SC20: International Conference for High Performance Computing, Networking, Storage and Analysis | 978-1-7281-9998-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/SC41405.2020.00026

Cell-List based Molecular Dynamics on Many-Core Processors: A Case Study on Sunway TaihuLight Supercomputer

Xiaohui Duan[a,d,e] Ping Gao[a,e] Meng Zhang[a,e] Tingjian Zhang[a,e] Hongsong Meng[e] Yuxuan Li[d,e]
Bertil Schmidt[b] Haohuan Fu[c,e] Lin Gan[d,e] Wei Xue[d,e] Weiguo Liu[a,e] Guangwen Yang[d,e]
sunrise.duan@mail.sdu.edu.cn qdgaoping@gmail.com missximon_7@163.com sdubeyhhhh@gmail.com
popobeauty@qq.com yuxuan-l17@mails.tsinghua.edu.cn bertil.schmidt@uni-mainz.de
{haohuan, lingan, xuewei}@tsinghua.edu.cn weiguo.liu@sdu.edu.cn ygw@tsinghua.edu.cn
a. School of Software, Shandong University, China
b. Johannes-Gutenberg University of Mainz, Germany
c. Ministry of Education Key Lab. for Earth System Modeling, and Department of Earth System Science, Tsinghua University, China
d. Department of Computer Science and Technology, Tsinghua University, China
e. National Supercomputing Center in Wuxi, China

***Abstract*—Molecular dynamics (MD) simulations are playing an increasingly important role in several research areas. The most frequently used potentials in MD simulations are pair-wise potentials. Due to the memory wall, computing pair-wise potentials on many-core processors are usually memory bounded. In this paper, we take the SW26010 processor as an exemplary platform to explore the possibility to break the memory bottleneck by improving data reusage via cell-list-based methods. We use cell-lists instead of neighbor-lists in the potential computation, and apply a number of novel optimization methods. Theses methods include: an adaptive replica arrangement strategy, a parameter-profile data structure, and a particle-cell cutoff checking filter. An incremental cell-list building method is also realized to accelerate the construction of cell-lists. Furthermore, we have established an open source standalone framework, ESMD, featuring the techniques above. Experiments show that ESMD is 50∼170% faster than previous ports on a single node, and can scale to 1,024 nodes with a weak scalibility of 95%.**

***Index Terms*—Molecular Dynamics, Sunway TaihuLight, Neighbor-list-free method**

I. Introduction

Molecular dynamics (MD) plays an increasingly important role in many research areas. By means of microscopic simulations, it enables scientists to study biochemical reactions and structures of macro-molecules. Over the last decades a number of MD packages have been developed and applied in biological research such as *NAMD* [1], *GROMACS* [2], *LAMMPS* [3], and *AMBER* [4].

MD applications evaluate the interactions between particles according to various potentials. The *Lennard-Jones* potential [5] is one of the most simple potentials which can approximate the van der Waal's force among particles. The *Tersoff* potential [6] models the interactions in crystal materials. The *AMBER* potential is a combination of various potentials that can evaluate interactions in biochemical systems.

Calculation of pair-wise interactions for evaluating non-bonded forces is typically the most time-consuming part for the computation of these potentials. It is also essential for computing many-body potentials, for example, to accumulate the electronic cloud density in *Embedded Atom Methods* [7] and generate a short neighbor list in *Tersoff* potentials.

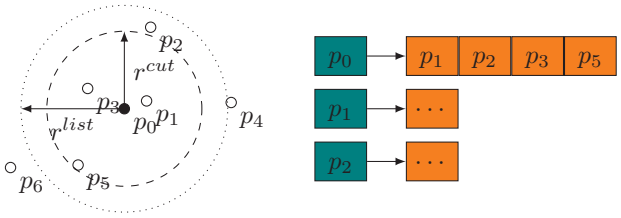
With the development of modern architectures, compute power increases faster than memory transfer speed. Thus, memory accesses become a bottleneck in various applications. As pair-wise interaction calculations usually contain a number of random memory accesses, data reuse can be low. Thus, corresponding implementations are often memory-bound.

The Sunway TaihuLight supercomputer [8] is manufactured by the National Research Center of Parallel Computer Engineering and Technology of China. It has a theoretical peak performance of 125 PFlops/s, making it the world's fastest supercomputer between 2016 and 2018.

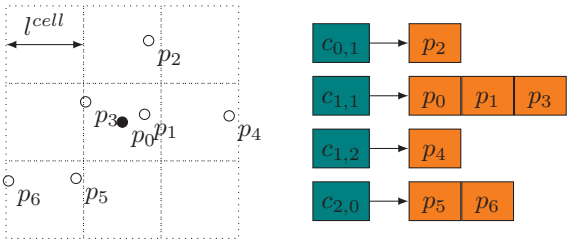
The TaihuLight is equipped with $40{,}960$ SW26010 processors. The SW26010 processor features an on-chip heterogeneous many-core architecture. The compute cores have access to a 64KB *local device memory (LDM)* instead of data caches. The LDM is controlled by DMA instructions, which makes it a good platform for the study of data reuse strategies and optimizations.

Recent studies [9] [10] [11] on MD application optimization on the SW26010 processor are based on neighbor lists. While various methods for handling random memory accesses are considered, the corresponding implementations still hit the memory wall. An alternative to the neighbor list data structure are cell lists (see Figure 1). They divide the simulation box into cells and build a list for particles within each cell. While building lists may require random memory access, the computation of pair-wise interactions based on cell lists can exhibit good data locality.

SC20, November 9-19, 2020, Is Everywhere We Are
978-1-7281-9998-6/20/$31.00 ©2020 IEEE



(a) Neighbor list



(b) Cell list

Fig. 1. Neighbor list and cell list.

In this paper, we present the design of a cell-list-based pair-wise interaction computation framework. By proposing various novel optimization methods including adaptive replica-summation for conflict-free parallelization, parameter profiles for flexible vectorization, and a particle-cell cutoff checking filter for reducing the computation workload, we can outperform all previously published pair-wise interaction implementations on the SW26010 processor. Some of our work is based on AMBER's *pmemd* simulation engine. Furthermore, we have established a standalone framework, ESMD, that includes the above techniques and a novel incremental cell list updating method. The evaluation of this cell list based interaction computation shows that our optimized AMBER has comparable single node performance to a Knights Landing (KNL) many-core processor. On a single node of SW26010, ESMD can have 50% to 170% speedup over the latest existing port of LAMMPS [11] with the same input and potential. Furthermore, ESMD can achieve 95% weak scaling efficiency on 1,024 nodes.

II. Background

A. Computation of Pair-wise Interactions

The computation of pair-wise potentials is a fundamental part of MD simulations used for the evaluation of non-bonded forces. One of the earliest proposed is the *Lennard-Jones* potential [5] which simply accumulates the van der Waal's (vdW) force among atoms without charges as shown in Equation (1), where r_{ij} denotes the distance between particle i and j, and σ_{ij}, ϵ_{ij} are empirical parameters depending on the considered particle types.

$$V_{LJ} = \sum_{i<j}^{n} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] \tag{1}$$

Because the vdW force converges quickly as the distance grows, a cutoff radius is typically used to speedup computation.

