A Scalable Parallel Algorithm for Dynamic Range-Limited *n*-Tuple Computation in Many-Body Molecular Dynamics Simulation

Manaschai Kunaseth[1,2], Rajiv K. Kalia[1], Aiichiro Nakano[1], Ken-ichi Nomura[3,1], Priya Vashishta[1]

[1]Collaboratory for Advanced Computing and Simulations
Department of Computer Science, Department of Physics & Astronomy, Department of Material Science,
University of Southern California, Los Angeles, CA 90089-0242, USA

[2]National Nanotechnology Center (NANOTEC)
National Science and Technology Development Agency,
Thailand Science Park, Klong Luang, Pathumthani 12120, Thailand

[3]Center for High-Performance Computing and Communications
University of Southern California, Los Angeles, CA 90089-0706, USA

{kunaseth, rkalia, anakano, knomura, priyav}@usc.edu

ABSTRACT

Recent advancements in reactive molecular dynamics (MD) simulations based on many-body interatomic potentials necessitate efficient dynamic *n*-tuple computation, where a set of atomic *n*-tuples within a given spatial range is constructed at every time step. Here, we develop a computation-pattern algebraic framework to mathematically formulate general *n*-tuple computation. Based on translation/reflection-invariant properties of computation patterns within this framework, we design a shift-collapse (SC) algorithm for cell-based parallel MD. Theoretical analysis quantifies the compact *n*-tuple search space and small communication cost of SC-MD for arbitrary *n*, which are reduced to those in best pair-computation approaches (*e.g.* eighth-shell method) for $n = 2$. Benchmark tests show that SC-MD outperforms our production MD code at the finest grain, with 9.7- and 5.1-fold speedups on Intel-Xeon and BlueGene/Q clusters. SC-MD also exhibits excellent strong scalability.

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Physical Sciences and Engineering – *physics, chemistry.*

General Terms

Algorithms, Performance, Theory.

Keywords

Dynamic range-limited *n*-tuple computation, Molecular dynamics, Parallel computing

SC '13, November 17 - 21 2013, Denver, USA

http://dx.doi.org/10.1145/2503210.2503235

1. INTRODUCTION

Molecular dynamics (MD) is a simulation method to study the dynamics of particles (*e.g.* atoms). It has broad applications in diverse fields such as physics, chemistry, biology, and materials science. MD simulation using a differentiable interatomic potential-energy function Φ was started by Rahman in 1964 [1] using a pair-wise potential energy, in which Φ is a sum of atomic-pair energies. Since then, scientists started performing many-body MD simulations that use *n*-tuple ($n \geq 3$) energy functions for accurate description of a wider range of materials. In one type of *n*-tuple computation (*i.e.* static *n*-tuple computation) used typically in biomolecular simulations [2], the list of atomic *n*-tuples is fixed throughout the simulation. In another (*i.e.* dynamic *n*-tuple computation), *n*-tuple lists within given interaction ranges are constructed at every simulation time step [3, 4]. Recent advancements in chemically reactive MD simulations [5] have renewed interests in efficient implementation of dynamic *n*-tuple computation [6]. Reactive MD describes the formation and breakage of chemical bonds based on a reactive bond-order concept [5]. In the ReaxFF approach, for example, *n* is 4 explicitly, and force computation involves up to $n = 6$ due to chain-rule differentiations through bond-order terms [7-9].

Scalable implementation of MD on massively parallel computers has been one of the major driving forces of supercomputing technologies [10-16]. Earlier parallel implementations of MD were based on spatial decomposition, in which the simulated physical volume is subdivided into spatially localized sub-volumes that are assigned to different processors [17]. For long-range pair ($n = 2$) computation, octree-based $O(N)$ algorithms (N is the number of atoms) [18] have highly scalable parallel implementations [19, 20]. For short-ranged (or range-limited) pair computation, Plimpton relaxed the conventional "owner-compute" rule (*i.e.*, computation is performed by a processor that has data) to design a force-decomposition algorithm to increase the concurrency [21]. Since then, various hybrid spatial-force decomposition algorithms have been designed [22]. On distributed-memory parallel computers, atomic data needed for range-limited pair computations are copied from neighbor processors. The most primitive scheme for these atom-caching

operations is full shell (FS), in which data from 26 (face-, edge-, and corner-sharing) neighbor sub-volumes are imported from other processors. In the half-shell (HS) scheme, Newton's third law is utilized to halve the number of imported sub-volumes to 13 [17]. Relaxation of the owner-compute rule further reduces this number to 7 in the eighth-shell (ES) scheme [23]. In the case of special-purpose computers with low network latency, neutral-territory (NT) [24] and related [25] schemes achieve asymptotically smaller import volumes for fine granularities, N/P (P is the number of processors). In addition to these parallel algorithms for dynamic pair computations, numerous schemes have been employed in biological MD codes to efficiently compute static n-tuple computations [22, 26].

In contrast to these remarkable progresses in parallel algorithms for dynamic pair (n = 2) and static n-tuple computations, parallel dynamic n-tuple computation is still in its infancy. Fundamental questions include: How can we generalize the computation-redundancy removal in the HS scheme and the import-volume reduction in the ES scheme developed for pair computation into arbitrary dynamic n-tuple computations? To answer these questions in a mathematically rigorous and systematic manner, we here develop a novel computation-pattern algebraic framework. Based on this framework and translation- and reflection-invariant properties of n-tuple computations, we then design a shift-collapse (SC) algorithm. Our algebraic framework allows not only a formal proof of force-computation completeness but also a quantitative analysis of n-tuple search cost and import volume. We will show, for general dynamic n-tuple computations, that: (1) the pair HS scheme can be generalized to a reflective-collapse (RC) scheme to tighten the n-tuple search space; and (2) the pair ES scheme can be generalized to an octant-compression (OC) shift scheme to minimize the import volume.

This paper is organized as follows. Section 2 provides background information on n-tuple MD. Section 3 presents the computation-pattern algebraic framework for n-tuple computation and the SC algorithm along with a proof of its correctness. Section 4 analyzes the search-space size and communication cost of the SC algorithm. Section 5 presents performance benchmark results on BlueGene/Q and Intel Xeon clusters. Conclusions are drawn in section 6.

2. BACKGROUND

In this section, we describe many-body MD, followed by the introduction of dynamic range-limited n-tuple computation in it.

2.1 Many-Body Molecular Dynamics

MD simulation follows the time evolution of an N-atom system by numerically integrating Newton's equations of motion:

$$m_i \frac{d^2\mathbf{x}_i}{dt} = \mathbf{f}_i = -\frac{\partial \Phi(\mathbf{X})}{\partial \mathbf{x}_i}, \quad (1)$$

where $\mathbf{X} = \{\mathbf{x}_0,\ldots,\mathbf{x}_{N-1}\}$ denotes a set of atomic positions, m_i and $\mathbf{f}_i$ are the mass of and force acting on atom i, and t is time. In Eq. (1), the many-body interatomic potential-energy function $\Phi(\mathbf{X})$ is a sum of n-body potential terms Φ_n:

$$\Phi(\mathbf{X}) = \Phi_2 + \Phi_3 + \ldots + \Phi_{n_{\max}} = \sum_{n=2}^{n_{\max}} \Phi_n, \quad (2)$$

where $n_{\max}$ is the maximum n and Φ_n is a function of n-tuples of atomic positions $(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})$.

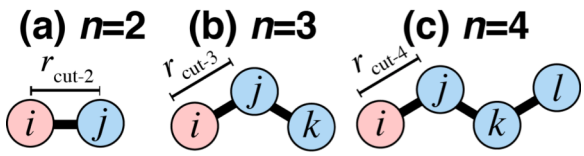

Figure 1. Schematic of range-limited n-tuples: (a) pair (n = 2), (b) triplet (n = 3), and (c) quadruplet (n = 4).

Force computation is the most time-consuming step in MD simulation. For each simulation time step, we need to map the current set of atomic positions, $\mathbf{R} = \{\mathbf{r}_0, \ldots, \mathbf{r}_{N-1}\}$, to n-body force terms for all $n \in [2, n_{\max}]$ and all atoms $i \in [0, N-1]$ as

$$\mathbf{f}_i^{(n)} = -\sum_{\forall(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1})\in\Gamma^{(n)}} \frac{\partial}{\partial \mathbf{x}_i} \Phi_n(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1}) \Big|_{(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})=(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1})}, \quad (3)$$

where $\Gamma^{(n)}$ denotes the set of all n-tuples of atom positions in $\mathbf{R}$. For a given n-tuple, $\chi = (\mathbf{r}_0,\mathbf{r}_1,\ldots,\mathbf{r}_{n-1})$ (see Figures 1 (a)-(c) for n = 2-4), forces exerted on all atoms in χ can be calculated simultaneously in one computational step:

$$\forall \mathbf{r}_i \in \chi : \mathbf{f}_i^{(n)} \leftarrow \mathbf{f}_i^{(n)} - \frac{\partial}{\partial \mathbf{x}_i} \Phi_n(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1}) \Big|_{(\mathbf{x}_0,\ldots,\mathbf{x}_{n-1})=\chi}. \quad (4)$$

Namely, the set of n-body forces on all atoms can be decomposed into partial contributions from different n-tuples χ:

$$\left\{\mathbf{f}_i^{(n)} \middle| i \in [0, N-1]\right\} = \mathbf{F}^{(n)} = \bigcup_{\forall \chi \in \Gamma^{(n)}} \mathbf{F}_\chi^{(n)}. \quad (5)$$

In all problems we consider, χ in Eq. (5) is undirectional, reflecting the Newton's 3rd law [9, 12]. Namely, $(\mathbf{r}_0,\mathbf{r}_1,\ldots,\mathbf{r}_{n-1})$ produces the same forces as $(\mathbf{r}_{n-1},\mathbf{r}_{n-2},\ldots,\mathbf{r}_0)$, thus they are not counted as separate entities. We call this reflective equivalence.

Assume that force calculation for each $\mathbf{F}_\chi^{(n)}$ can be executed in constant time. The computational complexity of n-body force computation is then proportional to the size of $\Gamma^{(n)}$, *i.e.*, $N!/[2(N-n)!] = O(N^n)$ for a system with $N >> n$ (which is the case in most MD simulations). Thus, the computational complexity is dictated by that of the largest n-body term, $O(N^{n_{\max}})$.

2.2 Dynamic Range-Limited n-Tuple Computation

Atomic interaction is often range-limited, *i.e.*, only atoms within short distances contribute to forces. Force computation for a range-limited n-body interatomic potential is defined as Eq. (3), where $\Gamma^{(n)}$ is replaced by its subset $\Gamma^{*(n)} \subseteq \Gamma^{(n)}$:

$$\Gamma^{*(n)} = \left\{(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \middle| r_{k,k+1} < r_{\text{cut-}n} \text{ for all } k \in \{0,\ldots,n-2\}\right\}, \quad (6)$$

where $r_{k,k+1}$ is the interatomic distance between $\mathbf{r}_k$ and $\mathbf{r}_{k+1}$ and $r_{\text{cut-}n}$ is the cutoff distance for n-body interaction, see Figures 1(a)-(c).

