Crystal-KMC: Parallel Software for Lattice Dynamics Monte Carlo Simulation of Metal Materials

Jianjiang Li, Peng Wei, Shaofeng Yang, Jie Wu, Peng Liu*, and Xinfu He

Abstract: Kinetic Monte Carlo (KMC) is a widely used method for studying the evolution of materials at the microcosmic level. At present, while there are many simulation software programs based on this algorithm, most focus on the verification of a certain phenomenon and have no analog-scale requirement, so many are serial in nature. The dynamic Monte Carlo algorithm is implemented using a parallel framework called SPPARKS, but it does not support the Embedded Atom Method (EAM) potential, which is commonly used in the dynamic simulation of metal materials. Metal material — the preferred material for most containers and components — plays an important role in many fields, including construction engineering and transportation. In this paper, we propose and describe the development of a parallel software program called Crystal-KMC, which is specifically used to simulate the lattice dynamics of metallic materials. This software uses MPI to achieve a parallel multiprocessing mode, which avoids the limitations of serial software in the analog scale. Finally, we describe the use of the parallel-KMC simulation software Crystal-KMC in simulating the diffusion of vacancies in iron, and analyze the experimental results. In addition, we tested the performance of Crystal-KMC in “meta -Era” supercomputing clusters, and the results show the Crystal-KMC parallel software to have good parallel speedup and scalability.

Key words: Kinetic Monte Carlo (KMC); communication optimization; parallel computation; Message Passing Interface (MPI)

1 Introduction

Kinetic Monte Carlo (KMC)[1], a method for studying the microstructural evolution of materials at the atomistic or micro scale, is widely used in many fields of scientific research, including chemical reactions, biology, and physical materials. Specifically, this method has been used to achieve important results in the process of simulating diffusion reactions at the microcosmic level[1], such as the simulation of metal surface gas absorption[2], film growth[3], grain growth[4], and material irradiation damage[5]. Hoffmann et al.[6] developed a kinetic Monte Carlo serial open-source software known as “KMOS” to study the oligodynamics in complex reaction networks by simulating the basic processes that occur on static lattice active sites. Borodin et al.[7] used the CASINO-LKMC program to simulate the formation process of helium-vacancy clusters in the iron of body-centered cubic structures. Zhang et al.[8] discussed a package management scheme based on containers. The newly developed method
can ease the maintenance complexity and reduce human mistakes.

Because the serial Monte Carlo method is limited by the complexity of the algorithm and the memory cost, the simulation time is long and the simulation scale is limited. With the rapid development and application of high-performance clusters, domestic and foreign scholars have studied and proposed many KMC parallel simulation algorithms and frameworks. In 2004, Shim and Amar[9] proposed a parallel-KMC semi-strict synchronous lattice subdomain algorithm for simulating thin film growth, which performs parallel optimization of diffusion functions and processed region partitioning. In 2008, Martínez et al.[10] proposed a KMC parallel algorithm that uses time synchronization to simulate a continuous diffusion reaction system. In 2010, the Sandia National Laboratory developed an open source KMC parallel simulation framework called SPPARKS[11] and a parallel-KMC simulation algorithm that supports multiple applications. In 2014, Leetmaa and Skorodumova[12] implemented a lattice dynamics simulation parallel framework called KMCLib which is easy to customize and integrate with other software programs. Since many research groups have designed their own physical models to simulate particular types of systems, it is difficult to develop parallel software that can meet the needs of all KMC applications. SPPARKS and KMCLib have shown better parallelism and scalability than others. However, in addition to KMC applications that have been implemented on the platform, users must implement their own physical models that correspond to the program interface. Also, because of the need for multi-model compatibility, the platform software logic and implementation is more complex, and requires more memory and higher computer performance. On the other hand, SPPARKS does not support the Embedded Atom Method (EAM) potential, which is typically used in metal-material dynamics simulations.

