OpenKMC: a KMC Design for Hundred-Billion-Atom Simulation Using Millions of Cores on Sunway Taihulight

Kun Li*
State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
School of Computer and Control Engineering, University of Chinese Academy of Sciences
Beijing, China
likungw@gmail.com

Honghui Shang*
State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
shanghui.ustc@gmail.com

Yunquan Zhang
State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
yunquan.zhang@gmail.com

Shigang Li
Department of Computer Science, ETH Zurich
Zurich, Switzerland
shigangli.cs@gmail.com

Baodong Wu
Institute of Computing Technology, Chinese Academy of Sciences
Sensetime Research
Beijing, China
wubd.cs@gmail.com

Dong Wang
College of Information Engineering, Dalian Ocean University
Dalian, China
yjswangdong@gmail.com

Libo Zhang
Fang Li
zlb03@hotmail.com
lifang56@163.com
Wuxi Jiangnan Institute of Computing Technology
Wuxi, China

Dexun Chen
National Supercomputing Center, Wuxi
Wuxi, China
adch@263.net

Zhiqiang Wei
Qingdao National Laboratory for Marine Science and Technology
Qingdao, China
weizhiqiang@ouc.edu.cn

ABSTRACT

With more attention attached to nuclear energy, the formation mechanism of the solute clusters precipitation within complex alloys becomes intriguing research in the embrittlement of nuclear reactor pressure vessel (RPV) steels. Such phenomenon can be simulated with atomic kinetic Monte Carlo (AKMC) software, which evaluates the interactions of solute atoms with point defects in metal alloys. In this paper, we propose OpenKMC to accelerate large-scale KMC simulations on Sunway many-core architecture. To overcome the constraints caused by complex many-core architecture, we employ six levels of optimization in OpenKMC: (1) a new efficient potential computation model; (2) a group reaction strategy for fast event selection; (3) a software cache strategy; (4) combined communication optimizations; (5) a Transcription-Translation-Transmission algorithm for many-core optimization; (6) vectorization acceleration. Experiments illustrate that our OpenKMC has high accuracy and good scalability of applying hundred-billion-atom simulation over 5.2 million cores with a performance of over 80.1% parallel efficiency.

CCS CONCEPTS

• **Theory of computation → Massively parallel algorithms**; • **Computing methodologies → Massively parallel and high-performance simulations**; • **Applied computing → Physics**; • **Software and its engineering →** *Massively parallel systems.*

KEYWORDS

Sunway TaihuLight, Multi/many-core Processor, KMC, large-scale simulation

*Both authors contributed equally to this research.

SC '19, November 17–22, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6229-0/19/11.
https://doi.org/10.1145/3295500.3356165

ACM Reference Format:
Kun Li, Honghui Shang, Yunquan Zhang, Shigang Li, Baodong Wu, Dong Wang, Libo Zhang, Fang Li, Dexun Chen, and Zhiqiang Wei. 2019. OpenKMC: a KMC Design for Hundred-Billion-Atom Simulation Using Millions of Cores on Sunway Taihulight. In *The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '19), November 17–22, 2019, Denver, CO, USA.* ACM, Denver, CO, USA, 12 pages. https://doi.org/10.1145/3295500.3356165

1 INTRODUCTION

Nuclear energy is an energy source released by nuclear reactions, which is most frequently utilized to provide emissions-free electricity around-the-clock in the nuclear power plant. Approximately 13% of the world's electricity is supplied by nuclear energy at present [28, 37]. However, the workloads on nuclear reactor components deteriorate them gradually and bring declining efficiency of their functionality. The primary contribution to the nuclear reactor pressure vessel (RPV) degradation can be attributed to the formation of Cu rich precipitates in RPV steels under neutron irradiation. Such solute clusters formation processes can be simulated at the atomistic level with kinetic Monte Carlo (KMC) methods [3, 20].

The Monte Carlo methods comprise several techniques such as atomic kinetic Monte Carlo (AKMC) [4], object kinetic Monte Carlo (OKMC) [2, 7, 9] and event kinetic Monte Carlo (EKMC) [19]. In this paper focus is laid on the AKMC simulation, where point defects are introduced in the system. In such simulation, the migration energy of the vacancy is highly important. These elementary properties are not known at the atomic level experimentally and can be obtained by *ab initio* calculations [5, 6]. Such calculations have been made to investigate the kind of interaction which exists between the point defects and the Cu, Ni, Mn and Si solute atoms [37], and used to model the microstructural evolution of a dilute FeCuNiMnSi complex alloy [27, 38].

Numerous research teams have been working on developing different AKMC approaches to simulate the microstructural evolution of metal alloy in RPV [1, 35, 37]. However, such softwares only have serial versions, and employ conventional empirical potential model extensively. Thus, their efficiencies are restricted in large-scale simulation. Here we propose a parallel AKMC software, OpenKMC, to accelerate simulation progress. We use a pair potential model in energy computation and divide the workloads on each process into fine-grained sectors for avoiding hop conflicts. Moreover, a series of optimization strategies are proposed to reduce the communication time and to avoid frequent global memory accesses.

In practice, there are also high demands to accelerate KMC simulations on supercomputers. Thus we implement and optimize our OpenKMC software on SW26010 processor, which is the main building block of the world-leading supercomputer Sunway TaihuLight.

The major contributions in this paper include:

- We employ a pair potential model to obtain the interaction energies without involving redundant computations.
- We partition the data into fine-grained sectors to solve hop conflicts at the shared boundary and use group reaction strategy to accelerate the selection of events in each simulation.
- A cache optimization strategy is implemented to fully utilize the memory bandwidth.
- We do a series of optimization for communication to eliminate the communication redundancy, reduce synchronous time and decrease communication frequency adaptively.
- We design a Transcription-Translation-Transmission algorithm and achieve vectorization to accelerate the computation procedure on SW26010 processor.

The remainder of this paper is organized as follows. In Sec. 2 we succinctly summarize the fundamental background of this study and discuss the challenges of parallel KMC on SW26010. Then

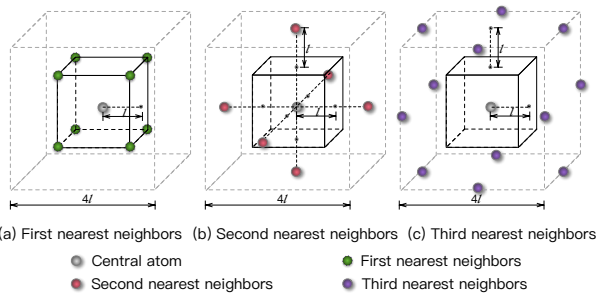

Figure 1: Three kinds of nearest neighbors for a bcc lattice.

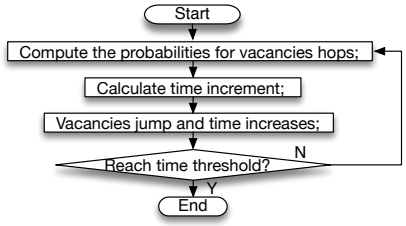

Figure 2: The serial AKMC method for vacancies hops.

our optimization schemes and their efficient implementation are discussed in Sec. 3. Furthermore, performance is evaluated exhaustively in Sec. 4. Eventually, Sec. 5 summarizes the main ideas and findings of this work.

2 BACKGROUND

2.1 Atomistic Kinetic Monte Carlo Method

The AKMC method is mainly utilized to provide an accurate solution that governs the evolution of the microscopic system. In our case, all events in simulation occur in the body-centred-cubic (bcc) lattice, and a diffusional hop event for point defects in it is considered as an exchange of a vacancy with the first 8 nearest neighbor atoms. Fig.1 shows the first, second and third nearest neighbors respectively for a central atom in a bcc lattice (length is $2l$). As Fig.2 shows, the solution could be built by four basic steps generally.

Computing the probabilities of various possible hops from the current state for vacancies is the first step, and this could rely directly on the corresponding transition rates. The probability for an event X in the evolution is formulated as Boltzmann factor frequency here, namely:

$$\Gamma_X = \Gamma_0 \cdot \exp(-\frac{E_a^X}{kT}) \ . \tag{1}$$

Here Γ_0 is an attempt frequency (s^{-1}), T is the absolute temperature, and E_a^X is the event migration energy (eV). The attempt frequency Γ_0 is set to 6×10^{12} s^{-1}. Taking the computation time and the precision of the results into account, we obtain an event migration energy E_a^X by employing an environment-dependent model [7], which satisfies the following balance rule:

$$E_a^X = E_a^0 + \frac{1}{2} \cdot (E_f - E_i) \ . \tag{2}$$

The reference activation energy E_a^0 (Fe:0.65 eV, Cu:0.56 eV used in this work) is assumed to depend only on the chemical nature of the migrating atom that exchanges position with the vacancy. E_i and E_f are the system total energies respectively before and after the vacancy hops [4, 30].

Then the residence time algorithm [41] is employed to calculate a time increment (Δt) in Equation 3, and it is proportional to the inverse of the sum of all possible event frequencies with a random number (r) according to its probability so that the event propagation is associated with each independent event.

$$\Delta t = \frac{-\ln r}{\sum_{X=1,8} \Gamma_X} \tag{3}$$

The next step is to determine a jump for each vacancy randomly with a weight corresponding to its probability, and update energies caused by these vacancies hops. At the same time, a time increment (Δt) is accumulated on the simulation. Finally, the AKMC method repeats the steps above until the simulation time has reached a preset threshold.

In addition, when computing energies in the first step, many models have been proposed by using the empirical interatomic potentials. One of the empirical potential is the embedded-atom method (EAM) [8], where the interaction energy of atoms is contributed by both the pairwise potential (up to the third nearest neighbor) and the many-body potential (the environmental dependence term)

$$E_{\mathrm{EAM}} = \sum_{\mu} \left[\frac{1}{2} \sum_{i=1}^{3} \sum_{\nu} \varepsilon_{(\mu\nu)}^{(n)} + F(\rho_\mu) \right] , \tag{4}$$

where μ, ν label the atomic index, F is the multi-body embedding energy, ρ_μ is the total electron density interpolated at host atom μ.

