A Computation- and Communication-Optimal Parallel Direct 3-Body Algorithm

Article · January 2015

DOI: 10.1109/SC.2014.35

Penporn Koanantakool
University of California, Berkeley

Katherine Yelick
University of California at Berkeley and Lawrence Berkeley National Laboratory

A Computation- and Communication-Optimal Parallel Direct 3-Body Algorithm

Penporn Koanantakool*† and Katherine Yelick*†
*Computer Science Division, University of California, Berkeley
†Lawrence Berkeley National Laboratory, Berkeley, CA
{penpornk,yelick}@cs.berkeley.edu

***Abstract*—Traditional particle simulation methods are used to calculate pairwise potentials, but some problems require 3-body potentials that calculate over triplets of particles. A direct calculation of 3-body interactions involves $O(n^3)$ interactions, but has significant redundant computations that occur in a nested loop formulation. In this paper we explore algorithms for 3-body computations that simultaneously optimize three criteria: computation minimization through symmetries, communication optimality, and load balancing. We present a new 3-body algorithm that is both communication and computation optimal. Its optional replication factor, c, saves c^3 in latency (number of messages) and c^2 in bandwidth (volume), with bounded load-imbalance. We also consider the k-body case and discuss an algorithm that is optimal if there is a cutoff distance of less than 1/3 of the domain. The 3-body algorithm demonstrates 99% efficiency on tens of thousands of cores, showing strong scaling properties with order of magnitude speedups over the naïve algorithm.**

Keywords*—parallel algorithms; communication-avoiding algorithms; particle methods; n-body;

I. Introduction

Particle simulations are used in various fields such as physics, chemistry, and material science. Traditionally one computes just pairwise interactions between particles, but there is increasing interest in calculations that require many-body, especially 3-body, interactions. The parallel 3-body calculation is a relatively unexplored problem, and most of the existing algorithms compute direct interactions rather than a tree-based approach as is common in the 2-body case. One shared concern among these methods is avoiding redundant computation. While the factor of 2 in the 2-body case (computing both the force f_{ij} of particle i on particle j as well as $f_{ji} = -f_{ij}$) is sometimes ignored, a triplet appears six times in the 3-body $O(n^3)$ interaction space. A good algorithm should only compute all of its related forces once, taking advantage of force symmetry. Multiple efforts have been put into avoiding redundancy and balancing work over processors, but none of these consider the communication complexity of the algorithms. Even though the problem is compute-intensive, communication takes time and energy and still comes into play when scaling aggressively. Our goal is to develop algorithms that are optimal in both computation (exploitation of symmetries) and communication.

Here, we present two communication-avoiding algorithms that are both computation and communication optimal, one for all-triplets interactions and the other for interactions with cutoff distance. We derive lower bounds for communication along the critical path and prove that both algorithms achieve their bounds. ‘Communication-avoiding’ means they can exploit extra memory to reduce the lower bounds; by making c replicas, they can reduce by factors of c^3 latency (number of messages) and c^2 bandwidth (number of words). Experimental results of the all-triplets algorithms on two large scale machines are also presented.

Our contributions are:

- Derivation of the communication lower bounds for k-body methods $(k \geq 2)$ with or without cutoff.
- A new *all triplets* algorithm for 3-body calculations without cutoff that is provably optimal in computation and communication and has provable less load imbalance than previous work.
- A new class of algorithms for any k-body interaction $(k \geq 2)$ with cutoff, where the cutoff limits interactions to less than 1/3 of the domain. These algorithms are also provably optimal in communication and computation.
- Application of replication to further avoid communication asymptotically.
- An implementation of the 3-body algorithm for massively parallel machines, including support for hybrid (shared and distributed memory) parallelism.
- Performance results showing near perfect strong scaling on tens of thousands of cores and the tradeoff between communication and load imbalance of the communication-avoiding technique

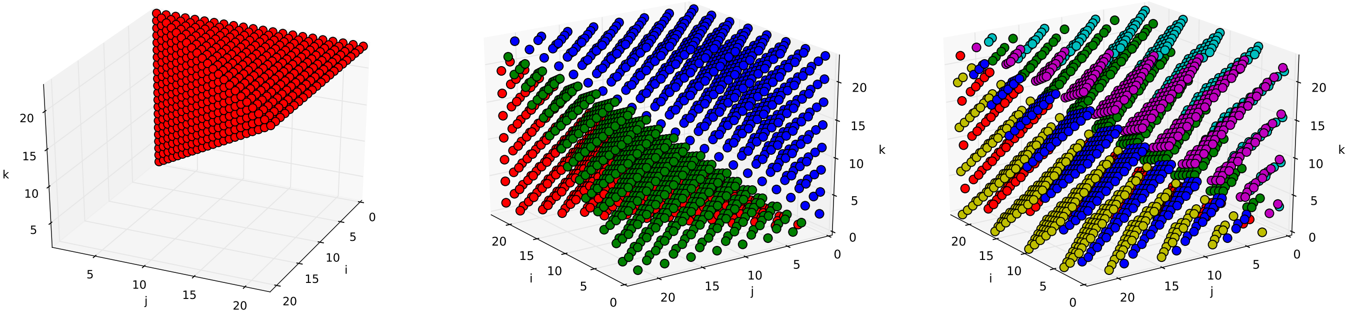
This paper is organized as follows. Section II describes many-body interactions in Molecular Dynamics, previous 3-body algorithms, and derives communication lower bounds for the all-triplets problem. Section III gives the all-triplets algorithm and extends it to support replication. Section IV shows performance results of the all-triplets algorithm. Section V adds cutoff, generalizes to any k-body interaction, and proves the communication lower bounds and optimality. Section VI concludes and discusses future work.

II. Background and Previous Work

A. Many-Body Interactions in Molecular Dynamics

Molecular Dynamics (MD) calculations compute forces acting on particles to simulate their movements over time. The

SC14, November 16-21, 2014, New Orleans



(a) Tetrahedron $(i < j < k)$

(b) Slice Symmetric Distribution

(c) Volume Symmetric Distribution

Fig. 1: Illustration of the elements to compute in the force cubes in Li et al. [1], [2], [3] and Sumanth et al. [4] ($n = 24$ particles)

total force on a particle i is computed from the equation,

$$f_i = -\nabla(\phi_1(i) + \sum_j \phi_2(i,j) + \sum_{j,k} \phi_3(i,j,k) + \ldots), \quad (1)$$

where ∇ is the gradient operator and ϕ_k is the k-body potential energy function. The effect of k-body potentials is generally lower as k increases, so the 2-body potentials have the largest contribution. Although MD applications have traditionally focused on pairwise interactions, there is interest more recently on force potentials involving many-body interactions.

Many-body potentials are significant in both atomic systems and molecular systems [5]. They enable more accurate modeling of specific materials such as metals and ceramics [6]. 4-body potentials are used in protein folding [7], [8]. 3-body potentials are the most commonly used after 2-body potentials, and are required to capture some properties such as phase equilibria in noble gases [9] and the second virial coefficient in systems such as water [10], [11].

The parallel 3-body problem poses a complicated symmetry challenge. It is trivial to exploit the force symmetry $f_{ij} = -f_{ji}$ in the 2-body problem, and the cost of ignoring symmetry is only a factor of two. However, in the 3-body problem, there are $n(n-1)(n-2)/6$ unique triplets, so there are roughly 6 times more triplet interactions if we do not eliminate redundant triplets. At $O(n^3)$ scale, this is costly. It is even worse for larger values of k as the symmetry factor grows with k!. As described in the next section, previous work focused on conditions to avoid redundant triplets and work partitioning to maintain load balance; we add the additional constraint of minimizing communication.

B. *3-Body Algorithms*

There are two approaches to computing 2-body interactions: direct and approximate. Direct approaches compute all required $O(n^2)$ interactions. Approximate or tree-based approaches treat a group of sufficiently far away particles as one big particle and compute significantly fewer interactions. For example, the Barnes-Hut algorithm [12] computes $O(n \log n)$ interactions and the Fast Multipole Method (FMM) [13] computes $O(n)$ interactions. Likewise, there are existing 3-body algorithms based on both direct and approximate approaches.

The work on the 3-body problem has begun rather recently. Nakano et al. [14] proposed a domain decomposition algorithm with multiple-time-step (MTS) to compute 2- and 3-body forces with cutoff distance in 1993. They used a separable 3-body calculation, i.e., decomposing a 3-body force calculation (i, j, k) to two 2-body interactions (i, j) and (i, k).

Li et al. [1], [2], [3] extended the 2-body's n-by-n *force matrix* to an n-by-n-by-n *force cube*, as shown in Figure 1a. The goal is to compute only unique triplets (i, j, k) where $i < j < k$, shown in red circles forming a tetrahedron. They presented four ways to assign work to processors: Force, Cyclic, Balanced Cyclic, and Precise Decompositions. Force Decomposition (FD) [2] partitions the force cube into subcubes, prunes the redundant subcubes from the job list, then assigns each subcube to a processor. Since not all subcubes have the same computation load – subcubes (i, j, k) where $i \neq j \neq k$ have the most load and $i = j = k$ the least, – the FD algorithm suffered the most load imbalance among all the algorithms they considered.

