MD-HM: Memoization-based Molecular Dynamics Simulations on Big Memory System

Zhen Xie
zxie10@ucmerced.edu
University of California, Merced

Wenqian Dong
wdong5@ucmerced.edu
University of California, Merced

Jie Liu
jliu279@ucmerced.edu
University of California, Merced

Ivy Peng
peng8@llnl.gov
Lawrence Livermore National Laboratory

Yanbao Ma
yma5@ucmerced.edu
University of California, Merced

Dong Li
dli35@ucmerced.edu
University of California, Merced

ABSTRACT

Molecular dynamics (MD) simulation is a fundamental method for modeling ensembles of particles. In this paper, we introduce a new method to improve the performance of MD by leveraging the emerging TB-scale big memory system. In particular, we trade memory capacity for computation capability to improve MD performance by the lookup table-based memoization technique. The traditional memoization technique for the MD simulation uses relatively small DRAM, bases on a suboptimal data structure, and replaces pair-wise computation, which leads to limited performance benefit in the big memory system. We introduce MD-HM, a memoization-based MD simulation framework customized for the big memory system. MD-HM partitions the simulation field into subgrids, and replaces computation in each subgrid as a whole based on a lightweight pattern-match algorithm to recognize computation in the subgrid. MD-HM uses a new two-phase LSM-tree to optimize read/write performance. Evaluating with nine MD simulations, we show that MD-HM outperforms the state-of-the-art LAMMPS simulation framework with an average speedup of 7.6× based on the Intel Optane-based big memory system.

CCS CONCEPTS

• **Computing methodologies** → **Modeling and simulation**; Computer graphics; • **Information systems** → *Data structures*.

KEYWORDS

Molecular Dynamics, Memoization, Big Memory, Key-Value Store, Moment Algorithm

ACM Reference Format:
Zhen Xie, Wenqian Dong, Jie Liu, Ivy Peng, Yanbao Ma, and Dong Li. 2021. MD-HM: Memoization-based Molecular Dynamics Simulations on Big Memory System. In *2021 International Conference on Supercomputing (ICS '21), June 14–17, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3447818.3460365

ICS '21, June 14–17, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8335-6/21/06.
https://doi.org/10.1145/3447818.3460365

1 INTRODUCTION

Molecular dynamics (MD) simulation computes the interaction between a collection of particles. It is a common and general computational method, and gives scientists the ability to track particles motion. MD plays important roles in various fields [9, 22, 40, 50, 85]. such as computational chemistry [34] and materials science [78], bio-informatics [11], high performance applications [18, 57], etc. For example, MD is an indispensable tool to interpret long time-scale trajectories of relevant bio-molecular system for new drugs and vaccines discovery [9, 40, 50]; MD at the nanoscale is often used in semiconductor and integrated circuit design [22, 85] to study thermodynamic properties. By performing MD simulation, these systems and their thermodynamic properties can be obtained more easily compared with experiments [36].

MD is typically compute-bound and not bounded by memory bandwidth or capacity [39]. In particular, MD usually involves a large number of particles; It iteratively computes energies and forces between particles based on the computation of inter-particle *potentials*. The calculation of potentials dominates the simulation time (at least 90% of the total time) and has high computation intensity (4.6-71.5 flops per byte) [29]. This calculation is based on a data structure of a few bytes to represent a particle, consuming small memory even for a large-scale simulation. For example, the bulk silicon simulation for 1M particles consumes only 3.6 GB memory, which is far less than typical memory capacity in a node. The traditional performance optimization on MD focuses on increasing instruction-level and thread-level parallelism by loop vectorization, data alignment, and structure transformation [3, 25, 49]. Such performance optimization is bounded by processor's theoretical peak performance.

In this paper, we introduce a new method to improve performance of MD by leveraging the emerging *big memory system*. In particular, we trade memory capacity for computation capability to improve MD performance by memoization. This method is motivated by the emergence of big memory systems. Such a big memory system is exemplified by the recent release of Intel Optane DC persistent memory module (PMM), which is able to provide up to 9TB main memory in a single machine. Such a big memory system cannot be leveraged by the traditional performance optimization on MD because of MD's small memory consumption and compute-boundness, but using memoization, we are able to transform large memory capacity into performance benefit.

The memoization technique, in nature, stores results of expensive computation to a data structure, such as a lookup table, such that when the same input happens, the results can be returned without performing expensive computation. Existing work in MD builds a small lookup table (hundreds of MB to tens of GB) on DRAM to store pre-computed results [32, 35, 47, 54, 79]. Those efforts cannot work well when applied to the big memory systems, because of the following reasons. These reasons fundamentally limit the feasibility of using the big memory to accelerate MD simulation.

First, the existing efforts replace the calculation of potential of each pair of particles, which brings limited performance benefit on the big memory systems. To ensure the performance benefit of using memoization, the memory access latency to use the lookup table must be smaller than the calculation of potential to be replaced. Depending on processor architecture and particle-based model in the MD simulation, the calculation of potential for each pair of particles is at the range of tens of nanoseconds, which is smaller or comparable to one-time search of the look up table on the traditional DRAM. However, on the big memory platform whose most capacity comes from slow memory (e.g., Optane DC persistent memory) with the latency of a few microseconds for one-time search, replacing the calculation of potential for a pair of particles with a search in the lookup table cannot have performance benefit.

Second, the existing efforts limit the size of lookup table to tens of GB due to the limited DRAM capacity, and the search performance in the lookup table is not optimized for the TB-scale of the big memory. In particular, the existing efforts employ a one-dimensional array. Such a data structure is not efficient to handle search and insertion operations for a large scale lookup table, hence shrinks the performance benefit brought by memoization on the big memory systems.

Third, the existing efforts build the lookup table before MD simulation. The table is loaded from hard drive at runtime. While this method is feasible for a small lookup table, it causes rather large storage cost for a TB-scale lookup table.

In this paper, we introduce a new memoization methodology to accelerate MD simulation to address the above problems. In particular, we partition the computation field in MD simulation into subgrids, and replace all pairwise computation in a subgrid as a whole. This brings much larger performance improvement than the traditional pairwise-based approach. Furthermore, the lookup table is based on a tree structure for fast search and dynamically built at runtime. Leveraging the big memory capacity of persistent memory, the lookup table can be at the scale of TB, leading to high-quality MD simulation. However, using the new memoization methodology, we face two challenges.

First, how to represent all pair-wise computation in a subgrid such that we can efficiently identify and replace it as a whole is challenging. Within a subgrid, particles are distributed randomly. Using coordinates of particles in the subgrid to represent their distribution as a record in the lookup table would lead to massive number of records, which causes high search overhead and large memory consumption. Furthermore, different distributions of particles can represent the same computation, which can be leveraged to increase the hit rate of the lookup table. For examples, a translational movement of particles in a subgrid causes a new distribution, but the distance between particles after the movement remains the same, hence computation of inter-particle potential remains the same. However recognizing the distribution similarity across movements is challenging, because of the difference in coordinates before and after the movement.

Second, the read/write memory access pattern to the lookup table changes over time, demanding the lookup table to provide high performance for both reads and writes. The memory accesses are dominated by writes in the beginning of the MD simulation, in order to populate lookup table. Once the lookup table is populated, the memory accesses are dominated by reads in the remaining of the MD simulation. The common data structures to build the lookup table, such as B^+-tree [10] and LSM-tree [59], can provide high performance for either reads or writes, but not both. Hence they lack the flexibility to accommodate the variance of the memory access pattern to the lookup table in the MD simulation.

To address the above challenges, we introduce a framework, named MD-HM, to enable high performance memoization-based MD simulation. To address the first challenge, we treat the distribution of particles in a subgrid as a *pattern* and introduce a lightweight pattern recognition algorithm. The algorithm is not sensitive to the translational movement of particles, which avoids repeated records in the lookup table.

To address the second challenge, we introduce a new data structure (named *two-phase LSM tree*), which is a variant of the LSM tree. The traditional LSM tree provides high performance for writes but not reads. To address the above problem, the two-phase LSM tree maintains the traditional multi-level structure in the LSM tree to enable high-performance writes when writes dominate memory accesses in the beginning, but is compacted into a two-level structure for a shorter read path when reads dominate memory accesses.

We summarize major constitutions as follows.

- We provide a memoization-based framework MD-HM that uses big memory systems to accelerate MD simulations.
- We propose a two-phase LSM-tree to support efficient write and read operations on lookup tables on NVM-DRAM;
- We introduce a computation replacement strategy to increase replaced computation per lookup with ensured simulation stability.
- We show that MD-HM outperforms the state-of-the-art framework by 7.6x on average in nine MD simulations, and that MD-HM on a single big memory node reaches the performance of the numerical simulation on eight nodes.

2 BACKGROUND

MD simulations [4] model ensembles of particles, e.g., atoms and molecules, in a gaseous, liquid, or solid state. MD simulations often proceed in iterations of time steps. At each time step, interatomic potentials are computed for each particle by prescribed potential functions, and the position and velocity of particles are updated by Newton's laws of motion [37] for the next time step. During MD simulations, stability is the most critical criterion to determine simulation quality [2, 77]. The traditional MD simulation checks some invariant (e.g., the energy invariant) during the simulation to determine simulation stability. We use the energy invariant to determine correctness and assist in our parameter selection.