Formal complexity of pruning $\Gamma^{(n)}$ to obtain $\Gamma^{*(n)}$ is exponential in n, which is not practical. However, it is possible to efficiently find a set of n-tuples $\mathbf{S}^{(n)} \subseteq \Gamma^{(n)}$ such that $|\mathbf{S}^{(n)}| << |\Gamma^{(n)}|$ and $\Gamma^{*(n)} \subseteq \mathbf{S}^{(n)}$ (Figure 2). After finding $\mathbf{S}^{(n)}$, elements in $\Gamma^{*(n)}$ can be obtained simply by exhaustive "filtering" from $\mathbf{S}^{(n)}$. Since $\mathbf{S}^{(n)}$ is used to compute forces, hereafter we denote $\mathbf{S}^{(n)}$ as a *force set*, and

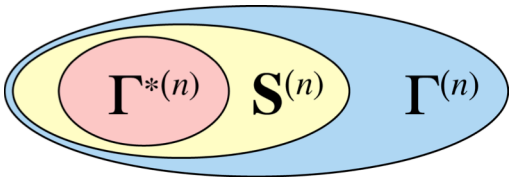

Figure 2. Set diagram showing the relationship of a bounding force set $\mathbf{S}^{(n)}$ with $\Gamma^{(n)}$ and $\Gamma^{*(n)}$.

$\mathbf{S}^{(n)}$ that satisfies $\Gamma^{*(n)} \subseteq \mathbf{S}^{(n)}$ as a *bounding force set*. Thus, MD force computation can be restated as a problem to find a bounding force set $\mathbf{S}^{(n)}$ for all $n \in \{2,\ldots,n_{\max}\}$. To solve this problem, an efficient pruning algorithm $\mathbf{A}:\mathbf{R} \Rightarrow \mathbf{S}^{(n)}$ is required. A widely used approach to achieve this is a cell method. It employs a cell data structure to prune $\Gamma^{(n)}$ in $O(N)$ time to obtain $\mathbf{S}^{(n)}$ that tightly bounds $\Gamma^{*(n)}$.

Cell-based methods are powerful yet simple. Its procedure is straightforward to implement for the case of pair computation. However, cell methods for general n-tuple computation have not been formalized mathematically. To address this problem systematically, we formulate a cell-based n-tuple MD problem using an algebraic framework in the next section.

3. COMPUTATION-PATTERN ALGEBRAIC FRAMEWORK

In this section, dynamic range-limited n-tuple MD is formalized using a computation-pattern algebraic framework. Subsection 3.1 introduces a uniform-cell pattern (UCP) formalism, which is an algebraic generalization of conventional cell methods to solve n-tuple MD problems for arbitrary n. In subsection 3.2, we propose a shift-collapse (SC) algorithm to efficiently solve n-tuple MD on the basis of UCP. Correctness of the SC algorithm is proved in subsection 3.3.

3.1 Algebraic Formulation of Cell-Based MD

3.1.1 Cell data-structure

Cell-based MD divides a simulation volume into a non-overlapping set of small cells with side lengths equal or slightly larger than $r_{\text{cut-}n}$. Each cell volume $\boldsymbol{\zeta}$ contains a subset of atoms in $\mathbf{R}$ that fall within its volume:

$$\mathbf{c} = \{\mathbf{r}_i \mid \forall \mathbf{r}_i \in \mathbf{R} \text{ and in volume } \boldsymbol{\zeta}\}, \tag{7}$$

see Figure 3(a). For simplicity, we consider a simple lattice of cubic cells. Then, each cell $\mathbf{c}(\mathbf{q})$ can be indexed using a 3-element vector $\mathbf{q} = (q_x, q_y, q_z)$, which specifies the Cartesian coordinate of the cell position in the lattice. Let L_x, L_y, and L_z be the number of cells in the x, y, and z directions, respectively. Then, the set of all cell indices constitutes a vector space $\mathbf{L}$:

$$\mathbf{L} = \left\{ \mathbf{q} = (q_x, q_y, q_z) \;\middle|\; \begin{array}{l} \forall q_x \in \{0,\ldots,L_x - 1\} \\ \forall q_y \in \{0,\ldots,L_y - 1\} \\ \forall q_z \in \{0,\ldots,L_z - 1\} \end{array} \right\}.$$

An essential ingredient of our algebraic formulation is a cell domain Ω, which is defined as a set of all cells:

$$\Omega = \{\mathbf{c}(\mathbf{q}) \mid \forall \mathbf{q} \in \mathbf{L}\}. \tag{8}$$

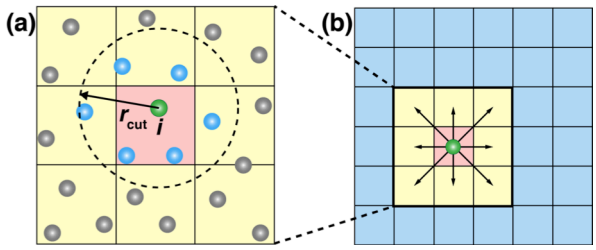

Figure 3. (a) Cell data-structure, where circles are atoms and small squares are cells. Dashed circle denotes the cutoff radius centered at atom *i*. (b) Uniform-cell pattern for pair computation, which generates all pairs between each cell and its neighbor cells.

In this paper, we assume periodic boundary conditions in all Cartesian directions. Then, for cell $\mathbf{c}(\mathbf{q})$, we define a cell-offset operation $\mathbf{c}(\mathbf{q}+\Delta) \rightarrow \mathbf{c}(\mathbf{q}')$ $(\Delta \in \mathbb{Z}^3)$ as

$$q'_\alpha = (q_\alpha + \Delta_\alpha)\%L_\alpha \quad (\alpha \in \{x,y,z\}) \;,$$

where % is a modulo operation. Note that Ω needs to be dynamically constructed every MD step because the positions of atoms are changing as the computation proceeds.

3.1.2 Uniform-cell pattern MD

UCP formalism is a generalization of a cell-based MD method using an algebraic approach (Figure 3(b)). To compute forces for each n-body potential term, UCP generates a force set $\mathbf{S}^{(n)}$ using information from cell domain Ω along with a *computation pattern* $\Psi^{(n)}$, which is a set of pre-define rules regarding interaction among cells. UCP applies $\Psi^{(n)}$ to each cell $\mathbf{c}$ and generates a *cell search-space* $\mathbf{S}_{\text{cell}}$, a set of n-tuples that forms a force subset $\mathbf{S}_{\text{cell}}(\mathbf{c}, \Psi^{(n)}) \subseteq \mathbf{S}^{(n)}$. By looping over all cells, it generates the entire $\mathbf{S}^{(n)}$. This is analogous to stencil computation for solving partial differential equations, in which a stencil (equivalent to computation pattern in our framework) defines local computation rules for each grid point and its neighbor grid points and the stencil is applied to all grid points in a lattice to solve the problem [27, 28].

To describe UCP, we first define a *computation path* $\mathbf{p}^{(n)}$ for n-tuple computation as a list of n vectors in $\mathbf{L}$ (see each arrow in Figure 3(b) as an example of pair-computation path):

$$\mathbf{p}^{(n)} = (\mathbf{v}_0,\ldots,\mathbf{v}_{n-1}) \in \mathbf{L}^n.$$

The inverse of $\mathbf{p}$ is defined as $\mathbf{p}^{-1} = (\mathbf{v}_{n-1},\ldots,\mathbf{v}_0)$. In addition, we define a *differential representation*, $\sigma(\mathbf{p}) \in \mathbf{L}^{n-1}$, of each path $\mathbf{p}$ as

$$\sigma(\mathbf{p}) = (\mathbf{v}_1 - \mathbf{v}_0,\ldots,\mathbf{v}_{n-1} - \mathbf{v}_{n-2}).$$

Then, a computation pattern is defined as a set of computation paths $\Psi^{(n)} = \{\mathbf{p}^{(n)}\}$ (*i.e.* set of all arrows in Figure 3(b)). Given a cell domain Ω and a computation pattern $\Psi^{(n)}$, a force set $\mathbf{S}^{(n)}$ is obtained as a union of cell search-spaces $\mathbf{S}_{\text{cell}}$:

$$\mathbf{S}^{(n)} = \text{UCP}(\Omega, \Psi^{(n)}) = \bigcup_{\forall \mathbf{c}(\mathbf{q}) \in \Omega} \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}), \Psi^{(n)}), \tag{9}$$

where cell search-space $\mathbf{S}_{\text{cell}}$ for each cell $\mathbf{c}(\mathbf{q})$ is defined as

Table 1. Uniform-cell pattern algorithm

Algorithm UCP
Input:
Ω : a cell domain
$\Psi^{(n)} \subseteq \mathbf{L}^n$: a computation pattern
Output:
$\mathbf{S}^{(n)} \subseteq \mathbf{R}^n$: a force set
Steps:
1. $\mathbf{S}^{(n)} = \varnothing$
2. for $\forall \mathbf{c}(\mathbf{q}) \in \Omega$
3. $\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q})) = \varnothing$
4. for $\forall \mathbf{p} = (\mathbf{v}_0,\ldots,\mathbf{v}_{n-1}) \in \Psi^{(n)}$
5. for $\forall \mathbf{r}_0 \in \mathbf{c}(\mathbf{q}+\mathbf{v}_0)$
6. ⋱
7. for $\forall \mathbf{r}_{n-1} \in \mathbf{c}(\mathbf{q}+\mathbf{v}_{n-1})$
8. $\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q})) \leftarrow \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q})) \cup \{(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1})\}$
9. $\mathbf{S}^{(n)} \leftarrow \mathbf{S}^{(n)} \cup \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}))$

$$\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\Psi^{(n)}) = \left\{ (\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \;\middle|\; \begin{array}{c} \forall \mathbf{p} = (\mathbf{v}_0,\ldots,\mathbf{v}_{n-1}) \in \Psi^{(n)} \\ \forall k \in \{0,\ldots,n-1\} : \forall \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_k) \end{array} \right\}. \quad (10)$$

Namely, $\mathbf{S}_{\text{cell}}$ is a set of n-tuples contained in all paths $\mathbf{p} \in \Psi^{(n)}$, which is associated with cell $\mathbf{c}(\mathbf{q})$, see Figure 4. An algorithm that performs operations in Eqs. (9) and (10) is given in Table 1. $\mathbf{S}^{(n)}$ in Eq. (9) is obtained by looping over all $\mathbf{c} \in \Omega$ (line 2), where each iteration adds $\mathbf{S}_{\text{cell}}(\mathbf{c})$ to $\mathbf{S}^{(n)}$ (line 9). In order for UCP to be a valid MD computation, its computation pattern $\Psi^{(n)}$ must satisfy the n-body completeness condition by generating a bounding force set:

$$\Gamma^{*(n)} \subseteq \text{UCP}(\Omega,\Psi^{(n)}) . \quad (11)$$

In this way, MD problem amounts to finding a computation pattern that satisfies Eq. (11). Such a computation pattern is regarded as *n-complete*. Existing cell-based methods can be expressed as particular computation patterns, and our algebraic formalism can be used to prove their completeness (*e.g.* HS and ES for pair-completeness). This will be discussed in section 4.3.