Cyclic Decomposition (CD) [1], [2], [3] slices the force cube into n planes and assigns planes cyclically to processors (processor r gets planes $r, r + p, r + 2p$, etc.). It still incurs a slight load imbalance as processors that are assigned the earlier planes always have less load than processors that get later planes. This led to Balanced Cyclic Decomposition (BD) [2], [3], which cyclically assigns odd planes first, then cyclically assigns even planes in reverse order to balance out excess load.

Precise decomposition (PD) [2] counts exactly how many elements are to be computed in each plane and assigns consecutive planes that have closest to perfect load to each processor. All of the latter three algorithms performed marginally better than FD, with PD slightly better than the other two. Still, none of them achieved high parallel efficiency. PD, BD, and CD were at 50% parallel efficiency while FD was at 40% on a system with only 35 processors.

Also based on the force cube, Sumanth et al. [4] exploited the symmetry to *transform* the force cube so that there is an equal number of unique elements to compute in each plane (Slice Symmetric) or volume space (Volume Symmetric). *Transforming* means determining if an element should be computed based on arithmetic conditions of its index (i, j, k). Figures 1b and 1c illustrate Slice Symmetric and Volume

Symmetric transformations. Circles indicate force elements to be computed. Colors are for illustration purpose only. This way, the load is optimally balanced no matter how the planes are assigned to the processors as long as they get the same number of planes or subcubes, and thus they can be assigned consecutively. Nevertheless, very few performance results are presented, most of which focused on load balance. The communication is not likely to scale well given that each plane involves all particles. The algorithms from both Li et al. and Sumanth et al. are $O(n^3)$ direct-interactions methods. They stated that cutoff distance can be supported but did not provide details.

Recently, Kunaseth et al. [15] proposed a systematic way to compute k-body interactions with cutoff (k-tuple computation in their terminology). Their *Shift-Collapse* algorithm uses a cell-based method, starting with iteratively generating all the cell domains the k-tuple could be in for each cell, then pruning out the cells that will contribute to redundant k-tuples. Despite the excessive work of generating all possible k-level neighbor cell combinations, their algorithm performed very well at large scale.

By extending their existing *multi-tree methods* framework for *generalized N-body problems* [16] in statistics, Lee et al. introduced for the first time approximation algorithms to compute any k-body interactions called the Multibody Multipole Methods [17]. The algorithms store particles in a k-d tree and only support multibody potentials that can be factorized as products of functions of pairwise Euclidean distances. An extension of FMM-like algorithms [13] is first discussed. They derived far- and near-field expansions for 3-body potentials but not for general multibody case. Instead, they opted for a simpler, nondeterministic algorithm with only a monopole approximation for monotonically decreasing k-body potentials. The algorithm uses Monte Carlo sampling based on the central limit theorem to guarantee the potential sum is within an error margin with probability $1 - \alpha$. The benefits of both algorithms are most prominent in clustered particle distribution in experiment. At 10,000 particles on a single core, the deterministic approach (FMM-like) achieved 20 times speedup against a naive algorithm and the Monte Carlo approach, with 90% probability guarantee that the relative error will be less than 0.1, gained a huge speedup of 1,000 times. All work is sequential. They did not provide parallelization but mentioned that it is underway.

C. Communication Lower Bounds

Interprocessor communication is more costly than computation in both time and energy and is often a major obstacle to application scalability. Overlapping with computation can only provide constant (up to 2×) speedup and does not help with energy consumption. As the gap between communication and computation time continues to grow [18], along with the concerns for energy [19], it is important to minimize communication.

Early work on communication lower bounds includes Hong and Kung's pebble game [20] for sequential algorithms and Irony et al.'s geometric approach for both sequential and parallel matrix multiplication [21], where they also consider data replication. Ballard et al. [22] extended the lower bounds to cover all direct-linear-algebra-like applications. Recently, Christ et al. [23] gave generalized bounds that cover general loop nests that reference arrays, subject to constraints on the type of subscripts, but general enough to include the k-body problem discussed here, which we explain further below. Solomonik et al. [24] explored tradeoffs between these communication and computation bounds in linear algebra applications with dependencies.

In 2013, Driscoll et al. [25] proved the communication lower bounds for the 2-body problem with and without cutoff and provided algorithms that use replication to avoid communication and successfully achieves the bounds. However, they did not utilize force symmetry which would save a factor of two in computation time.

We follow the framework in [23] to derive the lower bounds for the 3-body problem. If Z is the total number of arithmetic operations a processor has to perform to solve a problem and F is the maximum number of useful operations that can be performed using M words in memory, the lower bounds of the total number of messages S (latency), and words W (bandwidth) a processor must move along the critical path are:

$$S = \Omega\left(\frac{Z}{F}\right), \qquad W = \Omega\left(\frac{Z \cdot M}{F}\right). \tag{2}$$

Let n be the total number of particles in the system and p be the number of processors. The direct 3-body problem requires $O(n^3)$ interactions, so each processor has to do $Z = O(n^3/p)$ work. If each processor stores M words, it can perform at most $F = O(M^3)$ useful work. Thus, the lower bounds for the 3-body problem are:

$$S = \Omega\left(\frac{n^3/p}{M^3}\right), \qquad W = \Omega\left(\frac{n^3/p}{M^2}\right). \tag{3}$$

Normally each processor stores n/p particles. We would like to know how the bounds will change if we store c times more particles, so we write M in the form cn/p:

$$S = \Omega\left(\frac{p^2}{c^3}\right), \qquad W = \Omega\left(\frac{np}{c^2}\right). \tag{4}$$

We call c the replication factor because each group of cn/p particles has c copies among all processors. Since c is in the denominators of both S and W, this implies that increasing c decreases the lower bounds in both latency and bandwidth. However, there is a limit to c. Regardless of the algorithm, each processor has to send at least $O(1)$ messages, e.g., sending out results. Setting $S = O(1)$, we have the upper bound:

$$c \leq p^{2/3}. \tag{5}$$

Substituting this back to Equation (4) results in the memory-independent lower bounds:

$$S = \Omega\left(1\right), \qquad W = \Omega\left(\frac{n}{p^{1/3}}\right). \tag{6}$$

Algorithm 1 NAÏVE ALL-TRIPLETS ALGORITHM

```
**Input** $P_{rank}$ in buffer $b_0$, with all forces reset to 0.
**Output** The same $n/p$ particles updated in-place.
Copy $b_0$ to $b_1$ and $b_2$.
**for** $p$ steps **do**
    **for** $p$ steps **do**
        **if** have not processed this triplet **then**
            Calculate all interactions between three buffers
        **end if**
        $shift\_right(b_2)$
    **end for**
    $shift\_right(b_1)$
**end for**
Send particles in each buffer back to the owner processor
Receive up to 3 copies of my own particles
Sum the forces over all 3 copies
Move my particles according to their calculated forces
```

III. ALGORITHM

In this section, we present a naïve approach to computing all-triplets interactions, then refine it to only form unique triplets. In addition, we also present two variants of the all-unique-triplets algorithm that improve the communication cost. The first variant saves a constant factor of communication, while the second variant is the communication avoiding algorithm which replicates particles to reduce the communication asymptotically.

Previous work [1], [2], [4] on long-range interactions partitions work geometrically based on the force cube. Our approach can instead be seen as extending the all-pairs algorithm in [25] or the systolic/ring algorithm in [26] by adding an additional buffer. Denote the set of all particles in the system with P and the i^{th} of p processors by $proc_i$. We divide P into p equal disjoint subsets of n/p particles, $P_0, P_1, \ldots, P_{p-1}$. Processor $proc_i$ will own and be in charge of updating the particles in P_i, i.e., keeping track of forces applied to them and moving them accordingly in each timestep. Each processor has three buffers, b_0, b_1, and b_2, to hold the particle subsets $(P_i, P_j, P_k), 0 \leq i, j, k < p$.[1] Particle interactions are computed by forming all triplets using one particle from each buffer, calculating forces for all three particles in each triplet at once, and accumulating the forces within each particle. The algorithm alternates between computing interactions and exchanging buffers with other processors. The problem now turns into how to move particles around to cover all triplets of the particle subsets $P_0, P_1, \ldots, P_{p-1}$.

A. Naïve all-triplets algorithm

We arrange the processors into a virtual ring with left and right neighbors of $proc_i$ being $proc_{i-1}$ and $proc_{i+1}$, respectively. Cyclic indexing is used so that $proc_i = proc_{i\%p}$. Let the $shift\_right(b_i)$ operation pass the particles in buffer b_i to the right neighbor's b_i and replace them with the ones from the left neighbor's b_i. If each processor shifts b_2 to the right p^2 times, shifts b_1 every pth time b_2 is shifted, and computes interactions every time a buffer is shifted if they have not already been computed, we will get all necessary interactions from all possible triplets, albeit at the cost of abundant shifting. The pseudocode for one timestep is shown in Algorithm 1 in a SPMD fashion with variable $rank$ denoting the processor's rank.