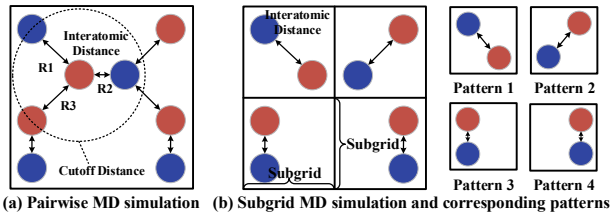

Figure 1: Two methods to build the lookup table.

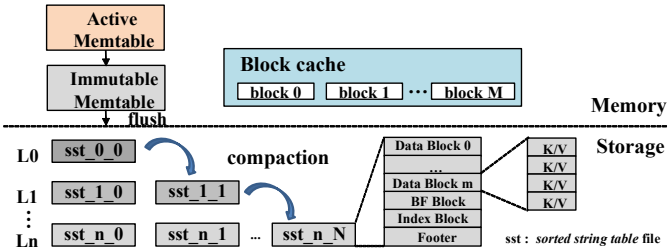

Figure 2: LSM-tree-based key-value store.

The computation of inter-atomic potentials dominates the simulation time because, for each particle, it requires $O(N^2)$ for modeling pair-wise interactions between N particles – $O(N^3)$ in total. On a modern processor, each pair computation typically takes 10^{-8} to 10^{-7} seconds [62, 86]. To reduce the computation cost, a cutoff distance (denoted as D_{cutoff}) is often used to limit pair-wise computation only to neighbor particles within D_{cutoff}. Figure 1(a) illustrates the pairwise computation for two types of particles.

Potential functions, such as Lennard-Jones, Tersoff, Gay-Berne, and Stillinger-Weber [20, 51, 76], are parameterized only by the distance between a pair. Therefore, if different pairs have the same distance, the computation of potentials is the same. However, in direct numerical simulations, such computation will be repeated.

Lookup tables are often used to reduce the repeated computation of potentials. For instance, pairwise lookup tables [35, 54] use the distance between a pair of particles as the key and save their potentials as the value. In particular, the cutoff distance is often discretized by linear or squared interpolations into many entries, each having its potential saved in the lookup table [47]. To replace the computation of potential, the distance between particles is looked up from the lookup table. Its closest match (called *target*) and other close-enough entries (called *adjacent entries*) are averaged to reduce interpolation error [27, 46]. Hence, one table lookup may replace one potential computation. Due to randomness in accesses and the latency to memory, pairwise lookup methods only bring limited performance benefits.

Subgrid-based lookup tables have been used in computational fluid dynamics simulations [12, 45, 52] that divide the whole force field into multiple subgrids. Such simulations progress by computing the potentials among all particles in a subgrid and then accumulate potentials between subgrids. Those lookup tables use the (hashed) coordinates of all particles in a subgrid as the key and save the computation results in the subgrid as the value. Hence, one table lookup could result in substantial computation reduction. Moreover, using Locality Sensitive Hashing (LSH) algorithms [13, 41, 72] for hashing the coordinates of particles in a subgrid would result in similar keys (i.e., adjacent locations in a table) for similar subgrids. Nevertheless, a large number of particles in a subgrid would lead to numerous combinations of particle coordinates (i.e., patterns), which could cause explosive memory consumption. For example, a liquid simulation with $2e+7$ patterns would require 1.2 TB memory and thus is infeasible on conventional DRAM-only systems.

Non-volatile memory-based big memory system. High-density, byte-addressable non-volatile memory (NVM) [5, 7] is promising for implementing large-capacity main memory. For instance, the recent Intel Optane DC Persistent Memory (DCPMM) could support up to 9 TB memory capacity on a single machine [33, 60] Large lookup tables that were unprecedented before have now become feasible. However, current NVMs typically have lower performance than DRAM, and its asymmetric performance in read and write makes write access a performance bottleneck. Therefore, heterogeneous memory systems that combine the performance of DRAM and the capacity of NVM are emerging.

In this work, we use an Intel DCPMM-based big memory system. The DCPMM can be configured in either *Memory*, *App Direct*, or *Hybrid* mode [33]. In Memory mode, DRAM becomes a hardware-managed cache to DCPMM, while in App Direct mode, accesses to DCPMM and DRAM can be controlled at the application level explicitly. DCPMM has a 174 ns and 304 ns latency for sequential and random read. Its peak bandwidth on a socket with six memory channels is 39 GB/s and 13 GB/s for read and write, respectively [60].

Log-structured merge tree. Lookup tables in MD simulations are essentially a key-value store system. Various data structures have been proposed to implement such storage systems [58]. In this work, we consider the performance characteristics of NVMs and access patterns in MD simulations and propose a new data structure based on the traditional Log-Structured Merge (LSM) tree [59]. We note that access to the lookup table during a simulation could be write-intensive when lots of new key-value pairs need to be inserted at runtime, and LSM-trees are write-optimized on NVM.

The traditional LSM-tree consists of a memory component and a storage component (Figure 2). The memory component, including active and immutable MemTables, is implemented in in-place sorted data structures, e.g., skip-list [64]. The storage component has a multi-level structure. Each level contains multiple sorted string table (SST) files. When the active MemTable is full, it is converted into the immutable MemTable, which is then compressed and *flushed* to the storage. Similarly, when a level of the storage component is full, its SST files are moved to the next level in a *compaction* process. Therefore, writes to storage only occur in *flush* and *compaction* operations, resulting in sequential writes that are friendly to storage media. Random read from an LSM-tree, however, may have a long latency because of the possibility of traversing multiple SST files on multiple levels.

3 OVERVIEW OF MD-HM

We propose a memoization-based MD simulation framework on Heterogeneous Memory (HM), called MD-HM. Figure 3 outlines

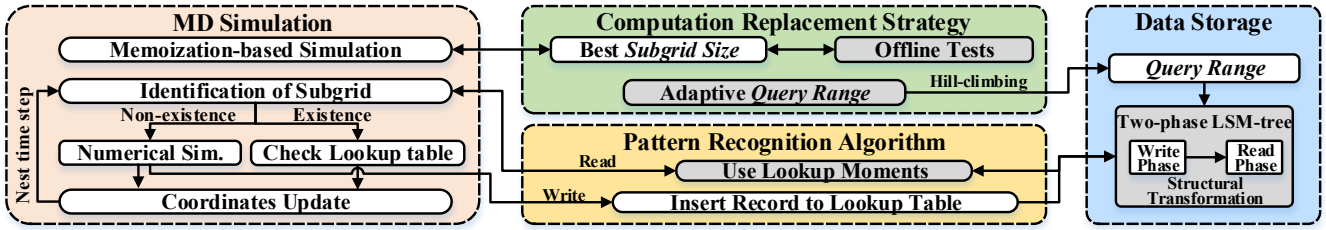

Figure 3: An overview of the MD-HM framework. MD simulations drive the execution flow and are accelerated by computation replacement strategy, shape matching-based pattern recognition, and HM-optimized data storage.

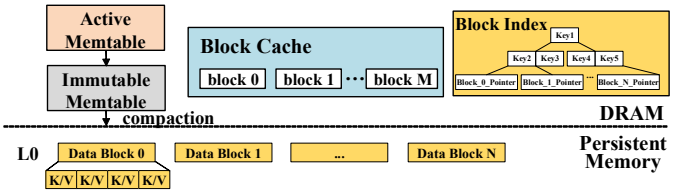

Figure 4: Main data structures in the two-phase LSM-tree in the read-dominant phase. The modification of data structure is highlighted in yellow.

its main components. The MD simulation drives the execution flow and queries the lookup table for computation replacement. MD-HM employs a pattern-matching algorithm to rapidly extract features of a subgrid and recognize it as a pattern to search for the matched subgrid in the lookup table. The lookup table is implemented in a two-phase LSM-tree designed to optimize both lookup and insertion operations on DRAM-NVM heterogeneous memory. Throughout a simulation, the computation replacement strategy adapts parameters for simulation stability and lookup efficiency.

The workflow of a MD-HM simulation starts from initializing the two-phase LSM-tree (Section 4) as the data storage for constructing the subgrid lookup table. Based on the input problem, MD-HM then builds a memoization-based simulation using suitable subgrid size (Section 5.1) and query range (Section 5.2). At each time step, to compute subgrid potentials, MD-HM tries to identify a subgrid based on proposed lookup moments (Section 6.2) from the lookup table. If a match is found (i.e., a hit), the stored potentials will be used. Otherwise (i.e., a miss), direct numerical computation needs to be performed, and a new subgrid-potentials pair will be inserted into the lookup table. After the potential computation, all particles will be updated to new positions, and their velocities will be updated for the next time step.

4 TWO-PHASE LSM-TREE

Lookup table-based MD simulations exhibit two phases distinct in read and write intensity. At the beginning of a simulation, the lookup table is mostly empty. Thus, most lookups will "miss" and result in inserting new entries of potentials from numerical computation into the lookup table, i.e., a simulation starts with a **write-dominant phase**. As the simulation progresses, the lookup table contains more entries so that lookups are more likely to "hit" in the table. This phase is characterized as **read-dominant**. The original LSM-tree is designed for high-throughput write but results in long-latency read due to its multi-level structure.