Algorithm 1 Three algorithms for Pair-wise Interactions

Require:
P: all of particles to be processed.
L: neighbor list for particles.
C: all cells in the image.
r^{cut}, r^{list}: interaction cutoff radius and neighbor list cutoff radius.
DIST(P_i, P_j): calculate the distance between particles P_i and P_j.
PAIR-UPD(F, P_i, P_j): calculate the pair-wise interactions between particles P_i and P_j and update the force status F of P_i and P_j.
NBR(C_i): find adjacent cells of cell C_i.
Ensure:
F: force status of atoms.

```
function NAÏVE-PAIR(P, F)
    for i ← 1, |P| do
        for j ← i + 1, |P| do
            if DIST(P_i, P_j) < r^cut then
                PAIR-UPD(F, P_i, P_j)
function NEIGHBOR-LIST-PAIR(P, F, L)
    for i ← 1, |P| do
        for j ∈ L_i do
            if DIST(P_i, P_j) < r^cut then
                PAIR-UPD(F, P_i, P_j)
function NEIGHBOR-FROM-CELL(P, C, L)
    for I ← 1, |C| do
        for J ∈ NBR(C_I) do
            for i ∈ C_I do
                for j ∈ C_J do
                    if DIST(P_i, P_j) < r^list then
                        L_i ← L_i ∪ P_j
```

Coulomb force calculation (Equation (2)), however, only converges slowly with growing distance. In Equation (2), k_e denotes the Coulomb constant, and q_i, q_j represent the charges of the two particles.

$$V_{Coulomb} = k_e \sum_{i<j}^{n} \frac{q_i q_j}{r_{ij}} \tag{2}$$

The computation of Coulomb forces is therefore usually separated into two parts: short-ranged and long-ranged. Equation (3) shows a corresponding example for the *Particle-Mesh Ewald* method [12].

$$V_{Coulomb} = k_e \left(\underbrace{\sum_{i<j}^{n} \frac{q_i q_j \mathrm{erfc}(\alpha r_{ij})}{r_{ij}}}_{\text{short range}} + \underbrace{\sum_{i<j}^{n} \frac{q_i q_j \mathrm{erf}(\alpha r_{ij})}{r_{ij}}}_{\text{long range}} \right) \tag{3}$$

erf and erfc in Equation (3) are the Gaussian error function and the complementary error function. erfc(x) reduces rapidly when x grows. Thus, the short range part can be computed by

applying a cutoff for pair-wise interactions. The long range part can be calculated by spectral methods.

A naïve method to calculate pair-wise interactions enumerates each pair of particles and calculates interactions if the distance is within a specific cutoff radius r^{cut} as shown in NAÏVE-PAIR in Algorithm 1, and α is a coefficient that can decide the converge distance of short-range interactions.

As the $O(n^2)$ naïve method is inefficient, most of MD implementations use a neighbor list or a cell list to find particles within a radius. For each particle, all particles within a list cutoff r^{list} are added to its neighbor list as shown in Figure 1(a). The value r^{list} is typically larger than r^{cut}, thus, the neighbor list can be reused if none of the particles has moved by a distance of over $\frac{r^{list}-r^{cut}}{2}$. To compute pair-wise interactions, we just need to enumerate particles in the neighbor list of each particle as shown in NEIGHBOR-LIST-PAIR in Algorithm 1.

To build the neighbor list efficiently, many MD implementations export the neighbor list from a cell list. The simulation box is divided into several cells of length $l^{cell} \geq r^{list}$. Particles are then grouped according to cells as shown in Figure 1(b). Thus, a particle can only have neighboring particles in adjacent cells and the neighbor list can be built efficiently with NEIGHBOR-FROM-CELL in Algorithm 1. Some MD implementations also use $l^{cell} \geq \frac{r^{list}}{n}$. Thus, they need to scan n cells in each direction. Cell lists are quite generic and can also be used for finding particles within r^{cut}. A few MD implementations therefore compute pair-wise interactions directly based on cell lists.

B. Related MD Packages

LAMMPS [3] is a fully featured MD package featuring widely supported potentials and optimizations on various platforms. MiniMD [13] is designed for evaluating potential optimizations on new architectures or supercomputers and follows almost the same algorithm as LAMMPS. Both LAMMPS and MiniMD build their particle-to-particle neighbor list based on a cell list. GROMACS [2] builds neighbor lists for clusters which consist of a number of particles for efficient vectorization. AMBER [4] is an MD simulator with its own AMBER potential. Its *pmemd* simulation engine has been optimized on a number of architectures [14] [15] [16] [17], and is currently one of the fastest MD simulation engines for biochemical systems. *pmemd* also computes pair-wise interactions based on neighbor lists, but contains the additional step of sorting atoms according to cells before building the neighbor list. Thus, atoms in the same cell are stored in a continuous interval, which can significantly improve cache efficiency.

We have profiled the runtime decomposition of LAMMPS and *pmemd*:

Application	Testcase	Neighbor%	Pair%
LAMMPS Official	in.lj	16	80
LAMMPS CPE [11]	in.lj	22	56
pmemd	STMV	11	58

As we can see from the table, if there are no intensive many-body interactions, computing pair-wise interactions and building the neighbor list are the largest kernels in MD simulations.

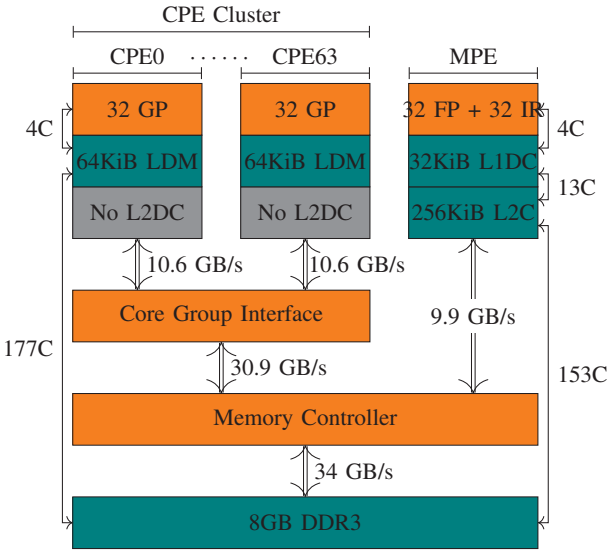


Fig. 2. The memory hierarchy of a CG on SW26010. Arrows with x GB/s show the bandwidth among two parts of the processor. Arrows with x C show the latency cycles among the two levels of memory. GP, FP, and IR represent general-purpose, floating point, and integer registers, respectively.

C. MD on TaihuLight

The TaihuLight supercomputer is equipped with 40,960 SW26010 processors. Each SW26010 processor is divided into 4 *core groups (CG)* and each CG contains a *management processing element (MPE)* and 64 *computation processing elements (CPE)* running at a clock rate of 1.45 GHz. The MPE is a traditional CPU core with two levels of cache while CPEs are simplified cores without data cache. To meet the requirement of low power consumption and high performance, each CPE has a 64KB LDM which is controlled by DMA instructions and can be used as a high-speed buffer for data caching.

The memory hierarchy of SW26010 is shown in Figure 2. Each CG has 8GB DDR3 shared memory. The MPE communicates directly with the memory controller. CPEs access shared memory via a core group interface, either loading at most 32B data directly to registers or use DMA instructions to guide the core group interface fetch/store data from/into its LDM. MPE can access the shared memory with a bandwidth of 9.9 GB/s while CPEs can access the shared memory via DMA instructions with a bandwidth of 10.6 GB/s in the case of exclusive access, but in the case of parallel access, the bottleneck of memory bandwidth is the bandwidth of the DDR3 shared memory (34 GB/s).

As for the compute units, basic SIMD floating point calculations are supported by both MPE and CPEs with a latency of 7 clock cycles, while division and square root need $\sim$ 30 cycles. CPEs support vector shuffle instruction which can select data from two vectors and store them into a new vector. Aligned and unaligned vector loading instructions are both supported by CPEs and MPE .

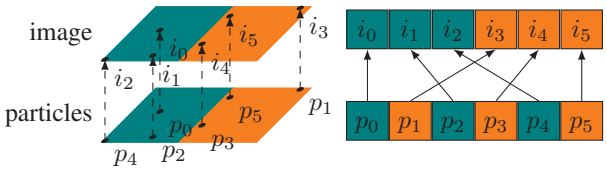


Fig. 3. Particle-to-image mapping: spatial mapping (left) and the corresponding memory mapping (right).

