ± 163.8

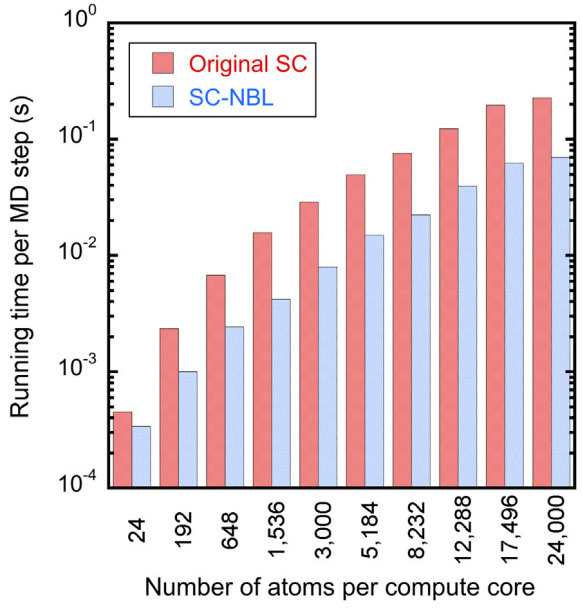

Fig. 5. Computation performance based on running time per MD step of SC-NBL compared to the original SC algorithm on 96 compute cores.

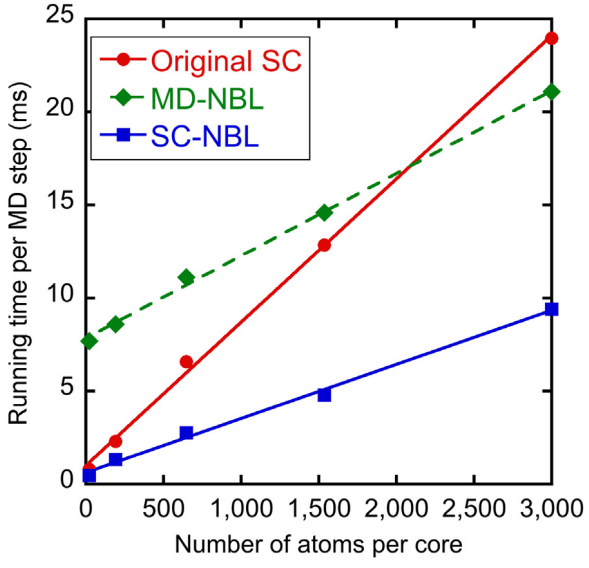

Fig. 6. Comparison of running time per MD step as a function of granularity (number of atoms per core) on 768 compute cores for the original SC, MD-NBL, and SC-NBL codes.

To compare performance of SC-NBL against the existing algorithms, we performed a performance benchmark on three codes: (1) original SC; (2) MD-NBL; and (3) SC-NBL codes. MD-NBL is a production code presented in Ref. [22], which employs cell method for 2-body force computation and NBL for 3-body force computation. MD-NBL code utilizes conventional full-shell computation and communication schemes in 2-body computation, which is common for most of DMBP-MD programs. The benchmark was carried out on 64 dual-hexcore Xeon 2.6 GHz nodes (768 compute cores total) by varying granularities (*i.e.*, numbers of atoms per core) from 24 to 3000. Fig. 6 shows benchmark results of running time per MD step of for three codes as a function of granularity. At the smallest granularity (*i.e.*, 24 atoms per core), original SC and SC-NBL considerably outperform MD-NBL such that the running time per MD step of MD-NBL is 9.7 and 16.2 folds of the original SC and SC-NBL, respectively. As the number of atoms per core increases, running time of original SC is gradually increased with respect to MD-NBL. Eventually, performance of original SC becomes slower than MD-NBL at 3000 atoms per compute core. This is due to the reduced 3-body force computation cost using NBL in MD-NBL, which was previously observed in [17]. On the other hand, the gap between running time per MD step of MD-NBL and SC-NBL remains large at all granularities. The running time ratios of MD-NBL over SC-NBL are ranging from 16.2 to 2.2 folds for granularities between 24 and 3000 atoms per core. This signifies that SC-NBL encompasses the performance of the original SC at small granularities and that of MD-NBL on larger granularities.

To evaluate sustained performance of SC-NBL, we performed large-scale benchmarks of 31,104 atom-system (24 atoms per MPI task) and 248,832 atom-system (192 atoms per MPI tasks) on

$9 \times 12 \times 12$ = 1,296 MPI tasks using 1,296 compute cores on 108 dual-hexcore Intel Xeon 2.6 GHz compute nodes. The results showed average running time per MD step of 0.65 and 1.72 ms over 500,000 MD steps for 24 atoms/core and 196 atoms/core systems, respectively. This is equivalent to 132.9 ns/day and 50.3 ns/day based on 1 fs MD time step for 31,104 and 248,832 atom-systems, respectively. Additionally, we investigated performance of SC and SC-NBL algorithms based on different phases of materials. The results show that performance of SC and SC-NBL for molten and crystalline SiO_2 phases is very similar, see Section S3 in supplementary data for details.