Algorithm 1 Structural Transformation for Read Phase

1: Define: *number of levels of LSM-tree* $\leftarrow N$
2: Set the 0-*th* level to directly compress to the N-*th* level
3: **for** $i \leftarrow 1$ to $N-1$ **do**
4: Compress the i-*th* level to the N-*th* level
5: Delete the i-*th* level
6: Flush out all the MemTables to the 0-*th* level
7: Suspend flush operation
8: **for** $j \leftarrow 1$ to *number of data blocks in the* 0-*th level* **do**
9: Compress the j-*th* data block to the N-*th* level
10: Delete the 0-*th* level and change N-*th* level to sorted 0-*th* level
11: Deserialize all the SST files to data blocks in the 0-*th* level
12: Build an index tree for each data block in the 0-*th* level
13: Modify the flush operation that moves data block from DRAM to NVM to compaction operation
14: Relaunch compaction operation between DRAM and NVM

In this work, we propose a two-phase LSM-tree as the data storage for the subgrid lookup table. Our two-phase LSM-tree supports high-throughput write and low-latency read operations in the write- and read- dominant phases, respectively. Two operations are supported – read operation that searches for a subgrid, and write operation that inserts a subgrid-potential pair. The main data structures include a block index in DRAM, one-level data blocks in NVM, and those in the original LSM-tree (e.g., MemTables and block cache). The block index and data blocks are only activated in the read-dominant phases, as illustrated in Figure 4.

In the write-dominant phase, the two-phase LSM-tree reuses data layout and implementations in the original LSM-tree design (see Section 2). When the hit rate on the lookup table exceeds a threshold value, the simulation is considered to have entered the read-dominant phase. The two-phase LSM-tree then triggers **phase transition** to transform to new data structures in the read-dominant phase.

4.1 Phase Transition

Phase transition compacts the multi-level SST files in NVM into a one-level structure of data blocks in NVM (i.e., *L0* in Figure 4) and activates the block index in DRAM. It also performs a compaction

operation to write MemTable into $L0$ to ensure data blocks are ordered. We summarize this process in Alg 1. MD-HM performs the transition in the background and the simulation progresses in the numerical simulation mode during the process.

First, all levels above the N-th level are compacted into the N-th level (Line 2-6). This step reduces the hierarchical levels in NVM. Next, all MemTables are flushed to the $0-th$ level, which is then directly compacted to the N-th level (Line 7-11). Then, all levels above the N-th level are deleted, and the N-th level is set to be 0-th level (i.e., $L0$). SST files in $L0$ are deserialized into non-overlapping and ordered data blocks (Line 12-13). Any records in the data blocks can be reached directly by a pointer because of the byte-addressability in NVM. Finally, an in-memory block index is built in DRAM using the Block Range Index [74] for data blocks in $L0$.

The block index in DRAM accelerates locating a data block in NVM on a given input key. Other data structures in the traditional LSM-tree, such as the active MemTable, immutable MemTable, and block cache, remain unchanged. The one-level structure of data blocks avoids multi-level search in NVM as required in the original LSM-tree and requires small footprint in DRAM (quantified in Figure 9(b)) to build the block index.

4.2 Read-Dominant Phase

MD-HM activates a new data structure – the block index in DRAM when a simulation enters the read-dominant phase (i.e, the hit rate is larger than 90%). MD-HM uses the block index to cache the location information of data blocks and act as a bookkeeper of available data blocks. This data block index organizes the in-memory index (location information) of data blocks in NVM for high performance.

A read operation first searches active MemTable and immutable MemTables to ensure the latest modified/new entry is fetched if any. If no record is found in MemTables, MD-HM will query the block index to find the required data block in NVM. If a pointer to the data block is returned, it will be used to retrieve the data block in NVM directly. Otherwise, it indicates no such data block has been saved in the lookup table and MD-HM would bypass all searches in NVM, unlike the original LSM-tree design. Consequently, at most one data block on NVM is accessed, in contrast to the traversal of multiple data blocks in the original LSM-tree. The block index in DRAM and one-level structure in NVM significantly reduce read amplification and the latency of read operations.

A write operation stores data to the MemTable in DRAM similarly to that in the write-dominant phase. Once the compaction finishes and new block are persisted into NVM, a new block pointer will be added to the block index. In the read-dominant phases, the frequency of compaction operations is so low that its impact on read operations is negligible.

5 COMPUTATION REPLACEMENT STRATEGY

The subgrid lookup table method divides the whole force field into multiple subgrids of a fixed subgrid size (denoted as $S_{subgrid}$). The computation of potentials in a subgrid is replaced by a query of the subgrid in the two-phase LSM-tree data storage. Performance improvement comes from the amount of computation replaced in one lookup that *hits* in the lookup table. We identify *subgrid size* as the key parameter that impacts both the hit rate in the lookup table and the amount of replaced computation per hit. We propose an empirical enumeration-based approach to determine the optimal subgrid size in Section 5.1.

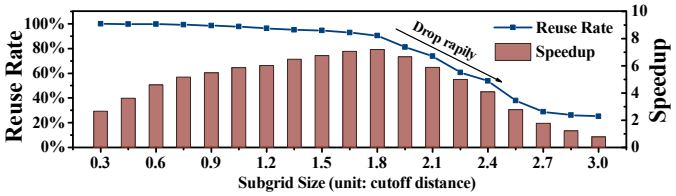

Figure 5: The impact of subgrid size on reuse rate and performance for a silicon simulation with the Tersoff potential function.

Simulation stability is another key factor that determines the success of a computation replacement strategy. As a lookup table based method, the subgrid lookup based method leverages interpolation to improve simulation accuracy, i.e., the returned results are interpolated from multiple 'neighboring' entries. We identify the *query range* as the key parameter, and introduce a dynamic adaptation algorithm to optimize the *query range* at runtime (see Section 5.2).

5.1 *Subgrid Size* Selection

We select the subgrid size under the constraint of memory capacity on a target platform and ensure high reuse rate of subgrids. Each subgrid lookup will replace the computation of all potentials in it. Therefore, larger subgrids increase computation replacement. However, as the information (such as coordinates and forces) for all particles in a subgrid needs to be stored in the lookup table, a larger subgrid will also increase memory consumption. With large-capacity NVM integrated into the memory subsystems, the memory capacity of a single machine is dramatically increased. Thus, the NVM-based systems can enable large subgrid sizes infeasible on conventional DRAM-based machines.

The reuse rate of subgrids is commensurate with the hit rate of the lookup table, i.e., queries that mostly hit mean that subgrids in the lookup table are reused. However, a larger subgrid size would reduce the chance of a subgrid being reused because of the increased number of possible subgrid patterns. We study the reuse rate and its impact on performance in nine MD simulations at various subgrid sizes and identify two critical observations for selecting the optimal subgrid size.

Observation 1: The optimal subgrid size for performance improvement can be achieved by enumeration.

Figure 5 reports the reuse rates and corresponding performance improvements over the numerical simulation LAMMPS at different subgrid sizes for a silicon simulation with the Tersoff potential function. The lookup table is built from the first 10% simulation steps. Under the memory limit of the testbed, the maximum subgrid size is three times the cutoff distance. When the subgrid size increases, the reuse rate decreases. Once the subgrid size exceeds 1.8 times the cutoff distance, the reuse rate starts to decline rapidly. Low

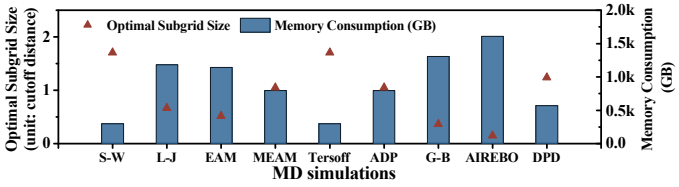

Figure 6: Optimal subgrid size and corresponding memory consumption for different MD problems.

reuse rates indicate that most queries 'miss' in the lookup table, i.e., requiring numerical simulation and insertion of new entries to the table. Thus, we observe a sharp decrease in performance. The peak performance improvement is achieved at the optimal subgrid size.

Observation 2: Simulations of the same materials are likely to share a common optimal subgrid size.

Figure 6 reports simulations of nine materials using different potential functions, their memory consumption, and optimal subgrid sizes. The results show that simulations of the same material have the same optimal subgrid size. For example, both the S-W and Tersoff potential functions are simulations for bulk silicon and perform the best at the subgrid size of 1.8 times the cutoff distance.

We design an empirical enumeration-based strategy to determine the optimal subgrid size for a simulation. For an MD simulation of new material, we explore feasible subgrid sizes under the constraints of memory capacity. The subgrid size that corresponds to the highest reuse rate is selected. This selection process is a one-time effort for each new material and the decision will be reused for different potential functions of the same material.

5.2 *Query Range* Adaptation

Lookup tables [6, 54] usually include multiple entries adjacent to the target for interpolation to improve simulation accuracy. The parameter that determines the furthermost adjacent entries to be included is denoted as *query range*.