There has been prior work on accelerating MD applications on the TaihuLight. The earliest attempt optimized the *Lennard-Jones* potential of LAMMPS [9] by carrying out a software prefetching technique to handle random memory access from the neighbor list. Another work optimized the non-bonded interaction computation of GROMACS [10]. The authors divide CPEs into cache cores and compute cores. Cache cores simulate a software cache to handle the memory access with indirect addressing from the neighbor list while compute cores perform interaction computation. The most recent work parallelized the *Lennard-Jones* potential and the *Tersoff* potential of LAMMPS [11] by optimizing the software cache performance according to the *average memory access time (AMAT)* [18] model and achieved better performance for the *Lennard-Jones* potential. In addition, a *hybrid memory updating* framework is employed for handling three-body interactions in *Tersoff* potentials. Although the work has gained further speedup over existing implementations on TaihuLight, the performance of pair-wise potential calculation is still far lower compared to peer many-core processors like the Xeon Phi Knights Landing (KNL).

While current TaihuLight implementations usually perform large amounts of random memory accesses, MD simulations actually exhibit spatial locality: nearby particles share quite a few neighboring particles. Cache based architectures may automatically take advantages of the spatial locality. However, for the SW26010 architecture, programmers must invest additional thoughts on how to exploit this kind of locality. While software caches can be a solution, they cannot reach the same performance as hardware ones due to the size of LDM and software overhead.

III. Optimization Methods

A. Optimization based on pmemd's Nonbonded Kernel

The non-bonded kernel in *pmemd* includes a vdW part (Equation (1)) and a Coulomb part (short range part in Equation (3)) which is computed via pair-wise interactions. The workflow of pair-wise computation in *pmemd* differs from LAMMPS and GROMACS. While LAMMPS and GROMACS build their neighbor lists directly from the cell linked lists of particles, *pmemd* introduces a middle layer (image) which is a remapping of particles. After building the cell linked list, *pmemd* puts the particles in the same cell into a continuous interval of the image, and then builds a neighbor list for particles in the image. The mapping of particles to images is shown in Figure 3, where p_* indicates the original index of particles while i_* is the index in the image. We have found that using this kind of middle layer leads to good memory locality.

Algorithm 2 Cell-wise Interactions Computation

Require:
C: all of the cells in the image (stored as image interval $[C.s, C.t]$ and 3-D discrete coordinate $(C.X, C.Y, C.Z)$).
P: particles in the image.
DIST, PAIR-UPD, r^{cut}: same as Algorithm 1.
CELL(X, Y, Z): find the cell with coordinate (X, Y, Z).
Ensure:
F: force status of atoms.

```
1:  function NEIGHBOR(C)
2:      neighbor ← ∅
3:      for z ← C.Z − 2, C.Z do
4:          if z = C.Z then
5:              yhi ← C.Y
6:          else
7:              yhi ← C.Y + 2
8:          for y ← C.Y − 2, yhi do
9:              if z = C.Z ∧ y = C.Y then
10:                 xhi ← C.X
11:             else
12:                 xhi ← C.X + 2
13:             for x ← C.X − 2, xhi do
14:                 neighbor ← neighbor ∪ CELL(x, y, z)
15:     return neighbor
16: F ← 0
17: for I ← 1, |C| do
18:     for J ← NEIGHBOR(C_I) do
19:         for i ← C_I.s, C_I.t do
20:             for j ← C_J.s, C_J.t do
21:                 if I ≠ J ∨ j < i then
22:                     if DIST(P_i, P_j) < r^cut then
23:                         PAIR-UPD(F, P_i, P_j)
```

1) Cell-wise Interactions based on Images: pmemd uses a cell length of $l^{cell} \geq \frac{r^{list}}{2}$. Thus, we need to scan two cells in each direction to find all of the particles that need to interact with particles in a specific cell. The design of our cell-wise interaction computation is shown in Algorithm 2. The outer loop (Line 17) iterates over cells and considers each cell as center cell. The inner loop (Line 18) iterates over its neighboring cell in a half-shell style. For each center cell (C_I), we locate its neighboring cells (all of the C_J) by calling the function NEIGHBOR(C). We then consider pair of particles located inside C_I and C_J for potential interaction. In order to avoid duplicate calculations, we use two strategies:

1) Only include a half-shell region of a center cell in the NEIGHBOR(C) function; i.e., we do iterations only if C_J is *above and to the left* of C_I.
2) If $J = I$, we only calculate pair-wise interactions of particle pairs $j < i$ (Line 21 in Algorithm 2).

An obvious benefit of using cell-wise interactions is that the main loop does not require indirect addressing (except

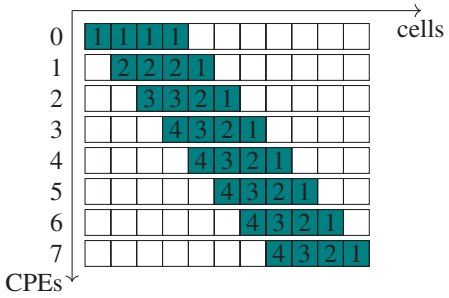


Fig. 4. An example of the inspector. Cells used by the CPE are indicated in green together with the number of replicas that should be used when updating the corresponding cell.

Algorithm 3 Replica arrangement algorithm

Require:

C: all of cells in the image.

OWNER(I): returns the CPE that processes C_I as a center cell.

NEIGHBOR(C): neighbor cells of C as described in Algorithm 2.

Ensure:

N: N_I indicates the number replicas allocated for C_I.

S: $S_{I,p}$ indicates p should initialize its replica of C_I when center cell is $S_{I,p}$.

R: $R_{I,p}$ indicates the replica number to use when p updates C_I.

1: $N \leftarrow 0$
2: $R \leftarrow nil$
3: **for** $I \leftarrow 1, |C|$ **do**
4: $\quad p \leftarrow$ OWNER(I)
5: $\quad$ **for** $J \in$ NEIGHBOR(C_I) **do**
6: $\quad\quad$ **if** $R_{J,p} = nil$ **then**
7: $\quad\quad\quad N_J \leftarrow N_J + 1$
8: $\quad\quad\quad$ allocate N_Jth replica for C_J
9: $\quad\quad\quad R_{J,p} \leftarrow N_J$
10: $\quad\quad\quad S_{J,p} \leftarrow I$

for accessing particle type related parameters). Thus, we do not need to consider how to swap atom data in-and-out. This greatly increases the data reuse ratio. Moreover, the continuous loop is more suitable for vectorization.

2) Adaptive Replica Summation: There can be write-conflicts in the pair-wise interactions with Newton's 3rd law since center particles on two different cores may share a neighboring particle. Thus, particle forces might be updated by different cores. Existing approaches on TaihuLight include: **1) full-replica-summation** [10]: create one replica of forces for each CPE, and do a summation after computation. **2) disabling Newton's 3rd law** [11] [9]: for a pair of particles, calculate the interactions, only update the forces of the center particle, and a pair of particles is calculated twice when either side is the center particle. Both approaches have obvious disadvantages: full-replica-summation occupies a lot of memory; disabling Newton's 3rd law doubles the computational workload.

We find that a number of finite volume (FV) or finite element (FE) applications use an inspector-executor scheme to arrange replicas for conflict-free updates on many-core processors. This scheme firstly runs an inspector procedure which simulates parallel decomposition to detect possible write-conflicts, and generates a schedule to avoid write-conflicts. Subsequently, it runs an executor to perform computation according to the generated schedule generated. This strategy is effective for FV and FE applications when they are based on a static unstructured grid. However, the situation is more difficult in the case of constantly moving particles.

To address this problem, we take the cell-list as a middle layer for the inspector-executor scheme. We run the inspector on the cell level, and arrange replicas with cells as the decomposition unit. Though particles move among cells, the decomposition of cells is kept unchanged which enables the reuse of replica arrangements.

Our replica arrangement algorithm is illustrated in Algorithm 3: We build an array $R_{I,p}$ for each cell C_I and CPE p, which indicates that CPE p uses the $R_{I,p}$th replica of C_I when updating it. The outer loop in Algorithm 3 (Line 3) iterates over each cell C_I. We know CPE $p =$ OWNER(I) will update it as well as its neighbors. The inner loop (Line 5) checks each neighbor of C_I. For each neighbor cell C_J, if the $R_{J,p}$ array is not assigned, we allocate a replica for C_J and assign it to $R_{J,p}$. An example is shown in Figure 4. In practice, $R_{I,*}$ can be bit encoded and $R_{I,p}$ can be calculated via a bit mask and *popcount* instructions.