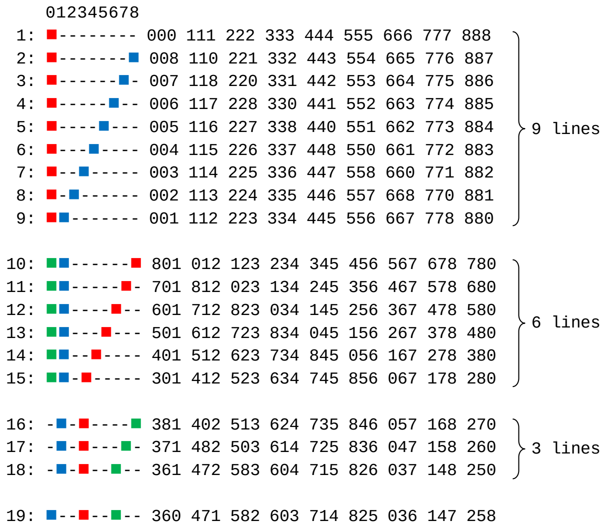
Despite its inefficiency, it is worth noting that Algorithm 1 is still asymptotically communication optimal. It shifts $O(n/p)$ words for $O(p^2)$ rounds with at most 2 messages per round, therefore it sends $O(p^2)$ messages and uses $O(np)$ bandwidth, achieving the bounds in equation (4) ($c = 1$).

B. All-unique-triplets algorithm

The *shift* operation has a nice uniqueness property. Provided that all processors start with different triplets of particle subsets in their buffers, they will still have different triplets from each other after a series of *shift*s on any buffer. Mathematically, $proc_i$ has $(P_{i+x}, P_{i+y}, P_{i+z})$ where $x, y, z \in \mathbb{Z}$ and $0 \leq x, y, z < p$. The only exception is when $3 \mid p$ and x, y, z are $0+q, p/3+q$, and $2p/3+q$ (in any order), where $q \in \mathbb{Z}$ is a constant offset. Algorithm 2 changes the shifting sequence in Algorithm 1 so that any triplet (P_i, P_j, P_k) only occurs once during the whole shifting process.

The new shifting scheme is surprisingly simple: pick a buffer, shift it p times, shift another buffer $p-3$ times, shift the last buffer $p-6$ times, then back to shifting the first buffer picked $p-9$ times, keep picking buffers in a round-robin manner and shift until the number of times to shift is $p\%3$. If p is a multiple of 3, one additional round is required.

Figure 2 shows the complete system status of the algorithm for $p = 9$ processors. Nine column groups on the right side represents the nine processors. The three columns in each group list what particle subsets (the number i of P_i) are in b_0, b_1, and b_2, respectively. On the left are boxes and dashes



```
    012345678
 1: 🟥-------- 000 111 222 333 444 555 666 777 888  ⎫
 2: 🟥-------🟦 008 110 221 332 443 554 665 776 887  ⎪
 3: 🟥------🟦- 007 118 220 331 442 553 664 775 886  ⎪
 4: 🟥-----🟦-- 006 117 228 330 441 552 663 774 885  ⎪
 5: 🟥----🟦--- 005 116 227 338 440 551 662 773 884  ⎬ 9 lines
 6: 🟥---🟦---- 004 115 226 337 448 550 661 772 883  ⎪
 7: 🟥--🟦----- 003 114 225 336 447 558 660 771 882  ⎪
 8: 🟥-🟦------ 002 113 224 335 446 557 668 770 881  ⎪
 9: 🟥🟦------- 001 112 223 334 445 556 667 778 880  ⎭

10: 🟩🟦------🟥 801 012 123 234 345 456 567 678 780  ⎫
11: 🟩🟦-----🟥- 701 812 023 134 245 356 467 578 680  ⎪
12: 🟩🟦----🟥-- 601 712 823 034 145 256 367 478 580  ⎬ 6 lines
13: 🟩🟦---🟥--- 501 612 723 834 045 156 267 378 480  ⎪
14: 🟩🟦--🟥---- 401 512 623 734 845 056 167 278 380  ⎪
15: 🟩🟦-🟥----- 301 412 523 634 745 856 067 178 280  ⎭

16: -🟦-🟥----🟩 381 402 513 624 735 846 057 168 270  ⎫
17: -🟦-🟥---🟩- 371 482 503 614 725 836 047 158 260  ⎬ 3 lines
18: -🟦-🟥--🟩-- 361 472 583 604 715 826 037 148 250  ⎭

19: 🟦--🟥--🟩-- 360 471 582 603 714 825 036 147 258
```

Fig. 2: Illustration of the all-unique-triplets algorithm. Each line shows what particle subsets are in what buffers in each round. In the first nine steps, the red and green boxes are both in processor 0.

[1] This is not to be confused with replication. There are always 3 buffers for the 3-body problem and we treat this factor of 3 as a constant, i.e., each processor stores $O(n/p)$ and does not replicate.

Algorithm 2 ALL-UNIQUE-TRIPLETS ALGORITHM

```
1:  Input $P_{rank}$ in buffer $b_0$, with all forces reset to 0.
2:  Output The same $n/p$ particles updated in-place.
3:  $i \leftarrow 2$                        // The 2nd buffer is picked
4:  Copy $b_0$ to $b_1$ and $b_2$.
5:  for $s \in \{p, p-3, p-6, ..., p\%3\}$ do
6:      for $s$ steps do
7:          if not the first step or $s \neq p$ then
8:              $shift\_right(b_i)$
9:          end if
10:         Calculate all interactions between three buffers
11:     end for
12:     $i \leftarrow (i+1)\%3$
13: end for
14: if $3 \mid p$ then                   // Special case: additional round
15:     $shift\_right(b_i)$
16:     Calculate one third of the interactions based on $\lfloor \frac{rank}{p/3} \rfloor$
17: end if
18: Send particles in each buffer back to the owner processors
19: Receive up to 3 copies of my own particles
20: Sum the forces over all 3 copies
21: Move my particles according to their calculated forces
```

depicting (P_i, P_j, P_k) that are in buffers b_0, b_1, and b_2 of $proc_0$. Red, green, and blue boxes represent b_0, b_1, and b_2. Dash means not in the buffer. This helps illustrate the offset patterns (x, y, z) of the particle subsets that all processors are forming.

In the first line, each processor calculates interactions between its own particles. In lines 2-9, it calculates interactions between two of its own particles and those of successive left neighbors, starting with the immediate left and moving further left cyclically until reaching its right neighbor. Then it switches to shifting the red buffer (b_0) to avoid getting a repeat offset pattern. During the next $9-3=6$ lines, interactions between itself and its right neighbor are calculated with the moving buffer, starting from its left neighbor and stopping short before reaching the right neighbor to avoid getting the same offset pattern as line 10 (recall that the permutation does not matter).

Then, the green buffer is shifted 3 times to compute interactions of particle subsets that are at least one neighbor apart. Again, it has to stop at line 18 or it will generate the same offset patterns as line 16. Finally, the blue buffer is shifted to cover the pattern with equally spaced boxes, which is the special case when p is divisible by 3. Notice that $proc_i$ has the same triplet of particle subsets as those of $proc_{i+p/3}$ and $proc_{i+2p/3}$. To maintain load balance, let each third of the processors calculate each third of the interactions in this case. ($proc_i$ calculates the $\lfloor \frac{i}{p/3} \rfloor^{\text{th}}$ third of interactions.) After computing all interactions, the particles are sent back to their owner processors, which will combine all interactions and update their positions.

C. Correctness

Here we provide formal proof that the algorithm does generate every unique triplet of particle subsets $P_0, P_1, \ldots, P_p$, and that no redundancy occurs apart from the special case when $3 \mid p$.

Lemma 3.1: There are $p - 3d$ offset patterns (x, y, z) with two boxes being d positions apart and the last box being at least d away from the other two.

Proof: Let us put the first box at position 0, the second box at position d, then the last box can be from position $2d$ to $p-1-d$ because any position greater than that it will be less than d away from the first box. There are $(p-1-d)-2d+1 = p-3d$ positions, therefore there are $p-3d$ patterns. ■

Lemma 3.2: The offset patterns generated in the algorithm create no redundant triplet apart from the exceptional case.

Proof: Notice that the algorithm generates all offset patterns following the proof in Lemma 3.1, starting from $d = 0$ to $d = \lfloor p/3 \rfloor$. Since we already excluded the special case, no redundancy *within* an offset pattern can occur. Any redundancy would be from getting the same offset pattern again in the process. Assume a redundant offset pattern occurs at distance d, it means at least one of the previously generated patterns has at least one pair of boxes with distance d from each other, which is not possible by construction. ■

Lemma 3.3: The algorithm generates all unique triplets.

Proof: We start by counting the number of box patterns generated by the algorithm. Let q and r be the quotient and remainder of $p/3$, so $p = 3q + r$ where $q = \lfloor p/3 \rfloor$ and $r \in \{0, 1, 2\}$.

$$
\begin{aligned}
\sum_{d=1}^{q}(p-3d) &= pq - \frac{3q(q+1)}{2} \\
&= p\left(\frac{p-r}{3}\right) - \frac{1}{2}\cdot(p-r)\left(\frac{p-r}{3}+1\right) \\
&= \frac{1}{6}\cdot(p-r)(p+r-3) \\
&= \begin{cases} \dfrac{p^2-3p}{6} & \text{if } 3 \mid p \\ \dfrac{(p-1)(p-2)}{6} & \text{otherwise} \end{cases}
\end{aligned}
$$

Now we count how many lines it takes to write all unique triplets in p columns,

$$
\binom{p}{3}/p = \frac{(p-1)(p-2)}{6} = \frac{p^2-3p+2}{6},
$$

which matches the number of lines the algorithm produces exactly when $3 \nmid p$ and is $1/3$ more otherwise. This corresponds to the one extra line in the algorithm, which also produces only one third of a line of unique triplets. Since all triplets generated beforehand are unique, the proof is complete. ■

D. Computation Optimality

There are two kinds of computation optimality we are looking for: avoiding non-redundant computation and ensuring load balance. Lemma 3.2 indicates that no triplets are repeated and therefore proves the first point.