As the preferred material for most containers and components, metal materials play an important role in many areas such as construction and transportation. Therefore, we developed a parallel kinetic Monte Carlo software program, Crystal-KMC, which supports the EAM potential[13] for metal materials. This software uses MPI programming technology[14], makes full use of the fact that the same processes share memory, and reduces the required volume of communications between parallel areas. After analyzing the process topology and communication relationships, this software converts duplicate communications into point-to-point communication and adjusts the analog sequence of the simulation area, thereby decreasing the overall number of communications and traffic level and significantly improving simulation efficiency. In this paper, we use Crystal-KMC to simulate the diffusion of the vacancies in iron, then analyzes the experimental results. In addition, we test the performance of Crystal-KMC in “meta-Era” supercomputing clusters.

The remainder of this paper is organized as follows: Section 2 introduces the parallelization of KMC, including the spatial region division, process communication, and parallel KMC algorithm. In Section 3, we test the performance of Crystal-KMC, then analyze the experimental results. In Section 4, we summarize our findings and future outlook.

2 KMC Parallelization

KMC parallelization is generally divided into two categories: The first is task parallelism, which performs a parallel calculation for a sub-process without data dependency during the simulation process, and in which the process of calculating the transition probability is generally parallelized. The other category is data parallelism[15], which splits the simulation area into blocks, each of which is independently simulated, thereby eliminating the impact of adjacent data communicated between processes. So, while the task parallelism speeds up the program calculation to improve simulation efficiency, it cannot break through the limitations of the computer’s memory at the simulation scale. The data parallelism eliminates the limitations of the computer’s memory at the computation scale and effectively accelerates the program’s execution, although it cannot be applied to models with strong overall data relevance or data distributions that change frequently. For an algorithm, the choice of parallelization method depends on the data model and algorithmic implementation. Solid material has a regular lattice structure that can be mapped into static lattice points[10], and the movement of atoms in the solid material is not very intense. Based on these characteristics, data parallelism is more suitable for the parallelization of the solid-state lattice kinetic Monte Carlo, so the parallelization method we use in the Crystal-KMC parallel software is data parallelism.

Strictly speaking, identifying an event from all events that may occur is essentially a serial algorithm. The configuration will change as the event occurs, which will
affect the choice of the next event. In KMC’s data parallel method, the entire system is divided into multiple sub-regions and then mapped to different processors. Each sub-region independently selects and executes events, which requires the spatial decoupling of events, which is why the whole system must be approximated. In the system, the greater the distance between the atoms, the smaller is the interaction and the less dependent the event. The dependency of an event at a position greater than a certain distance can be approximated as zero, to thereby achieve the spatial decoupling of events.

2.1 Spatial area division

In the Crystal-KMC parallel software we describe here, the whole simulation area is divided into cells with a length of $2r_{\text{KMC}}$ (small squares divided by dotted lines shown in Fig. 1). In the program, we calculate the $r_{\text{KMC}}$ value from the input in the run command and the configuration file, with $r_{\text{KMC}}$ being no smaller than the cut-off radius $r_{\text{cut}}$. (We assume, here, that the interaction between atoms and other atoms outside their cut-off radius is zero, or that the force is small enough to be negligible.) The simulation area of each process can contain multiple cells.

Each process has its own subdomain and outermost buffer cell. As shown in Fig. 1, the whole simulation area is divided into $12 \times 12$ cells in a two-dimensional environment and is assigned $3 \times 3$ processes, with each process having $4 \times 4$ cells.

Since the KMC method is based on the stochastic theory, if the regions of parallel execution are adjacent, it is possible that two atoms could jump into the same vacancy at the same time, which is a phenomenon known as critical position competition. In this study, we used the labeling method shown in Fig. 2a (simulation areas that perform KMC events at the same time are indicated by having the same background pattern), to ensure that the KMC simulations in adjacent areas were performed asynchronously to avoid any potential critical resource competition between adjacent areas. Figure 2b shows a regional simulation example to illustrate the executive situation of simulating the entire area at the same time (the slashed-lines area in the figure).