OWNER(I) can be implemented by various strategies according to the major goal of task decomposition. We can use a round robin distribution for better load balancing, or use block distribution in order to minimize the number of replicas. In the case of block distribution, it needs at most 10 replicas for a cell when 64 CPEs update an image of $44 \times 44 \times 16$ cells and over 1 million particles.

3) Vectorization of Cell-wise Interactions: After optimizing memory accesses, computation becomes the main bottleneck of the pair-wise interaction kernel. Thus, efficient vectorization can significantly improve its performance. The following problems are common when vectorizing pair-wise interactions.

1) Indirect addressing from neighbor lists: Particle data may be loaded according to the particle index stored in the neighbor list.

2) Indirect addressing from particle types: Some of the parameters in the potential vary according to particle types, e.g., ϵ and σ in the *Lennard-Jones* potential.

3) Divergence: There are a number of branch conditions in the interaction calculation (e.g., checking whether the distance is less than the cutoff), which may reduce the utilization ratio of vector lanes.

We have already avoided Problem **1)** by replacing neighbor lists with cell-wise interactions.

Problem **2)** is usually addressed by using *gather* or *shuffle* instructions. However, SIMD *gather* instructions are not as efficient as continuous loading. In addition, *shuffle* operations will add a number of extra instructions. Inspired by an SIMD implementation [19] of the Smith-Waterman algorithm for bi-

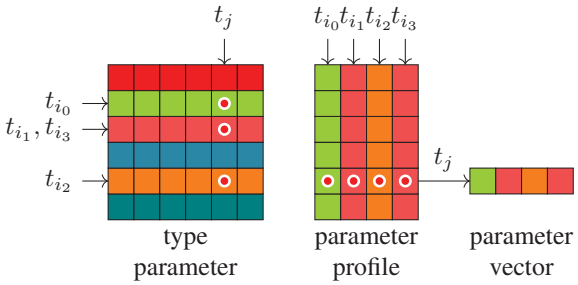


Fig. 5. An example of a parameter profile. We pick the corresponding rows of 4 types $t_{i0}, t_{i1}, t_{i2}, t_{i3}$, and convert them into a parameter profile. When loading a parameter vector for type $t_{i0}, t_{i1}, t_{i2}, t_{i3}$ vs t_j as shown by the red dots, we only need one SIMD instruction.

Algorithm 4 Parameter profile conversion algorithm

Require:

$t_{i0}, t_{i1}, t_{i2}, t_{i3}$: types of 4 i-atoms.

$p_{i,j}$: parameter for calculating pair-wise potential for atom type i, j

$types$: particle types involved in pair-wise potential calculation.

Ensure:

$\hat{p}_j$: parameter profile for calculating pair-wise potential with atom type j with the 4 i-atoms.

1: **for** $j \in type$ **do**

2: $\quad \hat{p}_j \leftarrow \langle p_{t_{i0},j}, p_{t_{i1},j}, p_{t_{i2},j}, p_{t_{i3},j} \rangle$

ological sequence alignment [20] that represents a substitution matrix $S(a_i, b_j)$ by a query-profile, we have designed a novel parameter profile strategy to load parameters for four fixed particles and one arbitrary particle. This is shown in Algorithm 4 and an example is illustrated in Figure 5. When we load four particles from the center cell into a vector register, we pack the parameters p for the particle types $t_{i_0}, t_{i_1}, t_{i_2}, t_{i_3}$ into a parameter profile $\hat{p}$ ($\hat{p}_x = \langle p_{t_{i_0},x}, p_{t_{i_1},x}, p_{t_{i_2},x}, p_{t_{i_4},x} \rangle$ for all $x \in [0, ntypes)$). Thus, when we perform vectorized calculations of these particles and a particle in a neighboring cell with type t_j, we just load the corresponding parameter vector $\hat{p}_{t_j}$ from the parameter profile which can be reused until all particles in the neighboring cell are enumerated.

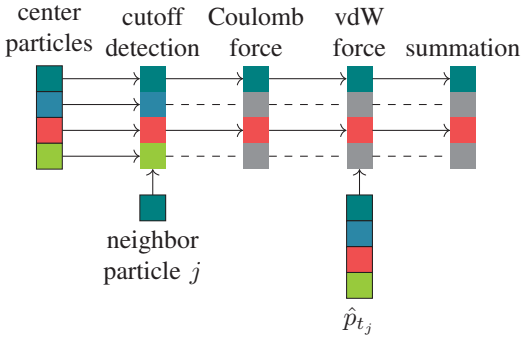


Fig. 6. An example of inter-i-atom vectorization. If any i-atom passes cutoff detection with the j-atom, the coulomb force and vdW force are calculated for the whole vector. We then perform a summation to update the forces. The parameter profile is also involved when calculating vdW forces.

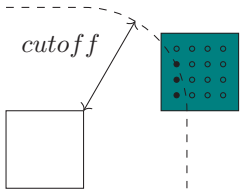


Fig. 7. Only a few atoms interact with far-away buckets. The blue box is the i-bucket, and the other box is a j-bucket. Filled dots in the i-bucket represent atoms that may interact with atoms in the j-bucket.

Algorithm 5 Particle-cell distance algorithm

Require:

$\langle x_o, y_o, z_o \rangle$: Center of a bounding box.

$\langle x_v, y_v, z_v \rangle$: Vertex of a bounding box.

$\langle x_p, y_p, z_p \rangle$: Coordinates of a particle.

Ensure:

d: Shortest distance between particle and bounding box.

1: $\vec{h} \leftarrow \langle x_v - x_o, y_v - y_o, z_v - z_o \rangle$

2: $\vec{v'} \leftarrow \langle x_p - x_o, y_p - y_o, z_p - z_o \rangle$

3: $\vec{v} \leftarrow \langle |\vec{v'}.x|, |\vec{v'}.y|, |\vec{v'}.z| \rangle$ ▷ map P to the first quadrant.

4: $\vec{u'} \leftarrow \vec{v} - \vec{h}$

5: $\vec{u} \leftarrow \langle \max(\vec{u'}.x, 0), \max(\vec{u'}.y, 0), \max(\vec{u'}.z, 0) \rangle$

6: $d \leftarrow |\vec{u}|$

Note that when performing vectorized calculations using the parameter profile, there should be four fixed particles, which can be four particles from the center cell (center particles). Then we can iterate over the particles in the neighboring cell (neighboring particle). An example of our vectorization is shown in Figure 6, we first do a cutoff check between center particles and the neighboring particle, then Coulomb forces and vdW forces are calculated, when calculating vdW forces, the parameter profile is used for better performance.

Figure 6 shows our approach to address Problem **3)**. If a center particle cannot pass the cutoff check, we leave the corresponding lane empty (dashed lines in Figure 6). In order to reduce the number of empty lanes as much as possible, we employ two optimizations:

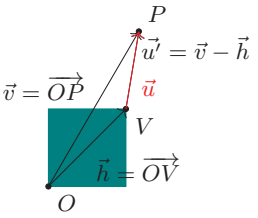
1) Shortcuts: We skip branches, if all of the lanes in a vector are empty by using an integer matching instruction of SW26010 to find if there is a non-empty lane in a vector register.

2) Sorting particles: We divide cells into $8 \times 8 \times 8$ sub-cells and generate a Hilbert space-filling curve to encode these sub-cells. We then sort particles by encoding their sub-buckets. Therefore, particles which are close in the 3-D space will also have a nearby 1-D index. Thus, data in a vector is more likely to go through the same branches.

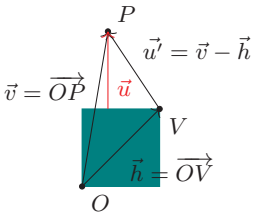
4) Particle-cell Cutoff Checking Filter: Performance differences between the cell list method and the neighbor list method can be explained by the successful check ratio, which we define as:

$$\eta = \frac{T_{success}}{T_{check}} \tag{4}$$

In Equation (4), $T_{success}$ denotes the number of pairs that close enough for pair-wise interactions calculation and T_{check}



(a) A case where all components of $\vec{u'}$ are not less than 0.