The EAM model has been widely used in a wide range of simulation applications [8, 11]. However, the EAM potential needs to be generated from *ab initio* calculations, and at present, there is no EAM potential for complex FeCuNiMnSi alloys and for the interstitials which are needed in the real nuclear engineering simulation.

On the other hand, the pair interaction model [34] can deal with such complex alloys [37] and the interstitials [38]. In the pair potential model, the interactions are considered up to the second nearest neighbor positions, and the chemical interactions between all the entities which compose the system have been described by the following equation

$$\begin{aligned} E_{\mathrm{pair}} = \quad & \textstyle\sum_j \varepsilon_{(\mathrm{Fe-Fe})}^{(i)} + \sum_k \varepsilon_{(\mathrm{V-V})}^{(i)} + \sum_l \varepsilon_{(\mathrm{Fe-V})}^{(i)} \\ & + \textstyle\sum_m \varepsilon_{(\mathrm{Fe-Cu})}^{(i)} + \sum_n \varepsilon_{(\mathrm{V-Cu})}^{(i)} + \sum_p \varepsilon_{(\mathrm{Cu-Cu})}^{(i)} , \end{aligned} \tag{5}$$

where i equals 1 or 2 and corresponds respectively to the first or second nearest neighbor interaction, j the number of Fe-Fe bonds, k the number of V-V bonds, l the number of Fe-V bonds, m the number of Fe-Cu bonds, n the number of V-Cu bonds, p the number of Cu-Cu bonds of the lattice.

The limitation of the pair interaction model is the lack of many-body potentials, so it cannot account for the influence of the local environment. In addition, the pair potential parameters need to be carefully tuned with experiments or thermodynamical data [37, 38].

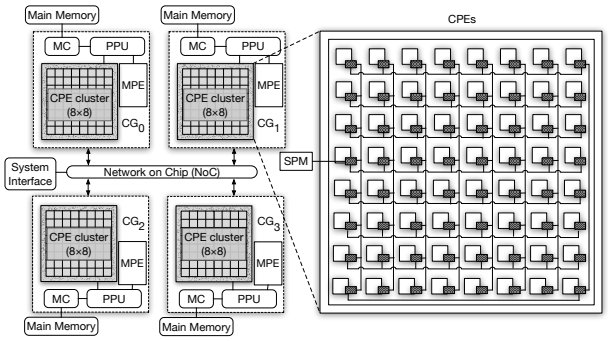

Figure 3: The general architecture of the SW26010 many-core processor.

The pair potential parameters used in this work are taken from Ref. [38], which can reproduce the EAM model result and also be extended in the dilute FeCuNiMnSi alloy studies.

2.2 Sunway TaihuLight and SW26010 Many-Core Processor

The world-leading supercomputer, Sunway TaihuLight, ranks as the third machine currently in the world and achieves a peak performance over 125 Pflops. The Sunway TaihuLight is mainly enabled by China's custom SW26010 many-core processor [13].

Fig. 3 exhibits the basic architecture for SW26010 processor. Each processor consists of four core groups (CGs) and they are connected to each other via the network on chip (NoC). Each CG includes 65 cores: 1 management processing element (MPE), and 64 Computing Processor Elements (CPEs), organized as an 8×8 mesh. The processor connects to other outside devices by a system interface (SI). Both the MPE and CPEs work at 1.45GHz and 256-bit vector instructions are supported. A CG has roughly 34.1 GB/s theoretical peak memory bandwidth and around 765 GFlops double-precision peak performance [12].

The SW26010 architecture is significantly different from other multi-core or many-core processors. For the memory hierarchy, the L1 data or instruction cache are both 32 KB on MPE, and a unified 256 KB L2 cache is also configured on it. Each CPE has a 16 KB L1 instruction cache, and a 64 KB local store (user-controlled scratch pad memory, SPM). For the internal communication of CPE mesh, we have a register communication mechanism including 8-column and 8-row communication buses respectively. The register communication in CPE mesh provides the possibility for establishing fast data transmission channels so that a significant data sharing capability is obtained on the CPE level [13, 23].

2.3 Parallel AKMC Implementation

To extend the time and length scales of the KMC simulation, the parallel KMC algorithm is desired. Here we implement the optimized algorithms in SPPARKS [17], and adapted the semirigorous synchronous sublattice method of Shim and Amar [32] to solve boundary inconsistencies and avoid global communications. As the effects could be neglected if the event occurs far away from the current site [31], the parallelism is enabled by decoupling events

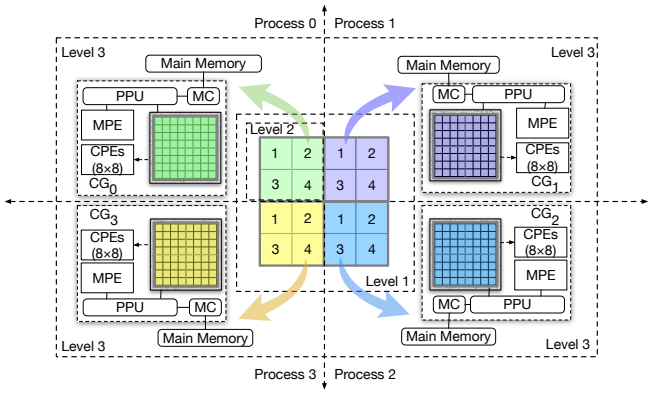

Figure 4: Parallel AKMC Implementation for 4 processes. Each process is sub-divided into 4 sectors and each sector is further distributed to CPEs.

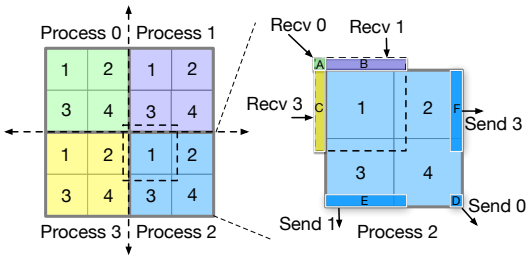

Figure 5: Ghost cells computation with 4 sectors per sub-domain. The left part shows the first blue sector has a surrounding dotted square of sites it needs, some of them owned by other processes. The right part exhibits the sites the process will send and receive in 3 exchanges (green, purple, yellow) from other processes to update the ghost cells surrounding its first quadrant.

spatially. We exploit this idea in three levels. First, all data are partitioned to the available processors. Fig. 4 illustrates this idea in a 2d simulation domain, split across a 2×2 grid of 4 processes on SW26010. However, in the traditional algorithm, when two or more processes perform events simultaneously near the shared boundary, they could execute in conflicting hops with incorrect probabilities.

The second level is to solve this problem by dividing the sub-domain into more fine-grained sectors, typically 4 quadrants in 2d or 8 octants in 3d [31]. As shown in Fig. 4, all processes execute events in sequence according to sector number on own sub-domain independently for a period of time Δt_{syn}. When moving to the computation of the next sector in a process, sites in the boundary region must be updated in advance. Fig. 5 presents the schematic diagram for data transfers of ghost cells, which we adopt in this paper. Here, we prepare the ghost cells from surrounding processes to send to the ghost region for the current sector. Similarly, ghost cells in local are also transferred to other processes. When the last sector has been simulated, the total execution time for the system increases Δt_{syn}. Since every time the simulation is restricted in one sector, more essential data are loaded to cache and higher cache hits are obtained in this way.

A third, based on the architecture of SW26010, the computation on sectors could be accelerated by utilizing CPEs in one CG. The simulation task distributed on each process is further divided. At this time, a part of CPEs play the role of messengers between MPE and other CPEs and the rest of CPEs execute computation task accurately. Here, a Transcription-Translation-Transmission algorithm is presented and details are discussed in Sec. 3.4.

2.4 Related Work and Challenges of Parallel OpenKMC

Different research teams have been devoting themselves to developing approaches of simulating the microstructural evolution of a dilute metal alloy material for RPVs with more accuracy. Électricité de France (EDF), owner of 58 power plants in France, has developed a simulation software named LAKIMOCA [19, 37]. Atomic behaviors of dilute metal alloys could be predicted accurately and it is suitable for long term kinetics. However, such AKMC software only has a serial version, so the simulations of large systems could be exceedingly time-consuming. Customized KMC models are supported in KMCLib [20]. For better load balancing, KMCLib rematches the sites with the processes at each step, and the overhead for it is expensive when running for large-scale simulation. SPPARKS [31] is a KMC software using the synchronous sublattice method [32] as the parallel algorithm. It includes several Monte Carlo models, such as vacancy diffusion model, H/He diffusion on an Erbium lattice model, Ising model, membrane model and Potts model for grain growth. Crystal-KMC [22] is a parallel KMC software, but it can only simulate vacancy diffusion in pure Fe lattice. Moreover, the correctness is not validated in their work, and the scalability for crystal-KMC is only 56% with 800 cores. Both SPPARKS and Crystal-KMC code [22] cannot deal with the vacancy diffusion in the dilute metal alloy systems such as vacancy assistant Cu precipitation in Fe lattice. Jiménez designed an OKMC software based on GPU to simulate evolution in irradiated metal materials [18]. Nonetheless, the AKMC is not supported and diffusions of point defects in dilute metal alloys cannot be simulated. Synchronous parallel kinetic Monte Carlo (spKMC) [29] and its extension [26] to discrete lattices were developed by E. Martínez*et al.*. Ising model is implemented in spKMC based on a synchronous time decomposition algorithm, and it scales to 256 cores for total 1.1 billion atoms with a parallel efficiency around 82%. N.J. van der Kaap and L.J.A. Koster developed a parallel kinetic Monte Carlo simulations [36] that runs on a General Purpose Computing on GPU (GPGPU) board. However, the application of it is limited for charge carrier transport simulations on GPGPU. Total 4,096 cores are employed in their work [36] and the efficiency of it changes from 85% to 0.4% dramatically at different concentrations. Even though the various KMC simulation softwares can help us understand the mechanisms, and the modern computing clusters provide rich computation resources, it is still challenging to develop a complete parallel KMC simulation software especially on Sunway processors with complex many-core architecture.