There are three possible workload numbers depending on the offset pattern. Let $m = n/p$ be the number of particles each processor owns. If all three buffers contain same particle parts, e.g., offset pattern 000, the load is $\binom{m}{3}$. If two buffers have same particle parts and the other is different, e.g., offset pattern

Algorithm 3 EMBEDDED ALGORITHM

```
Input $P_{rank}$ in buffer $b_1$, with all forces reset to 0.
Output The same $n/p$ particles updated in-place.
$i \leftarrow 0$                          // Start from the 0th buffer
Copy particles from my left neighbor to $b_0$.
Copy particles from my right neighbor to $b_2$.
for $s \in \{p-3, p-6, ..., p\%3\}$ do
    for $s$ steps do
        if not the first step or $s \neq p$ then
            $shift\_right(b_i)$
        else
            Calculate all $(b_1, b_1, b_1)$ interactions
            Calculate all $(b_1, b_1, b_2)$ interactions
            Calculate all $(b_0, b_0, b_2)$ interactions
        end if
        if $s = p - 3$ then
            Calculate all $(b_0, b_1, b_1)$ interactions
        end if
        Calculate all interactions between three buffers
    end for
    $i \leftarrow (i+1)\%3$
end for
if $3 \mid p$ then                    // Special case: additional round
    $shift\_right(b_i)$
    Calculate one third of the interactions based on $\lfloor \frac{rank}{p/3} \rfloor$
end if
Send particles in each buffer back to the owner processor
Receive up to 3 copies of my own particles
Sum the forces over all 3 copies
Move my particles according to their calculated forces
```

001, the load is $m\binom{m}{2}$. Otherwise, the load is m^3. However, since all processors are always working on the same offset pattern, the load will always be perfectly balanced (assuming n is a multiple of p for simplicity).

E. Communication Optimality

The algorithm sends a message each round for $\sum_{d=1}^{q}(p-3d) = O(p^2)$ rounds, therefore a total of $O(p^2)$ messages. Each message contains n/p words, so the bandwidth used is $O(p^2 \cdot n/p) = O(np)$. These costs match the lower bounds derived in equation (4) with $c = 1$ (no replication, $M = 1{\cdot}n/p$) so the algorithm is communication optimal. Still, the bounds only indicate optimality in an asymptotic sense. The algorithm can save at least a constant factor more; we can calculate the case where more than one particle is from the same particle subset without really having to have them in multiple buffers. These computations can be *embedded* into some other rounds.

Algorithm 3 outlines one of the numerous ways to do so. The only change is to skip the first p rounds in Algorithm 2 and do more computation at some specific rounds. To be more precise, each processor now starts with particles from its left neighbor, itself, and right neighbor in its three buffers instead of all particles from itself. It still alternates between interacting and shifting as before, but the number of times shifted starts from $p-3$. Extra calculations are performed in the first $p-3$ rounds. For example, let us follow the interactions of the 0th particle parts. In the first round interactions are computed on these combinations of buffers: $(b_1, b_1, b_1), (b_1, b_1, b_2), (b_0, b_0, b_2)$, and $(b_0, b_1.b_1)$, in addition to the usual (b_0, b_1, b_2), to get interactions of offset patterns 000, 001, 002, and 800. Offset patterns 700 to 300 are computed in the next p-4 rounds from the (b_0, b_1, b_1) interactions. See lines 10-19 of Figure 2 for an illustration of the full algorithm.

At this point, the algorithm appears to also be communication-optimal in a non-asymptotic sense. Each processor sends only a message every round for the least number of rounds possible, and utilizes all particles in each message.

F. Incorporating 1- and 2-body interactions

For simplicity, most previous 3-body algorithms compute 3-body potentials separately from those of 1- and 2-body potentials because the work is partitioned differently. With our approach, 1- and 2-body potentials can be computed together with 3-body the same way we *'embed'* the first p rounds of Algorithm 2 into Algorithm 3 and the load will still be perfectly balanced without any extra effort.

G. Communication-Avoiding All-Triplets Algorithm

This section demonstrates the application of the communication-avoiding (CA) technique to Algorithm 3. The main concept is to make each processor store more particles and cooperate on computing interactions with other processors who also have the same particle subsets.

Let us store c times more particles and arrange p processors logically into a p/c-by-c 2D torus instead of a p-ring. We divide particles into p/c equal subsets of cn/p particles, $P_0, P_1, \ldots, P_{p/c-1}$, and let processors in the same column own the same particle parts and cooperate on computing forces and updating them as a team. The communication-avoiding algorithm behaves similarly to Algorithm 3 with p/c processors; the only difference is that it divides all the rounds (lines in Figure 2) among processors in the same team, then performs a reduction on particles inside the teams first before sending particles back to owner processor teams.

Not all rounds have the same computation costs so round distribution has to be done wisely to avoid severe load imbalance. Currently, they are partitioned into consecutive rounds with accumulated load closest to the perfect load. Equation (7) predicts the cost to compute the i^{th} round. Again, it is trivial to support 1- and 2-body computation within the algorithm by updating the cost equation accordingly.

$$\mathit{cost} = \begin{cases} m^3 + 3m\binom{m}{2} + \binom{m}{3} & \text{if } i = 0 \\ m^3 + m\binom{m}{2} & \text{if } 0 < i < p/c - 3 \\ m^3/3 & \text{if last round and } 3 \mid p \\ m^3 & \text{otherwise} \end{cases} \tag{7}$$

The high-level algorithm is shown in Algorithm 4. The $predict\_pos$ function is introduced. It takes processor's row_id (rank within team) as an input and indicates for each processor what rounds in the absolute round number of Algorithm 3 it should compute and what particle subsets should be in each buffer at the start. The processor then handles

Algorithm 4 CA All-Triplets Algorithm

```
Input My own $cn/p$ particles $\in P_{col\_id}$ in buffer $b_0$.
Output The same $cn/p$ particles updated in-place.
$(i, srcs, start, end) \leftarrow predict\_pos(row\_id)$
for $j \in \{0, 1, 2\}$ do
    Put $P_{srcs_j}$ into $b_j$.
end for
for $s \in [start, end)$ do
    if not the first step or $s \neq p$ then
        $shift\_right(b_i)$
    end if
    Calculate interactions according to the $s$th round.
    $i \leftarrow (i + 1)\%3$
end for
Do a sum-reduction inside my processor team.
Send particles in each buffer back to the owner processor
Receive up to 3 copies of my own particles
Sum the forces over all 3 copies
Move my particles according to their calculated forces
```

the calculation of each round the same way as Algorithm 3 would.

Figure 2 can be used for illustration again, this time for 36 processors with replication factor $c = 4$. Processors are divided into 9 teams, resulting in total of 10 rounds of interactions (lines 10-19). According to equation (7), processors in rows 0, 1, 2, and 3 will compute lines 10-11, 12-13, 14-15, and 16-19, respectively. Then processors in the same team do a column reduction to get total interactions computed by all processors in the team before sending particles back to their owners, which will proceed to update them as usual.

The overall communication is now carried in two dimensions. When processors separately calculate their share of interactions in the team, they only need to shift horizontally in a ring as before. The additional direction comes from the column reduction at the end where team members need to communicate vertically. Regarding communication costs, there are p/c columns so there will be $O(p^2/c^2)$ total rounds. Dividing the rounds to c groups, each processor does approximately $O(p^2/c^3)$ rounds. A message is sent per round, so $O(p^2/c^3)$ messages are sent during shifting phases. Only $O(\log c)$ messages are required for column reduction so we can consider total number of messages sent to be just $O(p^2/c^3)$. Each message is of size cn/p, therefore the bandwidth used is $O(p^2/c^3) \cdot cn/p = O(np/c^2)$. Since the costs match the lower bounds in (4), we conclude the communication-avoiding algorithm is communication optimal.

There is a constraint on the replication factor c. If there are more processors per team than the number of rounds to compute, then some processor rows will be idle and the computation efficiency will decrease. Hence, we want

$$c \leq \binom{p/c}{3} \div (p/c)$$
$$6c^3 \leq (p - c)(p - 2c). \tag{8}$$

Any c that satisfies inequality (8) guarantees that all processor rows are utilized. Explicit solution of c is omitted due to its length and complexity. Simplifying inequality (8) to $c^3 \leq p^2$, we get the asymptotic upper bound of c,

$$c = O(p^{2/3}), \tag{9}$$

which is consistent with the upper bound in Equation (5).