(b) A case where one component of $\vec{u}$ is less than 0.

Fig. 8. Two examples of the particle-cell distance algorithm. Point O is the center of the bounding box of a cell, Point V is the vertex of the bounding box, and Point P is the particle. $\vec{u}$ is the shortest vector from P to the bounding box.

denotes the number of pairs that need cutoff checking. Assuming a uniform particle coordinate distribution, we can estimate η as follows: $\eta_{neigh} = (\frac{r^{cut}}{r^{list}})^3$ for the neighbor list method and $\eta_{cell} = \frac{4}{3}\pi(\frac{r^{cut}}{(2n+1)l^{cell}})^3$ when $l^{cell} \geq \frac{r^{cut}}{n}$ for the cell list method. For the common parameters, $r^{list} \simeq 1.11r^{cut}$ and $n = 2$, we have $\eta_{neigh} \simeq 0.73$ and $\eta_{cell} \leq 0.27$.

We have found that quite a few particles do not interact with any particles in the neighboring cell, especially when we take $n = 2$ as shown in Figure 7. We introduce a particle-cell cutoff check filter to reduce the number of pair-wise cutoff checks. Before calculating cell-wise interactions, we compute the axis aligned bounding box of each cell which is the minimum box which has edges parallel to the axes that cover all particles in that cell. For each particle and a neighboring cell, we first calculate the shortest distance from the particle to the bounding box of the neighboring cell. If the shortest distance is larger than the cutoff, we can just skip all the distance checks between the particle and the neighboring cell.

We implement a particle-cell distance check module based on a box-sphere intersection test algorithm [21]. The idea of this algorithm is as follows. Let the center of the box O be the origin of axes, $V(x_v, y_v, z_v)$ be the vertex of the box in the first quadrant, and assume that the particle is at $P(x_p, y_p, z_p)$ which is located in the first quadrant. For a point $X(x', y', z')$ in the box, its distance to P is

$$d = \sqrt{(x' - x_p)^2 + (y' - y_p)^2 + (z' - z_p)^2}.$$

Because the box is parallel to the axes, the three terms $(x' - x_p)^2$, $(y' - y_p)^2$ and $(z' - z_p)^2$ can be minimized independently. Thus, we can use greedy strategy. Taking $(x' - x_p)^2$ as an example, we have $x' \in [0, x_v]$, so we can set $x' = x_p$ if $x_p < x_v$, otherwise we pick $x' = x_v$ which can minimize $(x' - x_p)$. The same strategy can be applied for y' and z', then X is $(\min(x_p, x_v), \min(y_p, y_v), \min(z_p, z_v))$. The shortest vector from the box to the particle is $\vec{u} = \langle \max(0, x_p - x_v), \max(0, y_p - y_v), \max(0, z_p - z_v) \rangle$. If P is not in the first quadrant, we just take $P' = (|x_p|, |y_p|, |z_p|)$ because mirroring does not affect the distance. A simplified series of operations is presented in Algorithm 5.

5) Overview of Our Optimized Non-bonded Kernel: Our non-bonded kernel is shown in Algorithm 6. It contains a loop that iterates over cells and its neighbor in order to calculate

Algorithm 6 Overview of our pair-wise interactions kernel

Require:
pid: id of current CPE.
C, P, r^{cut}: the same as Algorithm 1.
R, OWNER: the same as Algorithm 3.
ϵ, σ: parameters for vdW force.
NEIGHBOR(bkt): find the neighbor bucket of bkt.
DIST: find the shortest distance between particles or particle to cell.
PROFILE$(\epsilon, \sigma, t_0, t_1, t_2, t_3)$: build parameter profile of parameter ϵ and σ for 4 particle types t_0, t_1, t_2, t_3.
PAIR(VPI, VPJ, PV): calculate forces between particles in vector VPI and VPJ and preprocessed parameter vector PV.

Ensure:
F^*: replicas for force status of atoms.

1: **for** $I \in \{x | x \in [1, |C|] \wedge \text{OWNER}(x) = pid\}$ **do** ▷ Cell-wise loop
2: fetch particles from $C_I.s$ to $C_I.t$ to LDM.
3: $Fi \leftarrow 0$
4: **for** $J \in \text{NEIGHBOR}(C_I)$ **do** ▷ Neighbor loop
5: $rid \leftarrow R_{J,pid}$
6: fetch particles from $C_J.s$ to $C_J.t$ to LDM.
7: fetch forces from $C_J.s$ to $C_J.t$ in F^*_{rid} to LDM.
8: $i \leftarrow C_I.s$
9: **while** $i \leq C_I.t$ **do** ▷ Iterate particles in C_I
10: $nvec \leftarrow 0$
▷ Particle-cell distance filter, and select 4 for vectorization:
11: **while** $nvec < 4 \wedge i <= C_I.t$ **do**
12: **if** $\text{DIST}(P_i, C_J) < r^{cut}$ **then**
13: $\hat{i}_{nvec} \leftarrow i$
14: $nvec \leftarrow nvec + 1$
15: $i \leftarrow i + 1$
16: $\hat{p} \leftarrow \text{PROFILE}(\epsilon, \sigma, P_{\hat{i}_0}.t, P_{\hat{i}_1}.t, P_{\hat{i}_2}.t, P_{\hat{i}_3}.t)$
17: $VPI \leftarrow \langle P_{\hat{i}_0}, P_{\hat{i}_1}, P_{\hat{i}_2}, P_{\hat{i}_3} \rangle$ ▷ Vector form selected particles
18: $VFI \leftarrow 0$ ▷ Vector form of temporary Fi
19: **for** $j \leftarrow C_J.s, C_J.t$ **do**
20: $VPI \leftarrow \langle P_j, P_j, P_j, P_j \rangle$ ▷ Convert P_j to vector
21: $Dij \leftarrow \text{DIST}(VPI, VPJ)$
22: **if** $\exists Dij < r^{cut}$ **then** ▷ Shortcut in III.A-3
23: $PV \leftarrow \hat{p}_{(P_j.type)}$ ▷ Inter-atom parameters
24: $\Delta F \leftarrow \text{PAIR}(VPI, VPJ, PV)$
25: $VFI \leftarrow VFI + \Delta F$
26: $F^*_{rid,j} \leftarrow F^*_{rid,j} - \text{SUM}(\Delta F)$
27: scatter and update VFI to Fi
28: **if** $I = J$ **then** ▷ assert: $I = J \Leftrightarrow J$ is last neighbor
29: add Fi to F^*_{rid} ▷ Merge force of C_I to C_J
30: store forces from $C_J.s$ to $C_J.t$ in F^*_{rid} to DDR.

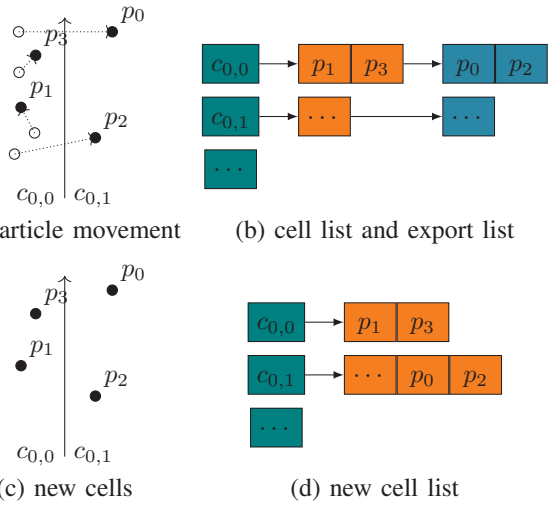


Fig. 9. An example of our incremental cell list rebuild strategy. (a) Particle movement: hollow points are the particles in previous step and solid points are the particles in the current step. (b) Cell list and export list. (c) Status of new cells. (d) New cell list.

pair-wise interactions. The replica allocation result is used in Line 5. The particle-cell filter is applied in Line 12. The parameter profile is built in Line 16 and used in Line 23.

