Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational Physics 227 (2008) 3804–3823
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Synchronous parallel kinetic Monte Carlo for continuum diffusion-reaction systems

E. Martínez [a,b,*], J. Marian [a], M.H. Kalos [a], J.M. Perlado [b]

[a] *Lawrence Livermore National Laboratory, Livermore, CA 94551, USA*
[b] *Instituto de Fusión Nuclear, Universidad Politécnica de Madrid, 28006 Madrid, Spain*

Received 13 April 2007; received in revised form 19 November 2007; accepted 23 November 2007
Available online 23 December 2007

Abstract

A novel parallel kinetic Monte Carlo (kMC) algorithm formulated on the basis of perfect time synchronicity is presented. The algorithm is intended as a generalization of the standard *n*-fold kMC method, and is trivially implemented in parallel architectures. In its present form, the algorithm is not rigorous in the sense that boundary conflicts are ignored. We demonstrate, however, that, in their absence, or if they were correctly accounted for, our algorithm solves the same master equation as the serial method. We test the validity and parallel performance of the method by solving several pure diffusion problems (*i.e.* with no particle interactions) with known analytical solution. We also study diffusion-reaction systems with known asymptotic behavior and find that, for large systems with interaction radii smaller than the typical diffusion length, boundary conflicts are negligible and do not affect the global kinetic evolution, which is seen to agree with the expected analytical behavior. Our method is a controlled approximation in the sense that the error incurred by ignoring boundary conflicts can be quantified intrinsically, during the course of a simulation, and decreased arbitrarily (controlled) by modifying a few problem-dependent simulation parameters.

PACS: 02.70.−c; 02.70.Uu; 61.72.Ss

Keywords: Kinetic Monte Carlo; Parallel computing; Diffusion; Scalability

1. Introduction

Kinetic Monte Carlo (kMC) has proven an extremely powerful method to simulate the time evolution of Markovian processes [1,2]. kMC relies on the *a priori* knowledge of a given set of transition rates characterizing the simulated processes, which are assumed to obey Poisson statistics. The scope of applications for kMC is extraordinarily wide, ranging from epidemiology and population kinetics to surface growth or radiation damage. Because of its versatility, ease of implementation, and wide range of applications, kMC has been

* Corresponding author. Address: Lawrence Livermore National Laboratory, Livermore, CA 94551, USA.
E-mail address: martinez210@llnl.gov (E. Martínez).

0021-9991/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2007.11.045

the object of a significant parallelization effort in order to take advantage of existing and upcoming tera- and peta-scale computing capabilities. However, the difficulty of parallelizing kinetic Monte Carlo lies in the intrinsic time discreticity underlying event-driven simulations, which are sequential in character, and do not lend themselves to trivial parallel implementations. The ultimate validity test for any parallel kMC (pkMC) algorithm is that it solve the same master equation as the sequential (serial) method rigorously. This does not necessarily imply that both approaches give the same sequence of events, but that, on average, both give the same kinetic evolution resulting in the same statistical distributions as a function of time.

Early attempts to use parallel algorithms achieved some speedup but failed to satisfy these requirements [3,4]. One family of methods that do ensure this compatibility between sequential and parallel processes is based on asynchronous kinetics, with different processors simulating events independently and then accepting or rejecting them on the basis of domain correlation schemes that may severely limit the computational efficiency [5–10]. Most of the recent work in this area has been inspired by Lubachevsky's original paper [6], which provides an exact parallel algorithm for discrete-event simulations. This class of algorithms attempts to advance a 'virtual time horizon' (VTH) asynchronously by a combination of kMC steps whose progression is controlled by relatively cumbersome acceptance/rejection techniques. The progress rate of the simulation depends on the density of local minima of the instantaneous VTH, which in turn depends on the relative occurrence of event roll-backs across domain boundaries. Depending on the problem at hand, VTHs can display a strongly fluctuating behavior, for which ingenious roughness-suppressing algorithms have been proposed [7,8]. Another interesting alternative for parallel event-driven simulations is Jefferson's *time warp* algorithm [10]. The time warp paradigm provides a protocol for minimizing the number of conservative synchronization updates by ignoring causality errors, which are later detected and retraced for their resolution. Whichever the method chosen, owing to their intrinsic implementation complexity and limited parallel efficiency, little use has been made of these methods in terms of physical applications.

