Massively Scaling the Metal Microscopic Damage Simulation on Sunway TaihuLight Supercomputer

Shigang Li
SKL Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
shigangli.cs@gmail.com

Baodong Wu
SKL Computer Architecture, Institute of Computing Technology, CAS; University of Chinese Academy of Sciences
wubd.cs@gmail.com

Yunquan Zhang
SKL Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
yunquan.zhang@gmail.com

Xianmeng Wang
University of Science and Technology Beijing
wangxianmeng@xs.ustb.edu.cn

Jianjiang Li
University of Science and Technology Beijing
lijianjiang@ustb.edu.cn

Changjun Hu
University of Science and Technology Beijing
huchangjun@ies.ustb.edu.cn

Jue Wang
Computer Network Information Center, CAS
wangjue@sccas.cn

Yangde Feng
Computer Network Information Center, CAS
ydfeng@sccas.cn

Ningming Nie
Computer Network Information Center, CAS
nienm@sccas.cn

ABSTRACT

The limitation of simulation scales leads to a gap between simulation results and physical phenomena. This paper reports our efforts on increasing the scalability of metal material microscopic damage simulation on the Sunway TaihuLight supercomputer. We use a multiscale modeling approach that couples Molecular Dynamics (MD) with Kinetic Monte Carlo (KMC). According to the characteristics of metal materials, we design a dedicated data structure to record the neighbor atoms for MD, which significantly reduces the memory consumption. Data compaction and double buffer are used to reduce the data transfer overhead between the main memory and the local store. We propose an on-demand communication strategy for KMC to remarkably reduce the communication overhead. We simulate $4 * 10^{12}$ atoms on 6,656,000 master+slave cores using MD with 85% parallel efficiency. Using the coupled MD-KMC approach, we simulate $3.2 * 10^{10}$ atoms in 19.2 days temporal scale on 6,240,000 master+slave cores with runtime of 8.6 hours.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**;
• **Computer systems organization** → *Multicore architectures*;

KEYWORDS

parallel scalability, microscopic damage simulation, Sunway TaihuLight, Molecular Dynamics, Kinetic Monte Carlo

1 INTRODUCTION

Microscopic damage in materials is difficult to observe by physical experiments. Alternatively, software simulation is a promising method to understand the evolution mechanism of the microscopic damage. The existing simulation software in materials science, such as Molecular Dynamics (MD) [7, 8, 22, 27] and Kinetic Monte Carlo (KMC) [14, 23, 31], has good performance in a specific temporal and spatial range. However, the microstructure and damage evolution process of materials have a much larger spatio-temporal scale. Thus, there is a big gap between simulation results and physical phenomena due to the limitation of simulation scales.

Typically, the simulation scales are limited due to the following two reasons: (1) Single model usually only performs well at a specific narrow range. For example, MD commonly simulates millions to billions of atoms at picoseconds to nanoseconds. In this paper, we use a coupled MD-KMC approach to achieve both high temporal and spatial scales. (2) Large-scale simulation has a high computational complexity and requires a great amount of computing resources. Although supercomputers are becoming more powerful, the existing software lags behind because of the high effort of code porting and optimization on new machines. We optimize our coupled MD-KMC code on the Sunway TaihuLight supercomputer, and utilize its rich computing resources to conduct large-scale microscopic damage simulation.

In the last decade, the computing capability of supercomputers has been growing rapidly. The top one system in 2016, Sunway TaihuLight [6], possesses 93 PFlops sustained Linpack performance. Besides, due to the constrains caused by the heat dissipation and power consumption issues, recent large systems come in the form of heterogeneous systems with both CPUs and many-core accelerators, such as GPUs [20], Intel Xeon Phi [3], and Sunway many-core accelerator [6]. To achieve good performance, applications should efficiently utilize both CPUs and accelerators.

Our application simulates the microscopic damage for the metal material composed of iron (Fe) atoms. Our application also supports the simulation of different atoms, e.g., the alloy materials. To achieve this, more interpolation tables should be used to calculate the embedded-atom method (EAM) potential, as discussed in Section 2.1.2. The atoms are modeled in Body-Centered Cubic (BCC) structure, and EAM potential [4] is used as physical interaction. We use MD to simulate the defect generation caused by

cascade collision, and use KMC to simulate the defect evolution and clustering. We aim at scaling the simulation with the increasing of the workload (larger temporal and spatial scale) and the available hardware resources. However, there are several challenges: (1) the memory consumption linearly increases with the number of the atoms, which may run out of the system memory and limits the spatial scale; (2) communication overhead caused by ghost data exchange and synchronization hinder the scalability with the increasing of computation nodes; (3) the small local store on the slave core of Sunway many-core processor makes it difficult to utilize the accelerator efficiently. We propose several methods, including a dedicated data structure for metal materials, on-demand communication, and data compaction, to overcome the above challenges. We simulate $4 * 10^{12}$ atoms on 6,656,000 master+slave cores using MD with 85% parallel efficiency (weak scaling). Strong scaling tests for MD show that it simulates $3.2 * 10^{10}$ atoms on 6,240,000 master+slave cores with 41.3% parallel efficiency. We simulate $3.2 * 10^{10}$ atoms in 19.2 days temporal scale on 6,240,000 master+slave cores using the coupled MD-KMC approach, with runtime of 8.6 hours.

The key contributions of this paper are as follows:

(1) We improve the previously proposed data structure, lattice neighbor list [11], which significantly reduces the memory consumption while deals with the run-away atoms more efficiently.
(2) We compact the interpolation table for force calculation, which remarkably reduces the data transfer overhead between the main memory of CPU and the local store of slave cores.
(3) We propose an on-demand communication strategy for KMC to reduce the communication overhead.
(4) Experimental results show that our coupled MD-KMC approach exhibits good scalability on the Sunway TaihuLight supercomputer. The simulation results successfully reveal the vacancy cluster phenomenon.

In the next section, we discuss the coupled MD-KMC approach for the microscopic damage simulation and the optimizations for increasing the scalability on the heterogeneous parallel system. Sections 2.1 and 2.2 discuss MD and KMC, respectively. Experimental results and analysis on Sunway TaihuLight are presented in Section 3. Section 4 discusses the related work, and Section 5 concludes.

2 A COUPLED MD-KMC APPROACH

We use a coupled MD-KMC approach to simulate the microscopic damage under the environment of irradiation for iron material. The iron atoms are modeled in Body-Centered Cubic (BCC) structure. BCC has one lattice point in the center of the cube and eight corner points. If a atom runs away from the lattice point, it forms a vacancy at the lattice point. The side length of the cube is called lattice constant. The BCC structure is shown in Figure 1. MD simulates the defect generation caused by cascade collision, and outputs the coordinates of vacancy and the information of atoms. KMC simulates the defect evolution and vacancies clustering. The coupled MD-KMC approach has been intensively studied in the field of physics [16, 24, 29]. This paper focuses on how to scale it up on supercomputers.

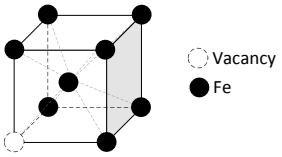


Figure 1: The structure of body-centered cubic.