When parallelizing the KMC simulation, a process has to synchronize with surrounding processes for sharing data or finishing

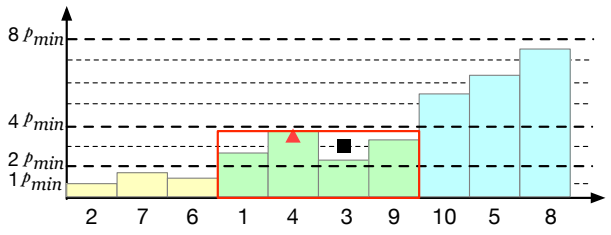

Figure 6: Group reaction strategy for event selection and update. The event that red triangle point falls into is selected, and the adjacent one with black square point is filtered out.

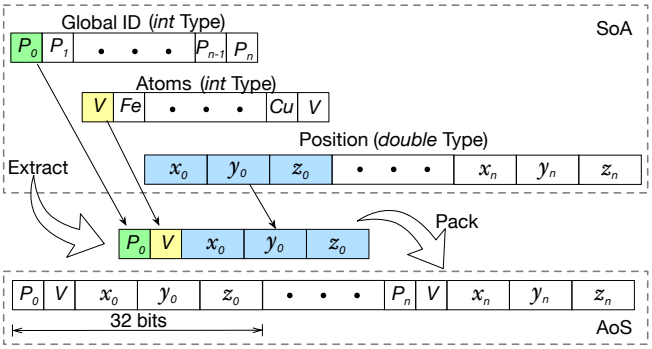

Figure 7: Packing data for DMA access to make accesses continuous and aligned by software cache strategy.

dependencies. On modern processors, especially many-core architectures, the cost for synchronization is exceedingly expensive and sometimes it dominates the time of the whole simulation. Furthermore, CPEs have no data cache, and the last buffer from each CPE to shared memory is the 64 KB SPM. This forces us to access the shared memory frequently for exchanging data if no other solution is proposed. At last, the Direct Memory Access (DMA) operation on SW26010 achieves its peak bandwidth only if the accessed address is in an alignment of 128 bytes [10, 23]. Thus the size of each cache line should be a multiplier of the atom data structure size, or we must do a transform between atom index and memory address, which has a non-negligible penalty on memory access time.

3 OPTIMIZATION

To achieve efficient KMC simulation in large-scale clusters, we have applied two categories of optimizations on OpenKMC. One includes a pair potential model proposed from the perspective of physics and the other is discussed based on parallel techniques.

In optimizations of the second category, group reaction strategy in Sec. 3.1 and cache optimization are achieved to decrease overheads of computation. Then we use a series of strategies to cut communication costs. At last, a Transcription-Translation-Transmission algorithm and vectorization are implemented to utilize the extra resources on CPEs. All these optimizations mentioned above are introduced in this section.

3.1 Group Reaction Strategy

In each iteration of the KMC simulation, an event needs to be picked up from a list of the events according to their relative probabilities. A common choice is to perform a linear scanning over all the events to select one, which scales as $O(N)$ for the total N events [31]. The list of the events can also be organized as a binary tree, which scales as $O(\log_2 N)$. However, the performance is still unsatisfactory and we turn to use the $O(1)$ scaling group reaction Strategy, which regroups propensities and is constant in selection [31, 33].

Fig. 6 illustrates group reaction strategy. Firstly, total N propensities from p_{min} to $p_{max}(8p_{min})$ are grouped to G groups. In this case, the upper bounds for each group are $2p_{min}$, $4p_{min}$ and $8p_{min}$ respectively. Then, a group is chosen randomly in selection step. Secondly, within the selected group, we consider a set of N_{local} reaction propensities with largest value $p_{localmax}$. We pick a uniform random integer i from 1 to N_{local} and a random value r from 0 to $p_{localmax}$ in this group. A reaction i is rejected if $p_i < r$, such as the black square point in Fig. 6. This step will loop until $p_i \geq r$ like the red triangle point, which means the event with the propensity p_4 is chosen. Thus the group reaction strategy scales as $O(1)$.

3.2 Cache Optimization Strategy

Since the simulation domain is partitioned across processors, each processor owns the computation cells within its sub-domain and a shell of surrounding ghost cells. The ghost cells on each processor will not execute events while they are involved in boundary sites computation. However, unlike contiguous storage for computation cells, the update of the ghost cells is promoted stochastically. These discrete and random memory accesses to different arrays lead fewer cache hits.

In computer science, average memory access time *(AMAT)* is a common metric to analyze memory system performance by employing three factors including hit time, miss rate, and miss penalty. Hit time is the time to hit in the cache. Miss rate (*MR*) is the frequency of cache misses, and miss penalty is the average cost of a cache miss in terms of time (*AMP*). Concretely it can be defined in Equation 6 [15].

$$AMAT = Hit\ time + Miss\ rate \cdot Miss\ penalty \tag{6}$$

To reduce *AMAT*, we can target each of these factors, and study how these can be decreased. On SW26010 processor, when cache line size is 256 bytes, it will reach the peak bandwidth even if the accessed address is distributed randomly [10]. Thus 256 bytes is the minimum size to achieve peak bandwidth for random memory access, and a larger size will increase *MR* and *AMP*. Though preprocessing techniques are not available in KMC simulation, we can rearrange the data structure and make full use of spatial locality in cache as much as possible.

Fig. 7 shows the reorganization of the atom data from structure of arrays (SoA) form to array of structures (AoS) form. We extract the essential attributes of a lattice site including ID, type and position, and pack them together into a new atom data structure of 32 bytes for ghost cells before communication. When we fetch cache lines from memory, memory access blocks are always aligned since cache lines are formed by 8 atom data structures (total 256 bytes). At the

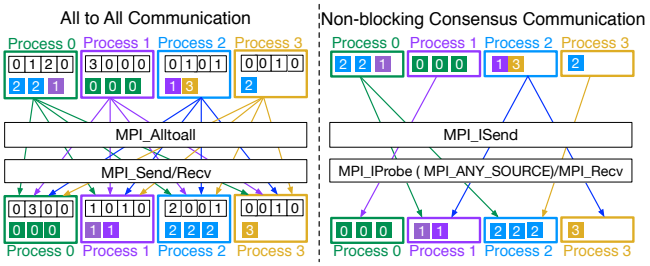

Figure 8: The comparison for MPI communication patterns used in convention and in our work. The left one is Personalized Exchange Algorithm (PEX) and the right one is Nonblocking Consensus Communication (NBX).

same time, we can also make full use of spatial locality of cache because adjacent ghost cells are generally accessed successively.

3.3 Communication Optimization Strategies

In this section, we perform communication optimization from three different aspects. First, ghost cells are computed to cut down the total amount of messages and eliminate communication redundancy. Then non-blocking consensus communication pattern is employed for decreasing synchronous time by point-to-point operations. At last, we present a self-adapting communication algorithm to reduce the frequency of communication adaptively.

3.3.1 Ghost Cells Computation. Sites in the boundary region must be updated in advance when a sector has been computed. Traditional methods for doing this, all data in surrounding sectors, are transferred to the local directly. However, lots of data are redundant since they are useless for the computation of the local sector. The key point of ghost cells computation is to eliminate redundant data transfers.

Ghost cells computation removes the data dependence by analyzing the effect ranges of cutoff distance, and then divides local data into more fine-grained chunks to transfer instead of transferring the whole data in local.

3.3.2 Nonblocking Consensus Communication. Ghost cells are required to transfer to other processes when a synchronous time Δt_{syn} is reached. As the left part of Fig. 8 shows, one common communication pattern is Personalized Exchange Algorithm (PEX). Each process writes the data sizes to send to each peer in a vector with P elements and the vector is redistributed with a personalized exchange *MPI_Alltoall* [14, 40]. Then all processes post their receive and respective send operations.

Though PEX communication is intuitive and easy to achieve, the *MPI_Alltoall* operation of it is quite expensive and the time needed to complete PEX on P processes is $\Theta(P)$ [24, 25]. Consequently, a scalable algorithm, Non-blocking Consensus Communication (NBX) [16], is adopted in this paper. The right part of Fig. 8 describes the NBX communication. This method utilizes non-blocking collective operations and they can be started and completed independently. The process uses *MPI_IProbe* to query the information beforehand, and launch *MPI_Recv* afterwards to receive the ghost cells data. Since non-blocking point-to-point operations are used, a mass of transmission and synchronous time are saved, and the time to perform NBX on P processes is $\Theta(\log(P))$.

3.3.3 Self-adapting Communication Algorithm . As discussed previously, the synchronous time Δt_{syn} of sub-domains is a crucial factor for communication. A smaller value of Δt_{syn} means the ghost cells are updated frequently, and accurate simulation results are obtained. However, frequent communication brings a poor parallel efficiency since not all communication is necessary. By increasing Δt_{syn}, the parallel efficiency is greatly improved, while the lower accuracy of simulation results is now much more pronounced.

Since the rate of growth of hop events could change during the simulation, it will be highly difficult to find a proper constant that satisfies both accuracy and efficiency requirements at long times. Therefore, we propose a self-adapting communication algorithm by utilizing the current state of the system including the propensities and hops number to perform communication adaptively.

First, we consider an idealized physical system consisting of a great number of identical events, each with propensity p. The total number of events is N, then the time for n events is computed by:

$$t = \frac{n}{N \times p} = \frac{n/N}{p} . \tag{7}$$

The fraction of the events in the sector is E, which equals to $(n/N)\times 100\%$. A good estimate for the number of events performed is given by this parameter. We exploit this relationship by defining p_s as the total propensity of events in a sector, divided by the number of active events in it. Then the initial synchronous time Δt_{syn} for sub-domains is obtained by

$$\Delta t_{syn} = \min(E/p_{max}, threshold - T) , \tag{8}$$

where p_{max} is the maximum p_s across all processors, threshold is a preset total simulation time, and T is current simulation time.