We use a range operation to achieve high-precision interpolation. The range operation first retrieves the entry closest to the target, and then siblings within the query range. The final result is the average of all included entries. A smaller query range would have fewer but closer siblings. When the query range is too small and no sibling or exact entry is found, MD-HM goes back to the traditional numerical simulation. Hence, using a small query range is helpful to improve accuracy but at a risk of losing performance. In contrast, a large query range prolonging the range operation increases the chance of finding siblings but may compromise the accuracy if siblings with drastically different values are included. As adjusting the query range can impact accuracy, we propose an online algorithm (Alg 2) to adapt the query range to ensure simulation stability.

In particular, at the end of each time step, MD-HM examines whether the simulation is stable (Line 19) by calculating the difference (ΔE) between the energy computed in the time step and an energy invariant. If $\Delta E > E_{thr}$ where E_{thr} is a threshold typically defined by the user in the traditional numerical simulation to indicate the maximum change in the energy allowed for simulation

Algorithm 2 Algorithm for Query Range Adaptation

```
1:  Objective: minimize query range, subjected to ΔE less than E_thr
2:  Define: query range ← subgrid size, counter ← 0
3:  Define: maximum number of energy-invariant-violating time
    step N
4:  while true do
5:    if counter < N then
6:      if ΔE < E_thr then
7:        Drawn ϵ from a Gaussian distribution
8:        if ϵ > 1/ln(ΔE) then
9:          Drop to a new value: query range ← query range+ϵ
10:       Move on to the next time step using lookup table
11:       counter ← 0
12:     else
13:       Scale down: query range ← query range*scale_factor
14:       Move on to the next time step using lookup table
15:       counter ← counter + 1
16:   else
17:     Go back to the last checkpoint
18:     Move on to the next time step using numerical simulation
19:   Calculate the energy difference (ΔE) in the current time step
```

stability, Alg 2 starts and adjusts the query range. Otherwise, simulation is stable and the query range is opportunistically increased to increase the chance of finding siblings and hit rate (Lines 6-9).

Once the simulation is detected to be unstable, Alg 2 scales down the query range (Line 13) by a predefined *scale_factor* (0.9 in this study) and then the MD simulation continues to the next time step using the new query range. Alg 2 continues scaling down the query range, until the simulation becomes stable or N time steps have been tried to change the query range (Lines 13-15). If N time steps have been tried, Alg 2 reverts to the last checkpoint (Line 17) where the MD simulation is stable. From the last checkpoint, the MD uses traditional numerical simulation until reaching the last energy-invariant-violating time step. From that time step, the MD resumes the lookup table approach.

In general, MD-HM uses a combination of checkpointing, a traditional mechanism of detecting simulation stableness, and adapting query range to ensure simulation stability. The number of time steps, N, is determined by considering computation loss due to revert to the last checkpoint. In particular, the computation loss should be smaller than the performance benefit from MD-HM. Alg 2 cannot use too many time steps to scale down the query range, because the computation loss will then outweigh the performance benefit.

6 PATTERN RECOGNITION ALGORITHM

In this section, we propose a moment-based subgrid-pattern matching algorithm for computing the key of the lookup table. A naive implementation of the lookup table would use the positions of particles in a subgrid as the key. If we visualize the positions of particles in a subgrid, they form a *shape*. One shape could be a result of a translational movement from another shape, and they have the same potentials because the distances between particles remain unchanged. Hence, we propose to use *subgrid pattern*, computed by

lookup moments as the key of the lookup table. We define lookup moments to be translation invariant but rotation and scaling sensitive. Using subgrid pattern as the key avoids storing redundant keys and reduces memory consumption.

6.1 Moments and Hu Moment Invariants

Moments have been extensively in shape matching in various applications [24, 30, 31]. The two-dimensional $(p+q)$-th order *raw moments* $m_{p,q}$, *central moments* $\mu_{p,q}$, and *normalized central moments* $\eta_{p,q}$ of a region D are defined as follows:

$$m_{p,q} = \iint_D x^p y^q f(x,y)\,dxdy \tag{1}$$

$$\mu_{p,q} = \iint_D (x-x_c)^p (y-y_c)^q f(x,y)\,dxdy \tag{2}$$

$$\eta_{p,q} = \frac{\mu_{p,q}}{\mu_{0,0}^{(p+q)/2+1}} \tag{3}$$

where p, q are non-negative integers, $x_c = m_{1,0}/m_{0,0}$ and $y_c = m_{0,1}/m_{0,0}$. The point (x_c, y_c) is the centroid of the region D. $f(x,y)$ is a continuous bounded function. For MD simulation problem, the coordinate precision for x and y is discrete and usually determined by lookup table density empirically [35]. The continuous function represents the atom type at the position (x,y), and the type is 0 if no particle exists. All moments of all orders exist unique values for different subgrid patterns.

For MD simulations, a subgrid pattern can be represented by a set of moments that extract features for distinguishing patterns. In particular, the moments should be translation invariant because a translation does not change the potentials. Rotations change the order of replacement between particles and scaling movements change the distance. Consequently, the moments should distinguish rotation and scale as different patterns. The required moment algorithm needs to be translation invariant but rotation and scaling sensitive.

Here we define three transformations to a pattern, and these transformations can be expressed as: *Translation*, $\hat{x} = x + t_x$, $\hat{y} = y + t_y$; *Rotation*, $\hat{x} = x\cos\theta - y\sin\theta$, $\hat{y} = x\sin\theta + y\cos\theta$; *Scaling*, $\hat{x} = x * s_x$, $\hat{y} = y * s_y$, where (t_x, t_y) is the translation vector or shift vector, θ is a particular angle of rotation from its origin, and (s_x, s_y) is the scaling factor in X and Y direction, respectively.

The central moments are invariants to translational movement but sensitive to rotational and scaling movements [8, 48]. Naturally, the central moments seem an appropriate choice. However, the central moments alone cannot extract enough features to represent patterns, e.g., different subgrid patterns may have the same central moments. The Hu moments [30] introduced seven moments polynomials formed from the three aforementioned moments to extract enough features for pattern recognition. However, the Hu moments are invariants to translational, rotational, and scaling movements. Therefore, a new set of moments is needed for the subgrid pattern recognition in MD simulations.

6.2 Lookup Moments

We propose *lookup moments* as a set of seven moments that combine the central moments and Hu moment polynomials to satisfy the unique requirements of subgrid pattern recognition. The seven moments are defined as follows:

$$\begin{aligned}
M_1 &= \mu_{0,2} + \mu_{0,3} \\
M_2 &= \mu_{2,0} + \mu_{3,0} \\
M_3 &= \mu_{1,2} + \mu_{1,3} \\
M_4 &= \mu_{2,1} + \mu_{3,1} \\
M_5 &= (\eta_{3,0} - 3\eta_{1,2})^2 + (3\eta_{2,1} - 3\eta_{0,3})^2 \\
M_6 &= (\eta_{3,0} + \eta_{1,2})^2 + (\eta_{2,1} + \eta_{0,3})^2 \\
M_7 &= 2\eta_{1,1}[(\eta_{3,0} + \eta_{1,2})^2 - (\eta_{2,1} + \eta_{0,3})^2] \\
&\quad - 2(\eta_{2,0} - \eta_{0,2})(\eta_{1,2} + \eta_{0,3})(\eta_{2,1} + \eta_{0,3})
\end{aligned} \tag{4}$$

where M_1-M_4 are composed of the central moments for invariance to translation and sensitivity to rotation and scaling. M_5-M_7 enrich feature extraction to distinguish subgrid patterns that may have the same central moments. Among them, M_5 and M_6 are adopted from the Hu moments while M_7 is a complement of M_5-M_6 by invariant theorems [66].

Previous analysis shows that the central moments eliminate the influence from translational movements and retain sensitivity to rotation and scaling. Combining moment polynomials in the Hu moments extracts adequate features to distinguish subgrid patterns indistinguishable by the central moments. Our lookup moments satisfy the requirements of subgrid pattern in MD simulation at a low computation cost.

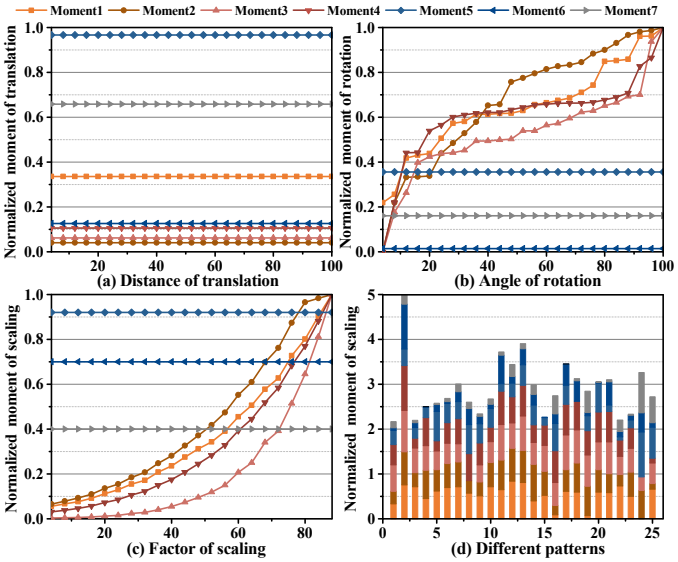

Figure 7: Normalized moments of translation(a), rotation(b), scaling(c), and the distribution of the lookup moments invariants(d).