Embedded-Atom Method (EAM) potential is used as physical interaction between the atoms. Typically, EAM potential consists of pair potential and embedding potential, as shown Equation (1), in which e denotes pair potential, F denotes embedding energy, and ρ denotes the electron cloud density. For a specific atom, pair potential e and electron cloud density ρ are the accumulated effect of the neighbor atoms within the cutoff radius, as shown in Equation (2) and Equation (3). For both MD and KMC, the EAM potential calculation is the core computation part. MD uses the EAM potential to calculate the forces of the atoms, and then updates the coordinates and the velocity of the atoms. KMC uses the EAM potential to calculate the probability of the vacancy transition.

$$E_{total} = \sum_{i=1}^{n} e_i + \sum_{i=1}^{n} F(\rho_i) \tag{1}$$

$$e_i = \frac{1}{2} \sum_{i \neq j} \phi_{ij}(r_{ij}) \tag{2}$$

$$\rho_i = \sum_{i \neq j} f_{ij}(r_{ij}) \tag{3}$$

To scale the application (both MD and KMC) across multiple computation nodes, we use the standard domain decomposition to equally partition the simulation box. Each computation node (i.e., each process) is responsible for a subdomain. To calculate the state of the atoms on the edge of the subdomain, the corresponding process has to access the data on the edge of the neighbor subdomains, which is commonly called ghost data. The ghost data should be updated with the increasing of the time steps. Thus, each process should communicate with the neighbor processes to exchange the ghost data after each time step (or several time steps).

2.1 Molecular Dynamics

2.1.1 Lattice neighbor list for the BCC structure. The simulation is based on the short-range forces. Thus, only the atoms within a specific cutoff radius interact with the central atom. Two data structures are commonly used to find the interaction atoms: neighbor list [22, 25] and linked cell [9, 19, 27]. For neighbor list, each atom maintains a list to store all the neighbor atoms within a distance which is equal to the cutoff radius plus a skin distance. Thus, the memory consumption of neighbor list is costly. The neighbor atoms should be updated after several time steps. Linked cell divides the simulation box into cubic cells, whose edge length is equal to the cutoff radius. Linked cell ensures that all interaction partners for any given atom are located either within the cell of itself or the surrounding cells. Each cell maintains all the atoms within it and the pointers to the neighbor cells. Compared with neighbor

list, linked cell consumes less memory. However, it should update the atoms within each cell at each time step, which leads to high computational overhead.

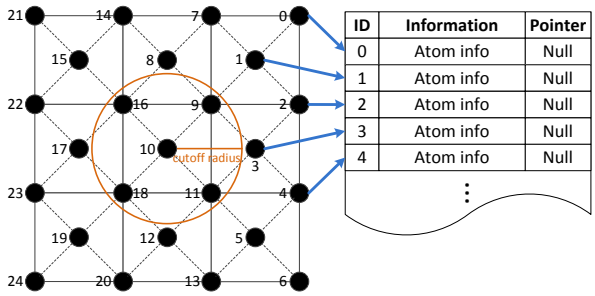


Figure 2: Lattice neighbor list for a 2-dimension simulation box. The black circles represent the atoms.

We focus on the microscopic damage simulation of the metal material with the BCC structure under the environment of irradiation. In this situation, most of the atoms stay very close to the lattice point and only a few atoms would break the constrain and run away from the lattice point. Based on this physical feature, we design a dedicated data structure, called **lattice neighbor list**, to store the atom information and record the neighbor atoms. We rank the atoms in the order of their spatial distribution. The information of the atoms, such as coordinates, velocity, force, and electron cloud density, is sequentially stored in a array in the order of the atoms ranks. An example with a 2-dimension simulation box is shown in Figure 2. Meanwhile, the neighbor atoms for a central atom are also regularly distributed in a region determined by the cutoff radius. Therefore, we can easily calculate the indexes of the neighbor atoms in the array for a central atom. As shown in Figure 2, the neighbor atoms of $atom_{10}$ are $atom_9$, $atom_{11}$, $atom_{16}$, and $atom_{18}$. For each central atom, the offsets of the neighbor atoms relative to the central atom are the same. This means the indexes of the neighbor atoms for each central atom can be calculated in the same way.

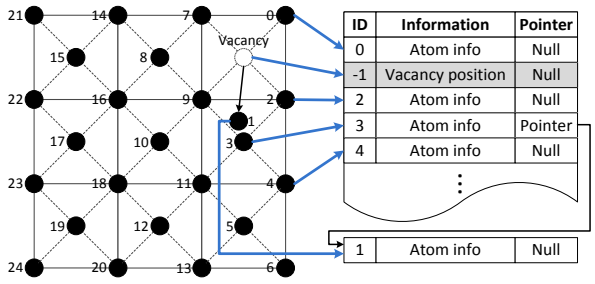


Figure 3: Data structures for run-away atoms. The black circles represent the atoms.

Next, we discuss how to deal with the atoms which run away from the lattice point in the lattice neighbor list structure. When an atom runs away from the lattice point, it generates a vacancy, as shown in Figure 3. We allocate another memory space to store the information of the run-away atom, and leave the original entry in the array to record the coordinates of the vacancy (ID is modified to a negative number to indicate this is a vacancy). If a run-away atom moves to a vacancy at the lattice point, the information of the vacancy in the array is overlapped by the run-away atom. Otherwise, the information of the run-away atom is linked to the entry of the nearest lattice point to facilitate further processing. In Figure 3, $atom_1$ runs away and its information is linked to the entry of the nearest regularly distributed neighbor $atom_3$. When a central atom (regularly distributed) finds its neighbor atoms within the cutoff radius, it not only checks the regularly distributed atoms, but also checks the run-away atoms linked to the lattice points. When a run-away atom finds its neighbor atoms within the cutoff radius, it checks the same neighbor atoms as the nearest lattice point it is linked to. Here, the extra overhead is the calculation of the nearest lattice point that the run-away atom is linked to. In our simulation, the number of the run-away atoms is only several millionth of the number of all the atoms. Therefore, the extra overhead can be ignored.

When exchanging the ghost data, the lattice points (either an atom or a vacancy) in the ghost region is packed (unpacked) and sent (received) according to the indexes in the array. For the ghost data at the lattice points, the communication pattern is static, which can be reused at each time step. For the run-away atoms, if they move into the subdomain or the ghost region of the neighbor processes, we pack their information and send it to the corresponding neighbor processes.

While the authors of [11] have discussed the lattice neighbor list structure, this paper further improves the structure by storing the run-away atoms using linked lists rather than an array. The benefit of using linked lists is two-fold. Firstly, the number of run-away atoms may exceed the size of the array. The linked list can overcome this drawback by having a dynamic size. Secondly, when using the array, the overhead of finding neighbors between the run-away atoms is $O(N^2)$ (where N is the number the run-away atoms), since the relative position information between the run-away atoms is lost. On the contrary, the linked lists can reduce this overhead to $O(N)$ since the run-away atoms are linked to the nearest lattice point. Compared with the traditional neighbor list and linked cell structures, our lattice neighbor list structure significantly reduces the memory and computation cost, since it does not have to maintain the extra structures and finds most of the neighbor atoms by static indexes. Compared with LAMMPS (using neighbor list) [22] and IMD (using linked cell) [27], our lattice neighbor list structure reduces the memory consumption significantly, and thus can simulate a larger number of atoms.