IV. Performance Results

As a proof of concept, we implemented Algorithm 4 with C MPI. A particle has three-dimensional double-precision coordinates and is of size 80 bytes. The Axilrod-Teller potential [27] is used for 3-body interactions. We exploited the force symmetry $f_{ijk} = f_{ikj}$ and calculate all forces related to all three particles in a triplet at once $(f_{ijk}, f_{jik},$ and $f_{kij})$. We did not do communication overlap since we wanted explicit communication time to fully measure the effect of the algorithm and because it is also a good indicator of energy usage. The correctness verification was done by comparing the algorithm's outputs to those of sequential program and confirming that the difference was no greater than a threshold relative to n and p.

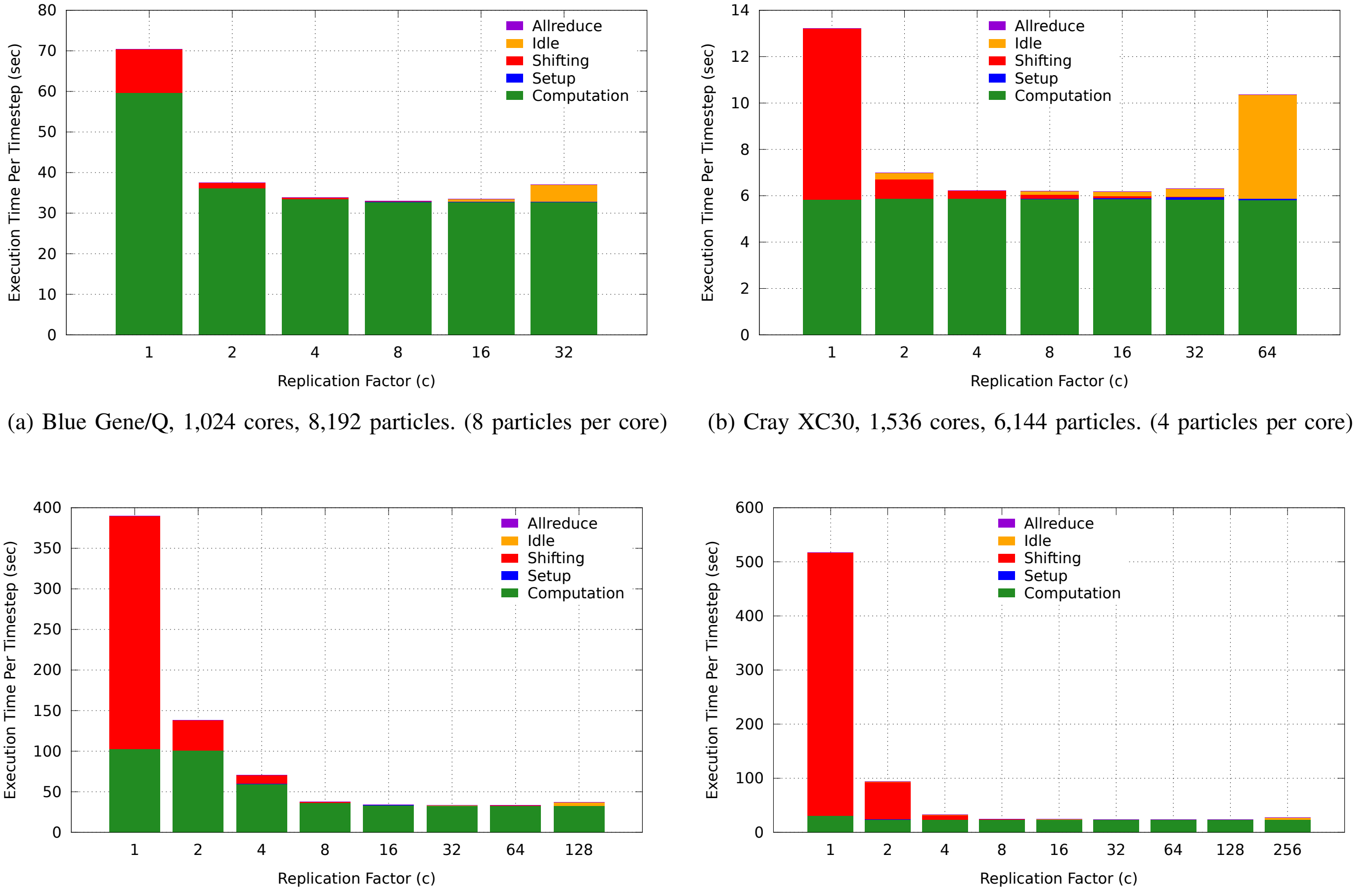
The program was benchmarked on two platforms with different network topology, Mira at Argonne Leadership Computing Facility (ALCF) and Edison at National Energy Research Scientific Computing Center (NERSC); Mira is a 10 PFLOPS IBM Blue Gene/Q supercomputer with a 5D torus network and topology-aware task mapping. There are 49,152 compute nodes, each has a 16-core PowerPC A2 1.6GHz processor with 4 hardware threads and 16 GB of memory; Edison is a 2.57 PFLOPS Cray XC30 supercomputer consisting of 5,576 compute nodes. Each node is equipped with 2 sockets of 12-core Intel Ivy Bridge processor at 2.4GHz and 64 GB memory. The machine has Cray Aries interconnection with Dragonfly topology.

While we used flat MPI with one MPI process per core on Cray XC30, we implemented hybrid MPI/OpenMP version for Blue Gene/Q to fully utilize its 4 hardware threads per core. Using one MPI process per core, we ran 1, 2, and 4 OpenMP threads per MPI process and selected the best results.

A. *Effects of replication*

Figure 3 shows time breakdowns per timestep of the program as we vary the replication factor c. The green, blue, red, yellow, and purple bars are the computation, setup, shifting, idle, and allreduce times, respectively. Setup time is the time to load required particle subsets to buffers at the beginning of the timestep and the time to send particles back to their owners at the end of the timestep combined. Idle time is the average time each processor has to wait for its teammates to reach the reduction point. Allreduce time is the reduction time among column teams.

Figure 3a and 3b are small scale results – 8K particles on 1K cores on Blue Gene/Q and 6K particles on 1.5K cores on Cray XC30. Figure 3c and 3d are large scale results – 16K particles on 8K cores on Blue Gene/Q and the extreme case, 24K particles on 24K cores on Cray XC30. All four graphs demonstrate same decreasing trend in shifting time, i.e., between 4 to 8 times reduction as c doubles. This is consistent with the bounds in equation (4) which says the algorithm can save factors of c^3 in messages and c^2 in bandwidth.



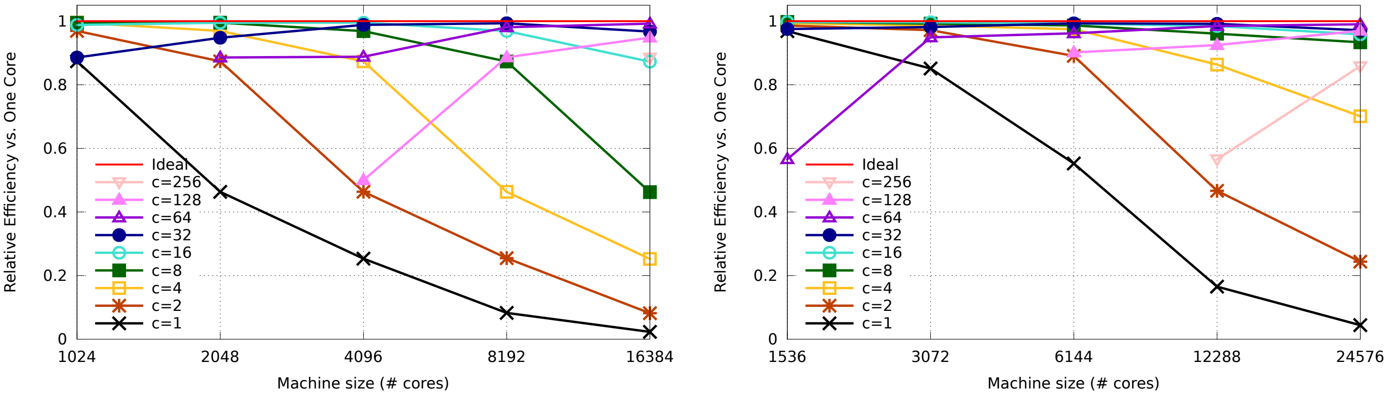
(a) Blue Gene/Q, 1,024 cores, 8,192 particles. (8 particles per core)

(b) Cray XC30, 1,536 cores, 6,144 particles. (4 particles per core)

(c) Blue Gene/Q, 8,192 cores, 16,384 particles. (2 particles per core)

(d) Cray XC30, 24,576 cores, 24,576 particles. (1 particle per core)

Fig. 3: Time breakdown for each replication factor on Blue Gene/Q and Cray XC30. All graphs illustrate the same decreasing trend in shifting time. The setup time is the cost to get desired particles into the buffers at the beginning and then sending the interacted particles back to the owner processor at the end of each timestep. The blocking time is due to load imbalance introduced when replicating. Not shown is the slightly increasing trend in reduction time happening at the end of each timestep.



(a) Blue Gene/Q, 16,384 particles

(b) Cray XC30, 24,576 particles.