2.2 Communication

In the simulation of data parallelism, we divided the entire simulation area into multiple sub-regions, using different processors to independently perform separate KMC processes. These sub-regions share a common boundary, and the area near the boundary has data dependency. As such, it is essential to carry out inter-process communication between corresponding critical areas. This way, these areas can receive the latest boundary information to ensure the correctness of the boundary simulation. Taking the data dependency in two-dimensional space as an example, where the area that could affect the evolutionary path of atoms in a cell is the area of the eight cells adjacent to it (cells’ side lengths are greater than or equal to $2r_{\text{cut}}$). That is, the region that interacts with a cell includes that of the cells circling it. To address this, we built a cell shell to surround each process area as a buffer. The buffer is used to store atomic information of the area for the neighboring process that interacts with it.

Next, we gave a simple example to illustrate the communication between processes. As shown in Fig. 3, we divided a two-dimensional process simulation area into four thread areas, which respectively correspond to threads 0, 1, 2, and 3. Each thread region is divided into four cells, respectively: cell$_0$, cell$_1$, cell$_2$, and cell$_3$. Before executing the KMC simulations for cell$_0$, the process receives atomistic information regarding any data dependency with the cell$_0$ region, which is sent by the neighboring processes on the left, top left corner, and upper bounds. At the same time, the process also sends information to the neighboring process on its right, lower right corner, and lower bounds. Similarly, before executing KMC simulation on cell$_1$, the process receives data from the upper, upper right, and right neighboring processes, and sends corresponding data to the left, lower left, and...
lower neighboring processes. Before executing a KMC simulation on cell2, the process receives data from the left, lower left, and lower neighboring processes, and sends relevant data to the right, upper right, and upper neighboring processes. Before executing a KMC simulation on cell3, the process receives data from the right, lower right, and lower neighboring processes, and sends corresponding data to the left, upper left, and upper neighboring processes. It takes 24 communications to execute a threshold-time KMC simulation for all areas in a two-dimensional environment.

Figure 3 shows the communication between the processes, each of which process contains four threads, with each thread divided into four cells: cell0, cell1, cell2, and cell3. The four graphs in the figure illustrate the communication processes that occur before the four cells executing the KMC simulation. This inter-process communication has some shortcomings. First, it is essential that they receive the atomistic boundary information from their upper neighboring processes prior to executing the simulation of cell0 and cell1, and any information received twice will have overlapping areas. Also, the process does not modify data from the cell2 and cell3 regions when simulating the cell0 and cell1 regions, so there is no change in the area where the process communicates with the above neighboring process before simulating the cell0 and cell1 regions. Similarly, this process has problems with data that is communicated prior to that other cell regions and those that are adjacent. As in the simulation of solid metal materials, the communicated data is only that within the cut-off radius, and this amount of data is small. Therefore, it is necessary to merge the communication between adjacent cell regions, as shown in Fig. 4.

Before the simulation, each process communicates with its adjacent processes in initializing its buffers, which are executed only once during the simulation. In the next simulation, data communication is executed according to the rules shown in Fig. 4. Before simulating the cell0 region, the process receives data from the adjacent processes in the upper and upper left corners while sending data to adjacent processes in the lower and lower right corners; before simulating the cell2 region, the process receives data from adjacent processes in the left and lower left corners while sending data to adjacent processes in the upper and upper right corners; before simulating the cell3 region, the process receives data from adjacent processes in the lower and lower right corners while sending data to adjacent processes in the upper and upper left corners.

In addition, we found that there is data (sent from the process in the top left corner) in the ghost areas of the upper processes, (i.e., in the upper left corner of the process area and at the edge of the adjacent upper process area), and their contents are the same. As shown in Fig. 5, four adjacent process areas are labeled process0 (dotted area), process1 (area with wavy line slashes), process2 (area with wavy lines), and process3 (area with vertical lines). In the first step, process0 and process2, respectively, send data that are on the boundary to process1 and process3. In the second step, process0 and process3, respectively send data that are on the boundary to process2 and process3.