2.1.2 Reduce the data transfer cost between main memory and local store. Next we discuss how to efficiently use slave cores on Sunway many-core processor to accelerate the calculation of EAM potential. First, we briefly introduce the Sunway many-core architecture [6]. As shown in Figure 4, it consists of four core groups (CGs). Each CG includes one management processing element (MPE), one computing processing element (CPE) cluster organized as an 8x8 mesh, and one memory controller (MC). For convenience, we call MPE as master core and call CPE as slave core in this paper. Each slave core has 64 KB local store, which can be configured as either

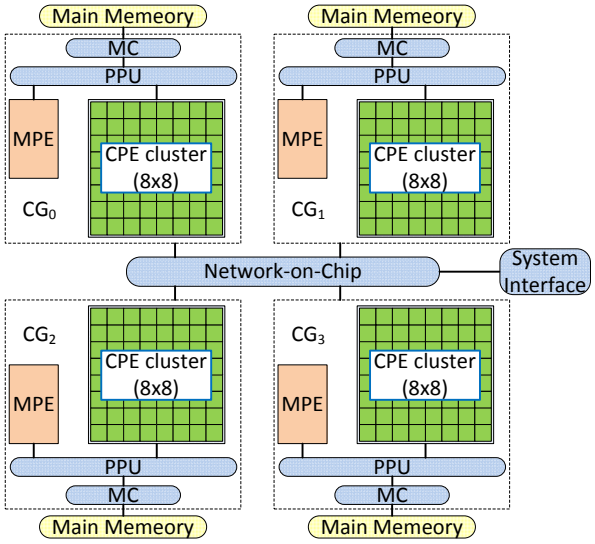


Figure 4: Sunway many-core architecture.

a user-controlled buffer or a software-emulated cache that achieves automatic data caching. Here we use it as a user-controlled buffer since it generally obtains better performance [6]. These four CGs are connected via the network on chip. The processor connects to other outside devices through a system interface (SI). To accelerate the computation using slave cores, we use one process on each master core, and each process launches 64 threads (running on 64 slave cores) using the *Athread* multithreading library supported by Sunway TaihuLight. The subdomain of each process is further equally partitioned into slabs, and each thread is responsible for one slab.

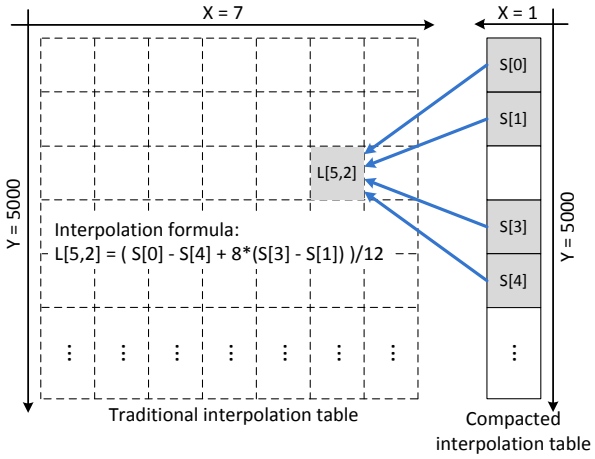


Figure 5: Interpolation table compaction (using pair potential table as an example). L[5,2] is calculated using S[0], S[1], S[3], and S[4] in the compacted table.

To calculate the EAM potential, we use the cubic spline interpolation method, which is also used in other MD software, such as LAMMPS [22] and CoMD [8]. Typically, three interpolation tables are used for EAM potential computation, including electron cloud density table, pair potential table, and embedding potential table. During the computation, these three tables will be accessed sequentially. Taking electron cloud density table as an example, table querying would return the value of the electron cloud density according to the distance between two atoms. Each traditional interpolation table (used in LAMMPS and CoMD) is a 5000*7 2D array, where "5000" is the maximum distance between two atoms, as shown in Figure 5. The interpolating function consists of 5000 segments (for different distances) of cubic functions. The 2D array is the coefficient matrix for the cubic functions, in which the columns 3-6 are the coefficients of a cubic function and the columns 0-2 are the coefficients of its derivative function. Thus, there are total 7 columns for the 2D array. The size of each traditional interpolation table is about 273 KB, which exceeds the size of local store (64 KB) on each slave core. Thus, traditional interpolation table can not be loaded in the local store at one time. Each slave core has to frequently uses DMA get operations (3 times for each neighbor atom at each time step) to transfer the table entries from main memory to local store, which severely damages the overall performance. To solve this problem, we use a **compacted interpolation table**, of which size is only 39 KB (1/7 of the traditional table). The compacted interpolation table contains the values of 5000 sampling points (i.e. the values of pair potentials between different distances). We load the whole compacted table into the local store at one time. Then, all the values in the traditional table can be calculated on the fly using the compacted table and a specific interpolation formula, as shown in Figure 5. Although this will bring extra computational overhead, it can be amortized by significantly reducing the data transfer overhead.

For alloy materials, more interpolation tables are used, since there are different kinds of interaction for different atomic pairs. Taking the Fe-Cu alloy as an example, there are three kinds of electron cloud density tables, for the atomic pairs of Fe-Fe, Cu-Cu, and Fe-Cu, respectively. The total size of these three compacted tables will exceed the size of local store. Thus, we only load the compacted table for the element with the highest content in the local store, since it would be the most frequently used, and leave the other tables in the main memory. Another method is to distribute all the tables to the local stores of neighbor slave cores, and use register communication supported by Sunway many-core architecture to transfer data between the local stores. However, since which data in the tables should be transferred cannot be known before runtime, it is very difficult to describe these irregular communications using register communication.

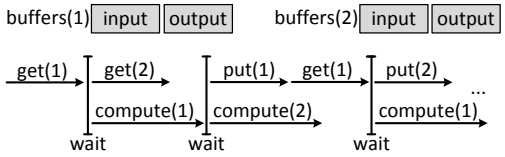


Figure 6: Double buffer for overlapping atoms information transfer with computation.

Since our simulation has large spatial scale, the atoms information of one slab cannot be loaded into the local store at one time either. Thus, each slab is further partitioned into blocks, and each slave core processes the blocks one by one. For each block, the slave core would get the atoms information into local store, and put back the output results (the updated position and velocity) to the main memory at each time step. To reduce the overhead of atoms information transfer, we propose two methods: (1) **Data reuse**. For the atoms information of one block, the data on the edge of the block is actually the ghost data for the next block. Thus, we keep the ghost data in the local store and reuse it in the next block, which reduces the overhead of data transfer to some extent. (2) **Double buffer**. Each slave core allocates two buffers on local store for the input and output data of each block. While carrying out DMA put or get on one buffer, it computes the potential or force on the other buffer, and vice versa. In this way, we overlap atoms information transfer with computation, as shown in Figure 6.

2.2 Kinetic Monte Carlo

Since the time step of MD is very short (typically at femtosecond scale), the temporal scale of MD simulation is generally limited to nanoseconds or less [32]. Kinetic Monte Carlo (KMC) is a stochastic method, which simulates the time evolution of some processes occurring with known transition rates among states. In our application, to couple MD with KMC, the temporal scale of MD should at least be 50 picoseconds. MD outputs the coordinates of vacancies and atoms, which are used as the input of KMC. KMC continues to simulate the vacancy clustering and evolution at a much larger temporal scale.