Fig. 4: Strong scaling on Blue Gene/Q and Cray XC30. Larger c's perform better as the number of cores increases, vice versa for smaller c's. Some large c's are available only with sufficiently many cores.

Up to 99.98% reduction in communication time, idle time included, is observed in the experiment. Maximum overall speedup occurs at the extreme scale at one particle per core, 22.13× on 16K cores on Blue Gene/Q (breakdown graph not shown) and 41.85× on 24K cores on Cray XC30.

Computation times are mostly equal as expected. For Blue Gene/Q, the computation time where $c \leq 4$ is significantly larger than others because each thread has too little work to do. As c increases, there are more interactions to compute per thread so the computation times are equal from $c > 4$. This shows that replication also helps maintain computation efficiency.

Idle time indicates load imbalance. It increases with c because storing more particles means more interactions per round and also less rounds to distribute among the processor rows (a.k.a. *team members*). Load balance is harder to maintain at coarser grain. However, this imbalance can be predicted and thus avoided since the work partitioning is static and does not depend on input data.

B. *Scalability*

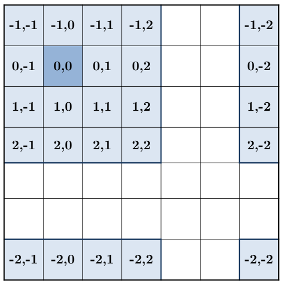
Figure 4 shows strong scaling on Blue Gene/Q and Cray XC30, with 16K and 24K particles, respectively. The y-axis is relative efficiency compared to one node, which is estimated from running the program on one node for 2,048 and 3,072 particles on Blue Gene/Q and Cray XC30, respectively. The red line at 1 indicates ideal efficiency and other lines are efficiencies for each replication factor c from 1 to 256. The benchmark achieved perfect strong scaling at 99% efficiency for all machine sizes on both machines with the best c.

In general, larger c's perform better than smaller c's with more cores. Some replication factors are only available with sufficiently many cores.

V. Extension for Cutoff Distance

This section extends the all-triplets algorithm to support a cutoff distance. Unlike pairwise interactions in which there is only one obvious way to apply the cutoff distance r_c, it is more complicated in the 3-body case. There have been various ways of applying cutoff distance to 3-body interactions, from triplets in which at least a pair of particles are less than r_c apart [28], [15], triplets where all particles are less than r_c apart [29], to triplets where the sum of all distances from the center of mass is less than r_c [11], etc. We opted to follow the second approach where all particles have to be less than r_c from each other because the first approach does not preserve Newton's third law [29] and because of its simplicity.

This cutoff algorithm can handle any k-body interactions ($k \geq 2$). Let the problem space be 3-dimensional and have periodic boundary conditions. We use domain decomposition and arrange p processors into a $p^{1/3} \times p^{1/3} \times p^{1/3}$ torus, map them to the simulation space, and let them own particles in their range. Let dim be the number of processor boxes in one dimension; $dim = p^{1/3}$ in 3-dimensional case. Assume that the cutoff distance r_c is no less than a processor box width but smaller than one third of the simulation box width ($1 \leq b_c < dim/3$, where b_c is the number of processors that r_c spans). Using bold letters to denote vectors, let



-1,-1	-1,0	-1,1	-1,2			-1,-2
0,-1	0,0	0,1	0,2			0,-2
1,-1	1,0	1,1	1,2			1,-2
2,-1	2,0	2,1	2,2			2,-2
-2,-1	-2,0	-2,1	-2,2			-2,-2

Fig. 5: A small system in 2-dimensional space with 49 processors arranged into 7×7 grid and $b_c = 2$. Cutoff range of $proc_{(1,1)}$ (highlighted in dark blue) is shown in light blue. Written inside each box are the relative coordinates with respect to position $(1,1)$.

- $\mathbb{U} = \{(x,y,z) \in \mathbb{Z}^3 \mid 0 \leq x,y,z < dim\}$
- $\mathbf{r} \in \mathbb{U}$ for rank $\mathbf{r}$ of $proc_{\mathbf{r}}$,
- $\mathbf{i_{1:k}}$ be a short notation for $\mathbf{i_1}, ..., \mathbf{i_k}$, for all $\mathbf{i_j} \in \mathbb{U}$,
- $x \circ y = \min(|x-y|, dim - |x-y|)$ (periodic distance of x and y, $0 \leq x, y, < dim$),
- $\mathbf{u} \circ \mathbf{v} = (u_0 \circ v_0, u_1 \circ v_1, u_2 \circ v_2)$ (periodic distance vector of $\mathbf{u}$ and $\mathbf{v}$, $\mathbf{u}, \mathbf{v} \in \mathbb{U}$),
- $\mathbf{u} \leq x$ if $u_j \leq x \ \forall j, 0 \leq j < q$ for a q-vector $\mathbf{u}$,
- $w(\mathbf{r}) = \{i \in \mathbb{U} \mid \mathbf{i} \circ \mathbf{r} \leq b_c\}$ (cutoff window of $proc_{\mathbf{r}}$),
- $w(\mathbf{i_{1:k}}) = w(\mathbf{i_1}) \cap ... \cap w(\mathbf{i_k})$ (intersected cutoff window),
- $x \ominus y = \begin{cases} x - y & \text{if } |x-y| \leq \frac{dim}{2} \\ sgn(y-x) \cdot x \circ y & \text{otherwise} \end{cases}$ (periodic difference)
- $\mathbf{u} \leq_{\mathbf{r}} \mathbf{v}$ when $\mathbf{u} \ominus \mathbf{r} \leq \mathbf{v} \ominus \mathbf{r}$ in lexicographic manner.

Figure 5 illustrates a simplified example of 7×7 processors in a 2-dimensional problem space. The cutoff window for $proc_{(1,1)}$ is shown in light blue. Indices in the pictures are relative coordinates compared to the point $(1,1)$ using the $\ominus$ operator. In other words, the processor labeled $(0,0)$ is actually processor $(1,1)$.

The algorithm is just a parallelization of the nested for loops in sequential k-body computation. Each $proc_{\mathbf{r}}$ computes the $\mathbf{r}^{\text{th}}$ iteration of the outermost loop which has exactly p iterations. It simply loops through boxes $\mathbf{i_2}$ in its cutoff range from $\mathbf{r}$ to the end of the cutoff window, then loops through boxes $\mathbf{i_3}$ in the intersected cutoff range of $\mathbf{i_2}$ and $\mathbf{r}$ from $\mathbf{i_2}$ to the end of the intersected cutoff window, and so on for k levels. Algorithm 5 shows the full pseudocode.

A. *Correctness*

Why does the algorithm not cause redundancy here? Is it correct? This section provides a formal proof that it generates all necessary triplets and creates no redundancy.

Lemma 5.1: All triplets that need to be interacted can be written as $(\mathbf{i_{1:k}})$, where $\mathbf{i_1} \leq_{\mathbf{i_1}} \mathbf{i_2} \leq_{\mathbf{i_1}} \dots \leq_{\mathbf{i_1}} \mathbf{i_k}$, when the cutoff range spans less than $1/3$ of the simulation space width.

Algorithm 5 PARALLEL K-BODY CUTOFF ALGORITHM

```
Input My rank $\mathbf{r} \in \mathbb{U}^3$
Input $P_\mathbf{r}$ with all forces reset to 0.
Output Particles updated in-place.
for $\mathbf{i_2} \in w(\mathbf{r}), \mathbf{r} \leq_\mathbf{r} \mathbf{i_2}$ do
    ...
    for $\mathbf{i_k} \in w(\mathbf{r}, \mathbf{i_{2:k-1}}), \mathbf{i_{k-1}} \leq_\mathbf{r} \mathbf{i_k}$ do
        Interact my particles with particles from $\mathbf{i_{2:k}}$
    end for
    ...
end for
Update my particles
```

Proof: Determining a sequence's starting point and order is a little tricky in a periodic boundary space. Sometimes such an order above does not exist, such as Figure 6, for example. The problem space is 1-dimensional and there are 6 processors forming a ring. The cutoff distance spans 2 processors. The cutoff windows of $proc_0$, $proc_2$, and $proc_4$ are shown in red dashed, green solid, and blue dotted frames, respectively. Assume, without losing generality, that we are computing 3-body interactions. Observe that the triplet $(0, 2, 4)$ cannot be written in any way that satisfies the inequality. It can not be $(0, 2, 4)$ and $(0, 4, 2)$ because $4 <_0 0$. $(2, 0, 4)$, $(2, 4, 0)$, $(4, 0, 2)$, and $(4, 2, 0)$ are also invalid in the same manners ($0 <_2 2$ and $2 <_4 4$).

This is because the triplet *wraps around* the simulation space. In other words, let $proc_b$ be any processor in the cutoff range of $proc_a$, the maximum width of the union of the cutoff windows of $proc_a$ and $proc_b$ exceeds dim, i.e., $(2b_c+1)+b_c > dim$. ($2b_c + 1$ for the cutoff window of $proc_a$. $proc_b$ being at one end of the cutoff window of $proc_a$ will give maximum (union) cutoff extension of b_c.) To find out the case this does not occur, we solve for b_c in

$$\begin{aligned} (2b_c + 1) + b_c &\leq dim \\ 3b_c &< dim \qquad (b_c, dim \in \mathbb{Z}) \\ b_c &< dim/3, \end{aligned} \tag{10}$$

which is our initial assumption. This completes the proof. ■

Lemma 5.2: Algorithm 5 computes all necessary k-tuples.