In SPPARKS [31], the propensity is regarded as dynamically changing so that a Δt_{syn} dependent on it is enough to adjust the simulation schedule. However, the propensity in AKMC simulation basically remains unchanged and a more accurate self-adapting algorithm is required. Thus, we further trace the number of events that the atom hops to another process during the synchronous interval. The self-adapting communication indicates that communication occurs only when jumps happen, or the simulation will continue without communication even though the current Δt_{syn} is satisfied.

The workflow of the self-adapting communication algorithm is shown in Algorithm 1. Taking the dual factors of propensity and jumps number, the adaptive algorithm could obtain a good balance between accuracy and efficiency.

3.4 Transcription-Translation-Transmission Algorithm

In this subsection, we propose a Transcription-Translation-Transmission (3T) algorithm to utilize the CPEs on SW26010 for further improving the performance of OpenKMC, which is inspired by the process of gene expression in biology and on the basis of the architecture of SW26010. Fig. 9 presents the whole process of our algorithm,

Algorithm 1 Self-adapting Communication Algorithm

Require:
E: The fraction of the events in the sector.
$threshold$: A preset total simulation time.
T: The current simulation time.
$p_{threshold}$: The propensity threshold for jumping events.
t_{stop} : A default synchronization interval.

$p_{local\,\max} \leftarrow 0$
for $i = 0$ to $sectors$ **do**
  $n_{active} \leftarrow$ Number of active sites in current sector.
  $p_s \leftarrow$ Total propensity of events in current sector.
  $p_s \leftarrow p_s / n_{active}$
  $p_{local\,\max} = \max(p_s, p_{local\,\max})$
end for
$p_{max} \leftarrow$ maximum $p_{local\,\max}$ across all processors.
if $p_{max} > 0$ **then**
  $\Delta t_{syn} \leftarrow E / p_{max}$
else
  $\Delta t_{syn} \leftarrow threshold - T$
end if
$\Delta t_{syn} = \min(\Delta t_{syn}, threshold - T)$
while $T \leq threshold$ **do**
  for $j = 0$ to $sectors$ **do**
    while $t < \Delta t_{syn} \&\& jumps_b <= 0$ **do**
      $p_{local\,\max} \leftarrow$ The total propensity of events.
      Execute jumping events.
      $jumps_b \leftarrow$ The number of boundary events.
      Calculate dt
      $t+ = dt$
    end while
    Communication.
  end for
  $T+ = \Delta t_{syn}$.
  $p_{max} \leftarrow$ maximum $p_{local\,\max}$ across all processors.
  if $p_{max} > 0$ **then**
    $\Delta t_{syn} \leftarrow t_{stop} / p_{max}$
  else
    $\Delta t_{syn} \leftarrow threshold - T$
  end if
  $\Delta t_{syn} = \min(\Delta t_{syn}, threshold - T)$
end while

where the green MPE core is *Template MPE*, the 8 purple cores are *Messenger CPEs* and the other 56 yellow cores are *Translation CPEs*.

The 3T algorithm describes the thread-level parallelism method and works in this way: First, the data are copied from MPE to the messenger CPEs and translation CPEs; and then, the data are computed on translation CPEs and copied to messenger CPEs via fast register communication; finally, the results are transferred back from the messenger CPEs to the MPE.

Transcription. Transcription is the first step of 3T algorithm. First, the data of atoms are diapatched from *Template MPE* to *Translation CPEs*. At the same time, lots of same parameters are also required

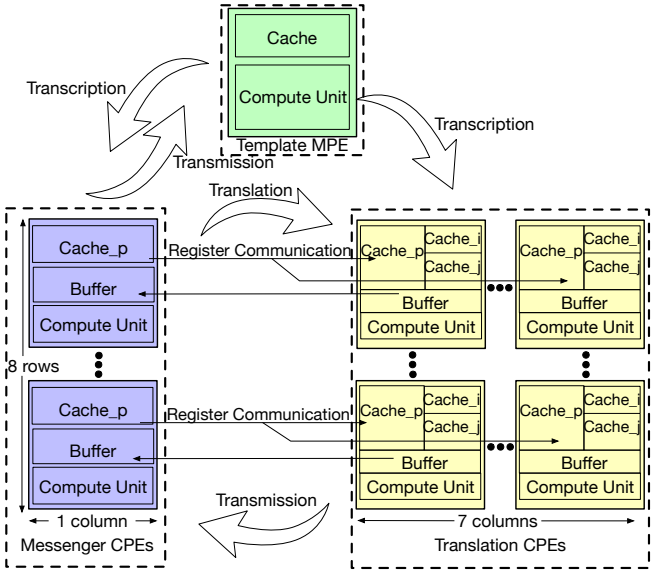

Figure 9: The Transcription-Translation-Transmission algorithm for accelerating computation by CPEs. The green, purple and yellow cores are Template MPE, Messenger CPEs and Translation CPEs respectively. Each Messenger CPE will be paired with 7 Translation CPEs in the same row and they will share data using fast row register communication.

for each *Translation CPE*. Because of the low communication efficiency between MPE and CPEs, the computation on *Translation CPEs* has to wait until all parameters have been transferred. Here we utilize *Messenger CPEs* to accomplish this task. When the data are diapatched to *Translation CPEs*, the *Messenger CPEs* start to fetch essential parameters simultaneously and the parameters are repacked to corresponding *Translation CPEs* by fast register communication directly. Thus the computation could start immediately when atoms data have been transferred to *Translation CPEs*.

A distance calculation between lattice site i and j is illustrated as an example here. The raw data of atoms i and j, including the type, ID and position, are repacked and transferred from *Template MPE* to *Translation CPEs* respectively. At the same time, *Messenger CPEs* obtain a series of parameters including cutoff radius and boundary conditions from *Template MPE*. Then the parameters are packed and transferred to the corresponding *Translation CPEs* in the same row by fast register communication. All data in cache of *Translation CPEs* wait for being processed by translation step.

Translation. The translation step in 3T algorithm is to process raw data into various simulation results on *Translation CPEs*. Every 7 *Translation CPEs* in the same row will be paired with one *Messenger CPE*. In distance calculation example, a mass of computation is performed to judge the neighbor relationship between two atoms. The results generated by computation are stored in **Buffer** at once for further judgment. If they are exactly neighbors to each other, the results will be sent to the corresponding *Messenger CPEs* via fast register communication instantly. Thus the time for computation on *Translation CPEs* and time for writing back to MPE are

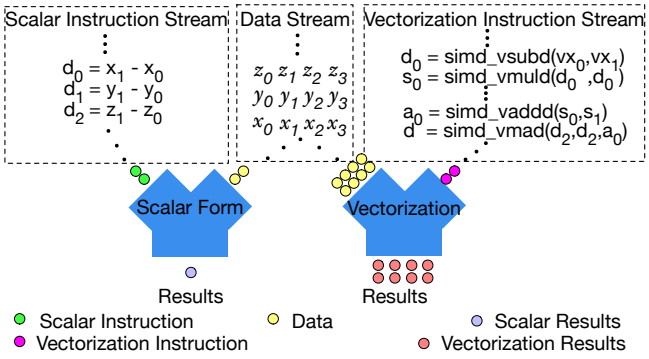

Figure 10: Inner j–sites vectorization. Four operations are performed at once by employing SIMD instructions.

overlapped and a large deal of communication time between MPE and CPEs is avoided.

Transmission. When results are sent to corresponding *Messenger CPEs* utilizing fast register communication, the *Messenger CPEs* will repack and transfer them to *Template MPE*. *Translation CPEs* will start the next round computation simultaneously and these steps will loop till all tasks dispatched by *Template MPE* are completed.

In fact, the *Template MPE*, *Messenger CPEs* and *Translation CPEs* will work together and run separately in 3T algorithm. Through decoupling the computation process of *Template MPE* and assigning different roles on CPEs precisely, all CPEs work with its own designated task. Thanks to the Transcription-Translation-Transmission algorithm, we accelerate KMC simulation efficiently by very fast register communication on CPEs.

3.5 Vectorization

To exploit the available 256-bit SIMD vector registers, we also implement vectorization in our work.

For distance calculation example in 3T algorithm, the pairwise calculation needs to compute a large number of j-sites for each i-site since massive sites are in the range of cutoff radius for every i-site. Therefore, we prefer to use the *sunway's SIMD extension* on CPEs to perform 8 computation of j-sites at once for the same i-site.

In practice, some tough cases that prevent efficient vectorization exist during the implementation on CPEs. One of the typical cases is that memory accesses are irregular when CPEs fetch data from MPE. The other case is that the overhead for writing back to MPE is also extremely expensive.

For the first case, we have proposed a cache optimization strategy on the data structure in MPE. The data are reorganized to an AoS form which occupies 32 bits for each atom and adjacent data in the data structure are accessed with high probabilities. Then only selection and repack operations on data are performed before they are transferred to CPEs. For the second case, the problem is solved by *Messenger CPEs*. The results generated by *Translation CPEs* are reorganized on *Messenger CPEs* simultaneously, which is profited from the fast register communication on CPEs, and then they are transferred to the cache in *Template MPE*.

Fig. 10 exhibits the comparison for scalar and vector operations in part of distance calculation. By employing SIMD operations, we exploit data-level parallelism and an inter-j-sites vectorization method is implemented, which leads a more efficient computation on CPEs.

4 EVALUATION

In this section, we describe our experimental evaluation elaborately. The correctness, single node performance and scalability are all evaluated, and the visualization for the Cu rich precipitates is also presented as a supplementary to experiments.