Satisfying Eq. (11) only guarantees that $\Psi^{(n)}$ is sufficient to generate a bounding force set using the UCP algorithm in Table 1. The generated force set could still contain duplicated or reflectively equivalent tuples as mentioned in section 2.1. In a practical implementation, these redundant tuples must be removed. Hence, the cost of the UCP algorithm, T_{UCP}, is a cost for filtering out the unnecessary tuples from cell search-space $\mathbf{S}_{\text{cell}}$ of every cell. This is proportional to a sum of all search-space sizes $|\mathbf{S}_{\text{cell}}|$:

$$T_{\text{UCP}} \propto \sum_{\forall \mathbf{c} \in \Omega} \left| \mathbf{S}_{\text{cell}}(\mathbf{c},\Psi^{(n)}) \right| . \quad (12)$$

3.1.3 Parallel UCP

In this paper, we consider parallel MD algorithms, where different sets of atoms $\mathbf{c}$ along with associated computation $\mathbf{S}_{\text{cell}}(\mathbf{c}, \Psi^{(n)})$ are assigned to different processors. Since $\mathbf{S}_{\text{cell}}(\mathbf{c}, \Psi^{(n)})$ requires data from other cells, these data must be imported if they reside in other processors. Depending on computation paths in $\Psi^{(n)}$, different cells need be imported. This leads to different import volumes (*i.e.* the number of imported cells), and hence different communication costs.

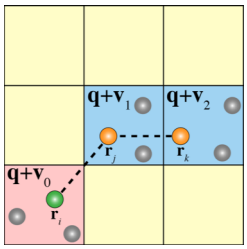

Figure 4. *n*-tuple (*n* = 3) generation for cell search-space $\mathbf{S}_{\text{cell}}$ of cell c(q) (colored in magenta) and computation path ($\mathbf{v}_0$,$\mathbf{v}_1$,$\mathbf{v}_2$). $\mathbf{S}_{\text{cell}}$ generates all triplets where the 1st, 2nd, and 3rd atoms in the triplet are in cells c(q+$\mathbf{v}_0$), c(q+$\mathbf{v}_1$), and c(q+$\mathbf{v}_2$), respectively. Dashed line shows one of the triplets ($\mathbf{r}_i$,$\mathbf{r}_j$,$\mathbf{r}_k$).

To quantify the import volume, we define the *cell coverage* Π of a computation pattern $\Psi^{(n)}$ as a set of cells that are needed in order to compute $\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}), \Psi^{(n)})$:

$$\Pi(\mathbf{c}(\mathbf{q}),\Psi^{(n)}) = \left\{ \mathbf{c}(\mathbf{q}+\mathbf{v}_k) \;\middle|\; \begin{array}{c} \forall \mathbf{p} = (\mathbf{v}_0,\ldots,\mathbf{v}_{n-1}) \in \Psi^{(n)} \\ \forall k : \mathbf{v}_k \in \mathbf{p} \end{array} \right\},$$

for an arbitrary cell $\mathbf{c}(\mathbf{q}) = \mathbf{c}$. We also define the *cell footprint* of a computation pattern to be the cardinality of $\Pi(\mathbf{c},\Psi^{(n)})$. The cell footprint is independent of cell, thus we denote it as $|\Pi(\Psi^{(n)})|$. Also, we define the *cell-domain coverage* as a union of the cell coverage of all cells in the domain:

$$\Pi(\Omega,\Psi^{(n)}) = \bigcup_{\forall \mathbf{c} \in \Omega} \Pi(\mathbf{c},\Psi^{(n)}) , \quad (13)$$

which is the set of cells that are needed to do computation of all cells in Ω. Here and in subsequent discussions of parallel MD, Ω denotes a cell domain for each processor. Then, the set of cells ω that needs to be imported from other processors are the cells that are in the cell-domain coverage but not in the cell domain:

$$\omega(\Omega,\Psi^{(n)}) = \Pi(\Omega,\Psi^{(n)}) - \Omega .$$

The import volume of n-tuple computation is defined as the size of all cells that need to be imported (*i.e.* the cardinality of ω):

$$V_{\omega}(\Omega,\Psi^{(n)}) = \left| \omega(\Omega,\Psi^{(n)}) \right| . \quad (14)$$

The overall import volume V_{import} is determined by the largest import volume among all n:

$$V_{\text{import}} = \max_n (V_{\omega}(\Omega,\Psi^{(n)})) .$$

3.1.4 Optimal UCP-MD problem

Based on the computational and communication cost functions of UCP in Eqs. (12) and (14), the parallel MD problem is reduced to the following optimization problem:

PROBLEM (OPTIMAL UCP-MD). *Given a cell domain Ω, find a set of n-complete computation patterns $\{\Psi^{*(n)}\}$ for all $n \in \{2,\ldots,n_{max}\}$ such that each $\Psi^{*(n)}$ satisfies the following:*

1. $$\Psi^{*(n)} = \underset{\Psi^{(n)}}{\text{argmin}} \left(\sum_{\forall \mathbf{c} \in \Omega} \left| \mathbf{S}_{\text{cell}}(\mathbf{c},\Psi^{(n)}) \right| \right)$$

2. $\Psi^{*(n)} = \underset{\Psi^{(n)}}{\operatorname{argmin}} \left(\left| \bigcup_{\forall \mathbf{c} \in \Omega} \Pi(\mathbf{c}, \Psi^{(n)}) - \Omega \right| \right)$

The first condition minimizes the search cost (*i.e.* filtering out the unnecessary tuples), while the second condition minimizes the import volume in parallel MD. In the next subsection, we present an SC algorithm to solve the optimal UCP-MD problem. We will show in section 4 that the SC algorithm in the case of pair computation is reduced to the best cell methods.

3.2 Shift-Collapse Algorithm

We propose an SC algorithm, which makes a complete computation-pattern while addressing the two optimality conditions presented in subsection 3.1.4. The SC algorithm consists of three main phases (see Table 2). The first phase, full-shell generation subroutine (GENERATE-FS in Table 3) performs (n-1)-fold nested loops (lines 2-5) to enumerate all possible paths to generate an n-complete pattern (*e.g.* for n = 3 in Figure 5(a)). This creates a computation pattern centered at the original cell (*i.e.* $\mathbf{c}(\mathbf{q})$ in Eq. (9)) surrounded by (n-1)-layers of nearest-neighbor cells (yellow region in Figure 5(a)). In the following, we refer to this computation pattern as *full shell* (FS).

In the second phase, octant-compression shift subroutine (OC-SHIFT) shifts all paths in the FS pattern to the first octant in order to compact the cell footprint $|\Pi(\Psi^{(n)})|$. To achieve this, all of the paths in FS are shifted toward the upper corner (the highest Cartesian coordinates in all 3 directions, see Figure 5(b)) of the FS cell coverage (see lines 3-4 in Table 4).

In the last phase, reflective collapse subroutine (R-COLLAPSE) finds and removes redundant paths that generate the same force set (Table 5). This is achieved by doubly nested loops (lines 2-3) over all pairs of paths to compare their equivalence (line 4). If a path-pair is found to be equivalent, one of the paths is removed from the pattern (line 5). The equivalence between two arbitrary paths $\mathbf{p}$ and $\mathbf{p}'$ is tested as $\sigma(\mathbf{p}') = \sigma(\mathbf{p}^{-1})$, which will be proven in Lemma 3 in section 3.3.

The most important property for the SC algorithm is that a path-shift operation does not alter the resulting force set, *i.e.*, translational invariance. Let $\mathbf{p}$ = ($\mathbf{v}_0$,…,$\mathbf{v}_{n-1}$) be an n-tuple computation path, then path shifting is defined as a translation of the origin of the computation path:

$$\mathbf{p} + \Delta = (\mathbf{v}_0 + \Delta, ..., \mathbf{v}_{n-1} + \Delta),$$

where $\Delta \in \mathbb{Z}^n$ denotes a shifting vector. The following theorem proves the path-shift invariance of force computation.

THEOREM 1 (PATH-SHIFT INVARIANCE). *Let Ω be a cell domain and $\mathbf{p}$ be an n-tuple computation path. For an arbitrary shift vector $\Delta \in \mathbb{Z}^n$, UCP(Ω,{$\mathbf{p}$})* = *UCP(Ω,{$\mathbf{p}$+Δ})*.

PROOF. For a particular cell $\mathbf{c}(\mathbf{q}) \in \Omega$ and $\mathbf{p}$ = {$\mathbf{v}_0$,…,$\mathbf{v}_{n-1}$}, $\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}), \{\mathbf{p}+\Delta\})$ reads

$$\begin{aligned} \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}), \{\mathbf{p}+\Delta\}) &= \{(\mathbf{r}_0,...,\mathbf{r}_{n-1}) \mid \forall \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_k+\Delta)\} \\ &= \{(\mathbf{r}_0,...,\mathbf{r}_{n-1}) \mid \forall \mathbf{r}_k \in \mathbf{c}((\mathbf{q}+\Delta)+\mathbf{v}_k)\} \\ &= \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}+\Delta), \{\mathbf{p}\}) \end{aligned} \quad (15)$$

The union of Eq. (15) over all cells in the domain yields

Table 2. Shift-collapse algorithm

Algorithm SC
Input:
$n \in \{2,3,\ldots\}$, the length of a tuple
Output:
$\Psi^{(n)}_{\text{SC}} \subseteq \mathbf{L}^n$: Shift-collapse pattern
Steps:
1. $\Psi_{\text{FS}} \leftarrow$ GENERATE-FS(n)
2. $\Psi_{\text{OC}} \leftarrow$ OC-SHIFT(Ψ_{FS})
3. $\Psi_{\text{SC}} \leftarrow$ COLLAPSE(Ψ_{OC})

Table 3. Full-shell generation subroutine

Subroutine GENERATE-FS(n)
Input:
$n \in \{2,3,\ldots\}$, the length of a tuple
Output:
$\Psi^{(n)}_{\text{FS}} \subseteq \mathbf{L}^n$: Full-shell computation pattern
Steps:
1. $\Psi^{(n)}_{\text{FS}} = \varnothing$, $\mathbf{v}_0 = (0,0,0)$
2. for $\forall \mathbf{v}_1 = (\alpha_1^x, \alpha_1^y, \alpha_1^z)$ where $-1 \le \alpha_1^{\text{xlylz}} \le 1$
3. for $\forall \mathbf{v}_2 = (\alpha_2^x, \alpha_2^y, \alpha_2^z)$ where $\alpha_1^{\text{xlylz}} - 1 \le \alpha_2^{\text{xlylz}} \le \alpha_1^{\text{xlylz}} + 1$
4. ⋱
5. for $\forall \mathbf{v}_{n-1} = (\alpha_{n-1}^x, \alpha_{n-1}^y, \alpha_{n-1}^z)$ where $\alpha_{n-2}^{\text{xlylz}} - 1 \le \alpha_{n-1}^{\text{xlylz}} \le \alpha_{n-2}^{\text{xlylz}} + 1$
6. $\Psi^{(n)}_{\text{FS}} \leftarrow \Psi^{(n)}_{\text{FS}} \cup (\mathbf{v}_0,...,\mathbf{v}_{n-1})$