Proof: The set of k-tuples that $proc_\mathbf{r}$ computes is

$$Q_\mathbf{r} = \{(\mathbf{r}, \mathbf{i_{2:k}}) \in w(\mathbf{r})^k \mid (\mathbf{r}, \mathbf{i_{2:k}}) \in w(\mathbf{r}, \mathbf{i_{2:k}})^k \text{ and } \mathbf{r} \leq_\mathbf{r} \mathbf{i_2} \leq_\mathbf{r} \ldots \leq_\mathbf{r} \mathbf{i_k}\}. \tag{11}$$

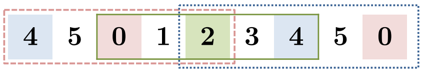


Fig. 6: A 1-dimensional problem space with 6 processors and the cutoff distance $b_c = 2$. The red dashed, green solid, and blue dotted frames illustrate the cutoff windows of $proc_0$, $proc_2$, and $proc_4$, respectively.

According to Lemma 5.1, the set of all unique k-tuples $proc_\mathbf{r}$ needs can be written as

$$R_\mathbf{r} = \{(\mathbf{i_{1:k}}) \in \mathbb{U}^k \mid (\mathbf{i_{1:k}}) \in w(\mathbf{i_{1:k}})^k \text{ and } \mathbf{i_j} = \mathbf{r},\ \exists j \text{ and } \mathbf{i_1} \leq_{\mathbf{i_1}} \mathbf{i_2} \leq_{\mathbf{i_1}} \ldots \leq_{\mathbf{i_1}} \mathbf{i_k}\}. \tag{12}$$

Since $R_\mathbf{r} \subseteq \bigcup_{\mathbf{i} \in \mathbb{U}} Q_\mathbf{i}$, Algorithm 5 computes all necessary k-tuples interactions for any $proc_\mathbf{r}$. ■

Lemma 5.3: $\mathbf{u} \ominus \mathbf{v} = -(\mathbf{v} \ominus \mathbf{u})$

Proof: We first prove the point for the scalar version of the operation: $x \ominus y = -(y \ominus x)$. There are two cases. If x and y are less than half the simulation box width apart, $x \ominus y$ is simply $x - y$ and thus

$$x \ominus y = x - y = -(y - x) = -(y \ominus x).$$

When $x \ominus y = sgn(y - x) \cdot x \circ y$, we have

$$\begin{aligned} x \ominus y &= sgn(y - x) \cdot x \circ y \\ &= -sgn(x - y) \cdot y \circ x \qquad (x \circ y = y \circ x) \\ &= -(y \ominus x) \end{aligned}$$

as well. Therefore, $x \ominus y = -(y \ominus x)$. Since the vector version of $\ominus$ is just element-wise scalar operations, it follows that $\mathbf{u} \ominus \mathbf{v} = -(\mathbf{v} \ominus \mathbf{u})$. ■

Lemma 5.4: No two processors compute the same k-tuples interactions.

Proof: Suppose the hypothesis is false and two processors of rank $\mathbf{u}$ and $\mathbf{v}$ work on the same k-tuples. Because indices are sorted lexicographically, the tuples have to be of form $(\mathbf{u}, \mathbf{i_{2:k}})$ and $(\mathbf{v}, \mathbf{j_{2:k}})$. Since $\mathbf{u} \neq \mathbf{v}$, the elements in both tuples must be in different orders. Let $\mathbf{u} = \mathbf{j_a}$ and $\mathbf{v} = \mathbf{i_b}$. They have to satisfy these following relative orders within tuple,

$$\mathbf{u} \leq_\mathbf{u} \mathbf{i_2} \leq_\mathbf{u} \ldots \leq_\mathbf{u} \mathbf{i_b} = \mathbf{v} \leq_\mathbf{u} \ldots \leq_\mathbf{u} \mathbf{i_k} \tag{13}$$

$$\mathbf{v} \leq_\mathbf{v} \mathbf{j_2} \leq_\mathbf{v} \ldots \leq_\mathbf{v} \mathbf{j_a} = \mathbf{u} \leq_\mathbf{v} \ldots \leq_\mathbf{v} \mathbf{j_k} \tag{14}$$

From (13), $\mathbf{u} \leq_\mathbf{u} \mathbf{v}$,

$$(\mathbf{0}, \mathbf{0}, \mathbf{0}) = \mathbf{u} \ominus \mathbf{u} \leq \mathbf{v} \ominus \mathbf{u} \qquad \text{(by definition)} \tag{15}$$

and from (14), $\mathbf{v} \leq_\mathbf{v} \mathbf{u}$,

$$(\mathbf{0}, \mathbf{0}, \mathbf{0}) = \mathbf{v} \ominus \mathbf{v} \leq \mathbf{u} \ominus \mathbf{v} \qquad \text{(by definition).} \tag{16}$$

Combining Lemma 5.3 with inequalities (15) and (16), we get

$$(\mathbf{0}, \mathbf{0}, \mathbf{0}) \leq \mathbf{u} \ominus \mathbf{v} \leq (\mathbf{0}, \mathbf{0}, \mathbf{0}). \tag{17}$$

The only solution to inequality (17) is $\mathbf{u} \ominus \mathbf{v} = (\mathbf{0}, \mathbf{0}, \mathbf{0})$, which only happens when $\mathbf{u} = \mathbf{v}$. This contradicts with the assumption that $\mathbf{u} \neq \mathbf{v}$ in the setup and hence the hypothesis has to be true. ■

We now conclude that Algorithm 5 operates correctly.

B. Computation Optimality

Again, we are looking for two kinds of optimality: no redundant work and ensuring load balance. Lemma 5.4 states that no k-tuple is computed twice and thus proves the first property. The load balance depends on particle distribution. If particles are uniformly distributed, all processors will compute equal number of interactions every round and the load is balanced. Otherwise, there will be load imbalance.

C. Communication Optimality

Here we derive the communication lower bounds for the uniformly-distributed cutoff case. For a d-dimensional problem space, let f be the ratio of the cutoff window volume to the problem space volume,

$$f = \left(\frac{2b_c + 1}{dim}\right)^d .$$

This means there are about fn particles in a particle's cutoff volume. The total work is $O\left(n \cdot (fn)^{k-1}\right)$ and thus each processor has work $Z = O(f^{k-1}n^k/p)$. Let us continue to write M, the number of particles per processor, as cn/p. The maximum useful work a processor can do with M particles in memory is $O(M^k)$, therefore, $F = c^k n^k/p^k$. Substitute this into equation (2), we get the general communication lower bounds for k-body problem,

$$S = \Omega\left(\frac{f^{k-1}p^{k-1}}{c^k}\right), \qquad W = \Omega\left(\frac{nf^{k-1}p^{k-2}}{c^{k-1}}\right). \quad (18)$$

For example, let $f = 1.0$, $k = 3$ and we get the same lower bounds as in equation (4). The case where $f = 1.0$, $k = 2$ also matches the 2-body communication lower bounds previously derived in [25].

Now let us analyze the communication costs of Algorithm 5. Since particles are distributed evenly and processors are responsible of equal domain size, there are roughly fp processor cells within a processor's cutoff volume and $O(fp)$ rounds are required to go through all of them. With k-way interaction, a processor needs to loop through all cells in range for $k-1$ levels, totaling of $O\left((fp)^{k-1}\right)$ rounds. In most rounds it sends only a message, except when the cells in upper levels also change, where it will send up to k messages per round. The program stores n/p particles in memory. Therefore, the total communication costs are,

$$S = O\left(kf^{k-1}p^{k-1}\right), \qquad W = O\left(knf^{k-1}p^{k-2}\right),$$

which are optimal if we consider k constant ($c = 1$).

Note that the costs of the all-triplet algorithms derived earlier did not have the k factor. This is because shifting is arranged in such a way that exactly one message is sent per round. We can try this optimization with the cutoff algorithm as well by mimicking Gray code generation, although special care has to be taken because different levels have different bounds. For example, consider 1-dimensional 3-body interactions for $proc_0$ with r_c spanning 3 processor cells ($p \geq 10$). Algorithm 5 will process particle subset triplets in this order: 000, 001, 002, 003, 011, 012, 013, 022, 023. A better schedule would be: 000, 001, 002, 003, 013, 011, 012, 022, 023. Notice the jump from 013 to 011 before going back to 012. Had it been 013, 012, 011, more than one message would be required to go to the next particle subset triplet. The communication costs for the modified schedule are

$$S = O\left(f^{k-1}p^{k-1}\right), \qquad W = O\left(nf^{k-1}p^{k-2}\right),$$

which are optimal regardless of k (with $c = 1$).

D. Communication-Avoiding Algorithm

A communication-avoiding algorithm can be derived the same way as in the all-triplets section. In brief, we divide processors into p/c teams with c team members each. Each team owns cn/p particles. All processors in a team cooperate to compute interactions for the particle subset their team owns. Work should be partitioned in a way that each processor calculates close to $1/c$ of all interactions of the subset. Finally, processors participate in a team sum-reduce to merge all interactions together, then update the particles. Due to the lack of space, we will only prove its communication optimality.