4.1 Correctness Validation

In order to validate the accuracy of OpenKMC, a series of thermal ageing simulations have been performed between 663 and 773 K for a binary Fe-1.34 at.%Cu alloy compared to the experiments of Lê *et al.* [21] as well as the simulation results of Vincent *et al.* [37]. We trace the evolution of simulation for 10^5 seconds (rescaled time), and a vacancy concentration accounting for 8×10^{-4} at.% is introduced within this space. The simulation time is rescaled in order to obtain a corresponding physical time scale by employing all simulation parameters according to Vincent *et al.*'s work [39]. For each simulation, a bcc lattice of 40 unit cells is employed in each of the three space dimensions with periodic boundary conditions. The precipitation advancement factor has been calculated as Equation 9 [21, 37], where $C_X(t)$ is the Cu concentration in the solid solution at time t, which tends towards the solubility limit $C_X(\infty)$. The precipitation kinetics predicted by our OpenKMC are performed on a single node of TaihuLight (4 MPEs with 256 CPEs) and a process of Intel Xeon E5-2670 both for EAM and pair potentials, and the results are compared to the real experiments obtained by electrical resistivity measurements in Fig. 11.

$$\zeta(t) = (C_X(0) - C_X(t))/(C_X(0) - C_X(\infty)) \tag{9}$$

It is clear that the Cu precipitates progressively during the thermal aging simulation and the precipitation kinetics predictions of our OpenKMC reproduce globally well the Vincent *et al.*'s work [39] on both architectures. As for the comparison with the experimental results [21], the simulation curves can, at least in a qualitative way, well describe the evolution of the system with time, although some quantitative deviation still exists, which comes from the simplified model (e.g. only one type of defects, i.e. the point defects, are taken into account) and the lack of many-body interactions and long-range interactions between atoms.

The accuracy is also evaluated when communication optimizations have been performed. We employ 16 processes both on Intel Xeon E5-2670 and SW26010. As Fig. 11 shows, the OpenKMC could obtain a sufficiently accurate simulation while the total runtime consumes only half under this case on two architectures. Thus a high simulation accuracy is guaranteed by OpenKMC with communication optimizations.

4.2 Single Node Evaluation

Fig. 12 demonstrates the performance improvements for different kinds of optimization methods with high and low concentration respectively on one SW26010 node which contains 260 cores (4 MPEs with 256 CPEs). A higher Cu concentration with 12.0 at.%

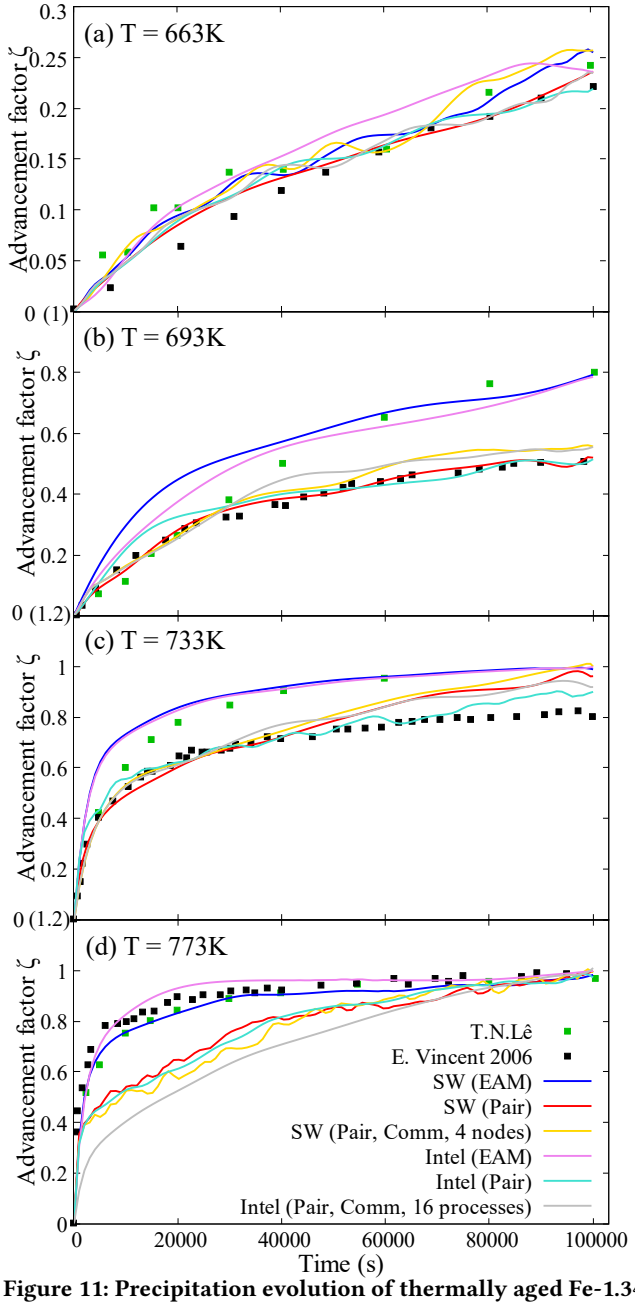

Figure 11: Precipitation evolution of thermally aged Fe-1.34 at.% Cu alloys. The green squares correspond to the experimental results of Lê *et al.* [21], the black squares to the Kinetic Monte Carlo results by Vincent *et al.* [37] and other curves to the OpenKMC results with EAM potentials or pair potentials on different platforms. The word 'Comm' in parentheses means the OpenKMC is evaluated with communication optimization.

of atoms and a vacancy concentration accounting for 12.8 at.% are used in high concentration evaluation, and other configurations are the same as the study case in Vincent *et al.*'s work [37].

As shown in Fig. 12(a) and Fig. 12(c) , initially, the energy computation kernel's performance is heavily dominated by EAM potentials, and we have a distinct speedup for this kernel after utilizing the optimized pair potential model from the physical perspective.

Fig. 12(b) and Fig. 12(d) give a comparison of different versions optimized by a series of methods from the algorithm perspective using pair potential. First, the traversal and updating time is decreased by group reaction strategy. Then the idea of packing atomic data also provides a slight speedup since it reduces many unaligned memory accesses. For the Self-adapting Communication algorithm, we can see that the simulation becomes much faster than the previous version and half of the communication time is reduced.

Next, we try to accelerate the computation process on CPEs by 3T algorithm and vectorization. In the thread-parallelism, the calculation of the vacancy events are loaded into CPEs. When high concentration is employed, enough computation tasks are guaranteed on each CPE and Fig. 12(b) gains an obvious speedup at this step. However, only one vacancy exists in simulation space if a low vacancy concentration 8×10^{-4} at.% is introduced. Thus the potential of many-core resources is not fully tapped and the employment of CPEs brings in extra overhead in Fig. 12(d) .

At last, the final design is 7.76 times faster than the first pair potential version in Fig. 12(b) and gain a more than 33.88 fold overall speedup compared to the original EAM version in Fig. 12(a) with a higher vacancy concentration.

To investigate the influence of vacancy concentration on performance, a detailed comparison is schematically depicted in Fig.13, where the many-core optimization adopts 3T algorithm and vectorization on CPEs compared to the general optimization. As the vacancy concentration is set to $8{\times}10^{-4}$ at.% (at the Cu concentration of 1.34 at.% and the temperature of 573 K), the running time in the simulation increases, rather than decreases, when the many-core optimization is applied. It is because when the number of the vacancies is too low, the amount of the computational tasks allocated to each core is so small that the time for many-core initiation and data communication occupies a large proportion of the simulation time, and thereby, makes the performance number significantly deceases (from 3.82 at the concentration of 12.8 at.% to only 0.31 at that of 8×10^{-4} at.%). However, when the vacancy concentration increases to 32.0 at.%, the performance number becomes even better (5.29) than that at 12.8 at.%.

It may be worth noting that the performance number depends on the number of vacancies per CPE, instead of the concentration of the vacancies. It is because the parallelization is over the vacancies in the simulated system. Therefore, when the total number of atoms in the system is sufficiently large, the performance number of the many-core optimization can still be satisfactory even at a very low concentration of vacancies.

4.3 Scalability

In this subsection, we evaluate the scalability of our OpenKMC on Sunway TaihuLight. To achieve a balance between simulation size for strong scalability and the limited memory on SW26010 (total 8 GB per CG), we take the performance of 5,000 CGs (5,000 MPEs with 320,000 CPEs) as a baseline, which simulates about 11 million (1.1×10^7) atoms per process. The memory needed per simulated

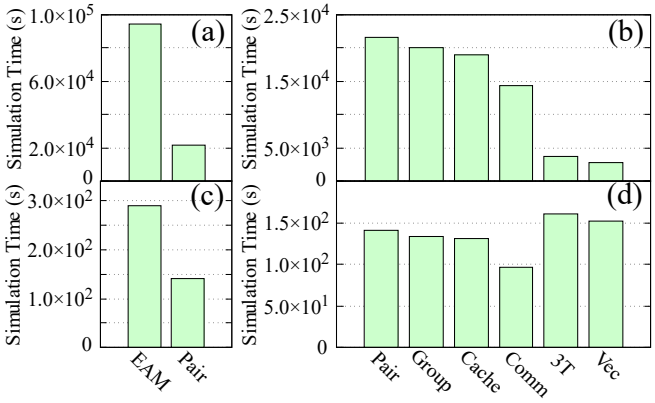

Figure 12: The performance comparison among different optimization steps cumulated on a SW26010 processor with a high concentration (12.8% for vacancies and 12.0% for Cu in (a) and (b)) and a low concentration (8×10^{-4}% for vacancies and 1.34% for Cu in (c) and (d)) respectively. (a) and (c) shows the comparison on physical optimization. (b) and (d) displays the breakthrough on algorithmic optimization.