In Fig. 5, the data marked with a circle were originally on the lower right corner of process0, and in the first communication it is sent to the ghost area of process1. In
the second communication, process \(1\) sends data to the ghost area of process \(3\). Since this double communication corresponds to the communication processes in cell \(1\) and cell \(0\) in Fig. 4, during the double communications, the data in the lower right corners of these processes does not change, i.e., the data marked with circle in Fig. 5 does not change. In the second communication, the data of the neighboring process in the upper left corner can be obtained by communicating with the upper process. In this way, before executing KMC simulations for each cell region, the process need only communicate with two corresponding sides, receiving data sent from one side, and sending data to the other side, as shown in Fig. 6. Thus, it takes just eight communications to execute a threshold-time KMC simulation for all the areas in a two-dimensional environment. Compared with the communication mode shown in Fig. 3, the number of communications is greatly reduced.

In three-dimensions, the simulation area cell block is divided as shown in Fig. 7. Its communication principle is similar to that in the two-dimensional environment, with all the processes and their adjacent processes executing point-to-point communications to update their ghost area data before initiating iterative simulation or after the iterative simulation is completed.

The execution order of each cell block in the iterative simulation process is as follows: cell \(0\) → cell \(1\) → cell \(3\) → cell \(2\) → cell \(6\) → cell \(7\) → cell \(5\). Before the simulation of cell \(0\), the left side of cell \(0\) receives critical-area data from the coplanar process, and updates its corresponding ghost area while sending the critical-area data to the right coplanar process. In the same way, before the simulation of cell \(1\), cell \(1\)’s top surface receives data and its bottom surface sends data; before the simulation of cell \(3\), cell \(3\)’s top surface receives data and its bottom surface sends data; before the simulation of cell \(2\), cell \(2\)’s bottom surface receives data and its top surface sends data; before the simulation of cell \(6\), cell \(6\)’s right surface receives data and its left surface sends data; before the simulation of cell \(7\), cell \(7\)’s top surface receives data and its bottom surface sends data; before the simulation of cell \(5\), cell \(5\)’s back surface receives data and its front surface sends data;
and before the simulation of cell4, cell4’s bottom surface receives data and its top surface sends data. Thus, it takes 16 communications to execute a threshold time KMC simulation for all areas.

2.3 Parallel-KMC algorithm

The proposed Crystal-KMC parallel software uses the direct KMC method, which is based on vacancy transition. Each step must produce just two random numbers evenly distributed between (0, 1], the first of which is used to select a vacancy and the transition direction of the vacancy, and the second determines the simulation times. First, the KMC step calculates the probability of all the vacancies in the eight directions where a transition may take place, and obtains the sum.prob by adding them together. Then, (1) the software generates a random number \( r_1 \) and calculates the system vacancies to be jumped and the direction the vacancies will jump, based on the sum.prob and \( r_1 \). (2) The software generates a random number \( r_2 \) and obtains the simulated running time in the KMC step based on the sum.prob and \( r_2 \). Finally, the software executes a vacancy transition and updates the data.