6.3 Effectiveness Evaluation

We evaluate the effectiveness of the proposed lookup moments by collecting $400k$ subgrids from nine MD simulations. We build variants of these subgrids and report their average lookup moments in Figure 7(a-c).

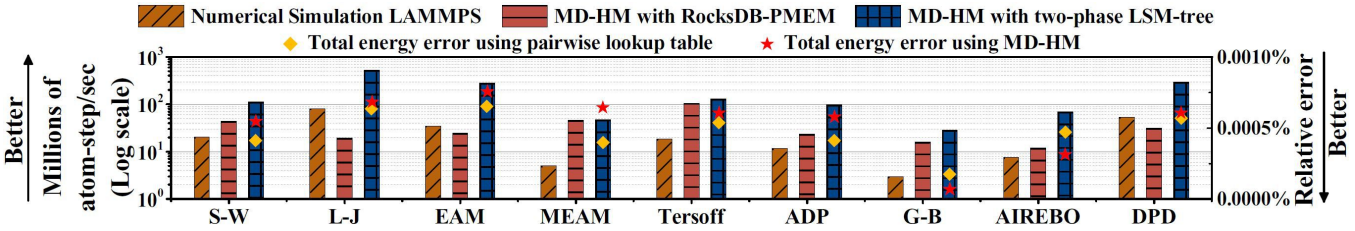

Figure 8: Performance comparison of nine MD simulations using LAMMPS, MD-HM with RocksDB-PMEM, and MD-HM with two-phase LSM-tree. And a relative error comparison using pairwise and subgrid lookup tables.

We apply a series of horizontal or vertical shifts to these subgrids. Figure 7(a) presents the seven moments after these translational movements. All moments remain stable when these translational movements are applied, indicating that lookup moments are translation invariant. To verify the sensitivity to rotation and scaling, 0-180 degrees rotation and 1-5 times scaling are applied. Figure 7(b-c) shows that the first four moments increase as the angle of rotation and the factor of scaling increases. Therefore, the lookup moments are sensitive to rotational and scaling movements. Finally, we visualize the moments on 25 randomly selected subgrids. Figure 7(d) shows that each subgrid reaches a specific value of the seven moments. The average coefficient of variation (COV) [1] of lookup moments is 68.37%, indicating high variability. For the low probability of different patterns resulted in the same moments, the checkpointing mechanism (Section 5.2) would suffice to ensure simulation stability.

6.4 Key Computation

We use the hash value of lookup moments as the key and the coordinates and potentials of particles inside a subgrid as the value to build the subgrid lookup table. In particular, We choose locality-sensitive hashing (LSH) [17] for hashing lookup moments so that similar subgrids will have similar keys. As the similarity between subgrids is embedded in keys, search for sibling subgrids within a query range in the range operation (see Section 5.2) is straightforward.

7 EVALUATION

Platform. We evaluate MD-HM on a server equipped with two Intel Xeon Gold 6252N 24-core processors running Linux 5.4.0. Each socket has 12 DIMM slots, six for 16-GB DDR4 DRAM modules, and six for 128-GB Optane DC modules. In total, the system has 192 GB DRAM and 1.5 TB NVM. We configure the Optane DC to App Direct Mode for maximum control. We also use a 32-node cluster based on EDR InfiniBand network, and each node has two Intel Xeon Platinum 8268 24-core processors and 128 GB of DRAM.

Input problems. We use nine molecular dynamics problems that cover a wide range of applications and come from LAMMPS benchmarks [63], summarized in Table 1 where *FLOPs* represents the number of floating-point operands per atom-step, *Patterns* stands for the number of patterns collected using MD-HM-based MD simulation, and *Size* is the capacity for storing all the subgrid patterns without using the lookup moments to detect repeated translational movements. The average number of neighbors ranges from 30 to 681, and each potential calculation requires 37 to 1502 floating-point operands. 100K particles are used, and the simulations run 1 million time steps for a typical setting [26, 42, 73].

Table 1: Input problems and parameters.

Problem	System	Neighs	FLOPs	Patterns	Size(TB)
S-W	bulk Si	30.0	151	3.25e+7	0.393
L-J	liquid	76.9	37	1.95e+7	1.155
EAM	bulk Cu	75.5	86	1.93e+7	1.113
MEAM	bulk Ni	48.8	610	3.31e+7	0.782
Tersoff	bulk Si	30.0	173	3.25e+7	0.393
ADP	bulk Ni	48.8	258	3.31e+7	0.782
G-B	ellipsoid	140	1017	6.51e+6	1.279
AIREBO	polyethylene	681	1502	3.39e+6	1.544
DPD	pure solvent	41.3	57	2.62e+7	0.526

Implementation and baselines. MD-HM is implemented based on LAMMPS [14]. We implement the computation replacement strategy and pattern recognition algorithm as a patch to LAMMPS. The two-phase LSM-tree is implemented as an extension of key-value store RocksDB-PMEM (v6.2.2) [21]. The statistics of modifications given by git diff is 13 files changed, 1582 insertions(+), and 47 deletions(-). The code is compiled with GCC-9.3 with the "-O3" option. Experiments run with 48 OpenMP threads, and the data block size in RocksDB-PMEM is 64 MB.

7.1 Overall Performance and Stability

We evaluate the performance and stability of MD-HM in nine MD simulations, including solid-state materials, soft matter, coarse-grained, and mesoscopic systems. For comparison, we use the numerical simulation LAMMPS as the baseline. To understand the benefit of the two-phase LSM-tree data storage, we also use a version of MD-HM built atop RocksDB-PMEM, the state-of-the-art key-value store. Figure 8 presents the end-to-end performance of these simulations, including computation phases (about 5-13% of the total execution time) that perform tasks other than potential computation, such as coordinates update and neighbor construction.

Overall, MD-HM outperforms the baseline numerical framework on all input problems by up to 9.71× and 7.62× on average. Without the two-phase LSM-tree data storage (i.e., MD-HM with RocksDB-PMEM), the subgrid lookup based framework outperforms the baseline on six problems (i.e., S-W, MEAM, Tersoff, ADP,

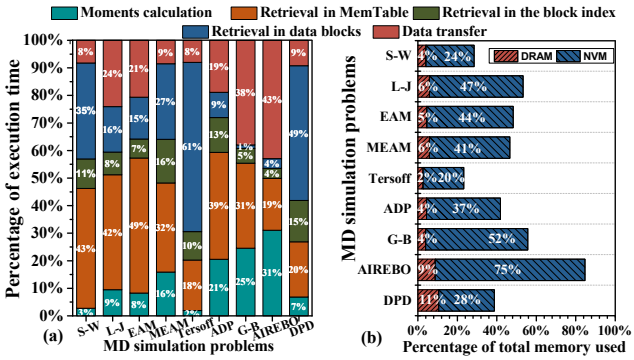

Figure 9: Overhead and memory consumption.

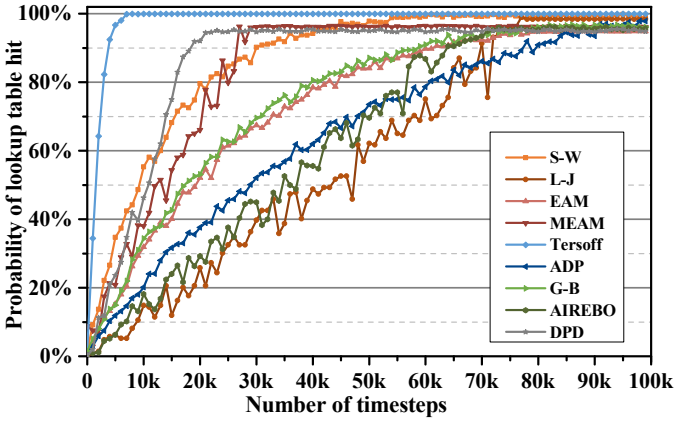

Figure 10: Hit rates on the lookup table as the simulation progresses.

G-B, and AIREBO), achieving an average 4.41× speedup. However, due to the high read latency of RocksDB-PMEM, MD-HM with RocksDB-PMEM underperforms the baseline in three problems (L-J, EAM, and DPD). Our analysis shows that these problems have low FLOPs (typically lower than 90) for the computation of potential, which increases the impact of read latency due to frequent queries to lookup table. This challenge is addressed by the shallow search hierarchy in our proposed two-phase LSM-tree data storage. Thus, MD-HM with the two-phase LSM-tree outperforms the RocksDB-PMEM based implementation by an average speedup of 2.76×. We highlight that for the computation part of MD, MD-HM with two-phase LSM-tree can achieve higher performance, and the average speedup is 27.93× (up to 48.07× for AIREBO problem).