Table 4. Octant-compression shift subroutine

Subroutine OC-SHIFT($\Psi^{(n)}$)
Input:
$\Psi^{(n)} \subseteq \mathbf{L}^n$: computation pattern
Output
$\Psi^{(n)}_{\text{OC}} \subseteq \mathbf{L}^n$: OC computation pattern
Steps:
1. $\Psi^{(n)}_{\text{OC}} = \varnothing$
2. for $\forall \mathbf{p} = (\mathbf{v}_0,...,\mathbf{v}_{n-1}) \in \Psi^{(n)}$
3. $\Delta^{\beta}_{\min} = \min_{0 \le i < n}(\mathbf{v}_i^{\beta})$ for all $\beta \in \{x,y,z\}$
4. $\mathbf{p}' \leftarrow \mathbf{p} - \Delta_{\min}$
5. $\Psi^{(n)}_{\text{OC}} \leftarrow \Psi^{(n)}_{\text{OC}} \cup \{\mathbf{p}'\}$

Table 5. Reflective-collapse subroutine

Subroutine R-COLLAPSE($\Psi^{(n)}$)
Input:
$\Psi^{(n)} \subseteq \mathbf{L}^n$: computation pattern
Output:
$\Psi^{(n)}_{\text{RC}} \subseteq \mathbf{L}^n$: non-redundant computation pattern
Steps:
1. $\Psi^{(n)}_{\text{RC}} = \Psi^{(n)}$
2. for $\forall \mathbf{p} \in \Psi^{(n)}$
3. for $\forall \mathbf{p}' \in \Psi^{(n)}$ where $\mathbf{p}' \ne \mathbf{p}$
4. if $\sigma(\mathbf{p}') = \sigma(\mathbf{p}^{-1})$
5. $\Psi^{(n)}_{\text{RC}} \leftarrow \Psi^{(n)}_{\text{RC}} - \{\mathbf{p}'\}$

$$\bigcup_{\forall \mathbf{c}(\mathbf{q}+\Delta)\in\Omega} \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}+\Delta),\{\mathbf{p}\}) = \bigcup_{\forall \mathbf{c}(\mathbf{q}')\in\Omega} \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}'),\{\mathbf{p}\}) = \text{UCP}(\Omega,\{\mathbf{p}\})$$

The last equality comes from the definition of UCP in Eq. (9). Therefore, $\text{UCP}(\Omega,\{\mathbf{p}+\Delta\}) = \text{UCP}(\Omega,\{\mathbf{p}\})$. □

3.3 Correctness of SC Algorithm

To prove the correctness of the SC algorithm, we will show that the computation pattern generated by it is complete as defined in Eq. (11). The SC algorithm consists of three phases. We will show that the first phase generates a complete pattern, while the second and third phases do not alter the resulting force set.

First, we prove that the GENERATE-FS routine generates a pattern that constitutes a bounding force set.

LEMMA 1. *Given a computation pattern* $\Psi^{(n)}_{FS}$ = *GENERATE-FS(n), the force set* $\mathbf{S}^{(n)}$ = *UCP(*Ω*,* $\Psi^{(n)}_{FS}$*) is a bounding force set for an arbitrary cell domain* Ω.

PROOF. GENERATE-FS(n) in Table 3 constructs $\Psi^{(n)}_{\text{FS}}$ as a union of $(\mathbf{v}_0,\ldots,\mathbf{v}_{n-1})$ such that $\mathbf{v}_{k+1} = (v_k^x \pm 1, v_k^y \pm 1, v_k^z \pm 1)$ for all $0 \le k < n$-1. Thus $\mathbf{c}(\mathbf{q}+\mathbf{v}_k)$ is a nearest neighbor cell of $\mathbf{c}(\mathbf{q}+\mathbf{v}_{k+1})$ for an arbitrary $\mathbf{q} \in \mathbf{L}$. Consider an arbitrary n-tuple $\chi = (\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \in \Gamma^{*(n)}$, where $r_{k,k+1} < r_{\text{cut}-n}$ for all $0 \le k < n$-1. We will prove that χ is included in a cell set $(\mathbf{c}(\mathbf{q}+\mathbf{v}_0),\mathbf{c}(\mathbf{q}+\mathbf{v}_1),\ldots,\mathbf{c}(\mathbf{q}+\mathbf{v}_{n-1}))$ using mathematical induction:

Base: $\mathbf{r}_0 \in \mathbf{c}(\mathbf{q}+\mathbf{v}_0)$,

Induction: if $\mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_k)$, then $\exists \mathbf{v}_{k+1} : \mathbf{r}_{k+1} \in \mathbf{c}(\mathbf{q}+\mathbf{v}_{k+1})$.

Base step is trivial because we choose $\mathbf{q}$ such that $\mathbf{r}_0 \in \mathbf{c}(\mathbf{q}+\mathbf{v}_0)$. For the induction step, atomic pair $(\mathbf{r}_k, \mathbf{r}_{k+1})$ with distance $r_{k,k+1} < r_{\text{cut}-n}$ is guaranteed to reside in cell $\mathbf{c}(\mathbf{q}+\mathbf{v}_k)$ and $\mathbf{c}(\mathbf{q}+\mathbf{v}_{k+1})$ because the cell size is larger than the cutoff $r_{\text{cut}-n}$. This proves the induction step. Hence, $\exists \mathbf{v}_k : \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_k)$ for all $0 \le k < n$-1 and thus, the entire n-tuple χ is contained in a cell set $(\mathbf{c}(\mathbf{q}+\mathbf{v}_0),\mathbf{c}(\mathbf{q}+\mathbf{v}_1),\ldots,\mathbf{c}(\mathbf{q}+\mathbf{v}_{n-1}))$. From the definition of $\mathbf{S}_{\text{cell}}$ in Eq. (10), for an arbitrary n-tuple in $\Gamma^{*(n)}$ there exists $\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\Psi^{(n)}_{\text{FS}})$. Therefore from the definition of UCP in Eq. (9), $\Gamma^{*(n)} \subseteq \text{UCP}(\Omega,\Psi^{(n)})$ is satisfied and $\text{UCP}(\Omega,\Psi^{(n)})$ is complete. □

Next, we prove that OC-SHIFT operation does not alter the resulting force set.

LEMMA 2. *Consider an arbitrary computation pattern* $\Psi^{(n)}$ *and a cell domain* Ω. *For* $\Psi^{(n)}_{OC}$ = *OC-SHIFT(*$\Psi^{(n)}$*),* $\text{UCP}(\Omega,\Psi^{(n)}_{OC}) = \text{UCP}(\Omega,\Psi^{(n)})$.

PROOF. The OC-SHIFT subroutine performs a series of shifting operations to all paths in $\Psi^{(n)}$. From Theorem 1, each shifting operation keeps the force set unchanged. Therefore, $\text{UCP}(\Omega,\Psi^{(n)}_{OC}) = \text{UCP}(\Omega,\Psi^{(n)})$ □

To prove that R-COLLAPSE keeps the force set unchanged, we first prove the equivalence between paths (*i.e.* line 4 in Table 5) used in R-COLLAPSE.

LEMMA 3 (REFLECTIVE INVARIANCE). *Consider a cell domain* Ω *and paths* $\mathbf{p}$, $\mathbf{p}'$ *of size n. If* $\sigma(\mathbf{p}^{-1}) = \sigma(\mathbf{p}')$, *then* $\text{UCP}(\Omega,\{\mathbf{p}\}) = \text{UCP}((\Omega,\{\mathbf{p}'\})$.

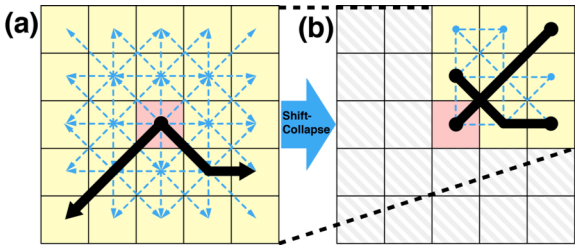

Figure 5. 2D schematic of a shift-collapse (SC) computation pattern. (a) FS computation pattern for n = 3, containing all possible paths of length 3 originated from the center cell (magenta). (b) SC computation pattern after OC-shift and R-collapse subroutines. Yellow cells denote the cell coverage of the center cell, which for SC is much smaller than that for FS.

PROOF. Let $\mathbf{p} = (\mathbf{v}_0,\ldots,\mathbf{v}_{n-1})$, $\mathbf{p}' = (\mathbf{u}_0,\ldots,\mathbf{u}_{n-1})$ and the inverse path of $\mathbf{p}$ be $\mathbf{p}^{-1} = (\mathbf{v}_{n-1},\ldots,\mathbf{v}_0)$. From the assumption $\sigma(\mathbf{p}^{-1}) = \sigma(\mathbf{p}')$, we have

$$\mathbf{u}_{k+1} - \mathbf{u}_k = \mathbf{v}_{n-2-k} - \mathbf{v}_{n-1-k}, \tag{16}$$

for all $k \in \{0,\ldots,n\text{-}2\}$. From the shift-invariant property in Theorem 1,

$$\bigcup_{\forall \mathbf{c}(\mathbf{q})\in\Omega} \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}'\}) = \bigcup_{\forall \mathbf{c}(\mathbf{q})\in\Omega} \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}'+\Delta\}).$$

Let $\Delta = \mathbf{v}_{n-1}$-$\mathbf{u}_0$, then $\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}'+\Delta\})$ reads

$$\begin{aligned}\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}'\}) &= \{(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \mid \forall \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{u}_k+\Delta)\} \\ &= \{(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \mid \forall \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_{n-1}-\mathbf{u}_0+\mathbf{u}_k)\}\end{aligned}, \tag{17}$$

Using telescoping, $\mathbf{q}+\mathbf{v}_{n-1}-\mathbf{u}_0+\mathbf{u}_k$ in Eq. (17) can be rewritten as

$$\mathbf{q}+\mathbf{v}_{n-1}-\mathbf{u}_0+\mathbf{u}_k = \mathbf{q}+\mathbf{v}_{n-1}+\sum_{i=0}^{k-1}(\mathbf{u}_{i+1}-\mathbf{u}_i). \tag{18}$$