The parallel-KMC algorithm, which we call the parallel-LKMC algorithm, improves upon the serial KMC algorithm, and introduces a time threshold and process communication to achieve time synchronization and data unification for the entire system. Table 1 lists and describes the associated variables in Algorithm 1.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{threshold}} )</td>
<td>Time threshold of each cell continuously executing KMC events at one time</td>
</tr>
<tr>
<td>( t )</td>
<td>Temporary variable in threshold time</td>
</tr>
<tr>
<td>( \Delta t )</td>
<td>Time of this KMC event occurred</td>
</tr>
<tr>
<td>( \text{totNumVacancy} )</td>
<td>Total number of vacancies in cell</td>
</tr>
<tr>
<td>( \text{numVacancy} )</td>
<td>Vacancy count variable</td>
</tr>
<tr>
<td>( \text{totNumDirection} )</td>
<td>Total number of transition directions</td>
</tr>
<tr>
<td>( \text{numDirection} )</td>
<td>Transition direction count variable</td>
</tr>
<tr>
<td>( P_{\text{numVacancy},\text{numDirection}} )</td>
<td>Probability that the vacancy transitions to a certain direction</td>
</tr>
<tr>
<td>( v )</td>
<td>Attempt frequency</td>
</tr>
<tr>
<td>( \Delta E_{A_i} )</td>
<td>Activation energy of event A</td>
</tr>
<tr>
<td>( k )</td>
<td>Boltzmann constant</td>
</tr>
<tr>
<td>( T )</td>
<td>Temperature</td>
</tr>
<tr>
<td>( \text{type} )</td>
<td>Atom type</td>
</tr>
<tr>
<td>( E_{A_0}(\text{type}) )</td>
<td>Atomic transfer energy in pure matrix</td>
</tr>
<tr>
<td>( E_i )</td>
<td>Total energy of system before atomic transition</td>
</tr>
<tr>
<td>( E_f )</td>
<td>Assumption of the total energy of system after the atomic transition</td>
</tr>
<tr>
<td>( \text{sum.prob} )</td>
<td>Total energy of all vacancies in cell</td>
</tr>
<tr>
<td>( r_1 )</td>
<td>Random number between (0,1]</td>
</tr>
<tr>
<td>( P_j )</td>
<td>Energy of the ( j )-th vacancy</td>
</tr>
<tr>
<td>( r_2 )</td>
<td>Random number between (0,1]</td>
</tr>
</tbody>
</table>
Algorithm 1 Parallel-LKMC algorithm

Input: location of the atom, atomic potential function table
Output: location of the atom, energy of the atom
1: \( t = 0; \)
2: while \( t < t_{\text{threshold}} \) do
3: for \( 1 \leq \text{numVacancy} < \text{totNumVacancy} \) do
4: for \( 1 \leq \text{numDirection} < \text{totNumDirection} \) do
5: \( \Delta E_i = E_{\text{eq}} \left( \text{type}_i \right) + \left( E_f - E_i \right)/2; \)
6: \( P_{\text{numVacancy}, \text{numDirection}} = \exp \left( -\Delta E_i / kT \right); \)
7: \( \text{sum_prob} = \text{sum_prob} + P_{\text{numVacancy}, \text{numDirection}}; \)
8: end for
9: end for
10: generate random number \( r_1 \) between 0 and 1;
11: \( P_{\text{numVacancy}, \text{numDirection}} = \sum_{\text{numDirection}=1}^{\text{totNumDirection}} P_{\text{numVacancy}, \text{numDirection}}; \)
12: \( \Delta t = -\ln(r_1) / \sum_{\text{numVacancy}=1}^{\text{totNumVacancy}} P_{\text{numVacancy}}; \)
13: if \( t + \Delta t < t_{\text{threshold}} \) then
14: generate random number \( r_2 \) between 0 and 1;
15: \( r_1 = r_2 \cdot \text{sum_prob} \)
16: if \( \sum_{\text{numVacancy}=1}^{k} P_{\text{numVacancy}} \leq r < \sum_{\text{numVacancy}=1}^{k+1} P_{\text{numVacancy}} \) then
17: select the \( k \)-th vacancy;
18: end if
19: if \( \sum_{\text{numDirection}=1}^{k-1} P_{\text{numVacancy}, \text{numDirection}} + \sum_{\text{numDirection}=1}^{d} P_{k-1, \text{numDirection}} \leq \sum_{\text{numDirection}=1}^{k} P_{\text{numVacancy}, \text{numDirection}} \) then
20: select the \( d \)-th direction;
21: end if
22: do jump;
23: \( t = t + \Delta t; \)
24: else
25: \( t = t_{\text{threshold}}; \)
26: end if
27: end while

(1) Lines 3–9: Calculate the total transition probability of the system. The activation barrier \( \Delta E_{\text{ai}} \) is a constant that is independent of the environment, and is only associated with the transitional atomic type.