We identify three characteristics for an MD simulation to benefit substantially from MD-HM. First, The potential function requires high FLOPs (more than 90 on our platform), which indicates that one query on the lookup table can replace a lot of computation. Second, a subgrid contains a large number of particles, which reduces the frequency of querying the lookup table. Finally, simulation phases other than the computation of potentials, e.g., updating particle position and constructing neighbor, only take a small portion of the total time. These characteristics help the user to decide the appropriate simulation approach for a given problem.

Figure 8 also reports simulation stability. We use the relative energy error [35] as the metric to evaluate simulation correctness. This metric is defined as $\Delta E = |E_{table} - E_{numeric}|/|E_{numeric}|$, where E_{table} and $E_{numeric}$ are the final energy values calculated with and without using lookup tables, respectively. We use the energy values calculated using the numerical simulation as ground truth. When using the lookup-based approach, we compare pairwise lookup table and subgrid lookup table (i.e., MD-HM using adaptive query range). Figure 8 shows that using MD-HM, the relative energy error is consistently below the error-tolerable threshold (0.001%) [35]. This indicates that MD-HM has high quality. Moreover, compared with the traditional pairwise lookup table, MD-HM has similar relative errors in seven problems, and smaller errors in two problems (G-B and AIREBO).

7.2 Performance Analysis

Overhead breakdown. Five components contribute most overhead of MD-HM framework – moments calculation, retrieval in MemTable, retrieval in index tree, retrieval in data block, and potentials transfer. Among them, moments calculation comes from the subgrid pattern recognition algorithm, potentials transfer comes from the computation replacement strategy, and the others come from the two-phase LSM-tree data storage. We report the breakdown of these overheads in the nine simulations in Figure 9(a).

We categorize the simulations into either moments-dominant or DRAM/NVM retrieval-dominant. The moments-dominant simulation is characterized by high moments calculation and high data transfer. Such problems have a large number of neighbors in a subgrid, resulting in more overhead in calculating the lookup moments and transferring retrieved potentials into the MD simulation. For instance, moments calculation and potential transfer attribute 40%, 63%, and 74% overhead in ADP, G-B, and AIREBO problems.

The remaining problems are dominated by overhead from accessing the two-phase LSM-tree, and they can be classified as either DRAM or NVM retrieval dominant. S-W, L-J, EAM, MEAM problems have a high cost in retrieval in MemTable and index tree that are stored on DRAM, accounting for 54%, 50%, 56%, and 48% overhead. Retrieval block index only takes an average of 10% overhead while MemTable takes 43-49% overhead because these problems have smaller subgrids and need to retrieve more entries in MemTable. Finally, Tersoff and DPD are dominated by NVM access. They have 61% and 49% overhead from retrieval in data block. The overhead of traversing the data block is proportional to the number of entries in the data block. As Tersoff and DPD problems have a smaller number of neighbors, a data block would contain more entries, costing more time in traversing data blocks.

Memory consumption. Figure 9(b) reports the DRAM and NVM usage compared to the total memory capacity (including DRAM and NVM). We find that using the lookup moments reduces the memory consumption by 27.4% on average, compared to the naive implementation without using the lookup moments to detect repeated translational movement.

Note that the native implementation would stores all coordinates and potentials in Table 1 in the lookup table. Our lookup moments approach recognizes subgrid patterns to avoid storing duplicate keys with translational movements. Also, most entries in

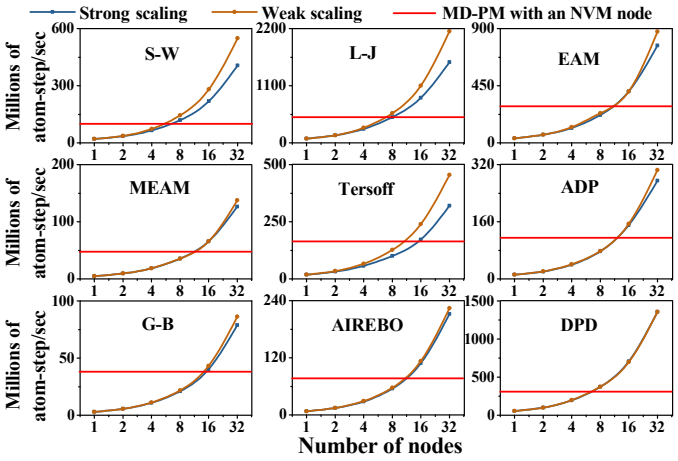

Figure 11: Comparing the numerical simulation on a cluster with MD-HM on an NVM-based node.

the lookup table are stored in NVM. The DRAM usage is less than 15%, indicating that our approach is applicable on machines with relatively small DRAM and can release DRAM space for large-scale MD simulation.

Effectiveness of Lookup Table. The subgrid lookup table is expected to capture most subgrid patterns in a simulation as the simulation advances, i.e., the hit rates on the table increases throughout a simulation. To evaluate the effectiveness of the lookup table in MD-HM, we report the hit rates at different timesteps in nine problems in Figure 10. Only the first 10% of the total timesteps are shown because the hit rates converge afterward.

The lookup table reaches over 95% hit rate in 90k timesteps for all problems. Different problems have a diverse profile of the hit rates. For instance, Tensoff has most patterns collected within the first 1k timesteps, and its hit rate converges at about 98.8%. Tensoff has a short construction time of the lookup table because the number of neighbors of each particle is relatively small, resulting in a small number of patterns that need to be collected. ADP and G-B achieve a hit rate of about 97% in 90k timesteps, i.e., about 9% of the total timesteps. Once most patterns are collected in the lookup table, subsequent timesteps are mostly read-intensive with a high lookup table hit rate, indicating the completion of the transition from intensive potential computation into read-intensive query lookups. The transition from write- to read-dominant phase is measured to complete in fewer than 5% of the total timesteps.

7.3 Scalability

We compare MD-HM running on a single NVM-based node with the traditional numerical simulation running on a cluster of 32 nodes without NVM. On the cluster, we evaluate the weak and strong scalability of nine MD simulations and report in Figure 11. The weak scalability indicates the ability to handle larger-scale simulations with more resources. The strong scalability indicates the speedup on a fixed problem with increased resources.

Figure 11 shows that MD simulations demonstrate good weak and strong scalability. This is because MD simulations often have high compute intensity and low communication overhead. The performance of the MD-HM on an NVM node is indicated by the red line on Figure 11. We show that MD-HM on one NVM node can outperform the numerical simulation on eight nodes commonly. In some cases, such as Tersoff and G-B, MD-HM can even reach the numerical simulation on 16 nodes. As MD-HM has similar scalability as the numerical simulation, the performance of MD-HM is expected to scale as the number of NVM-based nodes increases.

8 RELATED WORK

Memoization. The memoization approach works by storing the output of functions for later reuse with identical input. By converting the floating-point calculations to the memory access, the success of memoization critically depends on efficiently indexing structures and fast access to the data. The memoization technique has been studied in many fields. Franyell et al. [71] propose a fuzzy memoization scheme that avoids more than 24.2% of computation for RNN training. Liu et al. [44] replace a long sequence of instructions with a two-level lookup table. Keramidas et al. [38] memoized the outcomes of a fragment shaders that compute the final color value of pixel using arithmetic operations and texture fetches, thus the precision of arithmetic operations can be reduced more aggressively for high performance. Rahimi et al. [65] reused the result of instructions across different lanes of SIMD to reduce error recovery overhead. Arjun et al. [75] use large memoization tables to capture long intervals of repetition to benefit computationally intensive pure functions. Keramidas et al. [38] memoized the outcomes of a fragment shaders that compute the final color value of pixel using arithmetic operations and texture fetches, thus the precision of arithmetic operations can be reduced more aggressively for high performance. Lin et al. [55, 56] used LSH to reuse earlier computations in the field of DNN.

MD simulation using pairwise lookup table. Since MD simulation is based on potential functions, there are many works to explore the use of memoization for function-level reuse. Nilsson [38] proposed an efficient way to use lookup table without the need to calculate inverse square roots spline interpolation, and achieved 1.5x-2x speedup compared with standard calculations. Jaewoon et al. [35] proposed a short-range approach by defining energy and gradient as a linear function of inverse distance squared to enhance the accuracy of lookup table. However, these approaches are based on pairwise computation of replacing one potential computation with bounded speedup. In this work, we focus on subgrid-based simulation for higher performance improvement.

Non-volatile memory for HPC. HPC applications [15, 16, 28, 43, 67, 68, 70, 82–84] generally have significant memory consumption. A lot of work have explored the use of non-volatile memory in some HPC applications [23, 53, 61, 80, 81]. Nguyen et. al [53] introduce a multi-version octree on PM to enable adaptive mesh simulation on PM. Unimem [80] uses performance modeling to decide data placement for MPI-based HPC applications. Siena [61] explores rich organizations and configurations of HM architecture for HPC applications to determine optimal system designs. Tahoe [81] combines a machine learning model and an analytical model to predict application performance across multiple memory components for task-parallel programs. Some works leverage the non-volatility of

NVM for checkpoint [19, 69]. Our work is different from them by using the big memory capacity of NVM to accelerate MD simulations.