There are several different KMC approaches, such as atomistic KMC (AKMC) [1] and object KMC (OKMC) [15]. We choose to use AKMC to reveal the defect evolution for metal material. AKMC uses an on-lattice approximation method to map each atom or vacancy to a lattice point, and the atoms and vacancies are uniformly named as 'sites'. It describes the vacancy transitions (or events) as the position exchanges between atoms and vacancies. For the BCC structure in a 3D simulation box, as shown in Figure 1, there are eight possible events for a vacancy (since it may exchange with one of its eight nearest neighbors). The transition rate k_{ij} is calculated as

$$k_{ij} = v \exp(-\Delta E_{ij}/k_B T) \tag{4}$$

where v is the pre-exponential factor, ΔE_{ij} is the migration energy (calculated by EAM potential) from the state i to the state j, k_B is Boltzmann constant, and T is the temperature. We use the interpolation method to calculate the EAM potential, which is the same as MD and can be accelerated by the slave cores.

We use the semirigorous synchronous sublattice method [26] based on domain decomposition to scale KMC across multiple nodes. The flowchart for the parallel KMC algorithm is given in Figure 7. Firstly, we initialize the model by the outputs of MD simulation and other parameters, such as the coordinates of the sites, sites type (Fe or vacancy), and time threshold $T_{threshold}$. After domain decomposition, each process owns the sites within its subdomain and the surrounding ghost sites. Different from MD, KMC requires that all the sites in a subdomain be in the latest state even within a time step. Thus, to eliminate the conflicts on the boundary of the

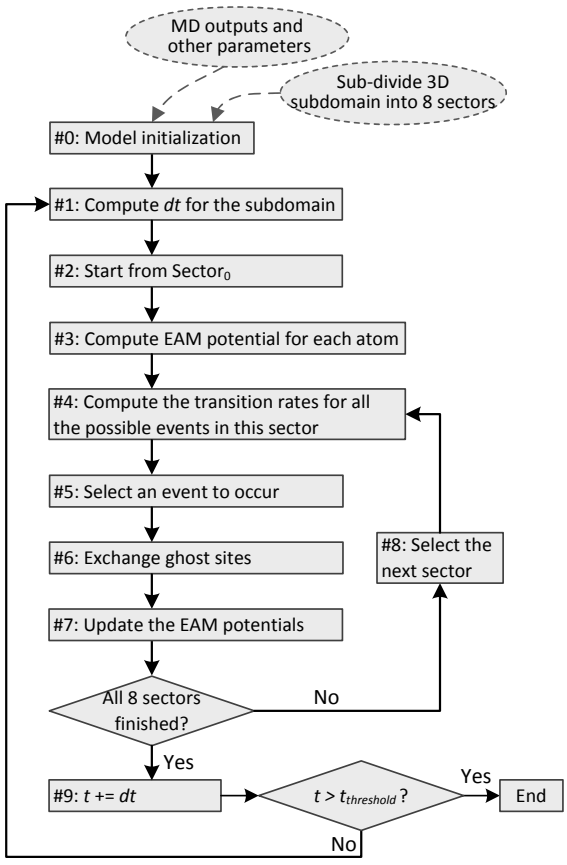
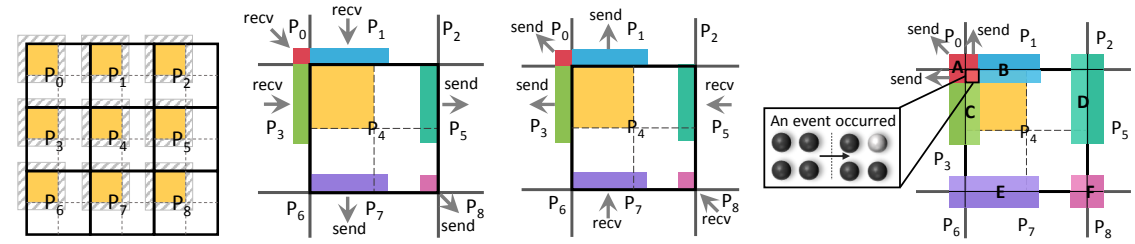


Figure 7: The flowchart of the parallel KMC simulation.

subdomain, each subdomain is further partitioned into sectors [26]. Typically, there are eight sectors for each process in a 3D simulation box and the eight sectors will be processed sequentially. Secondly, the time step dt is calculated and the simulation starts from the first sector. The transition rates for all the possible events within the current sector are calculated according to the EAM potentials. After the selected event occurs, it would exchange the ghost sites with the neighbor processes and update potential of the influenced atoms within the cutoff radius. At last, after finishing the simulation on all the eight sectors, it would repeat from computing the next time step dt until the time threshold $t_{threshold}$ is reached. In the algorithm, the communication is mainly for ghost sites exchange. Although the communication is regular with fixed neighbors and communication traffic, the communication overhead significantly increases with the increasing of computing nodes and the workload [31]. Thus, we present an on-demand communication strategy to reduce the communication traffic.

2.2.1 On-demand communication strategy. To discuss the on-demand communication strategy in detail, we use an example of nine processes on a 2D simulation box as shown in Figure 8. The 2D box is equally partitioned into nine subdomains, and each process further partitions its subdomain into four sectors. Each sector consists of local sites and the surrounding ghost sites (of which



(a) 2d domain partition for 9 processors (b) Get the latest ghost sites from neighbor processes (c) Put the ghost sites back to the neighbor processes (d) On-demand communication

Figure 8: The traditional communication pattern of KMC and the on-demand communication method.

width is determined by the cutoff radius). As shown in Figure 8(a), the solid lines represent the boundaries of the subdomains, the dotted lines represent the boundaries of the local sites of the sectors, and the shading region represents the ghost sites for a sector. Before processing a sector, each process has to get partial ghost sites (except those in the local subdomain) from the subdomains of its neighbor processes, as shown in Figure 8(b). After finishing the simulation of the current sector, each process has to put the ghost sites back to its neighbor processes, as shown in Figure 8(c). The philosophy of the two-time communication for each sector is to keep the sites in the subdomain and its ghost sites always in the latest state. This two-time communication pattern is widely used in the KMC software, such as SPPARKS [23] and KMCLib [14]. The neighbor processes and the send/receive buffers are determined at the initialization phase, and keep static as the simulation moves on. All the sites in the ghost region have to be transferred regardless of whether all the sites are updated or not. This causes a lot of redundant communication, since generally only a very small part of sites in the ghost region are affected when an event occurs. The communication redundancy is more expensive as the spatial scale of the simulation getting larger.