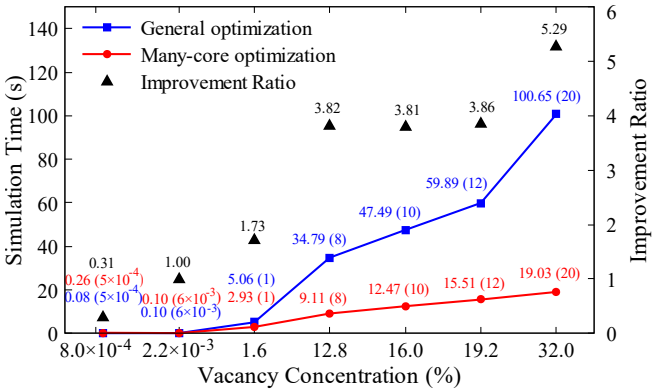

Figure 13: Performance comparison at different vacancy concentrations. The red and blue lines represent the running time in the OpenKMC simulations with and without the many-core optimization (3T algorithm and vectorization), respectively. The numbers in the parentheses denote the number of the vacancies allocated to each CPE. The improvement ratios are labelled with black upper triangles.

atom is approximately 0.72KB in this configuration. Accordingly, the weak scalability also initializes the simulation from one CG (1 MPE with 64 CPEs) with 11 million atoms as a baseline.

For the strong scalability, a case of 54 billion (5.4×10^{10}) atoms is presented with the thermal aging at 300°C (573K). Since the Cu clusters are formed by vacancies hops, we introduce 1.5×10^{-4} at.% of Cu atoms and 8×10^{-5} at.% of vacancies within this configuration. The total number of Cu atoms and vacancies are 43,200 and 81,000 respectively, which ensures at least one impurity element for each process on average when 5.2 million cores are used. The simulation time is rescaled in order to obtain a physical time scale and in this case, the simulation time of 2.26×10^{-2} is rescaled to 3 months.

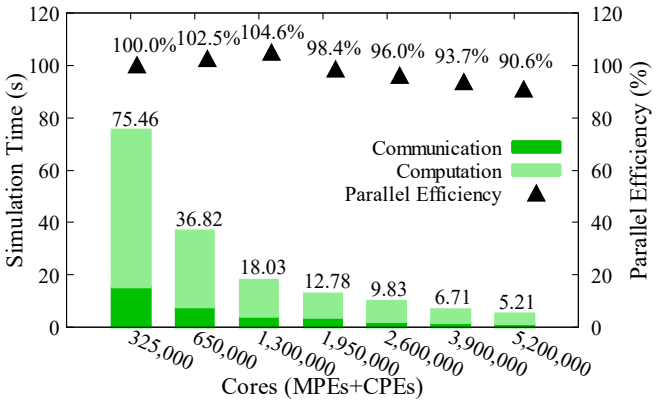

Figure 14: Strong scalability for 54 billion (5.4×10^{10}) atoms with x and y axis scaled. Green bars correspond to simulation time of the primary y axis on the left and parallel efficiency triangles correspond to the secondary y axis on the right. Simulation time and parallel efficiency values are annotated on the top of bars and triangles respectively.

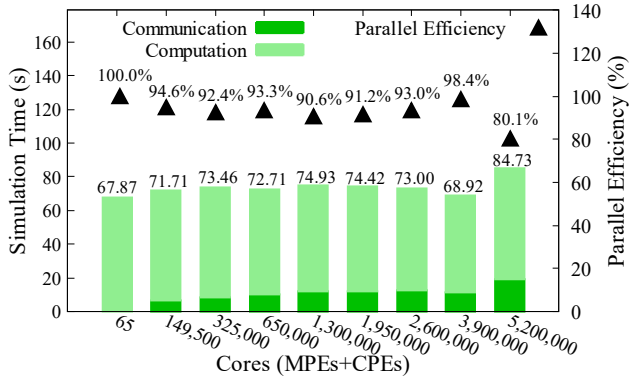

Figure 15: Weak scalability for 11 million (1.1×10^7) atoms per process, with x and y axis scaled. Green bars correspond to simulation time of the primary y axis on the left and parallel efficiency triangles correspond to the secondary y axis on the right. Simulation time and parallel efficiency values are annotated on the top of bars and ttriangles respectively.

Fig. 14 shows the performance to this case for strong scalability when the number of cores varies from 325,000 to 5,200,000. We can see that the computation time dominates a larger proportion of the whole simulation and both of the computation and communication time decrease with more cores. As Fig. 14 displays, even when the number of cores increases to 5.2 million, the parallel efficiency is still around 90%, which indicates good strong scalability for OpenKMC. The actual speedup is almost equivalent to the ideal one under this larger simulation size and especially when the cores are less than 1.3 million, the dual remarkable benefits brought by group reaction strategy and communication optimizations lead a superlinear speedup.

Fig. 15 illustrates the weak scaling performance with a baseline of one CG (1 MPE with 64 CPEs). Since each CG is a single process, the fast DMA operations between MPE and CPEs within a CG is not

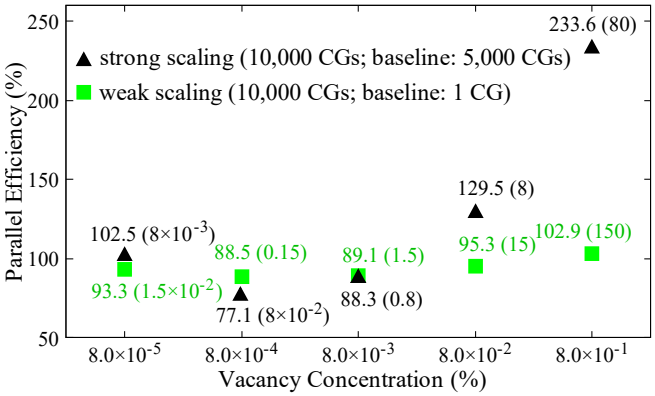

Figure 16: Parallel efficiencies comparison under different vacancy concentrations. The numbers in the parentheses denote the number of the vacancies allocated to each CPE.

considered as communication time. We initialize the simulation box with an average of 11 million (1.1×10^7) atoms per process. It can be seen that it keeps a high parallel efficiency with the growth of cores and ultimately reaches 5.2 billion cores. At last, it reaches about 840 billion (8.4×10^{11}) atoms and achieves a parallel efficiency of 80.1%. It's worth noting that nearly all parallel efficiencies approach 100% when the cores are less than 3.9 billion. This illustrates that OpenKMC has the ability to simulate a large-scale physical system with large-scale parallelism. As can be seen in Fig. 15, the computation time remains almost constant on different number of cores. However, the communication time is a little higher on 5.2 billion cores, which is mainly caused by communication contention.

The phenomenon that parallel efficiencies approach or exceed 100% in scaling experiments can be analyzed by

$$T_{total} = T_{event} + T_{comp} + T_{comm}\ , \tag{10}$$

where T_{total} is total execution time, T_{event} is time for selection and update of events, T_{comp} refers to the computation time and T_{comm} is communication time. First, group reaction strategy is a $O(1)$ algorithm. The number of groups is decreased with the number of sites per processor shrinks as processors are added. Thus, the overheads for event selection and update are extensively reduced, which contributes a lot to the performance improvement. Then the computation part could also be run in parallel over all available processes. At last, according to Equation **??** and Equation **??**, N_{comm} decreases obviously while v almost unchanged with highly increasing cores. Since the sub-domain shrinks when more processes are employed, the proportion of hop events on boundaries increases and Δt_{syn} will maintain a larger value for a longer time. Therefore, T_{comm} is reduced as processes increase. As for weak scalability, T_{event} and T_{comp} remain basically unchanged. With the increase of cores and simulation size, the performance drops gradually while the slight difference in Δt_{syn} brings a tremble occasionally.

The scaling studies on both strong and weak scalability for sufficient large systems (30,000 CGs are used) are also performed, which adopts the same vacancy concentration (8×10^{-4} at.%), Cu concentration (1.34 at.%) and temperature (573 K) with Vincent *et al.*'s work [37]. Results reveal that as the number of cores increases, the parallel efficiency does not change much.

In addition, with the Cu concentration (1.34 at.%) and the temperature (573 K) fixed, we have also performed the scaling studies at various vacancy concentrations, with the results shown in Fig.16. It can be seen that within a wide concentration range (from 8×10^{-5} at.% to 8×10^{-1} at.%), the parallel efficiencies are always satisfactory. Especially, at a vacancy concentration of 8×10^{-1} at.%, the efficiency of the strong scaling is as high as 233.6%, which is due to a large number of vacancies (80 per CPE) in this case.

4.4 Visualization

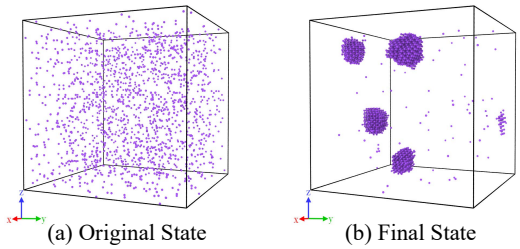

Figure 17: Visualization of the Cu precipitation before and after the thermal ageing of the Fe-Cu alloy.

To further prove the validity of OpenKMC for long-time evolution, simulations of thermal aging of a Fe-1.34Cu(at.%) are performed at 663K for a rescaled time of 100 years. Also, we use a rigid bcc lattice of 40 unit cells in each of the three dimensions, which is the same configuration as Sec. 4.2 and Vincent *et al.*'s work [37]. The original and final states are displayed in Fig. 17.

At the beginning of the simulation, it is clear that all Cu atoms are distributed randomly in the α-Fe ferrite matrix and no apparent Cu precipitates are formed. However, the microstructural aspect after 100 years of thermal aging is radically different than the one obtained at the beginning. Obviously, precipitates have occurred with the biggest Cu rich core composed of 542 atoms.

The formation of Cu precipitates in different size is coherent with the thermodynamic theory, as was explained by Vincent [37, 38]. Thus, the result obtained with long-time evolution is another powerful evidence for the validity of OpenKMC.

