Nuclear Instruments and Methods in Physics Research B 352 (2015) 27–30

Contents lists available at ScienceDirect

Nuclear Instruments and Methods in Physics Research B

journal homepage: www.elsevier.com/locate/nimb

Synchronous parallel Kinetic Monte Carlo: Implementation and results for object and lattice approaches

Ignacio Martin-Bragado [a,*], J. Abujas [b], P.L. Galindo [b], J. Pizarro [b]

[a] IMDEA Materials Institute, C/ Eric Kandel 2, 28906 Getafe, Madrid, Spain
[b] Departamento de Ingeniería Informática, Universidad de Cádiz, Puerto Real, Cádiz, Spain

ARTICLE INFO

Article history:
Received 20 June 2014
Received in revised form 9 December 2014
Accepted 10 December 2014
Available online 14 February 2015

Keywords:
Kinetic Monte Carlo
Parallel
Object
Lattice
Simulation

ABSTRACT

An adaptation of the synchronous parallel Kinetic Monte Carlo (spKMC) algorithm developed by Martinez et al. (2008) to the existing KMC code MMonCa (Martin-Bragado et al. 2013) is presented in this work. Two cases, general enough to provide an idea of the current state-of-the-art in parallel KMC, are presented: Object KMC simulations of the evolution of damage in irradiated iron, and Lattice KMC simulations of epitaxial regrowth of amorphized silicon. The results allow us to state that (a) the parallel overhead is critical, and severely degrades the performance of the simulator when it is comparable to the CPU time consumed per event, (b) the balance between domains is important, but not critical, (c) the algorithm and its implementation are correct and (d) further improvements are needed for spKMC to become a general, all-working solution for KMC simulations.

1. Introduction

The Kinetic Monte Carlo (KMC) algorithm has been widely used in many fields of computational physics with notable success since its inception [1]. Although there are several methods called Kinetic Monte Carlo, here we want to focus in two main ones: Object and Lattice Kinetic Monte Carlo.

Object KMC focuses only on the defects included in the system by discarding the lattice and following only some "objects" that mimic the defect behavior. It is mainly used to study the evolution of crystalline systems under irradiation, including metals (iron, tungsten, copper [2]), and semiconductors (silicon [3], germanium, silicon carbide, gallium arsenide).

Lattice KMC maintains the lattice of the system and uses it to follow the evolution of defects in the solid, for instance vacancy diffusion [4]. Another recent application of Lattice KMC is the simulation of epitaxial growth, either as the result of an internal phase transformation [5] or an external deposition [6].

Given the widespread use and power of KMC, it is understandable that a big effort towards its parallelization and scalability is being done. Such an effort tries to overcome one of the main limitations of KMC: it is computationally expensive to simulate realistic system sizes and times. Unfortunately KMC, in contrast to other algorithms (for instance Molecular Dynamics), is highly asynchronous, and thus a simple parallelization provided by domain splitting is not straightforward to implement and might even produce the wrong physical evolution of the system under consideration.

To overcome such limitation, several approaches have been taken. One is the implementation of asynchronous kinetics, with complex algorithms that accept or reject events based on domain correlation schemes [7–9]. A different approach is to advance time synchronously and to avoid boundary errors produce by neighboring threads running simultaneously [10,11].

Most recently, a synchronous parallel Kinetic Monte Carlo (spKMC) algorithm based on a generalization of the rejection-free *n*-fold method [1] has been proposed for continuum diffusion–reaction systems [12] and 3D Ising systems [13]. A question mark remains on whether such an algorithm can be successfully applied to a general OKMC or LKMC problem, with several objects and reactions, where high inhomogeneities are to be expected, and thus what is the impact of parallelization on existing KMC codes for damage evolution or crystal growth. This is the goal of this work, where we have taken an existing serial OKMC and LKMC code, MMonCa [14], applied and adapted the synchronous parallel KMC algorithm to it using OpenMP under the shared memory parallel paradigm, and used the parallel version in two problems that, instead of having been chosen to show strong parallelism, are of interest for computational physicists and engineers and the results of which have already been published [14,15]. This allows us to perform a fair comparison and assessment of the current state-of-the-art of parallelism in KMC for material sciences.

* Corresponding author. Tel.: +34 917871881; fax: +34 915503047.
E-mail address: ignacio.martin@imdea.org (I. Martin-Bragado).

http://dx.doi.org/10.1016/j.nimb.2014.12.081
0168-583X/

2. Model and implementation

This section explains the model and implementation used to transform the serial KMC algorithm into a parallel one. For details on the particular physical models for LKMC or OKMC previous publications can be read [14,15]. Our parallelization scheme is heavily based on the synchronous parallel KMC algorithm of Martínez et al. detailed in Ref. [13]. Such algorithm and our adaptation to `MMonCa` and to the shared memory parallel paradigm can be split in two parts: (a) the management of time, i.e., the synchronicity, and (b) the management of boundary conflicts.

2.1. Time tracking

The time tracking implementation is similar to the proposed in Ref. [12], which is an adaptation of the Gillespie KMC formulation [16]. It is written generically for a system with m domains, $(\Omega_i,\ 0 \leqslant i < m)$, each of them divided in p sub-domains or "colors" (see Fig. 1 and further explanation in Section 2.2). In our implementation, each domain is processed by a different thread, while the total memory is shared and accessible by all of them.