To eliminate the communication redundancy, we present an on-demand communication strategy. It changes the traditional static communication pattern to a dynamic communication pattern, namely the sites to be transferred and the neighbor processes are determined at runtime according to the state of the simulation system. When a vacancy transition (an event) occurs, it only affects the potential of atoms within the cutoff radius and the other sites keep steady. To keep the sites in the subdomain and the ghost sites always in the latest state, we only have to transfer the affected sites to the corresponding neighbor processes after the simulation of a sector within a time step is finished. As illustrated in Figure 8(d), after the top-left sector of P_4 is processed, it checks whether the sites in regions A, B, and C are affected and puts the influenced sites into the sending queue. Taking region A as an example, a site in region A is either a ghost site or a local site from the perspective of P_4' three neighbor processes (P_0, P_1, and P_3). Thus, if any site in region A is affected by the events, P_4 would send it to P_3, P_0, and P_1. Similar acts are carried out for regions B and C. Meanwhile, P_4 receives data from the neighbor processes and unpacks the data into either region D, E, or F to update the sites. In this way, only the affected sites are transferred. Considering the vacancy concentration is very low in our application, the communication traffic is reduced significantly compared with the traditional communication pattern. The on-demand communication strategy can be implemented using MPI [17] two-sided communication interfaces, and both sender and receiver have to know all the information about the message. Since the *source*, the *tag*, and the *size* of the messages are determined at runtime, the receiver does not know this information before receiving the messages. Thus, the receiver has to use *MPI_Probe* to query the information beforehand, and launch *MPI_Recv* afterwards to receive the actual data. However, since there is a match between the sender and the receiver, the sender has to send a zero-size message to the receiver even there is no update in the ghost sites. Alternatively, we can use MPI one-sided communication interfaces, by which only one side is involved in the communication, to eliminate these zero-size messages. Firstly, each process opens a globally-shared window on the subdomain. Secondly, each process puts the updates in the ghost sites to its neighbor processes. Thirdly, a global synchronization is carried out to guarantee the completion of the communications.

3 EVALUATION

Our experiments are conducted on the Sunway TaihuLight supercomputer. The Sunway TaihuLight has total 40,960 computing nodes. The architecture of the computing node has been introduced in Section 2.1.2. For a core group, there is total 8 GB DDR3 memory shared by a master core and 64 slave cores. Both master and slave cores work at 1.45GHz and support 256-bit vector instructions. Each master core has a 32 KB L1 cache and a 256 KB L2 cache, and each slave core has a 64 KB local store. The software environment of the system includes the customized 64-bit Linux kernel, and Sunway compiler version 5.4 supporting C/C++, Fortran, OpenMP, and OpenACC. The *Athread* multithreading library is provided to program on slave cores, and MPI library is provided for inter-node communication. In the experiments, we use the microscopic damage simulation for the Fe metal material at 600K temperature under the environment of irradiation. We demonstrate our application, implemented as the coupled MD-KMC model, achieves good performance and scalability on the Sunway TaihuLight supercomputer. It is unfair to compare the performance of our solution with the existing software (such as LAMMPS and SPPARKS) directly. The reasons include : (1) The existing software is not optimized for

Sunway architecture. (2) Our KMC implementation supports EAM potential and scales up to 100,000 processes; on the contrary, the existing parallel scalable software, like SPPARKS, does not support EAM potential.

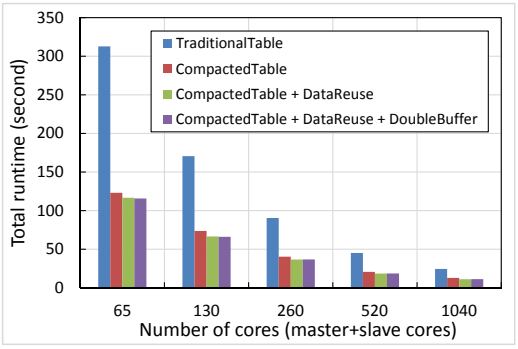


Figure 9: Performance comparisons for the optimizations of MD with $2 * 10^7$ atoms on Sunway many-core machines.

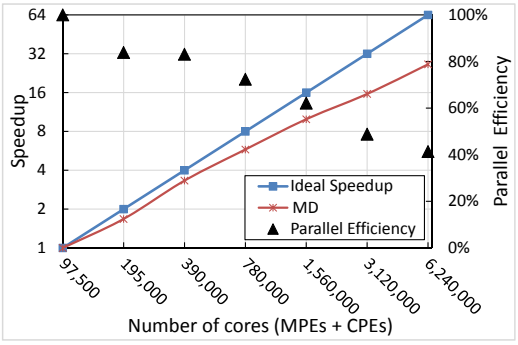


Figure 10: Strong scaling of MD with $3.2 * 10^{10}$ atoms using both master and slave cores.

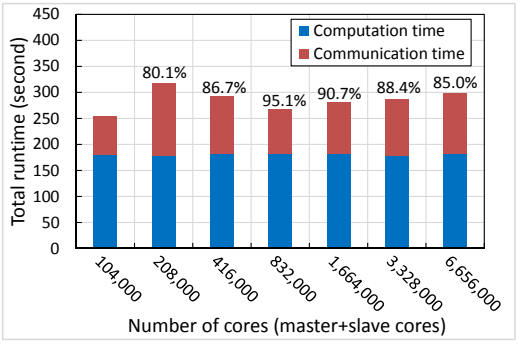


Figure 11: Weak scaling of MD using both master and slave cores, $3.9 * 10^7$ atoms per core group. Parallel efficiency is annotated on the top of the bar.

Firstly, we evaluate the optimization methods for reducing the data transfer cost between the main memory and the local store discussed in Section 2.1.2. We test the MD simulation with $2 * 10^7$ atoms using different numbers of cores as shown in Figure 9. Compared with the traditional interpolation tables, the compacted tables improve the performance by 54.7% on average in geometric mean. This is because using the compacted table significantly reduces the number of DMA operations. Ghost data reuse further improves the performance by 4% on average. However, double buffer does not bring obvious performance improvement, since there is not enough computation to overlap the data transfer.

Figure 10 shows the strong scaling test for our optimized MD simulation with $3.2 * 10^{10}$ atoms. For MD, the master cores are responsible for inter-node communication and the slave cores are responsible for the EAM computation. We can see that the parallel efficiency gradually decreases as the increasing of the computing resources, which is caused by the communication overhead. Scaling from 97, 500 cores to 6, 240, 000 cores, we achieve 26.4-fold speedup (41.3% parallel efficiency). Figure 11 shows the weak scaling test for our optimized MD simulation. As we increase the number of cores from 104, 000 (including 1, 600 master cores and 1, 024, 000 slave cores) to 6, 656, 000 (including 102, 400 master cores and 6, 553, 600 slave cores), the problem size increases from $6.25 * 10^{10}$ atoms to $4.0 * 10^{12}$ atoms to keep the workload per core fixed. Our MD code scales up to 6.656 million cores with total $4.0 * 10^{12}$ atoms by a 85% parallel efficiency. Using the traditional data structures (such as neighbor list), we only simulate about $8.0 * 10^{11}$ atoms on 6.656 million cores. The lower memory consumption of our lattice neighbor list structure contributes to a much larger spatial scale of MD. We can see that the computation time remains almost constant on different numbers of cores. However, the communication time for larger number of cores is a little higher, which is caused by the communication contention. The communication time on 208, 000 cores is a little higher than others, which is probably caused by the processes topology mapping. Overall, our MD code exhibits good strong and weak scalability.

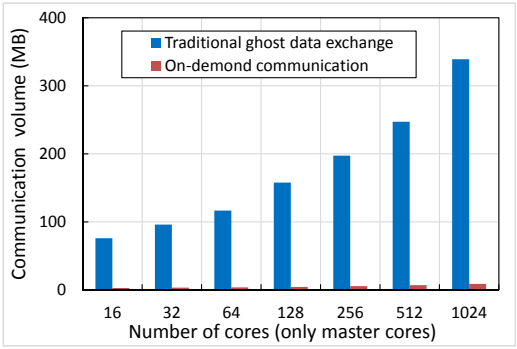


Figure 12: Communication volume comparison for KMC with $1.6 * 10^7$ sites using only master cores, vacancy concentration is $4.5 * 10^{-5}$.