5 CONCLUSION

In this work, we propose a new parallel KMC design as OpenKMC to simulate with hundred-billion-atom simulation on Sunway TaihuLight. Both the conventional EAM potential model and our optimized pair potential model are supported in OpenKMC. To develop a more efficient implementation on Sunway TaihuLight, we then employ a group reaction strategy to accelerate the selection and update processes for events and redesign the data structure to reduce the memory access cost. Moreover, a series of communication optimization strategies and a novel 3T algorithm on CPEs are utilized to further improve the performance markedly. Vectorization is also implemented on CPEs that brings extra performance benefits. Experiments show that OpenKMC provides high accuracy of KMC simulation. Sustained performance of 90.6% parallel efficiency with 54 billion atoms for strong scaling simulation and 80.1% parallel efficiency with 0.84 trillion atoms for weak scaling simulation are obtained when using 5.2 million cores on Sunway TaihuLight.

It is worth noting that many optimization strategies in our paper such as group reaction strategy and 3T algorithm are quite generic though they are optimized specially for SW26010. We believe that our proposed methods could provide an inspiring experience to algorithm design for other architectures, such as the heterogeneous clusters and next-generation Sunway supercomputer.

ACKNOWLEDGMENTS

The authors would like to thank all the reviewers for their insightful and valuable comments and suggestions. This work is supported by the the National Key Research & Development Program of China (2017YFB0202302), the Strategic Priority Research Program of Chinese Academy of Sciences with Grant No. XDC01000000, National Key Research & Development Program of China (2016YFB0200803), National Natural Science Foundation of China under Grant No. 61432018, National Natural Science Foundation of China under Grant No. 61502450, State Key Laboratory of Computer Architecture Foundation under Grant No. CARCH3504. The corresponding author of this paper is Honghui Shang (shanghui.ustc@gmail.com).

REFERENCES

[1] C. S. Becquart and C. Domain. 2010. Introducing chemistry in atomistic kinetic Monte Carlo simulations of Fe alloys under irradiation. *Phys. status solidi* 247, 1 (jan 2010), 9–22. https://doi.org/10.1002/pssb.200945251

[2] Charlotte S Becquart, Christophe Domain, Utpal Sarkar, Andrée Debacker, and Marc Hou. 2010. Microstructural evolution of irradiated tungsten: Ab initio parameterisation of an OKMC model. *Journal of nuclear materials* (2010).

[3] JA Blackman and PA Mulheran. 2001. Growth of clusters on surfaces: Monte Carlo simulations and scaling properties. *Computer physics communications* 137, 1 (2001), 195–205.

[4] N Castin, L Messina, C Domain, RC Pasianot, and Pär Olsson. 2017. Improved atomistic Monte Carlo models based on ab-initio-trained neural networks: Application to FeCu and FeCr alloys. *Physical Review B* 95, 21 (2017), 214117.

[5] C Domain and CS Becquart. 2001. Ab initio calculations of defects in Fe and dilute Fe-Cu alloys. *Physical Review B* 65, 2 (2001), 024103.

[6] C Domain, CS Becquart, and J Foct. 2004. Ab initio study of foreign interstitial atom (C, N) interactions with intrinsic point defects in α-Fe. *Physical Review B* 69, 14 (2004), 144112.

[7] C Domain, CS Becquart, and L Malerba. 2004. Simulation of radiation damage in Fe alloys: an object kinetic Monte Carlo approach. *Journal of Nuclear Materials* 335, 1 (2004), 121–145.

[8] C Domain, CS Becquart, and JC Van Duysen. 1998. Kinetic Monte Carlo simulations of FeCu alloys. *MRS Online Proceedings Library Archive* 538 (1998).

[9] C Domain, C S Becquart, and L Malerba. 2004. Simulation of radiation damage in Fe alloys: an object kinetic Monte Carlo approach. *J. Nucl. Mater.* 335, 1 (2004), 121–145. https://doi.org/10.1016/j.jnucmat.2004.07.037

[10] Xiaohui Duan, Dexun Chen, Xiangxu Meng, Guangwen Yang, Ping Gao, Tingjian Zhang, Meng Zhang, Weiguo Liu, Wusheng Zhang, Wei Xue, et al. 2018. Redesigning LAMMPS for peta-scale and hundred-billion-atom simulation on Sunway TaihuLight. In *Redesigning LAMMPS for Peta-Scale and Hundred-Billion-Atom Simulation on Sunway TaihuLight*. IEEE, 0.

[11] F Fang, XL Shu, HQ Deng, WY Hu, and M Zhu. 2003. Modified analytic EAM potentials for the binary immiscible alloy systems. *Materials Science and Engineering: A* 355, 1-2 (2003), 357–367.

[12] Jiarui Fang, Haohuan Fu, Wenlai Zhao, Bingwei Chen, Weijie Zheng, and Guangwen Yang. 2017. swDNN: A library for accelerating deep learning applications on sunway taihulight. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 615–624.

[13] 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.

[14] William D Gropp, William Gropp, Ewing Lusk, Anthony Skjellum, and Argonne Distinguished Fellow Emeritus Ewing Lusk. 1999. *Using MPI: portable parallel programming with the message-passing interface*. Vol. 1. MIT press.

[15] John L Hennessy and David A Patterson. 2011. *Computer architecture: a quantitative approach*. Elsevier.

[16] Torsten Hoefler, Christian Siebert, and Andrew Lumsdaine. 2010. Scalable communication protocols for dynamic sparse data exchange. *ACM Sigplan Notices* 45, 5 (2010), 159–168.

[17] Alex Slepoy ISteve Plimpton, Aidan Thompson. 2009. SPPARKS Kinetic Monte Carlo Simulator. https://spparks.sandia.gov/. [Online; accessed 10-July-2019].

[18] 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.

[19] Johnny Lazo. 2013. NUMERICAL SIMULATIONS OF MICROSTRUCTURAL EVOLUTION OF IRON ALLOYS UNDER IRRADIATION.

[20] 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.

[21] T.N Lê, A Barbu, D Liu, and F Maury. 1992. Precipitation kinetics of dilute FeCu and FeCuMn alloys subjected to electron irradiation. *Scripta Metallurgica et Materialia* 26, 5 (1992), 771 – 776. https://doi.org/10.1016/0956-716X(92)90436-I

[22] Jianjiang Li, Peng Wei, Shaofeng Yang, Jie Wu, Peng Liu, and Xinfu He. 2018. Crystal-KMC: parallel software for lattice dynamics monte carlo simulation of metal materials. *Tsinghua Science and Technology* 23, 4 (2018), 501–510.

[23] Kun Li, Shigang Li, Shan Huang, Yifeng Chen, and Yunquan Zhang. 2019. FastNBL: fast neighbor lists establishment for molecular dynamics simulation based on bitwise operations. *The Journal of Supercomputing* (2019), 1–20.

[24] Shigang Li, Baodong Wu, Yunquan Zhang, Xianmeng Wang, Jianjiang Li, Changjun Hu, Jue Wang, Yangde Feng, and Ningming Nie. 2018. Massively scaling the metal microscopic damage simulation on sunway taihulight supercomputer. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 47.

[25] Shigang Li, Yunquan Zhang, and Torsten Hoefler. 2018. Cache-oblivious MPI all-to-all communications based on Morton order. *IEEE Transactions on Parallel and Distributed Systems* 29, 3 (2018), 542–555.

[26] Enrique Martínez, Paul R Monasterio, and Jaime Marian. 2011. Billion-atom synchronous parallel kinetic Monte Carlo simulations of critical 3D Ising systems. *J. Comput. Phys.* 230, 4 (2011), 1359–1369.

[27] Luca Messina, Nicolas Castin, Christophe Domain, and Pär Olsson. 2017. Introducing ab initio based neural networks for transition-rate prediction in kinetic Monte Carlo simulations. *Physical Review B* 95, 6 (2017), 064112.

[28] Jana B Milford, Frank Kreith, and Charles F Kutscher. 2018. Nuclear Energy. In *Principles of Sustainable Energy Systems, Third Edition*. CRC Press, 247–268.

[29] Alfredo Moura and António Esteves. 2015. Synchronous parallel kinetic Monte Carlo simulation of AL3SC precipitation. (2015).

[30] Pär Olsson, TPC Klaver, and C Domain. 2010. Ab initio study of solute transition-metal interactions with point defects in bcc Fe. *Physical Review B* 81, 5 (2010), 054102.

[31] 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).

[32] Yunsic Shim and Jacques G. Amar. 2005. Semirigorous synchronous sublattice algorithm for parallel kinetic Monte Carlo simulations of thin film growth. *Phys. Rev. B* 71 (Mar 2005), 125432. Issue 12. https://doi.org/10.1103/PhysRevB.71.125432

[33] Alexander Slepoy, Aidan P Thompson, and Steven J Plimpton. 2008. A constant-time kinetic Monte Carlo algorithm for simulation of large biochemical reaction networks. *The journal of chemical physics* 128, 20 (2008), 05B618.

[34] F Soisson, A Barbu, and G Martin. 1996. Monte Carlo simulations of copper precipitation in dilute iron-copper alloys during thermal ageing and under electron irradiation. *Acta Materialia* 44, 9 (1996), 3789–3800.

[35] F. Soisson, C.S. Becquart, N. Castin, C. Domain, L. Malerba, and E. Vincent. 2010. Atomistic Kinetic Monte Carlo studies of microchemical evolutions driven by diffusion processes under irradiation. *J. Nucl. Mater.* 406, 1 (nov 2010), 55–67. https://doi.org/10.1016/j.jnucmat.2010.05.018

[36] NJ van der Kaap and L Jan Anton Koster. 2016. Massively parallel kinetic Monte Carlo simulations of charge carrier transport in organic semiconductors. *J. Comput. Phys.* 307 (2016), 321–332.

[37] E Vincent, CS Becquart, and C Domain. 2006. Solute interaction with point defects in α Fe during thermal ageing: A combined ab initio and atomic kinetic Monte Carlo approach. *Journal of nuclear materials* 351, 1-3 (2006), 88–99.