$\mathbf{u}_{i+1}$-$\mathbf{u}_i$ in the R.H.S. of Eq. (18) can be replaced by $\mathbf{v}_{n-2-i}$-$\mathbf{v}_{n-1-i}$ according to Eq. (16), which yields

$$\begin{aligned}\mathbf{q}+\mathbf{v}_{n-1}+\sum_{i=0}^{k-1}(\mathbf{u}_{i+1}-\mathbf{u}_i) &= \mathbf{q}+\mathbf{v}_{n-1}+\sum_{i=0}^{k-1}(\mathbf{v}_{n-2-i}-\mathbf{v}_{n-1-i}) \\ &= \mathbf{q}+\mathbf{v}_{n-1-k}\end{aligned}.$$

Therefore,

$$\mathbf{q}+\mathbf{v}_{n-1}-\mathbf{u}_0+\mathbf{u}_k = \mathbf{q}+\mathbf{v}_{n-1-k}. \tag{19}$$

Substituting Eq. (19) back into Eq. (17) yields,

$$\begin{aligned}\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}'\}) &= \{(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \mid \forall \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_{n-1-k})\} \\ &= \{(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \mid \forall \mathbf{r}_{n-1-k} \in \mathbf{c}(\mathbf{q}+\mathbf{v}_k)\} \\ &= \{(\mathbf{r}_{n-1},\ldots,\mathbf{r}_0) \mid \forall \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_k)\}\end{aligned}. \tag{20}$$

Based on the undirectionality of tuples described in section 2.1, $(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) = (\mathbf{r}_{n-1},\ldots,\mathbf{r}_0)$. Therefore Eq. (20) reads

$$\begin{aligned}\mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}'\}) &= \{(\mathbf{r}_0,\ldots,\mathbf{r}_{n-1}) \mid \forall \mathbf{r}_k \in \mathbf{c}(\mathbf{q}+\mathbf{v}_k)\} \\ &= \mathbf{S}_{\text{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}\})\end{aligned}.$$

Based on the definition of UCP in Eq. (9), this proves $\text{UCP}(\Omega,\{\mathbf{p}\}) = \text{UCP}(\Omega,\{\mathbf{p}'\})$. □

Using Lemma 3, we can now prove that the R-COLLAPSE subroutine preserves the completeness of a pattern.

LEMMA 4. *Let Ω be an arbitrary cell domain and $\Psi^{(n)}$ be a computation pattern. For $\Psi^{(n)}_{RC}$ = R-COLLAPSE($\Psi^{(n)}$), UCP($\Omega,\Psi^{(n)}_{RC}$) = UCP($\Omega,\Psi^{(n)}$).*

PROOF. In R-COLLAPSE subroutine (Table 5), lines 2-3 loop over all pairs of computation paths $\{\mathbf{p},\mathbf{p}'\}$ in $\Psi^{(n)}$. For each pair of paths, we remove one of them (*i.e.* collapsing) only when $\sigma(\mathbf{p}')=\sigma(\mathbf{p}^{-1})$ (lines 4-5). In such a case, $\Psi^{(n)}_{\mathrm{RC}} = \Psi^{(n)} - \{\mathbf{p}'\}$. Based on Lemma 3, $\mathbf{p}$ and $\mathbf{p}'$ produce the same force set, and thus

$$\mathrm{UCP}(\Omega,\Psi^{(n)}) = \mathrm{UCP}(\Omega,\Psi^{(n)}-\{\mathbf{p}'\}) . \quad (21)$$

Substituting $\Psi^{(n)}_{\mathrm{RC}} = \Psi^{(n)} - \{\mathbf{p}'\}$ into Eq. (21) yields $\mathrm{UCP}(\Omega,\Psi^{(n)}) = \mathrm{UCP}(\Omega,\Psi^{(n)}_{\mathrm{RC}})$, which proves the Lemma. □

Using Lemmas 1, 2, and 4, we can now prove the correctness of the SC algorithm by proving the completeness of SC pattern.

THEOREM 2. *Given an arbitrary cell domain Ω, a computation pattern $\Psi^{(n)}_{SC}$ generated from the shift-collapse algorithm is n-complete.*

PROOF. Lemma 1 states that GENERATE-FS produces $\Psi^{(n)}_{\mathrm{FS}}$, which is an n-complete pattern. Lemma 2 guarantees that for $\Psi^{(n)}_{\mathrm{OC}}$ = OC-SHIFT($\Psi^{(n)}_{\mathrm{FS}}$), UCP($\Omega,\Psi^{(n)}_{\mathrm{OC}}$) = UCP($\Omega,\Psi^{(n)}_{\mathrm{FS}}$) and thus $\Psi^{(n)}_{\mathrm{OC}}$ is also complete. According to Lemma 4, for $\Psi^{(n)}_{\mathrm{SC}}$ = R-COLLAPSE($\Psi^{(n)}_{\mathrm{OC}}$), UCP($\Omega,\Psi^{(n)}_{\mathrm{SC}}$) = UCP($\Omega,\Psi^{(n)}_{\mathrm{OC}}$). Therefore $\Psi^{(n)}_{\mathrm{SC}}$ is complete. □

4. THEORETICAL ANALYSIS

In this section, we perform theoretical analysis of the SC algorithm. First, we quantify the cost of n-tuple searches of the SC pattern. We then analyze and estimate the import volume of the SC the pattern. Finally, we discuss the relation of SC to previous works in the case of pair computation.

4.1 Search-Cost Analysis of SC Pattern

In this subsection, we show that the search cost for the SC pattern is approximately half that of FS. This size reduction arises from R-COLLAPSE, which removes all redundant computation paths using their reflective-invariant property in Lemma 3. First, we show that the size of a search space can be estimated by the cardinality of a computation pattern.

LEMMA 5. *Assume that the atom distribution is uniform. For an arbitrary cell domain Ω and a computation pattern $\Psi^{(n)}$, the search cost T_{UCP} is proportional to $|\Psi^{(n)}|$.*

PROOF. Let $\langle\rho_{\mathrm{cell}}\rangle$ be the average number of atoms per cell. For all paths $\mathbf{p}=(\mathbf{v}_0,\ldots,\mathbf{v}_{n-1})$ in $\Psi^{(n)}$ and an arbitrary cell $\mathbf{c}(\mathbf{q})\in\Omega$, the average number of atomic pairs in a cell-pair $\mathbf{c}(\mathbf{q}+\mathbf{v}_k)$ and $\mathbf{c}(\mathbf{q}+\mathbf{v}_{k+1})$ is $\langle\rho_{\mathrm{cell}}\rangle^2$. Thus, for an n-1 consecutive cell-pairs, the number of tuples in a cell search-space $\mathbf{S}_{\mathrm{cell}}$ centered at an arbitrary cell $\mathbf{c}(\mathbf{q})$ is

$$\left|\mathbf{S}_{\mathrm{cell}}(\mathbf{c}(\mathbf{q}),\{\mathbf{p}\})\right| = \left\langle\rho_{\mathrm{cell}}\right\rangle^{n-1} . \quad (22)$$

By summing Eq. (22) over all paths in $\Psi^{(n)}$, we obtain the average number of n-tuples associated with cell $\mathbf{c}(\mathbf{q})$ as

$$\left|\mathbf{S}_{\mathrm{cell}}(\mathbf{c}(\mathbf{q}),\Psi^{(n)})\right| = \left\langle\rho_{\mathrm{cell}}\right\rangle^{n-1}\left|\Psi^{(n)}\right| . \quad (23)$$

Summation of Eq. (23) over all cells in Ω yields the search cost in Eq. (12):

$$\sum_{\forall\mathbf{c}\in\Omega}\left|\mathbf{S}_{\mathrm{cell}}(\mathbf{c},\Psi^{(n)})\right| = T_{\mathrm{UCP}} = L\left\langle\rho_{\mathrm{cell}}\right\rangle^{n-1}\left|\Psi^{(n)}\right| . \quad (24)$$

Hence, $T_{\mathrm{UCP}} \propto |\Psi^{(n)}|$. □

Based on Lemma 5, we can estimate the costs of UCP($\Omega,\Psi^{(n)}_{\mathrm{FS}}$) and UCP($\Omega,\Psi^{(n)}_{\mathrm{SC}}$) in terms of $|\Psi^{(n)}_{\mathrm{FS}}|$ as follows.

For each path $\mathbf{p}$ in $\Psi^{(n)}_{\mathrm{FS}}$, there are n-1 consecutive cell-vector pairs ($\mathbf{v}_k$, $\mathbf{v}_{k+1}$) for $0 \le k < n$-1. For a particular cell $\mathbf{q}+\mathbf{v}_k$, there are 27 nearest-neighbor cells $\mathbf{q}+\mathbf{v}_{k+1}$. Thus for n-1 neighbor-search steps, we have the number of paths as

$$\left|\Psi^{(n)}_{\mathrm{FS}}\right| = 27^{n-1} . \quad (25)$$

Substituting Eq. (25) in Eq. (24), the search cost for UCP($\Omega,\Psi^{(n)}_{\mathrm{FS}}$) is obtained as

$$T_{\mathrm{UCP}} = L\left(27\left\langle\rho_{\mathrm{cell}}\right\rangle\right)^{n-1} .$$

Next, we analyze the search cost of UCP($\Omega,\Psi^{(n)}_{\mathrm{SC}}$). Paths in $\Psi^{(n)}_{\mathrm{SC}}$ are only removed in the R-COLLAPSE routine, where redundant paths are collapsed. To estimate $|\Psi^{(n)}_{\mathrm{SC}}|$, we need to find how many path are collapsed. Thus, we first categorize paths in $\Psi^{(n)}_{\mathrm{FS}}$ into collapsible and non-collapsible paths as

$$\Psi^{(n)}_{\mathrm{FS}} = \psi^{\mathrm{FS}}_{\mathrm{collapsible}} \cup \psi^{\mathrm{FS}}_{\mathrm{non\text{-}collapsible}} , \quad (26)$$

where $\psi_{\mathrm{collapsible}}$ and $\psi_{\mathrm{non\text{-}collapsible}}$ are disjoint sets of collapsible and non-collapsible paths, respectively. In the following, we prove that for each path in $\Psi^{(n)}_{\mathrm{FS}}$, there exists a unique path in $\Psi^{(n)}_{\mathrm{FS}}$ that produces the same force set.