9 CONCLUSION

Memoization techniques play an important role in many scientific and engineering applications. However, how to use them efficiently for MD simulation is challenging because of the enormous storage for patterns and the requirement of simulation stability. This paper designs a runtime framework called MD-HM, which builds a new data structure and eliminates redundant patterns. Results show that MD-HM yields better performance than the numerical simulation and the state-of-the-art key-value store. We expect more computation-intensive tasks can be inspired by our method to use big memory systems to accelerate performance.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. We also thank Profs. Liang Shi (UC Merced) and Zhen Li (Clemson) for their kind answers to our questions on LAMMPS. This work was partially supported by U.S. National Science Foundation (CNS-1617967, CCF-1553645 and CCF-1718194), the Chameleon Cloud, and Intel hardware donation. LLNL-CONF-821251. This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344.

REFERENCES

[1] Hervé Abdi. 2010. Coefficient of variation. *Encyclopedia of research design* 1 (2010), 169–171.
[2] Stewart A Adcock and J Andrew McCammon. 2006. Molecular dynamics: survey of methods for simulating the activity of proteins. *Chemical reviews* 106, 5 (2006), 1589–1615.
[3] Hasan Metin Aktulga, Joseph C Fogarty, Sagar A Pandit, and Ananth Y Grama. 2012. Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques. *Parallel Comput.* 38, 4-5 (2012), 245–259.
[4] Michael P Allen and Dominic J Tildesley. 2017. *Computer simulation of liquids*. Oxford university press.
[5] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. 2017. Emerging NVM: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 23, 2 (2017), 1–32.
[6] Bhargab Chattopadhyay and Ken Kelley. 2016. Estimation of the coefficient of variation with minimum risk: A sequential method for minimizing sampling error and study cost. *Multivariate Behavioral Research* 51, 5 (2016), 627–648.
[7] An Chen. 2016. A review of emerging non-volatile memory (NVM) technologies and applications. *Solid-State Electronics* 125 (2016), 25–38.
[8] Chaur-Chin Chen. 1993. Improved moment invariants for shape discrimination. *Pattern recognition* 26, 5 (1993), 683–686.
[9] Fabrizio Chiti, Niccolò Taddei, Paul M White, Monica Bucciantini, Francesca Magherini, Massimo Stefani, and Christopher M Dobson. 1999. Mutational analysis of acylphosphatase suggests the importance of topology and contact order in protein folding. *Nature structural biology* 6, 11 (1999), 1005–1009.
[10] Douglas Comer. 1979. Ubiquitous B-tree. *ACM Computing Surveys (CSUR)* 11, 2 (1979), 121–137.
[11] Valerie Daggett. 2006. Protein folding- simulation. *Chemical reviews* 106, 5 (2006), 1898–1916.
[12] Thibault Dairay, Eric Lamballais, Sylvain Laizet, and John Christos Vassilicos. 2017. Numerical dissipation vs. subgrid-scale modelling for large eddy simulation. *J. Comput. Phys.* 337 (2017), 252–274.
[13] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.
[14] LAMMPS Developers. 2021. lammps. https://https://github.com/lammps/lammps.
[15] Wenqian Dong, Jie Liu, Zhen Xie, and Dong Li. 2019. Adaptive neural network-based approximation to accelerate eulerian fluid simulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–22.
[16] Wenqian Dong, Zhen Xie, Gokcen Kestor, and Dong Li. 2020. Smart-PGSim: using neural network to accelerate AC-OPF power grid simulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.
[17] Donald Eastlake and Paul Jones. 2001. US secure hash algorithm 1 (SHA1).
[18] Peter Eastman, Jason Swails, John D Chodera, Robert T McGibbon, Yutong Zhao, Kyle A Beauchamp, Lee-Ping Wang, Andrew C Simmonett, Matthew P Harrigan, Chaya D Stern, et al. 2017. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS computational biology* 13, 7 (2017), e1005659.
[19] Hussein Elnawawy, Mohammad Alshboul, James Tuck, and Yan Solihin. 2017. Efficient checkpointing of loop-based codes for non-volatile main memory. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 318–329.
[20] Furio Ercolessi and James B Adams. 1994. Interatomic potentials from first-principles calculations: the force-matching method. *EPL (Europhysics Letters)* 26, 8 (1994), 583.
[21] Facebook. 2020. pmem-rocksdb. https://github.com/pmem/pmem-rocksdb.
[22] Hai Bo Fan, Edward KL Chan, Cell KY Wong, and Matthew MF Yuen. 2006. Investigation of moisture diffusion in electronic packages by molecular dynamics simulation. *Journal of Adhesion Science and Technology* 20, 16 (2006), 1937–1947.
[23] Pradeep Fernando, Ada Gavrilovska, Sudarsun Kannan, and Greg Eisenhauer. 2018. NVStream: Accelerating HPC Workflows with NVRAM-based Transport for Streaming Objects. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing* (Tempe, Arizona) *(HPDC '18)*. ACM, New York, NY, USA, 231–242. https://doi.org/10.1145/3208040.3208061
[24] Jan Flusser and Tomas Suk. 1993. Pattern recognition by affine moment invariants. *Pattern recognition* 26, 1 (1993), 167–174.
[25] Richard Fujimoto. 2015. Parallel and distributed simulation. In *2015 Winter Simulation Conference (WSC)*. IEEE, 45–59.
[26] Jens Glaser, Trung Dac Nguyen, Joshua A Anderson, Pak Lui, Filippo Spiga, Jaime A Millan, David C Morse, and Sharon C Glotzer. 2015. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Computer Physics Communications* 192 (2015), 97–107.
[27] Yongfeng Gu, Tom VanCourt, and Martin C Herbordt. 2006. Improved interpolation and system integration for FPGA-based molecular dynamics simulations. In *2006 International Conference on Field Programmable Logic and Applications*. IEEE, 1–8.
[28] Xin He, Yapeng Yao, Zhiwen Chen, Jianhua Sun, and Hao Chen. 2021. Efficient parallel A* search on multi-GPU system. *Future Generation Computer Systems* (2021).
[29] Fredrik Hedman. 2006. *Algorithms for Molecular Dynamics Simulations*. Ph.D. Dissertation. Institutionen för fysikalisk kemi, oorganisk kemi och strukturkemi.
[30] Ming-Kuei Hu. 1962. Visual pattern recognition by moment invariants. *IRE transactions on information theory* 8, 2 (1962), 179–187.
[31] Zhihu Huang and Jinsong Leng. 2010. Analysis of Hu's moment invariants on image scaling and rotation. In *2010 2nd International Conference on Computer Engineering and Technology*, Vol. 7. IEEE, V7–476.
[32] Afshan Ilyas, M Rizwan Khan, and Mohammad Ayyub. 2015. Lookup table based modeling and simulation of solar photovoltaic system. In *2015 Annual IEEE India Conference (INDICON)*. IEEE, 1–6.
[33] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, Subramanya R Dulloor, et al. 2019. Basic performance measurements of the intel optane DC persistent memory module. *arXiv preprint arXiv:1903.05714* (2019).
[34] Frank Jensen. 2013. Atomic orbital basis sets. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 3, 3 (2013), 273–295.
[35] Jaewoon Jung, Takaharu Mori, and Yuji Sugita. 2013. Efficient lookup table using a linear function of inverse distance squared. *Journal of Computational Chemistry* 34, 28 (2013), 2412–2420.
[36] Martin Karplus and J Andrew McCammon. 2002. Molecular dynamics simulations of biomolecules. *Nature structural biology* 9, 9 (2002), 646–652.
[37] Martin Karplus and Gregory A Petsko. 1990. Molecular dynamics simulations in biology. *Nature* 347, 6294 (1990), 631–639.
[38] Georgios Keramidas, Chrysa Kokkala, and Iakovos Stamoulis. 2015. Clumsy value cache: An approximate memoization technique for mobile GPU fragment shaders. In *Workshop on Approximate Computing (WAPCO'15)*.
[39] Tuomas Koskela, Zakhar Matveev, Charlene Yang, Adetokunbo Adedoyin, Roman Belenov, Philippe Thierry, Zhengji Zhao, Rahulkumar Gayatri, Hongzhang Shan, Leonid Oliker, Jack Deslippe, Ron Green, and Samuel Williams. 2018. A Novel Multi-level Integrated Roofline Model Approach for Performance Characterization. In *International Conference on High Performance Computing*.
[40] David Kozono, Masato Yasui, Landon S King, Peter Agre, et al. 2002. Aquaporin water channels: atomic structure molecular dynamics meet clinical medicine. *The Journal of Clinical Investigation* 109, 11 (2002), 1395–1399.

[41] Brian Kulis and Kristen Grauman. 2011. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, 6 (2011), 1092–1104.

[42] Scott Le Grand, Andreas W Götz, and Ross C Walker. 2013. SPFP: Speed without compromise—A mixed precision model for GPU accelerated molecular dynamics simulations. *Computer Physics Communications* 184, 2 (2013), 374–380.

[43] Jiawen Liu, Jie Ren, Roberto Gioiosa, Dong Li, and Jiajia Li. 2021. Sparta: High-performance, element-wise sparse tensor contraction on heterogeneous memory. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 318–333.