[38] E Vincent, CS Becquart, and C Domain. 2008. Microstructural evolution under high flux irradiation of dilute Fe–CuNiMnSi alloys studied by an atomic kinetic Monte Carlo model accounting for both vacancies and self interstitials. *Journal of Nuclear Materials* 382, 2-3 (2008), 154–159.

[39] E. Vincent, C. S. Becquart, C. Pareige, P. Pareige, and C. Domain. 2008. Precipitation of the FeCu system: A critical review of atomic kinetic Monte Carlo simulations. *J. Nucl. Mater.* 373, 1-3 (2008), 387–401. https://doi.org/10.1016/j.jnucmat.2007.06.016

[40] 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.

[41] W M Young and E W Elcock. 1966. Monte Carlo studies of vacancy migration in binary ordered alloys: I. *Proceedings of the Physical Society* 89, 3 (nov 1966), 735–746. https://doi.org/10.1088/0370-1328/89/3/329

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

1.Abstract

Our artifact provides the code of OpenKMC for Sunway architecture (the building block of Sunway TaihuLight Supercomputer), the benchmarks, along with the scripts to run these benchmarks to evaluate our algorithm's performance. As the code can be compiled and run only on Sunway processor, we also briefly introduce how to log in and use Sunway TaihuLight Supercomputer.

1.1 Artifact check-list

-Program: OpenKMC for Sunway architecture

-Compilation: Using the provided scripts via sw5cc compiler (the customized compiler for Sunway architecture).

-Hardware: We provide a user account for evaluating our OpenKMC on Sunway TaihuLight Supercomputer.

-Dataset: All of the experimental data used in our paper can be found and run in the Home directory of the user on Sunway TaihuLight.

-Run-time environment: Linux.

-Experiment Workflow: Login the provided user account; Run build scripts; Run test scripts.

-Publicly available?: Yes

1.2 Hardware dependencies

The OpenKMC proposed in this work only uses SW26010 processor on Sunway TaihuLight.

1.3 Software dependencies

The OpenKMC requires sw5cc compiler and MPI library, which have already been installed on Sunway TaihuLight Supercomputer. Moreover, a visualization tool Ovito is also needed.

1.4 Login to Sunway TaihuLight

-Land the homepage of the National Supercomputing Center in Wuxi: http://www.nsccwx.cn/wxcyw/

-Select one VPN service: 'Telecom', 'Unicom' or 'China Mobile' on the top of the website. Please choose the best one for a better connection.

-Login to Sunway TaihuLight Supercomputer:ssh 41.0.0.188.

1.5 Experimental configuration

For the correctness in Fig.11, we compare our optimized OpenKMC on a single node of TaihuLight (4 MPEs with 256 CPEs) and Intel Xeon E5-2670 v3 both for EAM and pair potentials to the reference of Vincent's work respectively. We trace the evolution of solute atoms for 100000 seconds simulation (rescaled time) with the thermal ageing at 663K,693K,733K,773K of FeCu alloys. For each simulation, a bcc lattice of 40 unit cells is employed in each of the three space dimensions with periodic boundary conditions. Then a configuration of 1.34 at.% of Cu atoms and a vacancy concentration accounting for 0.0008 at.% are introduced within this space.

For single node performance in Fig.12, both a high concentration (12.8% for vacancy and 12% for Cu) and a low concentration (0.0008% for vacancy and 1.34% for Cu) are used in the evaluation. Other configurations are the same with the study case in correctness validation.

For the strong scalability in Fig.14 , a case of 54 billion atoms is presented with the thermal aging at 573K. Then we introduce 0.00015 at.% of Cu atoms and 0.00008 at.% of vacancies within this configuration. The simulation time of 0.0226 is rescaled to 3 months. For the weak scalability in Fig.15, we initialize the simulation box with an average of 11 million atoms per process.

For the visualization in Fig.17, simulations of thermal aging of a Fe-1.34Cu(at.%) are performed at 663K for a rescaled time as 100 years. Also, we use a rigid bcc lattice of 40 unit cells in each of three dimensions. The visualizations are all done using Ovito software.

1.6 Experiments for reproducing Fig.11

-Build the binary.

$ cd ./online1/OpenKMC/

$ ./makeall.sh

-Entry the dir.

$ cd ./correct/

-Run a case.

$ ./test.sh -I test_name

1.7 Experiments for reproducing Fig.12

-Build the binary.

$ cd ./online1/OpenKMC/

$ ./makeall.sh

-Entry the dir.

$ cd ./breakdown/

-Run a case.

$ ./test.sh -I test_name

1.8 Experiments for reproducing Fig.14 to Fig.15

-Build the binary.

$ cd ./online1/OpenKMC/

$ ./makeall.sh

-Entry the dir.

$ cd ./scalability/

-Run a case.

$ ./test.sh -I test_name

1.9 Experiments for reproducing Fig.17

-Run Ovito.

-Load the output file generated by OpenKMC.

-Sort the atoms and obtain a rendered image.

-Save file.

ARTIFACT AVAILABILITY

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

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

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

Proprietary Artifacts: No author-created artifacts are proprietary.

List of URLs and/or DOIs where artifacts are available:
`http://www.nsccwx.cn/wxcyw/`

BASELINE EXPERIMENTAL SETUP, AND MODIFICATIONS MADE FOR THE PAPER

Relevant hardware details: SW26010 processor consists of four core groups (CGs) and they are connected to each other via the network on chip (NoC). Each CG includes 65 cores: 1 management processing element (MPE), and 64 Computing Processor Elements (CPEs), organized as an 8 multiply 8 mesh. The processor connects to other outside devices by a system interface (SI). Both the MPE and CPEs work at 1.45GHz and 256-bit vector instructions are supported. A CG has roughly 34.1 GB/s theoretical peak memory bandwidth and around 765 GFlops double-precision peak performance.

Operating systems and versions: Linux version 2.6.32-431.29.2.lustre.el6.x86_64

Compilers and versions: SWCC Compilers: Version 5.421-sw-500 (the customized compiler for Sunway architecture)

Applications and versions: OpenKMC

Libraries and versions: mpicc for SW version 2.2a

Key algorithms: self-adapting communication algorithm; Transcription-Translation-Transmission Algorithm

Output from scripts that gathers execution environment information.

```
$ cd ./online1/OpenKMC/
$ ./makeall.sh
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M variable.cpp > variable.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M universe.cpp > universe.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M timer.cpp > timer.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M spparks.cpp > spparks.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M solve_tree.cpp > solve_tree.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M solve_linear.cpp > solve_linear.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M solve_group.cpp > solve_group.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M solve.cpp > solve.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M shell.cpp > shell.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M set.cpp > set.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M region.cpp > region.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M region_block.cpp > region_block.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M read_sites.cpp > read_sites.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M random_park.cpp > random_park.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M random_mars.cpp > random_mars.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M potential.cpp > potential.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M pair_lj_cut.cpp > pair_lj_cut.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M pair.cpp > pair.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M output.cpp > output.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M memory.cpp > memory.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M math_extra.cpp > math_extra.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M main.cpp > main.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M library.cpp > library.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M lattice.cpp > lattice.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M input.cpp > input.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M image.cpp > image.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M groups.cpp > groups.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M finish.cpp > finish.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M error.cpp > error.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M dump_text.cpp > dump_text.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M dump_image.cpp > dump_image.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M dump.cpp > dump.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M domain.cpp > domain.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M diag_propensity.cpp > diag_propensity.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M diag_energy.cpp > diag_energy.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M diag.cpp > diag.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M diag_cluster.cpp > diag_cluster.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M diag_array.cpp > diag_array.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M create_sites.cpp > create_sites.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M create_box.cpp > create_box.d
```

```
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M comm_lattice.cpp > comm_lattice.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M cluster.cpp > cluster.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M app_vacancy.cpp > app_vacancy.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M app_lattice.cpp > app_lattice.d
sw5cc -O -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪     -M app.cpp > app.d
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c app.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c app_lattice.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c app_vacancy.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c cluster.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c comm_lattice.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c create_box.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c create_sites.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c diag_array.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c diag_cluster.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c diag.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c diag_energy.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c diag_propensity.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c domain.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c dump.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c dump_image.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c dump_text.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c error.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c finish.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c groups.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c image.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c input.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c lattice.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c library.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c main.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c math_extra.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c memory.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c output.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c pair.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c pair_lj_cut.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c potential.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c random_mars.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c random_park.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c read_sites.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c region_block.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c region.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c set.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c shell.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c solve.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c solve_group.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c solve_linear.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c solve_tree.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c spparks.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c timer.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c universe.cpp
mpiCC -O  -DSPPARKS_GZIP  -I/usr/sw-mpp/mpi2/include
↪      -host -fpermissive  -c variable.cpp
```

```
mpiCC -O -hybrid  -L/usr/sw-mpp -L/lib
↪  -I/usr/sw-mpp/mpi2/include       app.o
↪  app_lattice.o app_vacancy.o cluster.o
↪  comm_lattice.o create_box.o create_sites.o
↪  diag_array.o diag_cluster.o diag.o diag_energy.o
↪  diag_propensity.o domain.o dump.o dump_image.o
↪  dump_text.o error.o finish.o groups.o image.o
↪  input.o lattice.o library.o main.o math_extra.o
↪  memory.o output.o pair.o pair_lj_cut.o
↪  potential.o random_mars.o random_park.o
↪  read_sites.o region_block.o region.o set.o
↪  shell.o solve.o solve_group.o solve_linear.o
↪  solve_tree.o spparks.o timer.o universe.o
↪  variable.o     slave.o -o ../test_name
   text           data            bss
   ↪  dec            hex        filename
5205254        2740952
↪  217960        8164166
↪  7c9346        ../test_name
$ cd ./correct/
$ ./test.sh -I test_name
waiting for dispatch. . . . . .
dispatching. . . . . .
Job <45767573> has been submitted to queue
↪  <q_sw_share>.
```