(2) Lines 10–12: Calculate the evolution time of this KMC event.

(3) Lines 13–26: If the sum of the successive evolution times is less than or equal to the time threshold, the algorithm chooses to perform KMC events. Otherwise, the cell’s successive evolution time is equal to the time threshold. Lines 16 and 17: Vacancies that undergo place transition. Lines 19 and 20: Direction of the vacancies’ transitions. Line 22: Execute a vacancy transition event.

Pseudo-code for the execution of the MPI parallel-KMC simulation based on data parallelism is shown in Table 2.

Step description:

Lines 4 and 21: The process and surrounding processes both execute point-to-point communications. Line 4 initializes the ghost area and line 21 writes all the data in the ghost area back to the simulation area.

Lines 5, 7, 9, 11, 13, 15, 17, and 19: The command “communicate.ASendBRecv” indicates that the process sends data to the process adjacent to the A side, and receives data sent from adjacent processes on the B side.

The communication mode is synchronous.

Lines 6, 8, 10, 12, 14, 16, 18, and 20: The execute C-KMC in cell_i indicates that the program executes KMC simulation on cell_i.

3 Test and Analysis

3.1 Crystal-KMC’s correctness test

In this section, we use the Crystal-KMC parallel software to simulate a practical example: the diffusion of vacancies in iron[12]. The experimental subject is a simple rigid cube box whose solute is iron and solvents are randomly distributed vacancy defects. The size of the simulation box is \( 100 \times 100 \times 100 \), the simulation temperature is \( 600 \) K, the potential function is the EAM, the green grids represent vacancies (vacancies account for 1.25%), and the black grids are iron atoms. Figure 8 shows the simulation result.
Fig. 8 The process of vacancy spread and void formation in the iron when the temperature is 600 K. The simulation time is (a) 0 mcs, (b) $1.25 \times 10^{-4}$ mcs, (c) $1.25 \times 10^{-3}$ mcs, and (d) $1 \times 10^{-2}$ mcs, where mcs is Monte Carlo time.

In the evolutionary system, green dots represent vacancies, and green clusters are vacant clusters that occur due to the formation of pores. This is consistent with the results of the SPPARKS simulations reported in Ref. [16]. The vacancies continuously aggregate to form vacancy clusters, and eventually, the pores will continue to coarsen until only a large pore remains.

### 3.2 Multi-process simulation test

We performed all the tests described in this section on the new generation of supercomputing clusters known as “meta-Era”. There are a total of 270 clusters, a two-way blade server CB60-G16, and overall CPU performance is 120.96 TFlops. Each blade compute node is configured with two Intel E5-2680 V2 (Ivy Bridge || 10C || 2.8 GHz) processors and 64 GB DDR3 ECC 1866 MHz of memory, and each processor contains 10 processing cores. The operating system uses the RedHat Linux 4.4.7-3 version, the compiler is an Intel C++ 2013_sp1 version, the MPI environment is the MVAPCH2-intel 1.9 version, and the time measurement is performed by “MPI_Wtime()”.

### 3.3 Crystal-KMC’s performance test

In this study, we performed the multi-process simulation using the Crystal-KMC parallel software program in an experiment on the vacancy dissolution in iron. The size of simulation box was $200 \times 200 \times 200$, the number of atoms was $1.6 \times 10^7$, in which the number of vacancies was 500, the copper atom concentration was 1%, the temperature was 600 K, the potential function was the EAM, and the KMC simulation time was $0.625 \times 10^{-6}$ s. Table 3 shows the running times for the number of processes, i.e., 1, 2, 4, 8, 16. As we can see from Table 3, the higher the number of processes, the shorter is the running time of the program.