An obvious way to avoid roll-backs due to time evolution mismatches is to advance time synchronously. However, in this case, boundary errors due to conflicts among neighboring processors may still occur. There have been several parallel algorithms involving the so-called synchronous relaxation scheme that treat these conflicts rigorously [11–14]. However, although these algorithms effectively advance a flat VTH front (hence the term 'synchronous'), they still rely on an alternative form of roll-backs whereby an iterative scheme is used to ensure consistency among the sequences of events generated in each processor. This may result in poor efficiency and a large communications overhead that is seen to grow logarithmically with the number of processors [12,13]. These limitations can be partially mitigated by using more approximate methods, such as the synchronous sublattice algorithm recently proposed by Shim and Amar in the context of thin film growth simulations [15]. Although this method is only semi-rigorous, it has proven very promising due to its straightforward scheme for solving boundary errors and the absence of global communications. Nevertheless, despite the recent progress in parallel discrete-event simulations, both synchronous and asynchronous, the development and application of rigorous efficient parallel algorithms for kMC simulations remains a significant challenge.

In this article we propose a synchronous, parallel generalization of the rejection-free n-fold kMC method of Bortz et al. [16] (hereafter referred to as BKL for brevity). Our algorithm ensures a flat VTH construction, thereby rendering all communications between domains essentially trivial and facilitating its implementation. While our algorithm is not exact in its present form, we show that, for many practical applications, errors are essentially negligible. Next, we describe the algorithm in detail, discuss its correctness and the treatment of boundary conflicts, and we study its potential intrinsic performance. Then, in Section 3, we validate the method and study its scalability by solving several well-understood diffusion problems.

2. Parallel kMC algorithm

In BKL, a system with N walkers, each with rate r_i ($i = 1 \ldots N$), is evolved in time by randomly selecting an event with probability r_i/R_{tot}, where $R_{\text{tot}} = \sum_i^N r_i$ is what we hereafter term the *frequency line*, *i.e.* the aggregate of all the individual r_i. The system is then advanced in time by randomly sampling from an exponential distribution $\exp(-R_{\text{tot}}\delta t_{\text{BKL}})$ [16]. We build upon the BKL framework to formulate our parallel algorithm.

2.1. General algorithm

The algorithm is based on the introduction of 'null' events to achieve perfect synchronicity. The idea was originally proposed by Hanusse and Blanché to study large diffusion-reaction systems with their improved minimal process method [17]. The concept has been elaborated further by other authors [18–20], although always in the context of serial calculations. Here, we use null events to solve diffusion-reaction master equations in parallel. In our algorithm the computational cell is divided into K arbitrary subdomains Ω_k (where, for consistency with the parallel computing literature, K is the number of *processing units*, PE's). A parallel kMC step consists of the following:

(1) A frequency line is constructed for each Ω_k as the aggregate of the individual rates, r_{ik}, of all the walkers located within each subdomain k:

$$R_k = \sum_i^{n_k} r_{ik}$$

where n_k and R_k are, respectively, the number of walkers and the total rate in subdomain k, $R_{\text{tot}} = \sum_k^K R_k$, and $N = \sum_k^K n_k$.

(2) The maximum rate, $R_{\max}$ is defined as:

$$R_{\max} = \max_{k=1,\ldots,K}\{R_k\}$$

(3) We assign a *null* event with rate r_{k0} to each frequency line in each subdomain k such that:

$$r_{k0} = R_{\max} - R_k$$

where, in general, the r_{k0} will all be different. From this, it follows that:

$$\exists\ \Omega_\alpha, \quad \alpha \in \{k\}, \quad | R_\alpha \equiv R_{\max} \Rightarrow r_{\alpha 0} = 0,$$

i.e., there is at least one domain where there are no null events.

(4) In each Ω_k an event is carried out with probability $p_{ik} = r_{ik}/R_{\max}$, including null events chosen with $p_{k0} = r_{k0}/R_{\max}$. For this step, we must ensure that independent sequences of random numbers are produced for each K, using an appropriate parallel random number generator.

(5) Communicate interdomain processes and boundary events. If a walker leaves its subdomain of origin during a diffusive event, it is transferred to the corresponding Ω_k. In continuum systems with unbounded jump length (see Sections 3.1 and 3.2) this requires a global communication, as all processors need to be ready to receive any diffusing particle from any given Ω_k. In systems with finite range, *e.g.* lattice-based diffusion, local rules may suffice to transfer information from one domain to the next, such as neighbor lists or 'ghost' regions (cf. Ref. [13]). For systems with interacting particles, boundary conflicts are subsequently checked for and the corresponding actions to resolve them, if any, taken (cf. Section 2.3).[1]

(6) As in standard BKL, a simple time increment is sampled from an exponential distribution:

$$\delta t_{\text{p}} = -\frac{\ln \xi}{R_{\max}}$$

where $\xi \in (0, 1)$ is a suitable random number. By virtue of Poisson statistics, δt_{p} becomes the global time step for all of the parallel processes.