[44] Zhenhong Liu, Amir Yazdanbakhsh, Dong Kai Wang, Hadi Esmaeilzadeh, and Nam Sung Kim. 2019. AxMemo: hardware-compiler co-design for approximate code memoization. In *Proceedings of the 46th International Symposium on Computer Architecture*. 685–697.

[45] Bona Lu, Wei Wang, and Jinghai Li. 2009. Searching for a mesh-independent sub-grid model for CFD simulation of gas–solid riser flows. *Chemical Engineering Science* 64, 15 (2009), 3437–3447.

[46] Miha Luksic, Christopher J Fennell, and Ken A Dill. 2014. Using interpolation for fast and accurate calculation of ion–ion interactions. *The Journal of Physical Chemistry B* 118, 28 (2014), 8017–8025.

[47] Jinping Luo, Lijun Liu, Peng Su, Pengbo Duan, and Daihui Lu. 2015. A piecewise lookup table for calculating nonbonded pairwise atomic interactions. *Journal of molecular modeling* 21, 11 (2015), 288.

[48] Sidhartha Maitra. 1979. Moment invariants. *Proc. IEEE* 67, 4 (1979), 697–699.

[49] Jason E Miller, Harshad Kasture, George Kurian, Charles Gruenwald, Nathan Beckmann, Christopher Celio, Jonathan Eastep, and Anant Agarwal. 2010. Graphite: A distributed parallel simulator for multicores. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 1–12.

[50] Muhammad Usman Mirza, Shazia Rafique, Amjad Ali, Mobeen Munir, Nazia Ikram, Abdul Manan, Outi MH Salo-Ahen, and Muhammad Idrees. 2016. Towards peptide vaccines against Zika virus: Immunoinformatics combined with molecular dynamics simulations to predict antigenic epitopes of Zika viral proteins. *Scientific reports* 6 (2016), 37313.

[51] Y Mishin, Diana Farkas, MJ Mehl, and DA Papaconstantopoulos. 1999. Interatomic potentials for monoatomic metals from experimental data and ab initio calculations. *Physical Review B* 59, 5 (1999), 3393.

[52] Parviz Moin, Kyle Squires, W Cabot, and Sangsan Lee. 1991. A dynamic subgrid-scale model for compressible turbulence and scalar transport. *Physics of Fluids A: Fluid Dynamics* 3, 11 (1991), 2746–2757.

[53] Bao Nguyen, Hua Tan, and Xuechen Zhang. 2017. Large-scale adaptive mesh simulations through non-volatile byte-addressable memory. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[54] Lennart Nilsson. 2009. Efficient table lookup without inverse square roots for calculation of pair wise atomic interactions in classical simulations. *Journal of computational chemistry* 30, 9 (2009), 1490–1498.

[55] Lin Ning, Hui Guan, and Xipeng Shen. 2019. Adaptive deep reuse: Accelerating cnn training on the fly. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1538–1549.

[56] Lin Ning and Xipeng Shen. 2019. Deep reuse: streamline CNN inference on the fly via coarse-grained computation reuse. In *Proceedings of the ACM International Conference on Supercomputing*. 438–448.

[57] Noriaki Okimoto, Noriyuki Futatsugi, Hideyoshi Fuji, Atsushi Suenaga, Gentaro Morimoto, Ryoko Yanai, Yousuke Ohno, Tetsu Narumi, and Makoto Taiji. 2009. High-performance drug discovery: computational screening by combining docking and molecular dynamics simulations. *PLoS Comput Biol* 5, 10 (2009), e1000528.

[58] Ahmed Oussous, Fatima-Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. 2018. Big Data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences* 30, 4 (2018), 431–448.

[59] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33, 4 (1996), 351–385.

[60] Ivy B. Peng, Maya B. Gokhale, and Eric W. Green. 2019. System Evaluation of the Intel Optane Byte-addressable NVM. In *Proceedings of the International Symposium on Memory Systems*. ACM. https://doi.org/10.1145/3357526.3357568

[61] I. B. Peng and J. S. Vetter. 2018. Siena: Exploring the Design Space of Heterogeneous Memory Systems. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 427–440. https://doi.org/10.1109/SC.2018.00036

[62] Levi CT Pierce, Romelia Salomon-Ferrer, Cesar Augusto F. de Oliveira, J Andrew McCammon, and Ross C Walker. 2012. Routine access to millisecond time scale events with accelerated molecular dynamics. *Journal of chemical theory and computation* 8, 9 (2012), 2997–3002.

[63] Steve Plimpton. 1993. *Fast parallel algorithms for short-range molecular dynamics*. Technical Report. Sandia National Labs., Albuquerque, NM (United States).

[64] William Pugh. 1990. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM* 33, 6 (1990), 668–676.

[65] Abbas Rahimi, Luca Benini, and Rajesh K Gupta. 2013. Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures. *IEEE Transactions on Circuits and Systems II: Express Briefs* 60, 12 (2013), 847–851.

[66] Thomas H. Reiss. 1991. The revised fundamental theorem of moment invariants. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 8 (1991), 830–834.

[67] Jie Ren, Jiaolin Luo, Kai Wu, Minjia Zhang, Hyeran Jeon, and Dong Li. 2021. Sentinel: Efficient Tensor Migration and Allocation on Heterogeneous Memory Systems for Deep Learning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 598–611.

[68] Jie Ren, Kai Wu, and Dong Li. 2020. Exploring Non-Volatility of Non-Volatile Memory for High Performance Computing Under Failures. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 237–247.

[69] Jie Ren, Kai Wu, and Dong Li. 2020. Exploring Non-Volatility of Non-Volatile Memoryfor High Performance Computing Under Failures. In *IEEE International Conference on Cluster Computing*.

[70] Jie Ren, Minjia Zhang, and Dong Li. 2020. HM-ANN: Efficient Billion-Point Nearest Neighbor Search on Heterogeneous Memory. *Advances in Neural Information Processing Systems* 33 (2020).

[71] Franyell Silfa, Gem Dot, Jose-Maria Arnau, and Antonio Gonzàlez. 2019. Neuron-level fuzzy memoization in rnns. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 782–793.

[72] Malcolm Slaney and Michael Casey. 2008. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal processing magazine* 25, 2 (2008), 128–131.

[73] John E Stone, David J Hardy, Ivan S Ufimtsev, and Klaus Schulten. 2010. GPU-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modelling* 29, 2 (2010), 116–125.

[74] Michael Stonebraker and Lawrence A Rowe. 1986. The design of POSTGRES. *ACM Sigmod Record* 15, 2 (1986), 340–355.

[75] Arjun Suresh, Erven Rohou, and André Seznec. 2017. Compile-time function memoization. In *Proceedings of the 26th International Conference on Compiler Construction*. 45–54.

[76] JJPRB Tersoff. 1989. Modeling solid-state chemistry: Interatomic potentials for multicomponent systems. *Physical review B* 39, 8 (1989), 5566.

[77] Søren Toxvaerd, Ole J Heilmann, and Jeppe C Dyre. 2012. Energy conservation in molecular dynamics simulations of classical systems. *The Journal of chemical physics* 136, 22 (2012), 224106.

[78] CZ Wang and KM Ho. 1997. Material simulations with tight-binding molecular dynamics. *Journal of phase equilibria* 18, 6 (1997), 516.

[79] D Wolff and WG Rudd. 1999. Tabulated potentials in molecular dynamics simulations. *Computer physics communications* 120, 1 (1999), 20–32.

[80] K. Wu, Y. Huang, and D. Li. 2017. Unimem: Runtime Data Management on Non-Volatile Memory-based Heterogeneous Main Memory. In *International Conference for High Performance Computing, Networking, Storage and Analysis*.

[81] Kai Wu, Jie Ren, and Dong Li. 2018. Runtime Data Management on Non-Volatile Memory-Based Heterogeneous Memory for Task Parallel Programs. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*.

[82] Zhen Xie, Zheng Cao, Zhan Wang, Dawei Zang, En Shao, and Ninghui Sun. 2016. Modeling traffic of big data platform for large scale datacenter networks. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 224–231.

[83] Zhen Xie, Wenqian Dong, Jiawen Liu, Hang Liu, and Dong Li. 2021. Tahoe: tree structure-aware high performance inference engine for decision tree ensemble on GPU. In *Proceedings of the Sixteenth European Conference on Computer Systems*. 426–440.

[84] Zhen Xie, Guangming Tan, Weifeng Liu, and Ninghui Sun. 2019. IA-SpGEMM: An input-aware auto-tuning framework for parallel sparse matrix-matrix multiplication. In *Proceedings of the ACM International Conference on Supercomputing*. 94–105.

[85] Yuan Xu, Yi Zhang, Ephraim Suhir, and Xinwei Wang. 2006. Thermal properties of carbon nanotube array used for integrated circuit cooling. *Journal of Applied Physics* 100, 7 (2006), 074302.

[86] Wei Zhang, Riccardo Mazzarello, Matthias Wuttig, and Evan Ma. 2019. Designing crystallization in phase-change materials for universal memory and neuro-inspired computing. *Nature Reviews Materials* 4, 3 (2019), 150–168.