### 3.4 Crystal-KMC’s scalability test

Due to the limited number of CPU cores, to achieve higher execution performance, we cannot set the number of internal threads to be too large. But the number of processes can be expanded by increasing the number of CPUs. To analyze the scalability of the Crystal-KMC parallel software program, we tested the performance of the program when the number of threads was 1 and the number of processes was 100, 200, 400, and 800, respectively. The test application was the diffusion of vacancies in copper-iron alloy. For this experiment, the size of the simulation box was $800 \times 800 \times 800$, the number of atoms was $1.024 \times 10^9$, the number of vacancies was $1 \times 10^5$, the copper concentration was 1%, and the temperature was 600 K. The potential function was the EAM, and the time of the KMC simulation was $0.6 \times 10^{-4}$ s. Figure 9 shows the test results, which reveal that the parallel acceleration ratio of the Crystal-KMC parallel software program increases with an increasing number of processes. As a reference, we used the computation time when the number of processes was 100. The speed-up ratio of the parallel software was 3.91 when the number of processes was 800, which indicates the good scalability of this parallel software program.

### 4 Summary and Outlook

In this study, we determined that the parallel lattice kinetic Monte Carlo software program Crystal-KMC can overcome the limitations of serial software, both in the scale and scalability of the simulation, which makes it applicable to large-scale simulations. We used the parallel-KMC simulation software Crystal-KMC to simulate the diffusion of vacancies in iron, and verified its correctness.

<table>
<thead>
<tr>
<th>Number of processes</th>
<th>Running time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1156.72</td>
</tr>
<tr>
<td>2</td>
<td>522.16</td>
</tr>
<tr>
<td>4</td>
<td>264.75</td>
</tr>
<tr>
<td>8</td>
<td>185.36</td>
</tr>
<tr>
<td>16</td>
<td>126.71</td>
</tr>
</tbody>
</table>

Table 3 Running time when the number of processes is 1, 2, 4, 8, 16.
by analyzing the experimental results. On the other hand, we also tested the performance of proposed Crystal-KMC parallel software program on the new generation super-computing cluster “meta-Era”, and the experimental results indicate that parallel software has better parallel performance and scalability.

Acknowledgment

This work was supported by the National Key R & D Program of China (Nos. 2017YFB0202003 and 2017YFB0202104).

References


Jianjiang Li is currently an associate professor at University of Science and Technology Beijing, China. He received the PhD degree in computer science and technology from Tsinghua University in 2005. He was a visiting scholar at Temple University from Jan. 2014 to Jan. 2015. His current research interests include parallel computing and parallel compilation.

Peng Wei is a master student at University of Science and Technology Beijing, China. He received the bachelor degree from Qufu Normal University in 2016. His current research interests include parallel computing, cloud computing, parallel compilation, and big data.

Jie Wu is a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, USA, he was a program director at the National Science Foundation and a distinguished professor at Florida Atlantic University. He received the PhD degree from Florida Atlantic University in 1989. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. He is a CCF Distinguished Speaker and a fellow of the IEEE.

Shaofeng Yang is a computer engineer in CAS Computing Institute. She received the master degree from University of Science and Technology Beijing in 2016. Her current research interests include cloud computing and parallel computing.

Peng Liu received the BS and MS degrees from Hangzhou Dianzi University in 2001 and 2004, respectively, and PhD degree in computer science and technology from Zhejiang University in 2007. Currently he is an associate professor at Hangzhou Dianzi University. His research interests include wireless sensor networks, delay tolerant networks, parallel computing, and mobile computing.

Xinfu He received the BS degree from Hunan Normal University, China in 2004, and the PhD degree in nuclear engineering from China Institute of Atomic Energy in 2015. He is currently an associate professor in the Department of Nuclear Reactor Engineering at China Institute of Atomic Energy. His research interests include multiscale modeling of radiation damage in nuclear materials.