Figure 12 presents the communication volume comparison between the on-demand communication strategy for KMC and the traditional ghost data exchange method (used in SPPARKS [23] and KMCLib [14]). The experiments are conducted on different number

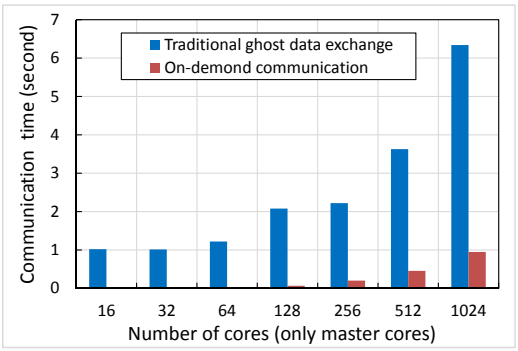


Figure 13: Communication time comparison for KMC with $1.6 * 10^7$ sites using only master cores, vacancy concentration is $4.5 * 10^{-5}$.

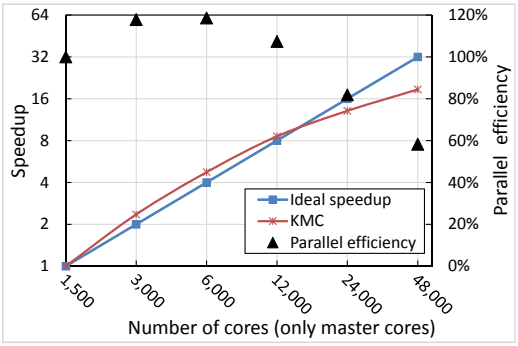


Figure 14: Strong scaling of KMC with $3.2 * 10^{10}$ sites using only master cores, vacancy concentration is $4.5 * 10^{-5}$.

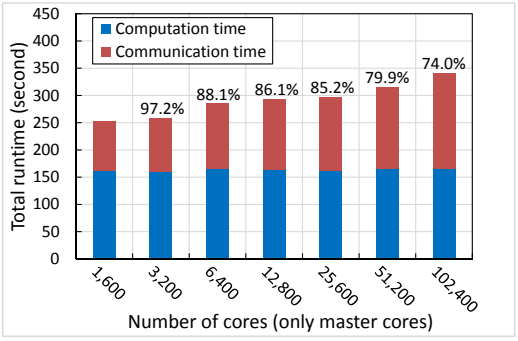


Figure 15: Weak scaling of KMC using only master cores, 10^7 sites per core, vacancy concentration is $2 * 10^{-6}$. Parallel efficiency is annotated on the top of the bar.

of cores (only master cores are used). The on-demand communication strategy reduces the communication volume to 2.6% of the traditional method on average. This is because the vacancy concentration is very low and only a few sites in the ghost region are updated after each time step. Figure 13 presents the communication time comparison between the on-demand communication strategy and the traditional ghost data exchange method. Compared

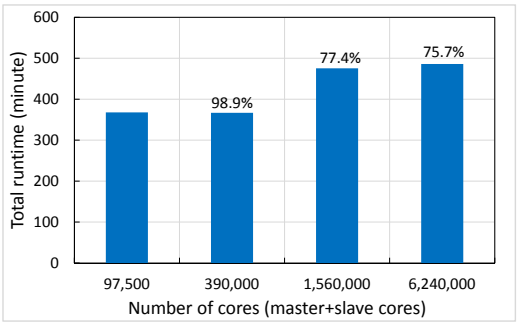
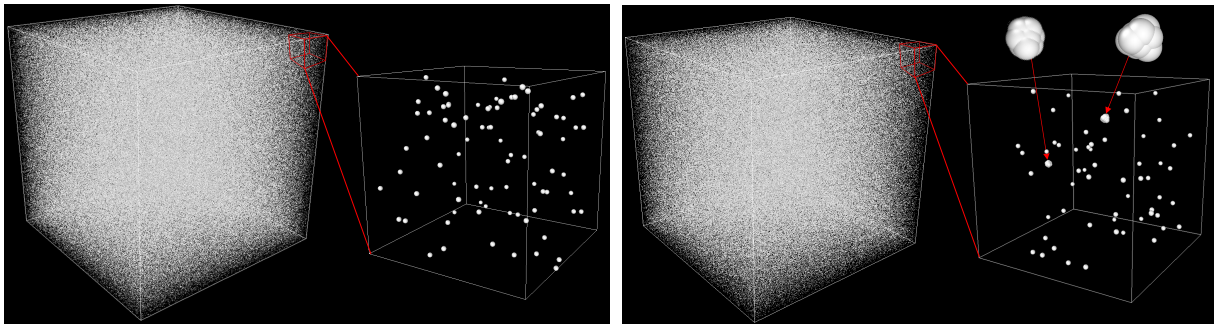


Figure 16: Weak scaling of the coupled MD-KMC approach using both master and slave cores, $3.3 * 10^5$ atoms per core group. Parallel efficiency is annotated on the top of the bar.

with the traditional method, the on-demand communication strategy obtains 21x speedup on average in terms of communication time. Figure 14 presents the strong scaling test for KMC, in which only master cores are used. The baseline runs on 1,500 cores with $3.2 * 10^{10}$ sites. Our KMC algorithm exhibits 18.5-fold speedup on 48,000 cores, indicating 58.2% parallel efficiency in strong scaling. The super-linear speedup from 3,000 to 12,000 cores is due to the benefit of L2 cache on the master cores, which can store the entire dataset. However, with the number of cores increasing, the communication cost becomes the major limitation. Figure 15 shows the weak scaling test for KMC in which only master cores are used. We keep 10^7 sites per core as the number of cores increases from 1,600 to 102,400. We observe that the computation time remains almost constant while the communication time increases gradually. The increased communication time is due to the collective operations used for time synchronization. Our KMC code scales up to 102,400 cores with 74% parallel efficiency. Our KMC code exhibits good strong and weak scalability.

We also present the weak scaling results for the coupled MD-KMC model in Figure 16. The number of cores increases from 97,500 to 6,240,000 while the number of atoms increases from $5.0 * 10^8$ to $3.2 * 10^{10}$. The results show that our coupled MD-KMC model achieves good weak scalability, and attains 75.7% parallel efficiency on 6,240,000 cores.