In Appendix A we prove that, if boundary conflicts (Section 2.3) are treated correctly, this algorithm solves the same master equation as the serial case.

[1] Similarly, if a particle does not leave its subdomain of origin during a diffusion event but falls within a distance from a domain boundary that is less or equal than its capture radius, its presence is communicated locally to the neighboring processor. After communication, appropriate action is taken to take care of particle interactions or boundary conflicts, if any. Again, a ghost or skin region is a very useful solution when the range of the walkers is known and short-ranged. As we shall see, in our case, for walkers with fixed jump rates (*i.e.* where *a priori* knowledge of the local environment is not needed) and unbounded (*i.e.* technically infinite) diffusion distance, there is no need for such an approach.

2.2. Parallel efficiency

The length conservation of the frequency lines in all Ω_k guarantees exact synchronicity. This is a key aspect of our algorithm: time is advanced exactly the same amount in all processors, which enables direct communication across domain boundaries trivially, eliminating the need for sophisticated rejection minimization schemes across boundaries commonly found in other parallel methods [10–14]. However, note that, because in principle the spatial decomposition may be arbitrary (*i.e.* it does not affect the global kinetics), optimal efficiency is not guaranteed *per se*. Evidently, an optimum decomposition will be that which renders $\{\sum_k r_{k0}\}$ minimum, but the solution is not unique and the catalog of options is quite varied. For example, in his proposed improvement of the original minimal process method, ben-Avraham chose a cell-coarsening scheme that preserves the half-life of the particle concentration [18]. Our method of choice is to perform a domain decomposition using the method of orthogonal recursive bisection (ORB) [21] so as to equally subdivide the aggregate of all the rates in each subdomain after each recursive partition. In the ideal limit of numbers of walkers that are an exact multiple of K, with equal rates, such decomposition yields optimum gain by producing $\{r_{k0} = 0, \forall k\}$. The deviation from this optimum behavior can be measured by the *utilization ratio* (UR), defined as the fraction of 'real' – rather than 'null' – events in the entire system:

$$\mathrm{UR} = 1 - \frac{1}{KR_{\max}} \sum_k r_{k0} \tag{1}$$

The intrinsic time step gain of the method is governed by this utilization ratio. A domain decomposition (or any other distributed decomposition) that prescribes $\{r_{k0} = 0, \forall\ k\}$ will yield ideal gain (UR = 100%). Under these conditions, $R_{\max} = R_{\mathrm{tot}}/K$ and, hence, on average, $\delta t_{\mathrm{p}} = K\delta t_{\mathrm{BKL}}$. UR = 100% is the theoretical efficiency limit and acts as an upper bound to the time step gain. Of course, generally, for discrete systems with varying rates r_i, UR $\leqslant$ 100%, $R_{\max} \geqslant R_{\mathrm{tot}}/K$, and $\delta t_{\mathrm{p}} \leqslant K\delta t_{\mathrm{BKL}}$. The general relation between δt_{p} and K, which represents the *true* time step gain with respect to an equivalent serial BKL simulation, is therefore:

$$\delta t_{\mathrm{p}} = K \cdot \mathrm{UR} \cdot \delta t_{\mathrm{BKL}} \tag{2}$$

As interdomain migration occurs, the $\{r_{k0}\}$ must be recomputed to continue ensuring synchronicity. However, the communication of $R_{\max}$ to all processors is needed only if, after a given pkMC step, the sum $\sum_i r_{ki}$ in any subdomain is greater than the current $R_{\max}$. Thus, the sixth step of our algorithm can be expressed as follows:

(7) if $(\sum_i r_{ki} > R_{\max})$, communicate $R_{\max}$ globally

However, even when this condition is not satisfied, the UR may drop if $R_{\max}$ is not updated regularly (*e.g.* if the r_{k0} are preponderant in the frequency line), leading to inefficient simulations. Depending on the problem at hand, an optimum balance between updating and communicating $R_{\max}$ can be achieved to ensure maximum efficiency. Communicating $R_{\max}$ only when condition (7) is satisfied presents some technical difficulties when it comes to the practical implementation of the algorithm. Indeed, we have not been able to eliminate the use of global calls to check (7), which limits the total efficiency of our method (see Appendix B). Nevertheless, this pertains to the technical aspects of the implementation, which we separate from the formulation of the method.