4. Conclusion

In this work, we have developed a SC-NBL algorithm for reactive many-body MD algorithm. Significant performance improvement of SC-NBL is achieved through the combination of communication-lean SC algorithm and a computation-lean NBL-based algorithm. Our benchmark indicated that SC-NBL-algorithm speeds up computation time over the original SC algorithm by 1.33× up to 3.29× for MD simulation with 24 and 24,000 atoms per compute core, respectively. SC-NBL surpasses the conventional NBL method for DMBP-MD at large granularity with running time speedup of 2.2 folds at 3000 atoms per core, which was the shortcoming of the original SC algorithm. In addition, the performance benchmark also showed a sustained remarkable performance of SC-NBL at very small granularity on 1,296 compute-cores parallel MD simulation, thereby achieving computation performance of 0.65 and 1.72 ms per MD step. This highlights a significant performance improvement on both strong- and weak-scaling regimes, which is remarkably challenging for large-scale parallel DMBP-MD simulations. SC-NBL algorithm significantly improves time-to-solution of reactive atomistic simulations, while allowing large-size and long-time scale DMBP-MD studies to be performed using modest computation resources.

Acknowledgments

This work was supported by the Thailand Research Fund Grant for New Researcher under the Contract No. TRG5780238. Computational resources were generously provided by NANOTEC, Thailand's National e-Science Infrastructure Consortium, NSTDA Computing Infrastructure, and Center for High-Performance Computing at University of Southern California.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cpc.2018.09.021.

References

[1] M.P. Allen, D.J. Tildesley, Computer Simulation Liquid, 1987.
[2] D.C. Rapaport, The Art of Molecular Dynamics Simulation, Second ed., Cambridge University Press, Cambridge, UK, 2004.
[3] M.S. Daw, M.I. Baskes, Phys. Rev. B 29 (1984) 6443–6453.
[4] J. Tersoff, Phys. Rev. B 37 (1988) 6991–7000.
[5] P. Vashishta, R.K. Kalia, J.P. Rino, I. Ebbsjo, Phys. Rev. B 41 (1990) 12197–12209.
[6] A.C.T. van Duin, S. Dasgupta, F. Lorant, W.A. Goddard, J. Phys. Chem. A 105 (2001) 9396–9409.
[7] K. Nomura, R.K. Kalia, A. Nakano, P. Vashishta, Comput. Phys. Commun. 178 (2008) 73–87.
[8] S.B. Sinnott, D.W. Brenner, MRS Bull. 37 (2012) 469–473.
[9] S.J. Plimpton, A.P. Thompson, MRS Bull. 37 (2012) 513–521.
[10] T.P. Senftle, S. Hong, M.M. Islam, S.B. Kylasa, Y. Zheng, Y.K. Shin, C. Junkermeier, R. Engel-Herbert, M.J. Janik, H.M. Aktulga, T. Verstraelen, A. Grama, A.C.T. van Duin, Npj Comput. Mater. 2 (2016) 15011.
[11] S. Naserifar, D.J. Brooks, W.A. Goddard, V. Cvicek, J. Chem. Phys. 146 (2017) 124117.
[12] A.K. Rappe, C.J. Casewit, K.S. Colwell, W.A. Goddard, W.M. Skiff, J. Am. Chem. Soc. 114 (1992) 10024–10035.
[13] M. Levitt, Angew. Chem., Int. Ed. 53 (2014) 10006–10018.
[14] A. Shekhar, K. Nomura, R.K. Kalia, A. Nakano, P. Vashishta, Phys. Rev. Lett. 111 (2013) 184503.
[15] K. Nomura, R.K. Kalia, Y. Li, A. Nakano, P. Rajak, C. Sheng, K. Shimamura, F. Shimojo, P. Vashishta, Sci. Rep. 6 (2016) 24109.
[16] A. Nakano, P. Vashishta, R.K. Kalia, Comput. Phys. Commun. 77 (1993) 303–312.
[17] M. Kunaseth, R.K. Kalia, A. Nakano, K. Nomura, P. Vashishta, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, ACM, Denver, Colorado, 2013, pp. 1–12.
[18] M. Kunaseth, R.K. Kalia, A. Nakano, P. Vashishta, International Conference on Scientific Computing, 2010.
[19] K.J. Bowers, R.O. Dror, D.E. Shaw, J. Comput. Phys. 221 (2007) 303–329.
[20] K.J. Bowers, R.O. Dror, D.E. Shaw, J. Chem. Phys. 124 (2006) 184109.
[21] D.E. Shaw, A. fast, J. Comput. Chem. 26 (2005) 1318–1328.
[22] A. Nakano, R.K. Kalia, P. Vashishta, T.J. Campbell, S. Ogata, F. Shimojo, S. Saini, Supercomputing, ACM/IEEE 2001 Conference, 2001, pp. 10–10.