Furthermore, in order to demonstrate the ability of the coupled MD-KMC model to simulate at large spatio-temporal scale, we conduct a simulation of microscopic damage evolution in Fe material with $3.2 * 10^{10}$ atoms on 6,240,000 cores. The lattice constant is set to 2.855. MD simulates the defect generation caused by cascade collision in the temporal scale of 50 picoseconds (time step is set to 1 femtosecond), and outputs the coordinates of vacancy and the information of atoms. KMC continues to simulate the vacancy clustering and evolution in the temporal scale of days. The temporal scale (real time) of KMC simulation can be calculated by the formula $t_{real} = t_{threshold}\, C_v^{MC}/C_v^{real}$ [2]. Here, $t_{threshold}$ is the threshold for the time steps, C_v^{MC} is the vacancy concentration in the simulation box, and C_v^{real} is the real vacancy concentration in the experiment. $t_{threshold}$ is set to 0.0002. C_v^{MC} is 0.000002, which is easily obtained by calculating the percentage of vacancies in atoms. C_v^{real} is obtained by $C_v^{real} = exp(-E_v^+/K_B\, T)$, where E_v^+ is vacancy



(a) The distribution of vacancies after MD

(b) The distribution of vacancies after KMC

Figure 17: The simulation results for $3.2 * 10^{10}$ atoms in 19.2 days temporal scale. The white points represent vacancies.

formation energy, k_B is the Boltzmann constant, and T is set to 600K. After calculation, the temporal scale t_{real} is equal to 19.2 days. The total runtime of the application on 6,240,000 cores is 8.6 hours. The simulation results are shown in Figure 17, in which only the vacancies are shown. The evolution of the system after MD simulation is shown in Figure 17(a), from which we can see that the vacancies are very dispersive. The evolution of the system after KMC simulation is shown in Figure 17(b), from which we can see the vacancies are relatively more aggregative and several vacancy clusters are forming.

4 RELATED WORK

Molecular dynamics simulation has been intensively studied in materials computing. There are different kinds of parallel software for large-scale MD simulation, such as LAMMPS [22], ls1-MarDyn [19], IMD [27], and CoMD [8]. LAMMPS [22] is a widely used MD simulator, which supports simulation in solid-state and soft materials. It supports various platforms, such as CPUs, GPUs, and Xeon Phi, and uses MPI to scale across nodes. LAMMPS adopts the neighbor list structure to record the interaction atoms within cutoff radius. IMD [27] is a software package designed for classical MD simulations, which allows the particles to interact via pair and multi-body potentials. In 1997 and 1999, IMD had twice held the MD world record by simulating $5 * 10^9$ particles focusing on multi-body potentials. Ls1-MarDyn [19] is designed to simulate homogeneous and heterogeneous fluid systems containing large numbers of molecules. It focuses on rigid molecules, and only constant-volume ensembles are supported. Ls1-MarDyn held the world record of 2014 by simulating $4.125 * 10^{12}$ single-site Lennard-Jones (LJ) particles on the SuperMUC system. CoMD [8] is a parallel software for a broad class of molecular dynamics simulations, which supports LJ and EAM potentials. IMD, ls1-MarDyn, and CoMD use the linked cell structure to record the neighbor atoms. Different from the mainstream software, we design the lattice neighbor list structure to reduce memory consumption.

Streitz et al. [28] applied the developed ddcMD code to simulate the first atomic-scale model of metal solidification, and achieved 103 TFlops performance on a BlueGene/L machine. Swaminarayan et al. [7] rewrote SPaSM code for the heterogeneous Roadrunner supercomputer. They performed a standard LJ potential benchmark and achieved 369 Tflops performance in double precision. Höhnerbach et al. [10] optimized the Tersoff potential by vectorization, and obtained good performance on different architectures. Dong et al. [5] transplanted the Tersoff potential computation in LAMMPS on the Sunway TaihuLight supercomputer and optimized the performance by vectorization, which achieved good parallel efficiency for up to 10^8 particles. Hu et al. [12] discussed the multi-threading and SIMD optimizations for molecular dynamics simulation on Intel Xeon Phi.

For KMC, Shim et al. [26] presented a semirigorous synchronous sublattice algorithm for parallel KMC simulation based on domain decomposition. Nandipati et al. [18] carried out realistic parallel KMC simulations of Ag(111) island coarsening using a large database. KMCLib [14] is designed to facilitate the implementation of customized KMC models. For better load balancing, KMClib would re-match the sites with the processes at each step, which makes it less efficient when running on a large number of processes. SPPARKS [23] defines three kinds of Monte Carlo models and facilitates the implementation of customized KMC models. For the Potts model using pair potentials, SPPARKS has been proven to achieve excellent scalability on thousands processors [21]. Wu et al. [30, 31] discussed the communication optimizations for SPPARKS using neighborhood collectives and shared-memory communication. Different from the existing work, we propose an on-demand communication strategy for KMC, which significantly reduces the communication volume. Jiménez et al. [13] proposed a GPU-based OKMC algorithm for the simulation of defects evolution in irradiated materials. Morishita et al. [16] proposed a coupled MD-KMC model to simulate the helium-vacancy clusters in Fe.

5 CONCLUSION

We design a scalable coupled MD-KMC algorithm to simulate the microscopic damage of metal material in BCC structure. Many-body potential, EAM, is used as physical interaction. MD simulates the vacancy generation caused by cascade collision in 50 picoseconds. The connected AKMC approach extends the temporal scale of the simulation to weeks. The algorithm is implemented and optimized on the Sunway TaihuLight supercomputer. For MD, we simulate $4 * 10^{12}$ atoms on 6,656,000 master+slave cores with 85% parallel efficiency (weak scaling). Strong scaling tests for MD show it

simulates $3.2 * 10^{10}$ atoms on 6,240,000 master+slave cores with 41.3% parallel efficiency. Using the coupled MD-KMC approach, we simulate $3.2 * 10^{10}$ atoms in 19.2 days temporal scale on 6,240,000 master+slave cores with runtime of 8.6 hours. The simulation results successfully reveal the vacancy cluster phenomenon.

Although the algorithm is implemented on the Sunway TaihuLight supercomputer, our optimization methods are not hardware-specific. The proposed lattice neighbor list structure for MD provides a valuable reference for metal materials simulations if their spatial scale is limited by the memory consumption. The on-demand communication strategy for KMC are also useful on other big machines if its communication overhead hinders the scalability with the increasing of the computation nodes. Potential table compaction, ghost data reuse, and double buffer strategies would contribute to performance improvement on heterogeneous architectures where the data transfer overhead between main memory and local store is the bottleneck.

MD and KMC are typically considered as computation-intensive. However, their computation features, such as potential computation based on interpolation table and frequent global synchronization, make it not easy to scale up even on the state-of-the-art supercomputer. Although our software optimizations have alleviated the above problems to some extent, we also come up with some suggestions to solve the problems from the system perspective. For example, the high-performance register communication is supported to overcome the problems caused by the shortage of local memory. However, the register communication interfaces work similarly to the MPI two-sided communication, which makes them difficult to describe irregular data transfers (i.e., dynamically changing data access patterns), like loading the data in the interpolation tables in MD. Thus, efficient one-sided register communication, which facilitates the describing of irregular data transfers, is a promising alternative.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China under Grant No. 61502450, Grant No. 61432018, and Grant No. 61521092; National Key R&D Program of China under Grant No. 2017YFB0202302, Grant No. 2016YFE0100300, and Grant No. 2016YFB0200800.

REFERENCES

[1] CS Becquart and C Domain. 2010. Introducing chemistry in atomistic kinetic Monte Carlo simulations of Fe alloys under irradiation. *physica status solidi (b)* 247, 1 (2010), 9–22.

[2] Nicolas Castin, Maria Ines Pascuet, and Lorenzo Malerba. 2011. Modeling the first stages of Cu precipitation in α-Fe using a hybrid atomistic kinetic Monte Carlo approach. *The Journal of chemical physics* 135, 6 (2011), 064502.