LEMMA 6 (REFLECTIVE PATH-TWIN (RPT)). *For an arbitrary cell domain Ω and a path $\mathbf{p}=(\mathbf{v}_0,\ldots,\mathbf{v}_{n-1})\in\Psi^{(n)}_{FS}$, there exists a unique reflective path-twin RPT($\mathbf{p}$) = $\mathbf{p}^{-1}$-$\mathbf{v}_{n-1}\in\Psi^{(n)}_{FS}$, such that $\sigma(\mathbf{p}^{-1})=\sigma$(RPT($\mathbf{p}$)).*

PROOF. For $\mathbf{p}=(\mathbf{v}_0\ldots,\mathbf{v}_{n-1})$, we have $\mathbf{p}^{-1}=(\mathbf{v}_{n-1},\ldots,\mathbf{v}_0)$. From the definition of UCP (Eq. (9)) for the case of a single-path pattern $\{\mathbf{p}\}$ and the undirectionality of n-tuple, it is straightforward to prove that

$$\mathrm{UCP}(\Omega,\{\mathbf{p}\}) = \mathrm{UCP}(\Omega,\{\mathbf{p}^{-1}\}) .$$

For a particular path $\mathbf{p}'=\mathbf{p}^{-1}-\mathbf{v}_{n-1}$, we have

$$\begin{aligned}\mathbf{p}' &= \mathbf{p}^{-1}-\mathbf{v}_{n-1}\\ &= (\mathbf{v}_{n-1}-\mathbf{v}_{n-1},\mathbf{v}_{n-2}-\mathbf{v}_{n-1},\ldots,\mathbf{v}_0-\mathbf{v}_{n-1}) .\\ &= (\mathbf{0},\mathbf{v}_{n-2}-\mathbf{v}_{n-1},\ldots,\mathbf{v}_0-\mathbf{v}_{n-1})\end{aligned}$$

Since the first cell offset of $\mathbf{p}$ is (0,0,0), we have $\mathbf{p}'\in\Psi^{(n)}_{\mathrm{FS}}$ from the generation of $\Psi^{(n)}_{\mathrm{FS}}$ (line 1 in Table 3). Thus, for every path $\mathbf{p}\in\Psi^{(n)}_{\mathrm{FS}}$, there exists a path-twin RPT($\mathbf{p}$) = $\mathbf{p}'=\mathbf{p}^{-1}-\mathbf{v}_{n-1}\in\Psi^{(n)}_{\mathrm{FS}}$.

The a differential representation of $\mathbf{p}'$ is

$$\sigma(\mathbf{p}') = (\mathbf{v}_{n-2} - \mathbf{v}_{n-1} - \mathbf{0}, \ldots, (\mathbf{v}_0 - \mathbf{v}_{n-1}) - (\mathbf{v}_1 - \mathbf{v}_{n-1})) = (\mathbf{v}_{n-2} - \mathbf{v}_{n-1}, \ldots, \mathbf{v}_0 - \mathbf{v}_1),$$

while that of $\mathbf{p}^{-1}$ is

$$\sigma(\mathbf{p}^{-1}) = (\mathbf{v}_{n-2} - \mathbf{v}_{n-1}, \ldots, \mathbf{v}_0 - \mathbf{v}_1).$$

Hence $\sigma(\mathbf{p}') = \sigma(\mathbf{p}^{-1})$. From Lemma 3, this proves the equivalence of $\mathbf{p}$ and its twin RPT($\mathbf{p}$). Next we consider the uniqueness of RPT($\mathbf{p}$). Since all paths in a computation pattern begin with the same cell, they are all inequivalent if the path is directional. The only source of equivalence therefore arises from the undirectionality of paths, which is a direct consequence of the undirectionality of n-tuples stated in section 2.1. The reflective equivalence is two-fold by definition, and it can produce at most one redundant path.□

From Lemmas 3 and 6, there exists a unique path twin that produces the identical force set for every path $\mathbf{p}$. However, if $\mathbf{p} = \mathbf{p}^{-1}$, then RPT($\mathbf{p}$) is $\mathbf{p}$ itself. Thus, it is not collapsible.

COROLLARY 1 (SELF-REFLECTION). *For an arbitrary cell domain Ω and a path $\boldsymbol{p} \in \Psi^{(n)}_{FS}$. If $\boldsymbol{p} = \boldsymbol{p}^{-1}$, RPT($\boldsymbol{p}$) = $\boldsymbol{p}$.*

PROOF. Let $\mathbf{p}$ be $(\mathbf{v}_0,\ldots,\mathbf{v}_{n-1})$. From Lemma 6, RPT($\mathbf{p}$) = $\mathbf{p}^{-1}$ – $\mathbf{v}_{n-1}$. From the assumption, $\mathbf{p} = \mathbf{p}^{-1}$, thus RPT($\mathbf{p}$) = $\mathbf{p} - \mathbf{v}_{n-1}$. Also, $\mathbf{p} = \mathbf{p}^{-1}$ implies $\mathbf{v}_0 = \mathbf{v}_{n-1}$. Since $\mathbf{p} \in \Psi^{(n)}_{\text{FS}}$, $\mathbf{v}_0 = \mathbf{0}$. Hence, RPT($\mathbf{p}$) = $\mathbf{p} - \mathrm{v}_{n-1} = \mathbf{p} - \mathbf{v}_0 = \mathbf{p}$. □

According to the self-reflection in Corollary 1, $\psi_{\text{non-collapsible}} = \{\mathbf{p} \mid \forall \mathbf{p} \in \Psi^{(n)}_{\text{FS}}: \mathbf{p} = \mathbf{p}^{-1}\}$. To estimate $|\psi_{\text{non-collapsible}}|$, we need to count the number of paths that are self-reflective. Note that

$$\mathbf{p} = \mathbf{p}^{-1} = \begin{cases} (\mathbf{v}_0,\ldots,\mathbf{v}_{(n-1)/2-1},\mathbf{v}_{(n-1)/2},\mathbf{v}_{(n-1)/2-1},\ldots,\mathbf{v}_0) & ;n \text{ is odd} \\ (\mathbf{v}_0,\ldots,\mathbf{v}_{n/2-1},\mathbf{v}_{n/2-1},\ldots,\mathbf{v}_0) & ;n \text{ is even} \end{cases},$$

thus, the number of self-reflective (*i.e.* non-collapsible) paths is

$$\left|\psi_{\text{non-collapsible}}\right| = 27^{\left\lfloor \frac{n+1}{2} \right\rfloor - 1}. \tag{27}$$

Substituting Eq. (27) in Eq. (26) yields

$$\begin{aligned} \left|\psi^{\text{FS}}_{\text{collapsible}}\right| &= \left|\Psi^{(n)}_{\text{FS}}\right| - \left|\psi^{\text{FS}}_{\text{non-collapsible}}\right| \\ &= 27^{n-1} - 27^{\left\lfloor \frac{n+1}{2} \right\rfloor - 1} \end{aligned}. \tag{28}$$

Lemma 6 states that there is a unique reflective path-twin of $\mathbf{p}$ for each collapsible path $\mathbf{p}$ such that one of them can be removed. Thus, the number of collapsed paths in SC pattern is half of the collapsible paths in Eq. (28):

$$\begin{aligned} \left|\Psi^{(n)}_{\text{SC}}\right| &= \frac{1}{2}\left|\psi^{\text{FS}}_{\text{collapsible}}\right| + \left|\psi^{\text{FS}}_{\text{non-collapsible}}\right| \\ &= \frac{1}{2}(27^{n-1} - 27^{\left\lfloor \frac{n+1}{2} \right\rfloor - 1}) + 27^{\left\lfloor \frac{n+1}{2} \right\rfloor - 1} \\ &= \frac{1}{2}27^{n-1} + O(27^{n/2}) \end{aligned}. \tag{29}$$

Hence, the search cost of SC is roughly half that of FS for large n.

4.2 Communication Analysis of SC Pattern

In this subsection, we analyze the communication cost of the parallel SC-MD. The communication cost is defined as

$$T_{\text{comm}} = T_{\text{bandwidth}} + T_{\text{latency}}. \tag{30}$$

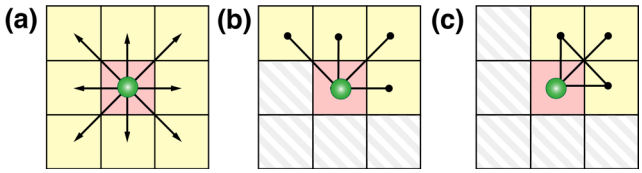

Figure 6. Schematic of shell methods for pair computation: (a) full-shell, (b) half-shell, and (c) eighth-shell patterns.

Here, we assume that the bandwidth cost $T_{\text{bandwidth}}$ is proportional to the amount of imported data. The imported data in parallel MD is proportional to the imported volume V_{import} in subsection 3.1.3 in the average case. On the other hand, the latency cost T_{latency} is proportional to the number of nodes $n_{\text{comm\_node}}$ from which data needs to be imported. Hence, Eq. (30) can be rewritten as

$$T_{\text{comm}} = c_{\text{bandwidth}} V_{\text{import}} + c_{\text{latency}} n_{\text{comm\_nodes}}, \tag{31}$$

where $c_{\text{bandwidth}}$ and c_{latency} are prefactors. To quantify T_{comm} for SC-MD, we first determine the import volume of the SC pattern. For simplicity, we assume that the given cell domain has a cubic shape such that $L_x = L_y = L_z = l$.

From the definition of import volume in Eq. (14), we need to find the union of cell coverage of all cells in the domain. To analyze the cell coverage, we define a layered cell coverage in the range $[a, b]$ (where a and b are nonnegative integers such that $a \le b$) as

$$\mathbf{c}[a,b]_{\mathbf{q}} = \left\{\mathbf{c}(\mathbf{q}+\vec{\delta}) \middle| \forall \delta_\beta \in \{a, a+1, \ldots, b\} \text{ for all } \beta \in \{\mathrm{x},\mathrm{y},\mathrm{z}\}\right\}.$$

In the SC algorithm, the key step that affects cell coverage is the OC-SHIFT subroutine. OC-SHIFT performs 1st-octant compression, in which all computation paths in the pattern are shifted to the first octant of cell coverage (*i.e.* non-negative cell indices along x, y, and z directions relative to the center cell). Thus, the cell coverage $\Pi(\mathbf{c}(\mathbf{q}),\Psi^{(n)}_{\text{SC}})$ for an arbitrary cell $\mathbf{c}(\mathbf{q})$ is

$$\Pi(\mathbf{c}(\mathbf{q}), \Psi^{(n)}_{\text{SC}}) = \mathbf{c}[0, n-1]_{\mathbf{q}}.$$

From Eq. (13), the cell-domain coverage of the SC pattern is

$$\Pi(\Omega, \Psi^{(n)}_{\text{SC}}) = \bigcup_{\forall \mathbf{c}(\mathbf{q}) \in \Omega} \mathbf{c}[0, n-1]_{\mathbf{q}}.$$

Hence, the import volume of the SC pattern reads

$$V_\omega(\Omega, \Psi^{(n)}_{\text{SC}}) = \left| \bigcup_{\forall \mathbf{c}(\mathbf{q}) \in \Omega} \mathbf{c}[0, n-1]_{\mathbf{q}} - \Omega \right|. \tag{32}$$

Since $\Omega \subseteq \Pi(\Omega, \Psi^{(n)}_{\text{SC}})$, the cardinality in the R.H.S. of Eq. (32) can be taken individually. Hence the import volume of $\Psi^{(n)}_{\text{SC}}$ is

$$\begin{aligned} V_\omega(\Omega, \Psi^{(n)}_{\text{SC}}) &= \left| \bigcup_{\forall \mathbf{c}(\mathbf{q}) \in \Omega} \mathbf{c}[0, n-1]_{\mathbf{q}} \right| - |\Omega| \\ &= (l+n-1)^3 - l^3 \end{aligned}. \tag{33}$$

Regarding the latency cost, unlike some methods such as NT [24] that minimize the bandwidth cost at the expense of latency cost, all cell-based methods have small constant latency. In SC-MD, we only need to import atom data from 7 nearest processors using only 3 communication steps via forwarded atom-data routing.