The proposed optimization techniques are quite generic and can be summarized as follows:

- Adaptive replica-summation can provide better parallel efficiency with respect to thread-level parallelization.
- Parameter profiles can be implemented on any modern CPU for better vectorization efficiency.
- The particle-cell cutoff filter can reduce the computational workload in pair-wise interactions and help to build neighbor lists of classical MD applications.

B. Optimizations of ESMD

We have established a minimum set pair-wise interactions related kernel in a standalone framework, called *ESMD*, which includes our methods for cell list building and pair-wise interactions, as well as the integration in a velocity-Verlet scheme [22] like LAMMPS.

1) Incremental Cell List Building: Building a cell list on many-core processors can be difficult since particles may be located in any cell according to their coordinates. Thus, we need mutex variables or atomic operations to avoid write conflicts. An existing implementation in *pmemd.CUDA* [15] uses parallel sort on all particles to avoid such situations.

We find that particles move continuously, i.e, a particle from one cell can only move to an adjacent cell. Thus, rebuilding of cell lists can be performed locally by utilizing the previous cell list. We have thus implemented an incremental cell list building scheme to handle the parallel construction of cell lists.

Incremental cell list building contains two stages as shown in Figure 9: **1) export stage:** for each cell, we scan its particles and find the particles that are no longer within the cell, remove those particles and export them to an export list. **2) import stage:** for each cell, we scan the particles in the export list of its adjacent cells, and import the particles that moved into the cell. Therefore, global operations in cells are no longer needed and thus can be done in parallel without write conflicts.

2) Cell-based Packing: Communication between processes in MD applications is another problem for heterogeneous systems since we need to gather particle data to a send buffer and scatter particle data from a receive buffer to random positions in the particle data.

Using cell lists, we can replace the original gather and scatter by a series of memcopies so that we can efficiently exploit the available memory bandwidth. Incorporated with incremental cell list building, we also exchange exported particles to neighboring processes for particles that move across processes.

TABLE I
CONFIGURATION OF OUR TEST CASES.

Benchmark	# atoms	r^{cut}	$r^{switch\dagger}$	fix‡
HMR	1,147,732	12.0Å	10.0Å	NVT
STMV	1,067,095	9.0Å	None	NVE
STMV-12	1,067,095	12.0Å	None	NVE

† force of particle pairs in $[r^{switch}, r^{cut}]$ is multiplied by a smooth function which equals to 1 at r^{switch} and 0 at r^{cut}.
‡ indicates that the simulation has some constant property. N, V, E, T are number of atoms, volume, total energy and temperature, respectively.

IV. Performance Evaluation

We have evaluated both our optimized AMBER and the stand-alone ESMD framework. AMBER is evaluated on a single node on different platforms. The performance of ESMD is compared with the single node performance of LAMMPS. Additionally, the scalability of ESMD is evaluated for up to 1,024 nodes. All applications are executed in MPI+CPE mode, which dispatch 4 processes per node. Reported timing is the average of 3 runs. The kernel occupation is calculated as $\sum T_{kernel}/\sum T_{total}$ for 3 runs so that we can keep a total occupation of 100%, where T_{kernel} is run time of the kernel and T_{total} is the run time of the whole application.

A. AMBER

Our optimization is based on the officially released symmetric multiprocessor (SMP) version of the *pmemd.MPI* code in the official AMBER 2016 release. In our experiments, the three test cases shown in Table I have been used. *HMR* is a real-world scenario which covers most code paths and is used for validating optimizations. *STMV* is an official benchmark of AMBER with 9Å as the cutoff value. We have also run *STMV* with a cutoff of 12Å (*STMV-12*) because longer cutoffs are often used to improve the stability of simulations in practice.

We use the performance of the officially released SMP code running on 4 MPEs of an SW26010 as baseline. Figure 10 shows incremental performance evaluations with different optimization techniques on a single TaihuLight node. We can see that the SMP code running on 4 MPEs has roughly the same performance as one E5-2650v4 core. After refactoring the code to cell list based interactions, performance slightly decreases. Parallelization of the interaction computation improves performance compared to MPE-only by a factor of 12,

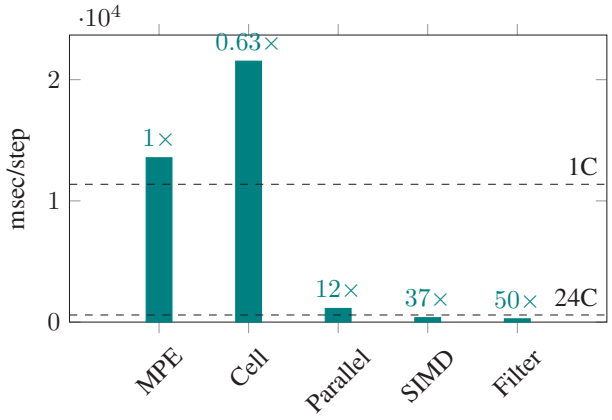


Fig. 10. Seconds per timestep of *HMR* for different optimization steps (*y*-axis is logarithmically scaled): *MPE* is the performance of the optimized *pmemd.MPI* running on 4 MPEs. *Cell* is the cell list based pair-wise interaction running on 4 MPEs. *Parallel*, *SIMD* and *Filter* are the cell list version running on 4 Core Groups with the basic parallelization, vectorization, and fast intersection filter optimizations, respectively. The two horizontal lines are the performance of the optimized *pmemd.MPI* running on a dual E5-2650v4 processor with one core (1C) and 24 cores (24C) respectively. The relative performance over MPE of 1C and 24C are 1.16× and 23.1×, respectively.

TABLE II
PLATFORMS INVOLVED IN THE COMPARISON.

Expr	Platform	Version†	Compiler	Kernel
MPE	SW26010	SMP	swcc 5.421-sw-500	3.8.0
KNL	Xeon Phi 7210	MIC2	icc 18.0.3	3.10.0
X86	E5-2650v4 × 2	SMP	icc 18.0.0	3.10.0
CPE	SW26010	CPE	swcc 5.421-sw-500	3.8.0

† The three versions *SMP*, *MIC2*, and *CPE* indicate the officially optimized *pmemd.MPI* code for symmetric multiprocessing CPUs, officially released code for Xeon Phi 72x0 processors and our optimized version, respectively.

but is still not faster than multi-threaded AMBER running on a dual E5-2650v4 CPU with 24 cores. SIMD vectorization, and particle-cell cutoff check filter optimizations to the code leads to significant speedups. The best speedup over the officially released SMP code running on 4 MPEs is around 50×.

Furthermore, we have compared the performance of our

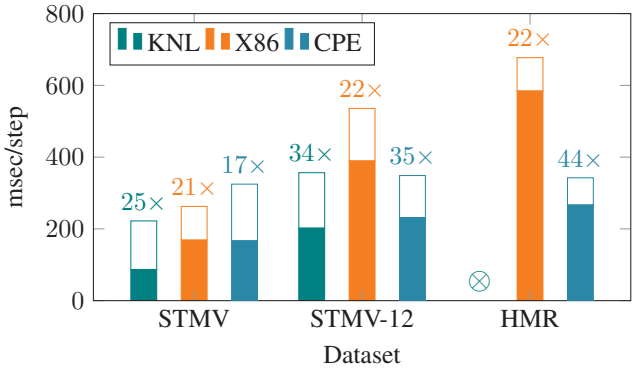


Fig. 11. Performance comparison to other platforms: A white segment in bars indicates the time occupied by neighbor list building or image building. We have excluded the performance on MPE in this figure since the bar is too high, and multipliers above the bar is the relative performance over MPE.

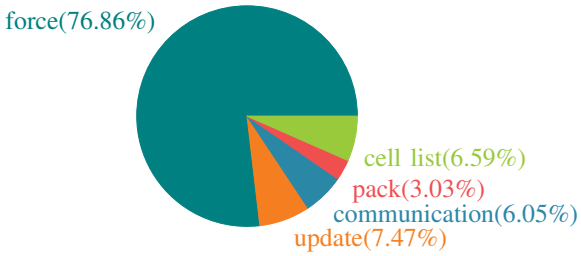


Fig. 12. Profiling of ESMD on a single SW26010 node with 16,384,000 particles.

implementation with other popular AMBER implementations. Table II shows the configuration of these platforms.