[3] George Chrysos. 2014. Intel® Xeon Phi™ coprocessor-the architecture. *Intel Whitepaper* 176 (2014).

[4] Murray S Daw and Michael I Baskes. 1984. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Physical Review B* 29, 12 (1984), 6443.

[5] Wenqian Dong, Letian Kang, Zhe Quan, Kenli Li, Keqin Li, Ziyu Hao, and Xiang-Hui Xie. 2016. Implementing Molecular Dynamics Simulation on Sunway TaihuLight System. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on*. IEEE, 443–450.

[6] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. 2016. The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences* 59, 7 (2016), 072001.

[7] Timothy C Germann, Kai Kadau, and Sriram Swaminarayan. 2009. 369 Tflop/s molecular dynamics simulations on the petaflop hybrid supercomputer 'Roadrunner'. *Concurrency and Computation: Practice and Experience* 21, 17 (2009), 2143–2159.

[8] Riyaz Haque, Sam Reeve, Luc Juallmes, Sameer Abu Asal, Aaron Landmehr, Sanian Gaffer, Gheorghe Teodor Bercea, and Zach Rubinstein. 2014. *CoMD Implementation Suite in Emerging Programming Models.* Technical Report. Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States).

[9] Roger W Hockney and James W Eastwood. 1988. *Computer simulation using particles.* crc Press.

[10] Markus Höhnerbach, Ahmed E. Ismail, and Paolo Bientinesi. 2016. The Vectorization of the Tersoff Multi-body Potential: An Exercise in Performance Portability. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16).* IEEE Press, Piscataway, NJ, USA, Article 7, 13 pages.

[11] Changjun Hu, He Bai, Xinfu He, Boyao Zhang, Ningming Nie, Xianmeng Wang, and Yingwen Ren. 2017. Crystal MD: The massively parallel molecular dynamics software for metal with BCC structure. *Computer Physics Communications* 211 (2017), 73–78.

[12] Changjun Hu, Xianmeng Wang, Jianjiang Li, Xinfu He, Shigang Li, Yangde Feng, Shaofeng Yang, and He Bai. 2017. Kernel optimization for short-range molecular dynamics. *Computer Physics Communications* 211 (2017), 31–40.

[13] F Jiménez and CJ Ortiz. 2016. A GPU-based parallel object kinetic Monte Carlo algorithm for the evolution of defects in irradiated materials. *Computational Materials Science* 113 (2016), 178–186.

[14] Mikael Leetmaa and Natalia V Skorodumova. 2014. KMCLib: A general framework for lattice kinetic Monte Carlo (KMC) simulations. *Computer Physics Communications* 185, 9 (2014), 2340–2349.

[15] Ignacio Martin-Bragado, Antonio Rivera, Gonzalo Valles, Jose Luis Gomez-Selles, and María J Caturla. 2013. MMonCa: An Object Kinetic Monte Carlo simulator for damage irradiation evolution and defect diffusion. *Computer Physics Communications* 184, 12 (2013), 2703–2710.

[16] Kazunori Morishita, Ryuichiro Sugano, and BD Wirth. 2003. MD and KMC modeling of the growth and shrinkage mechanisms of helium–vacancy clusters in Fe. *Journal of nuclear materials* 323, 2 (2003), 243–250.

[17] MPI Forum. 2012. MPI: A Message-Passing Interface standard. Version 3.0.

[18] Giridhar Nandipati, Yunsic Shim, Jacques G Amar, Altaf Karim, Abdelkader Kara, Talat S Rahman, and Oleg Trushin. 2009. Parallel kinetic Monte Carlo simu ations of Ag (111) island coarsening using a large database. *Journal of Physics: Condensed Matter* 21, 8 (2009), 084214.

[19] Christoph Niethammer, Stefan Becker, Martin Bernreuther, Martin Buchholz, Wolfgang Eckhardt, Alexander Heinecke, Stephan Werth, Hans-Joachim Bungartz, Colin W Glass, Hans Hasse, et al. 2014. ls1 mardyn: The massively parallel molecular dynamics code for large systems. *Journal of chemical theory and computation* 10, 10 (2014), 4455–4464.

[20] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. 2008. GPU computing. *Proc. IEEE* 96, 5 (2008), 879–899.

[21] Steve Plimpton, Corbett Battaile, Mike Chandross, Liz Holm, Aidan Thompson, Veena Tikare, Greg Wagner, E Webb, X Zhou, C Garcia Cardona, et al. 2009. Crossing the mesoscale no-man's land via parallel kinetic Monte Carlo. *Sandia Report SAND2009-6226* (2009).

[22] Steve Plimpton, Paul Crozier, and Aidan Thompson. 2007. LAMMPS-large-scale atomic/molecular massively parallel simulator. *Sandia National Laboratories* 18 (2007).

[23] S Plimpton, A Thompson, and A Slepoy. 2010. SPPARKS kinetic Monte Carlo simulator.

[24] Joshua M Pomeroy, Joachim Jacobsen, Colin C Hill, Barbara H Cooper, and James P Sethna. 2002. Kinetic Monte Carlo–molecular dynamics investigations of hyperthermal copper deposition on Cu (111). *Physical Review B* 66, 23 (2002), 235412.

[25] Dennis C Rapaport, Robin L Blumberg, Susan R McKay, Wolfgang Christian, et al. 1996. The art of molecular dynamics simulation. *Computers in Physics* 10, 5 (1996), 54–58.

[26] Yunsic Shim and Jacques G Amar. 2005. Semirigorous synchronous sublattice algorithm for parallel kinetic Monte Carlo simulations of thin film growth. *Physical Review B* 71, 12 (2005), 125432.

[27] J Stadler, R Mikulla, and H-R Trebin. 1997. IMD: a software package for molecular dynamics studies on parallel computers. *International Journal of Modern Physics C* 8, 05 (1997), 1131–1140.

[28] Frederick H Streitz, James N Glosli, Mehul V Patel, Bor Chan, Robert K Yates, Bronis R de Supinski, James Sexton, and John A Gunnels. 2005. 100+ TFlop solidification simulations on BlueGene/L. In *Proceedings of IEEE/ACM Supercomputing*, Vol. 5.

[29] Angela Violi, Adel F Sarofim, and Gregory A Voth. 2004. Kinetic Monte Carlo–molecular dynamics approach to model soot inception. *Combustion science and technology* 176, 5-6 (2004), 991–1005.

[30] Baodong Wu, Shigang Li, and Yunquan Zhang. 2015. Optimizing Parallel Kinetic Monte Carlo Simulation by Communication Aggregation and Scheduling. In *National Conference on Big Data Technology and Applications*. Springer, 282–297.
[31] Baodong Wu, Shigang Li, Yunquan Zhang, and Ningming Nie. 2017. Hybrid-optimization strategy for the communication of large-scale Kinetic Monte Carlo simulation. *Computer Physics Communications* 211 (2017), 113–123.
[32] Haixuan Xu, Yuri N Osetsky, and Roger E Stoller. 2012. Self-evolving atomistic kinetic Monte Carlo: fundamentals and applications. *Journal of Physics: Condensed Matter* 24, 37 (2012), 375402.