4.3 Relation to Previous Works

Numerous algorithms have been designed to utilize the cell data structure for efficient range-limited n-tuple search. For the case of pair computation ($n = 2$), in particular, shell-based methods—such as full-shell (FS), half-shell (HS), and eighth-shell (ES)—are used extensively for efficient search of pairs in $\Gamma^{*(2)}$. These methods can be described systematically in terms of UCP. In the following subsections, we provide their unified descriptions and compare them with SC.

4.3.1 Full-shell method

FS is the simplest among shell methods and it produces a bounding force set. FS searches all atoms within the nearest-neighbor cells (*i.e.* cells of indices within ±1 in the x, y, z directions) surrounding the center cell [17]. In our algebraic notation, FS is equivalent to $\Psi^{(2)}_{FS}$, and thus is 2-complete according to Lemma 1. However, FS is not optimal in the light of the search cost: $|\Psi^{(2)}_{FS}| = 27$ (see Figure 6(a)). Since FS has reflective redundancy, the extra computational cost is involved for filtering the collapsible pairs.

4.3.2 Half-shell method

HS reduces the search cost of FS using the symmetric property of reflected pairs to eliminate redundant search (see Figure 6(b)) [17]. This amounts to Ψ_{HS} = R-COLLAPSE($\Psi^{(2)}_{FS}$). Thus, the search cost and import volume are reduced by nearly half compared to that of FS, *i.e.*, $|\Psi_{HS}| = 14$.

4.3.3 Eighth-shell method

ES improves over HS by relaxing the owner-compute rule, thereby interacting only with the neighbor cells in the upper-corner octant [23]. This amounts to Ψ_{ES} = OC-SHIFT(Ψ_{HS}) = $\Psi^{(2)}_{SC}$. Hence, ES is a special case of the SC algorithm for $n = 2$. OC-shift reduces the cell footprint of ES to $|\Pi(\Psi_{ES})| = 7$, see Figure 6(c). This in turn reduces the import volume for parallel computation, which is Eq. (33) for $n = 2$.

5. PERFORMANCE BENCHMARKS

In this section, we benchmark the performance of SC-MD and two existing n-tuple computation codes—FS-MD and Hybrid-MD—for a real many-body MD application. Specifically, we consider MD simulation of silica (SiO_2), which involves dynamic pair and triplet ($n = 2$ and 3) computations [4]. In this application, $r_{cut\text{-}3}$ is smaller than $r_{cut\text{-}2}$, *i.e.*, $r_{cut\text{-}3}/r_{cut\text{-}2} \sim 0.47$. FS-MD uses a computation pattern generated from GENERATE-FS in Table 3 without further performing OC-shift and R-collapse. Hybrid-MD is a production code presented in Ref. [12]. In this code, the pair computation is done by constructing a dynamic pair list (called Verlet neighbor list) within UCP for $\Psi^{(2)}_{FS}$. Then, Hybrid-MD exploits the shorter cutoff of the triplet computation by pruning the triplet search directly from the pair list without using the cell data structure with $r_{cut\text{-}3}$. Although this hybrid cell/Verlet-neighbor-list approach reduces the triplet search cost in this particular situation, the import volume is not reduced from that of FS-MD.

Performance evaluations in this section are performed on two platforms: BlueGene/Q at Argonne National Laboratory and an Intel-Xeon cluster at the Center for High Performance Computing and Communication of the University of Southern California (USC-HPCC). The BlueGene/Q consists of 49,152 compute nodes, where each node has 16 PowerPC A2 cores. Each core supports 4 hardware threads, which is clocked at 1.6 GHz. The network topology of BlueGene/Q is a 5D torus. Four MPI tasks are spawn on each core of BlueGene/Q to fully take advantage of integer pipeline and cache-latency hiding in the BlueGene/Q architecture. Details of the BlueGene/Q architecture can be found in Ref. [29]. Tests on the USC-HPCC cluster are performed on dual 6-core processor nodes with 2.33 GHz Intel Xeon X5650 processors and 48 GB memory per node.

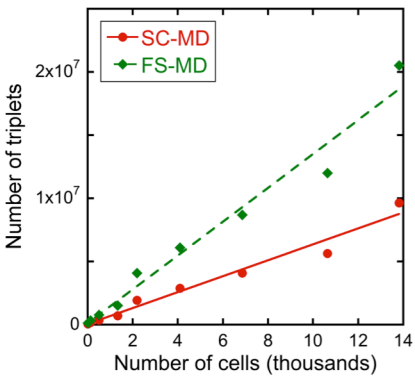

Figure 7. Average number of triplets as a function of domain size. Error bars are too small to be visible in the plot.

5.1 Search Cost of SC and FS Algorithms

According to the analysis in section 4.1, the search cost of SC is much smaller than that of FS (asymptotically half for large n). To confirm this assertion, we measure the actual number of n-tuples in the force set for SC-MD and FS-MD. Figure 7 shows the measured number of triplets per MD step (averaged over 10,000 time steps) as a function of the number of cells, where the average cell density $\langle \rho_{cell} \rangle$ is fixed for each measurement. The plot shows that the triplet count of FS-MD is ~2.13 times of that of SC-MD.

5.2 Fine-Grain Parallelism

We compare the runtime of SC-MD with those of FS-MD and Hybrid-MD to characterize the performance as a function of granularity (*i.e.* the number atoms per core, N/P) on 48-64 nodes for small grains (N/P = 24 – 3,000).

Figure 8(a) shows an average runtime over 10,000 MD steps of the three codes on 48 nodes of the Intel-Xeon cluster. The average grain size is varied from 24 to 3,000 atoms per core. At the smallest grain ($N/P = 24$), the runtime per MD step of SC-MD is much shorter than those of FS-MD and Hybrid-MD—by factors of 10.5 and 9.7, respectively. This is mainly due to the small import volume of SC-MD. Accordingly, SC-MD is faster than FS-MD for all granularities. However, the Hybrid-MD performance relative to that of SC-MD improves gradually as the granularity increases. This can be understood as follow. While SC-MD reduces the import volume as compared to Hybrid MD, Hybrid-MD reduces the triplet search cost by taking advantage of the special cutoff condition. For larger granularities (or larger computation/communication ratios), the advantage of smaller import-volume for SC-MD becomes overshadowed by its larger triplet search cost, and hence Hybrid MD becomes more advantageous. The crossover of performance advantage from SC-MD to Hybrid-MD occurs at the granularity of 2,095 atoms per core.

Figure 8(b) shows an average runtime over 10,000 MD steps of the three codes on 64 nodes of BlueGene/Q. In accordance with the result on the Intel Xeon platform, the finest-grain result of SC-MD shows 5.7- and 5.1-fold speedups over FS-MD and Hybrid-MD, respectively. In this test, the crossover of performance

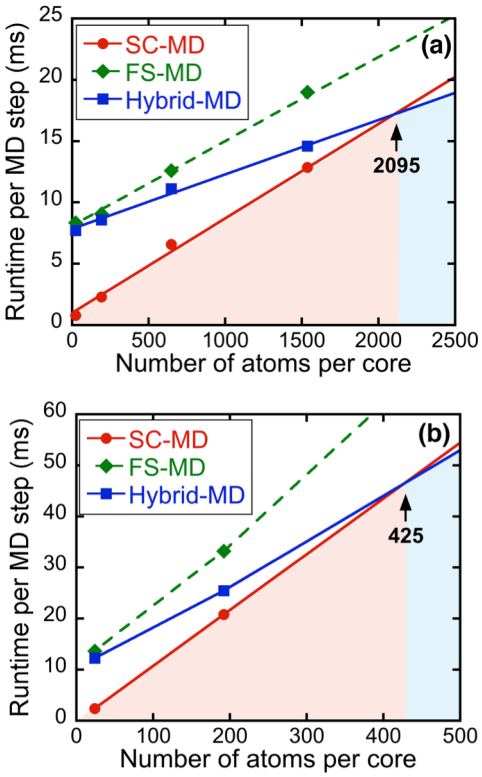

Figure 8. Runtime of SC-MD (red), FS-MD (green), and Hybrid-MD (blue) as a function of the granularity on (a) 48 Intel Xeon nodes and (b) 64 BlueGene/Q nodes. The plot shows that SC-MD is the fastest for $N/P < 2{,}095$ and 425 on Intel Xeon and BlueGene/Q platforms, respectively.

advantage from SC-MD to Hybrid MD on BlueGene/Q is found to be at $N/P = 425$, which is considerably smaller than that on Intel Xeon. This is likely due to the lower computational power per core of BlueGene/Q compared with that of Intel Xeon. Consequently, the benefit of smaller triplet search space for Hybrid-MD is emphasized to shift down the trade-off point between the search cost and import-volume size.

5.3 Strong-Scaling Benchmark

In this subsection, we perform a strong-scaling benchmark of SC-MD, FS-MD, and Hybrid-MD on 1–512 BlueGene/Q nodes (*i.e.* 16–8,192 cores) and on 1–64 Intel Xeon nodes (*i.e.* 12–768 cores). Strong-scaling speedup is defined as

$$S_{\text{strong}} = \frac{T_{\text{reference}}}{T_{\text{parallel}}}, \tag{34}$$

where $T_{\text{reference}}$ denotes the time spent on a reference test (*i.e.* the timing result on a single node) and T_{parallel} is the time spent on a larger parallel run to solve the same problem. The corresponding parallel efficiency is $\eta_{\text{strong}} = S_{\text{strong}}/(P_{\text{parallel}}/P_{\text{reference}})$, where P_{parallel} and $P_{\text{reference}}$ are the numbers of cores used in the parallel and reference runs, respectively. The total number of atoms used in the benchmark is fixed at 0.88 and 0.79 millions atoms on Intel-Xeon and BlueGene/Q platforms, respectively. Atoms in both systems are uniformly distributed.