1. The total cumulative rate for each sub-domain i, having n_i events to simulate, is built as $R_i = \sum_j^{n_i} r_{ij}$.
2. A maximum of all the cumulative rates is computed: $R_{\max} \geqslant \max_{i=0}^{m} R_i$.
3. Null events rate are assigned to each sub-domain: $r_i^{\text{null}} = R_{\max} - R_i$.
4. A random number is used to choose one, and only one, "color" for all domains.
5. For each sub-domain of the right color, one event is chosen proportionally to its rate and executed. This is the part done in parallel.
6. Time is advanced by $\Delta t = \log(1/s)/(p * R_{\max})$, being $s \in (0,1)$ a random number.
7. The algorithm is repeated until the requested simulated time is reached.

The logic behind this algorithm is that the addition of the null events makes all sub-domains equiprobable with a total cumulative rate of $p * R_{\max}$, thus allowing us to pick up which one is to be executed with a linearly distributed random number.

2.2. Boundary conflicts

The solution of boundary conflicts when particles interact across domain boundaries is implemented using the chessboard sub-lattice technique for KMC proposed by Shim and Amar [10], but in contrast with previous implementations and as shown in Fig. 1, we split the space between two adjacent domains in at least three sub-domains. In principle, the isolation produced by one sub-domain should be enough to prevent boundary conflicts of physical nature, i.e., to decide what happens when two neighboring particles interact across the boundary by removing the possibility of adjacent sub-domains to have contradictory events performed. Nevertheless, this mechanism alone is not enough to avoid memopry access related problems in our shared memory paradigm. Since in our approach the position of each particle determines the region where it belongs, and each region has a block of memory to be accessed, conflicts can arise when particles from different threads enter the same region, and consequently two threads try to write into the same memory block at the same time. This can be solved by implementing locks and semaphores that control and put order to shared regions accesses, but that implies an increase in code complexity plus an extra synchronization overhead. We have taken the alternate, simpler approach, of having three sub-domains per domain to assure that, even if diffusing particles are crossing to a neighboring boundary, two threads will never access the same region at the same time, which translates under the correct implementation in threads never accessing the same memory at the same time, and thus requiring no thread synchronization or boundary communication of any type.

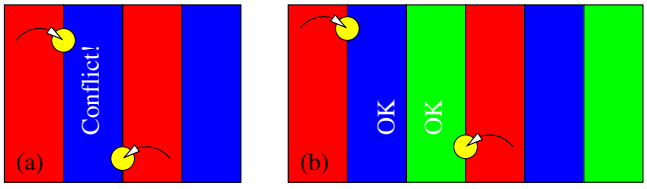

Fig. 1. Schematic representation of the subdivision of domains. Each thread executes the same color (red in this figure). In (a), the parallel diffusion of particles inside the same area is prone to generate memory conflicts. This is solved in (b) increasing the number of subdivisions. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3. Results and discussion

3.1. Object Kinetic Monte Carlo

The benchmark chosen for the parallel Object Kinetic Monte Carlo simulations has been the reproduction of the abrupt changes in electrical resistivity studied in Ref. [17] and simulated in Refs. [18,14]. In particular, we will compare simulated results between our serial code [14] and our parallel implementation shown in this work. An isochronal annealing of 2×10^{-4} dpa irradiated α-iron has been simulated in a box sized $143.5 \times 143.5 \times 143.5$ nm^3 using two different random seeds. Serial versions of the code, together with parallel simulations using 1, 2, 4, 6 and 8 threads have been simulated. The simulation results for the evolution in time and its first derivative of the total number of defects can be observed in Fig. 2. The similar shape of all the curves between them and with the previous serial implementation validates the parallel implementation.

Fig. 3 shows the performance of the OKMC parallelization, by comparing the total CPU time for the different cases explained

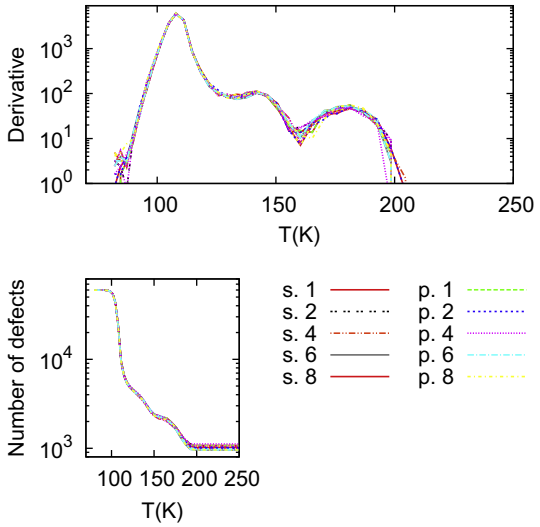

Fig. 2. Comparison of damage evolution and its first derivative for the Object KMC serial binary (s) and the parallel binary (p) with 1, 2, 4, 6 and 8 threads. The agreement validates the implementation.