Figure 11 shows that our implementation does not have advantages when the cutoff is short (*STMV*). This can be explained by the high frequency of image building ($\sim$ 3 timesteps per image build) and the small bucket size. When the cutoff is larger (*STMV-12*), our implementation is slightly faster than KNL. When r^{switch} is set, KNL can not produce a correct result, and our work takes only half of the computation time compared to 24 E5-2650v4 cores (2 sockets) per timestep.

Because pair-wise interactions do not have a high computation to memory-access ratio, previous work on pair-wise interactions on SW26010 do not have comparable performance compared with KNL. For example, KNL has a peak Flops/s rate (3.072T for 7210) comparable to SW26010 (3.06T), but an implementation reported in the most recent previous work [11] only achieves one-fifth of the performance of a KNL for pair-wise interaction calculation due to a low data reuse ratio. Thus, our current work achieves much better performance than previous work.

B. ESMD

ESMD is evaluated on the TaihuLight supercomputer using the *swcc 5.421-sw-500* compiler, TaihuLight's customized Linux kernel with version number 3.8.0, and TaihuLight's customized MPI library with version number 2.2a. The experiment is configured with face-centered-cubic lattice (*fcc*) and a lattice constant at 0.8442, which is identical LAMMPS's official benchmark. The potential function implementation is the same as LAMMPS's pair/lj/cut style. The cutoff radius and number of particles differ in each experiment. The cell length is configured equal to r^{cut}.

1) Single node evaluation: We have profiled ESMD with 16,384,000 particles with a cutoff radius of 3.2 on a single node with $l^{cell} = r^{cut}$. The profiling result in Figure 12 shows that the pair-wise interaction kernel is responsible for more than 75% of the execution time. This means that rebuilding the cell list with our incremental building method is efficient.

Since the intremental cell-list build is a new part in ESMD, we also have profiled it with different number of particles in a single node with cutoff distance of 2.5 and 3.2. The result is shown in Figure 13. We can see the incremental cell list building has better performance when r^{cut}=3.2, that is because there are less cells in the system, and the possibility for a particle leaving a cell is lower because of larger cell length.

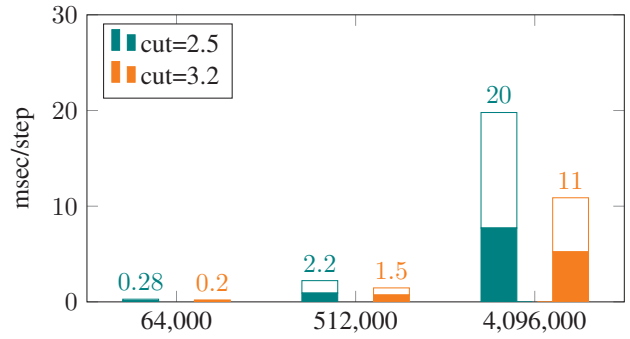


Fig. 13. Execution time of cell list building part with different number of particles per node. The number under X axis indicates the number of particles. Filled segment indicates the export stage and white segment indicates the import stage.

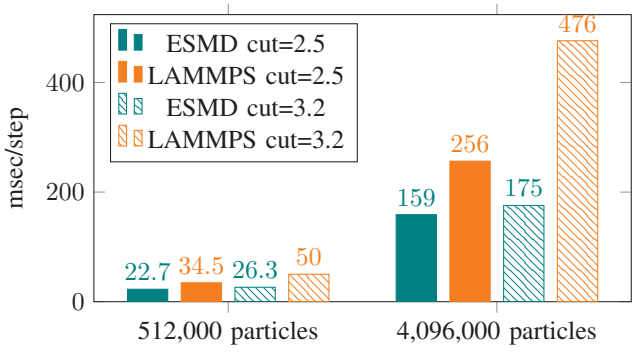
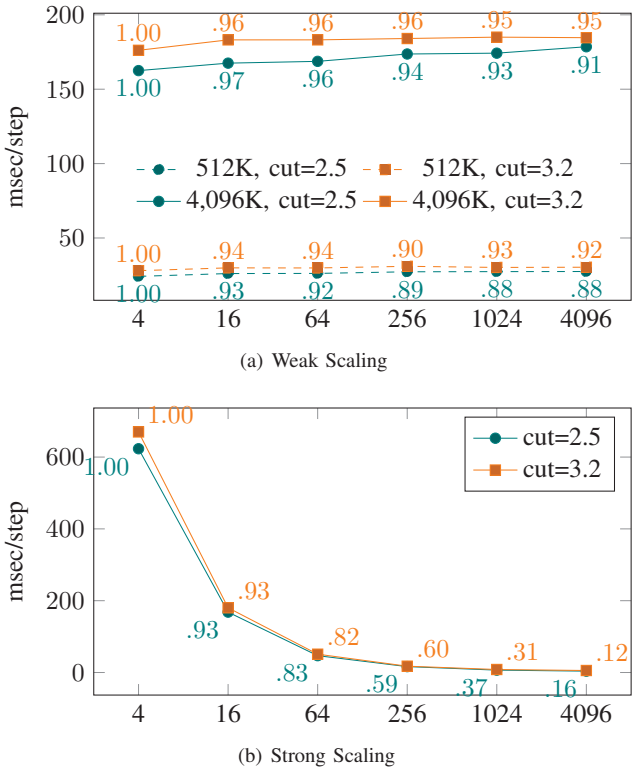


Fig. 14. Performance of ESMD and LAMMPS on a single TaihuLight node.



(a) Weak Scaling

(b) Strong Scaling

Fig. 15. Weak scaling and strong scaling result of ESMD. Axes of strong scaling are logarithmic scaled. X-axis is number of processes. The number near nodes is scaling efficiency.

This indicates we can introduce a *skin* like neighbor list based approach to get better performance when the cutoff radius is short.

We have evaluated the performance of the *Lennard-Jones* potential of ESMD in comparison to the existing LAMMPS [11] optimization[1] on a single TaihuLight node. Both ESMD and LAMMPS are configured as *fcc* lattice with a density of 0.8442, and with the same lattice dimension. We have selected cutoff distances of 2.5 and 3.2, and lattice dimension of $80 \times 40 \times 40$ (512,000 particles) and $160 \times 80 \times 80$ (4,096,000 particles) for comparison.

As shown in Figure 14, ESMD is at least 50% faster than LAMMPS. When there are more particles or the cutoff is longer, the performance difference increases. For the simulation of 4,096,000 particles with a cutoff of 3.2, ESMD is even 170% faster than LAMMPS. This is because the particle distribution is more random when the simulation box is larger and the cell list based computation has a more stable memory accessing strategy. Longer cutoff distances also make the particle-cell cutoff checking filter more effective.

2) Scalability: Figure 15 shows the results of our scalability evaluation of ESMD for up to 1,024 TaihuLight nodes. For weak scaling, we keep the number of particles per process constant at 512,000 or 4,096,000. For strong scaling, we have fixed the total number of particles at 16,384,000. Strong scaling begin to drop at 256 processes. Simulations with cutoff of 3.2 drops faster than simulations with cutoff of 2.5. This is because a larger cutoff leads to a larger ghost region, and the ratio of local particles drops faster. Weak scaling is almost linear when a node has 4,096,000 particles. Note that weak scaling results are better when the cutoff is 3.2, since computation takes longer for a larger cutoff radius.

[1]based on LAMMPS 11-Aug-2017, available at https://github.com/dxhisboy/lammps-sunway. Optimization for TaihuLight has no further change after that.

V. Conclusion

In this paper, we have presented a cell-list-based method for pair-wise interactions on the SW26010 processor. Compared to previous approaches that have optimized MD simulations on the SW26010 processor, we overcome the data reuse problems by using a cell list based method. We have introduced a series of new optimization methods for the CPEs computation procedure including the parameter profile and the particle-cell cutoff checking filter. Experiments show that our redesign of AMBER on a single SW26010 processor achieves 50 fold speedup compared to the official, optimized SMP version of AMBER executed on MPEs. Moreover, our implementation on a single SW26010 node can provide comparable performance to a Xeon Phi 7210 processor and can outperform two E5-2650v4 processors with 24 cores. Our ESMD framework can gain over 50% speedup over the most recent previous approach on SW26010 and can achieve a weak scaling efficiency of up to 95% on 1,024 TaihuLight nodes.