Assume once again that particles are uniformly distributed and let f be the ratio of the cutoff volume to the problem space volume. There are p/c teams, each has to interact with approximately $O(fp/c)$ particle parts from other teams. $O\left((fp/c)^{k-1}\right)$ rounds of interactions are required in k-way interactions. If the work is partitioned perfectly, a processor has to compute $O\left((fp/c)^{k-1}/c\right)$ rounds. Assuming optimal shifting schedule (1 message per round) is used, the total communication costs are

$$S = O\left(\frac{f^{k-1}p^{k-1}}{c^k}\right), \qquad W = O\left(\frac{nf^{k-1}p^{k-2}}{c^{k-1}}\right),$$

matching the communication lower bounds. The limit on c is $c \leq f^{\frac{k-1}{k}} p^{\frac{k-1}{k}}$. The memory-independent bounds are

$$S = O\left(1\right), \qquad W = O\left(nf^{\frac{k-1}{k}}p^{-\frac{1}{k}}\right).$$

VI. Conclusion

We presented a direct long-range 3-body algorithm and proved that it is both computation and communication optimal. We also provided a communication-avoiding version that, by making c replicas, decreases the total number of messages and bandwidth usage by c^3 and c^2, respectively.

The communication-avoiding algorithm introduces some load imbalance, but it is predictable based on a given replication factor, c, and grows with increasing values of c. Thus, there is a tradeoff between reducing shifting time through bandwidth and latency reductions and increasing idle time. Since we know the amount of load imbalance for each c ahead of time and the increase in reduction time is insignificant compared to shifting time, we suggest picking large c's with reasonably small load imbalance.

Large scale experimental results on up to 16K cores on Blue Gene/Q and 24K cores on Cray XC30 were consistent with the communication costs predicted for the communication-avoiding algorithm and also exhibited strong scalability. The algorithm experienced up to 99.98% reduction in communication time and 41.85× speedup, enabling strong scaling up to 99% efficiency.

Finally, we presented a generalized algorithm that supports k-body computations with a large cutoff distance that limits interactions to 1/3 of the total particles. We derived the communication lower bounds for this k-body algorithm and showed that it is also both computation and communication optimal. A specific piece of future work is to implement of this algorithm with cutoff and then compare it to the algorithm without cutoff for both accuracy and performance in the context of a real application scenarios such as water simulation.

We believe that our algorithmic framework, which represents a class of algorithms for various values of k and varying amounts of memory for replication is flexible enough to be useful in multiple application settings. The algorithms were provably optimal in communication and computation, and had bounded load imbalance. Overall, this work has shown the importance of both communication avoidance and computation avoidance for scalable k-body algorithms in both a theoretical and experimental setup.

Acknowledgments

This research used resources of the National Energy Research Scientific Computing Center and the Argonne Leadership Computing Facility, both supported by the Office of Science of the U.S. Department of Energy under Contracts No. DE-AC02-05CH11231 and DE-AC02-06CH11357, respectively. This work was also supported by the Department of Energy's Office of Advanced Scientific Computing Research X-Stack program under University of California, Berkeley contract DE-SC0008700 and Lawrence Berkeley National Laboratory contract DE-AC02-05CH11231. The first author was supported by a Fulbright Scholarship.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] J. Li, Z. Zhou, and R. J. Sadus, "A cyclic force decomposition algorithm for parallelising three-body interactions in molecular dynamics simulations," in *Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences - Volume 1 (IMSCCS'06) - Volume 01*, ser. IMSCCS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 338–343. [Online]. Available: http://dx.doi.org/10.1109/IMSCCS.2006.3

[2] ——, "Modified force decomposition algorithms for calculating three-body interactions via molecular dynamics," *Computer Physics Communications*, vol. 175, no. 11-12, pp. 683–691, 2006.

[3] ——, "Parallel algorithms for molecular dynamics with induction forces," *Computer Physics Communications*, vol. 178, no. 5, pp. 384–392, 2008.

[4] J. V. Sumanth, D. R. Swanson, and H. Jiang, "A symmetric transformation for 3-body potential molecular dynamics using force-decomposition in a heterogeneous distributed environment," in *Proceedings of the 21st Annual International Conference on Supercomputing*, ser. ICS '07. New York, NY, USA: ACM, 2007, pp. 105–115. [Online]. Available: http://doi.acm.org/10.1145/1274971.1274988

[5] M. J. Elrod and R. J. Saykally, "Many-body effects in intermolecular forces," *Chemical reviews*, vol. 94, no. 7, pp. 1975–1997, 1994.

[6] S. J. Plimpton and A. P. Thompson, "Computational aspects of many-body potentials," *MRS bulletin*, vol. 37, no. 05, pp. 513–521, 2012.

[7] H. H. Gan, A. Tropsha, and T. Schlick, "Lattice protein folding with two and four-body statistical potentials," *Proteins: Structure, Function, and Bioinformatics*, vol. 43, no. 2, pp. 161–174, 2001.

[8] C. W. Carter Jr, B. C. LeFebvre, S. A. Cammer, A. Tropsha, and M. H. Edgell, "Four-body potentials reveal protein-specific correlations to stability changes caused by hydrophobic core mutations," *Journal of Molecular Biology*, vol. 311, no. 4, pp. 625–638, 2001.

[9] G. Marcelli and R. J. Sadus, "Molecular simulation of the phase behavior of noble gases using accurate two-body and three-body intermolecular potentials," *The Journal of chemical physics*, vol. 111, no. 4, pp. 1533–1540, 1999.

[10] O. Matsuoka, E. Clementi, and M. Yoshimine, "Ci study of the water dimer potential surface," *The Journal of Chemical Physics*, vol. 64, no. 4, pp. 1351–1361, 2008.

[11] L. D. O'Suilleabhain, "Three Body Approximation to the Condensed Phase of Water," Master's thesis, University of California, Berkeley, Berkeley,CA, 2013.

[12] J. Barnes and P. Hut, "A hierarchical o(n log n) force-calculation algorithm," 1986.

[13] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of computational physics*, vol. 73, no. 2, pp. 325–348, 1987.

[14] A. Nakano, P. Vashishta, and R. K. Kalia, "Parallel multiple-time-step molecular dynamics with three-body interaction," *Computer physics communications*, vol. 77, no. 3, pp. 303–312, 1993.

[15] M. Kunaseth, R. K. Kalia, A. Nakano, K. Nomura, and P. Vashishta, "A scalable parallel algorithm for dynamic range-limited *n*-tuple computation in many-body molecular dynamics simulation," in *SC*, W. Gropp and S. Matsuoka, Eds. ACM, 2013, p. 71.

[16] A. G. Gray and A. W. Moore, "N-body problems in statistical learning," in *NIPS*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2000, pp. 521–527.

[17] D. Lee, A. Ozakin, and A. G. Gray, "Multibody multipole methods," *Journal of Computational Physics*, vol. 231, no. 20, pp. 6827–6845, 2012.

[18] S. H. Fuller, L. I. Millett *et al.*, *The Future of Computing Performance: Game Over or Next Level?* National Academies Press, 2011.

[19] J. Demmel, A. Gearhart, B. Lipshitz, and O. Schwartz, "Perfect strong scaling using no additional energy," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 649–660. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2013.32

[20] H. Jia-Wei and H. T. Kung, "I/O complexity: The red-blue pebble game," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, ser. STOC '81. New York, NY, USA: ACM, 1981, pp. 326–333.

[21] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1017–1026, 2004.

[22] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in linear algebra," *SIAM J. Mat. Anal. Appl.*, vol. 32, no. 3, 2011.

[23] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. A. Yelick, "Communication lower bounds and optimal algorithms for programs that reference arrays - part 1," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-61, May 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.html

[24] E. Solomonik, E. Carson, N. Knight, and J. Demmel, "Tradeoffs between synchronization, communication, and work in parallel linear algebra computations," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-8, January 2014.

[25] M. Driscoll, E. Georganas, P. Koanantakool, E. Solomonik, and K. Yelick, "A communication-optimal n-body algorithm for direct interactions," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 1075–1084. [Online]. Available: http://dx.doi.org/10.1109/IPDPS.2013.108

[26] S. Harfst, A. Gualandris, D. Merritt, R. Spurzem, S. P. Zwart, and P. Berczik, "Performance analysis of direct n-body algorithms on special-purpose supercomputers," *New Astronomy*, vol. 12, no. 5, pp. 357 – 377, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138410760600131X

[27] B. Axilrod and E. Teller, "Interaction of the van der waals type between three atoms," *Journal of Chemical Physics*, vol. 11, pp. 299–300, 1943.

[28] C. Cornwell and L. Wille, "Parallel molecular dynamics simulations for short-ranged many-body potentials," *Computer physics communications*, vol. 128, no. 1, pp. 477–491, 2000.

[29] G. Marcelli, "The role of three-body interactions on the equilibrium and non-equilibrium properties of fluids from molecular simulation," Ph.D. dissertation, Swinburne University of Technology, 2001.