Figure 9(a) shows the strong-scaling speedup of SC-MD, FS-MD, and Hybrid-MD as a function of the number of cores on the Intel-Xeon cluster. The plot shows that SC-MD achieves excellent

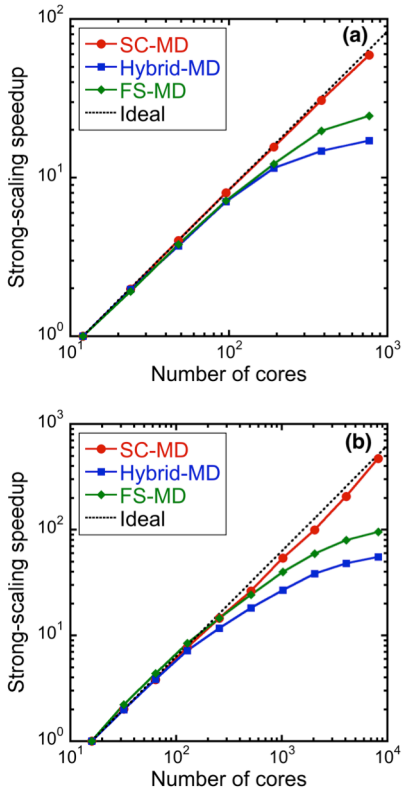

Figure 9. Strong scaling speedup of SC-MD, Hybrid-MD, and FS-MD on (a) Intel Xeon cluster and (b) BlueGene/Q.

scalability from 1–64 nodes with a 59.3-fold speedup (or 92.6% parallel efficiency) on 768 Intel Xeon cores. On the other hand, the scalability of FS-MD and Hybrid-MD decline after the number of cores exceeds 96, so that S_{strong} becomes 24.5 and 17.1 on 768 cores, respectively. The corresponding parallel efficiencies are 38.3% and 26.8%. The number of cores and system size used in this benchmark are in an affordable range for general scientists using commodity clusters.

The second benchmark measures strong scalabilities of the three codes on BlueGene/Q. Similarly to the previous benchmark, the results in Figure 9(b) indicate that SC-MD maintains excellent strong scalability with $S_{\text{strong}} = 465.6$ (or $\eta_{\text{strong}} = 90.9\%$) on 8,192 cores. This demonstrates an excellent strong scalability of SC-MD on larger platforms. On the other hand, FS-MD and Hybrid-MD maintain decent parallel efficiency only on small numbers of nodes, where 7.1- and 7.0-fold speedup is observed on 8 nodes. After that, their scalabilities decrease gradually, and only 55.1- and 95.2-fold speedups (10.8% and 18.6% efficiencies) are observed on 8,192 cores ($N/P \sim 100$ or ~26 atoms per MPI task).

To confirm that SC-MD scales on extreme-scale clusters, we perform a larger strong-scaling benchmark for a 50.3 million-atom system involving up to 32,768 BlueGene/Q nodes (or 2,097,152 MPI tasks on 524,288 cores for the largest measurement). The result shows an excellent speedup $S_{\text{strong}} = 3{,}764.6$ (or 91.9% parallel efficiency) on 524,288 cores compared to the reference timing of 128-core run on 8 BlueGene/Q nodes.

6. CONCLUSION

We have developed a computation-pattern algebraic framework to formalize dynamic *n*-tuple computation in many-body MD. This new formalism allows us to perform systematically and

mathematically proven analysis of dynamic range-limited *n*-tuple MD, which to the best of our knowledge, has not been done before. The new formulation and analysis have led to the development of the SC algorithm, which generalizes some of the best pair ($n = 2$) computation algorithms to an arbitrary n. Benchmark tests have shown that SC-MD outperforms our production Hybrid-MD code for fine granularities. Excellent strong scalability has also been observed for SC-MD. The results demonstrate the advantage of SC-MD over Hybrid-MD when the time-to-solution (rather than simulating the largest possible system size) is the major goal.

There is an additional benefit of SC-MD compared with Hybrid-MD that combines cell and Verlet-neighbor-list methods. Namely, SC exposes maximal concurrency on heterogeneous architectures such as GPU-accelerated and many-core clusters. Since SC executes different *n*-tuple computations independently, they can be assigned to different hardware (*e.g.* multiples GPUs). In contrast, Hybrid-MD has a sequential dependence, *i.e.*, the Verlet-neighbor list must be constructed within the pair computation before any $n > 2$ computation can be performed. Another issue is the cell size. Though we have restricted ourselves to the cell size larger than $r_{\text{cut-}n}$ for simplicity, it is straightforward to generalize the SC algorithm to a cell size less than $r_{\text{cut-}n}$ as was done, *e.g.*, in the midpoint method [30]. In this case, the SC algorithm improves the midpoint method by further eliminating redundant searches. Relative advantages between ES and midpoint methods have been thoroughly discussed by Hess *et al*. [26].

7. ACKNOWLEDGMENTS

This work was partially supported by DOE-BES/EFRC/INCITE, NSF-CDI/PetaApps, and ONR.

8. REFERENCES

[1] A. Rahman, "Correlations in the motion of atoms in liquid argon," *Physical Review,* vol. 136, pp. A405-A411, Oct 1964.

[2] J. A. Mccammon, B. R. Gelin, and M. Karplus, "Dynamics of folded proteins," *Nature,* vol. 267, pp. 585-590, Jun 1977.

[3] F. H. Stillinger and T. A. Weber, "Dynamics of structural transitions in liquids," *Physical Review A,* vol. 28, pp. 2408-2416, Oct 1983.

[4] P. Vashishta, R. K. Kalia, J. P. Rino, and I. Ebbsjo, "Interaction potential for SiO_2 - a molecular-dynamics study of structural correlations," *Physical Review B,* vol. 41, pp. 12197-12209, Jun 1990.

[5] S. B. Sinnott and D. W. Brenner, "Three decades of many-body potentials in materials research," *MRS Bulletin,* vol. 37, pp. 469-473, May 2012.

[6] S. J. Plimpton and A. P. Thompson, "Computational aspects of many-body potentials," *MRS Bulletin,* vol. 37, pp. 513-521, May 2012.

[7] A. C. T. van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard, "ReaxFF: a reactive force field for hydrocarbons," *Journal of Physical Chemistry A,* vol. 105, pp. 9396-9409, Oct 2001.

[8] A. Nakano, R. K. Kalia, K. Nomura, A. Sharma, P. Vashishta, F. Shimojo, A. C. T. van Duin, W. A. Goddard, R. Biswas, D. Srivastava, and L. H. Yang, "De novo ultrascale atomistic simulations on high-end parallel supercomputers," *International Journal of High Performance Computing Applications,* vol. 22, pp. 113-128, Feb 2008.

[9] K. Nomura, R. K. Kalia, A. Nakano, and P. Vashishta, "A scalable parallel algorithm for large-scale reactive force-field molecular dynamics simulations," *Computer Physics Communications,* vol. 178, pp. 73-87, Jan 2008.

[10] P. S. Lomdahl, P. Tamayo, N. Gronbech-Jensen, and D. M. Beazley, "50 Gflops molecular dynamics on the CM-5," *Proceedings of Supercomputing (SC93),* ACM/IEEE, 1993.

[11] T. Narumi, R. Susukita, T. Koishi, K. Yasuoka, H. Furusawa, A. Kawai, and T. Ebisuzaki, "1.34 Tflops molecular dynamics simulation for NaCl with a special-purpose computer: MDM," *Proceedings of Supercomputing (SC00),* ACM/IEEE, 2000.

[12] A. Nakano, R. K. Kalia, P. Vashishta, T. J. Campbell, S. Ogata, F. Shimojo, and S. Saini, "Scalable atomistic simulation algorithms for materials research," *Proceedings of Supercomputing (SC01),* ACM/IEEE, 2001.

[13] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale, "NAMD: biomolecular simulation on thousands of processors," *Proceedings of Supercomputing (SC02),* ACM/IEEE, 2002.

[14] T. C. Germann, K. Kadau, and P. S. Lomdahl, "25 Tflop/s multibillion-atom molecular dynamics simulations and visualization/analysis on BlueGene/L," *Proceedings of Supercomputing (SC05),* ACM/IEEE, 2005.

[15] J. N. Glosli, D. F. Richards, K. J. Caspersen, R. E. Rudd, J. A. Gunnels, and F. H. Streitz, "Extending stability beyond CPU millennium: a micron-scale atomistic simulation of Kelvin-Helmholtz instability," *Proceedings of Supercomputing (SC07),* ACM/IEEE, 2007.

[16] D. E. Shaw, R. O. Dror, J. K. Salmon, J. P. Grossman, K. M. Mackenzie, J. A. Bank, C. Young, M. M. Deneroff, B. Batson, K. J. Bowers, E. Chow, M. P. Eastwood, D. J. Ierardi, J. L. Klepeis, J. S. Kuskin, R. H. Larson, K. Lindorff-Larsen, P. Maragakis, M. A. Moraes, S. Piana, Y. Shan, and B. Towles, "Millisecond-scale molecular dynamics simulations on Anton," *Proceedings of Supercomputing (SC09),* ACM/IEEE, 2009.

[17] D. C. Rapaport, "Large-scale molecular-dynamics simulation using vector and parallel computers," *Computer Physics Reports,* vol. 9, pp. 1-53, Dec 1988.

[18] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics,* vol. 73, pp. 325-348, Dec 1987.

[19] A. Nakano, R. K. Kalia, and P. Vashishta, "Multiresolution molecular-dynamics algorithm for realistic materials modeling on parallel computers," *Computer Physics Communications,* vol. 83, pp. 197-214, Dec 1994.

[20] S. Ogata, T. J. Campbell, R. K. Kalia, A. Nakano, P. Vashishta, and S. Vemparala, "Scalable and portable implementation of the fast multipole method on parallel computers," *Computer Physics Communications,* vol. 153, pp. 445-461, Jul 2003.

[21] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics,* vol. 117, pp. 1-19, Mar 1995.

[22] L. Kale, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, "NAMD2: greater scalability for parallel molecular dynamics," *Journal of Computational Physics,* vol. 151, pp. 283-312, May 1999.

[23] K. J. Bowers, R. O. Dror, and D. E. Shaw, "Zonal methods for the parallel execution of range-limited N-body simulations," *Journal of Computational Physics,* vol. 221, pp. 303-329, Jan 2007.

[24] D. E. Shaw, "A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions,"

Journal of Computational Chemistry, vol. 26, pp. 1318-1328, Oct 2005.

[25] M. Snir, "A note on N-body computations with cutoffs," *Theory of Computing Systems,* vol. 37, pp. 295-318, Mar-Apr 2004.

[26] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation," *Journal of Chemical Theory and Computation,* vol. 4, pp. 435-447, Mar 2008.

[27] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," *Proceedings of Supercomputing (SC08),* ACM/IEEE, 2008.

[28] H. Dursun, K. Nomura, L. Peng, R. Seymour, W. Wang, R. K. Kalia, A. Nakano, and P. Vashishta, "A multilevel parallelization framework for high-order stencil computations," *Proceedings of the International European Conference on Parallel and Distributed Computing (Euro-Par 2009),* 2009.

[29] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L. T. Chiu, P. A. Boyle, N. H. Chist, and K. Changhoan, "The IBM Blue Gene/Q Compute Chip," *IEEE Micro,* vol. 32, pp. 48-60, Mar/Apr 2012.

[30] K. J. Bowers, R. O. Dror, and D. E. Shaw, "The midpoint method for parallelization of particle simulations," *Journal of Chemical Physics,* vol. 124, p. 184109, May 2006.