While the presented implementation is specific to the TaihuLight architecture, the proposed optimization methods are generic and are therefore also applicable to other modern architectures. The adaptive replica-summation can provide high parallel efficiency for thread-level parallelization. The parameter profile enables the efficient vectorization of MD applications. The particle-cell cutoff checking can reduce the computation workload in pair-wise interactions based on cell lists and can also help to build the neighbor lists of classical MD applications. Furthermore, our strategy to build incremental cell lists is well-suited for many-core platforms with low random memory access performance.

We envisage possible future enhancements of ESMD with respect to the following aspects: 1) Addition of long-ranged force support such as PME and MSM that takes advantage of the data locality of cell lists to speedup the interpolation procedure. 2) Incorporation of $skin$ support so that we do not need to rebuild the cell list each time a particle moves. This would also be useful for supporting listed forces such as bonds, valance angles and torsion angles in bio-molecular systems. 3) Adaption to other architectures such as the future series of Sunway supercomputers and GPUs. Furthermore, we are considering the implementation of bond-order potentials like Tersoff potential and ReaxFF based on the presented cell-list-based method.

ACKNOWLEDGEMENT

We would thank all anonymous reviewers of this paper for their valuable suggestions. This work is partially supported by NSFC Grants 61972231, 61672312, 41374113, 91530323, 61962051, 51761135015, U1806205, U1839206; National Key R&D Program of China (grant no. 2016YFA0602200, and 2017YFA0604500); the Key Project of Joint Fund of Shandong Province (Grant No. ZR2019LZH007); the Shenzhen Basic Research Fund (Grant No. JCYJ20180507182818013); the PPP project from CSC and DAAD; Center for High Performance Computing and System Simulation, Pilot National Laboratory for Marine Science and Technology (Qingdao). Corresponding authors of this paper are: Haohuan Fu (haohuan@tsinghua.edu.cn), Lin Gan (lingan@tsinghua.edu.cn), Wei Xue (xuewei@tsinghua.edu.cn), Weiguo Liu (weiguo.liu@sdu.edu.cn).

REFERENCES

[1] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, "Scalable molecular dynamics with namd," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[2] H. J. Berendsen, D. van der Spoel, and R. van Drunen, "Gromacs: a message-passing parallel molecular dynamics implementation," *Computer physics communications*, vol. 91, no. 1-3, pp. 43–56, 1995.

[3] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of computational physics*, vol. 117, no. 1, pp. 1–19, 1995.

[4] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, "The amber biomolecular simulation programs," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1668–1688, 2005.

[5] J. E. Jones, "On the determination of molecular fields.–ii. from the equation of state of a gas," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 106, no. 738, pp. 463–477, 1924.

[6] J. Tersoff, "New empirical approach for the structure and energy of covalent systems," *Physical Review B*, vol. 37, no. 12, p. 6991, 1988.

[7] M. S. Daw and M. I. Baskes, "Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals," *Physical Review B*, vol. 29, no. 12, p. 6443, 1984.

[8] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, W. Zhao, X. Yin, C. Hou, C. Zhang, W. Ge, J. Zhang, Y. Wang, C. Zhou, and Y. Guangwen, "The sunway taihulight supercomputer: system and applications," *Science China Information Sciences*, vol. 59, no. 7, p. 072001, 2016.

[9] W. Dong, L. Kang, Z. Quan, K. Li, K. Li, Z. Hao, and X.-H. Xie, "Implementing molecular dynamics simulation on sunway taihulight system," in *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2016, pp. 443–450.

[10] Y. Yu, H. An, J. Chen, W. Liang, Q. Xu, and Y. Chen, "Pipelining computation and optimization strategies for scaling gromacs on the sunway many-core processor," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2017, pp. 18–32.

[11] X. Duan, P. Gao, T. Zhang, M. Zhang, W. Liu, W. Zhang, W. Xue, H. Fu, L. Gan, D. Chen, X. Meng, and Y. Guangwen, "Redesigning lammps for peta-scale and hundred-billion-atom simulation on sunway taihulight," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 148–159.

[12] P. P. Ewald, "Die berechnung optischer und elektrostatischer gitterpotentiale," *Annalen der physik*, vol. 369, no. 3, pp. 253–287, 1921.

[13] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving performance via mini-applications," *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, vol. 3, 2009.

[14] A. W. Götz, M. J. Williamson, D. Xu, D. Poole, S. Le Grand, and R. C. Walker, "Routine microsecond molecular dynamics simulations with amber on gpus. 1. generalized born," *Journal of chemical theory and computation*, vol. 8, no. 5, pp. 1542–1555, 2012.

[15] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, and R. C. Walker, "Routine microsecond molecular dynamics simulations with amber on gpus. 2. explicit solvent particle mesh ewald," *Journal of chemical theory and computation*, vol. 9, no. 9, pp. 3878–3888, 2013.

[16] S. Le Grand, A. W. Götz, and R. C. Walker, "Spfp: Speed without compromise–a mixed precision model for gpu accelerated molecular dynamics simulations," *Computer Physics Communications*, vol. 184, no. 2, pp. 374–380, 2013.

[17] P. Needham, A. Bhuiyan, and R. C. Walker, "Extension of the amber molecular dynamics software to intel's many integrated core (mic) architecture," *Computer Physics Communications*, vol. 201, pp. 95–105, 2016.

[18] W. A. Wulf and S. A. Mckee, "Hitting the memory wall: implications of the obvious," *ACM Sigarch Computer Architecture News*, vol. 23, no. 1, pp. 20–24, 1995.

[19] M. Farrar, "Striped smith–waterman speeds database searches six times over other simd implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2006.

[20] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[21] J. Arvo, "A simple method for box-sphere intersection testing," in *Graphics gems*. Academic Press Professional, Inc., 1990, pp. 335–339.

[22] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson, "A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters," *The Journal of Chemical Physics*, vol. 76, no. 1, pp. 637–649, Jan. 1982. [Online]. Available: https://doi.org/10.1063/1.442716

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

We ran the original version and optimized version of PMEMD in AMBER16 on the Sunway TaihuLight supercomputer. Input dataset are introduced in the paper. For comparison, we also ran PMEMD of AMBER16 on Xeon Phi 7250 and E5-2650 v4.

We ran the optimized version of ESMD on the Sunway TaihuLight supercomputer, scaling from 1 to 1024 nodes, configurations are with FCC lattice and 1000 timesteps, scale of input is described in the paper. For comparison, we also ran an optimized version of LAMMPS which is available at https://www.github.com/dxhisboy/lammps-sunway.

ARTIFACT AVAILABILITY

Software Artifact Availability: Some author-created software artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Hardware Artifact Availability: There are no author-created hardware artifacts.

Data Artifact Availability: Some author-created data artifacts are NOT maintained in a public repository or are NOT available under an OSI-approved license.

Proprietary Artifacts: There are associated proprietary artifacts that are not created by the authors. Some author-created artifacts are proprietary.

Author-Created or Modified Artifacts:

```
Persistent ID: https://doi.org/10.5281/zenodo.3859321
Artifact name: Source Code and Guide for ESMD

Persistent ID: https://doi.org/10.5281/zenodo.3859319
Artifact name: Non-proprietary Part of our Optimized
↪  Amber
```

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: Sunway TaihuLight supercomputer, with SW26010 processor.

Operating systems and versions: Customized Linux with kernel version 3.8.0

Compilers and versions: SWCC Compilers: Version 5.421-sw-500

Applications and versions: AMBER 2016, ESMD

Libraries and versions: athread/mvapich-2.2

Key algorithms: molecular dynamics

Input datasets and versions: STMV in AMBER gpu benchmarks, Unique L-J input with FCC lattice