Steps (1)–(7) above provide a synchronous, parallel algorithm in closed form. So far, no numerical arguments have been made as regards the computational efficiency of the method. However, in the event that the time evolution of the density profile results in a spatial redistribution of particles that deviates from the original optimum decomposition, the utilization ratio may drop below what may be considered an acceptable parallel performance. The metrics chosen to establish reasonable tolerance limits on UR are typically problem-dependent (*e.g.* diffusion coefficients, cell sizes, etc.). In general, when this occurs, the domain decomposition must be updated, either by performing some type of dynamic load balancing, or by generating a new decomposition (such as a global ORB). Irrespective of the method chosen, this process can be integrated between steps (7) and (1) of our kMC algorithm:

if (UR < TOL), then update domain decomposition (perform ORB)

where TOL is a problem-dependent tolerance. It is worth stressing that this is an optional modification that does not detract from the generality of our algorithm, since correct kinetics (see the following section) arise independent of the domain partition scheme chosen. Refreshing the domain decomposition is aimed simply at optimizing the intrinsic efficiency, although, as we shall discuss below, it can also help in mitigating the effect of boundary conflicts.

2.3. Analysis of boundary conflicts

Owing to the intrinsic asynchronicity of discrete-event kinetics, boundary errors due to subdomain interactions are bound to occur in parallel kMC simulations of any kind. Consider for example the Ising system, where the flip probability of each particle depends on its local spin distribution. In this case, no two neighboring particles can flip their spins at the same time, as this would result in a causality error that could lead to the wrong kinetic evolution. Another potential source of conflicts in parallel is the violation of the lattice 'exclusion' principle. A case in point is solute diffusion via a vacancy mechanism, where only one solute atom can jump into a vacant site at a given instant. These processes are trivially modeled using a serial kMC method such as BKL, for such conflicts never arise. Methods to treat them have also been devised within the framework of asynchronous parallel algorithms, as seen in Section 1, *e.g.* roll-back schemes. Described in this fashion, these conflicts are mostly pertinent to discrete system kinetics, where particles can only occupy specified sites – or small deviations thereof – such as in lattice kMC or other lattice-based methods. In continuum media, the definition of conflict is somewhat blurrier, as no two particles ever occupy the same location. In this case, we may consider that a conflict has occurred whenever two particles move concurrently within d_c of one another, where d_c is a suitable interaction distance. In this sense, it is worth remarking that one of the most salient advantages of our method with respect to parallel asynchronous algorithms is that conflicts only arise when co-occurring particles interact. In other words, the reaction between a particle jumping across a domain boundary and an inactive one in the neighboring subdomain is trivial using our algorithm. Not so for asynchronous methods, where such instances must be considered carefully to avoid causality errors.

Here we restrict ourselves to the study of diffusion-reaction systems in continuum media, and we leave the treatment of conflicts in discrete systems for future studies. In any case, the solution of these conflicts determines the 'correctness' of the method, *i.e.* whether a parallel method can rigorously simulate the kinetics of a given problem as obtained with a serial simulation.[2] In our case, the event histories generated in each Ω_k are independent of each other as long as events in one subdomain do not affect events in others. When this occurs, interactions, *e.g.* as when two particles A and B moving concurrently during the same pkMC iteration within a distance d_c of each other,[3] may give rise to boundary conflicts. Fig. 1 shows schematically such an instance, with A and B following random trajectories to their final positions A′ and B′ within the same time step. Formally, a conflict occurs whenever the probability of interaction between two particles in parallel, $P_p(I)$, randomly selected in different domains, differs from the equivalent serial probability, $P_s(I)$.

This *equivalency* condition is simply that the number of kMC events required in either case to arrive at the same final configuration be the same. For example, if a conflict involves m particles from as many subdomains, the condition for correctness is that $P_p(I)$ and $P_s(I)$ be equal after m events (m kMC steps) have been considered. For the simplified case shown in Fig. 1 involving two particles, the interaction probability in serial is obtained from all the different possibilities:

$$P_s(I) = P_1(\mathrm{A}) + P_1(\mathrm{B}) + P_2(\mathrm{AA}) + P_2(\mathrm{BB}) + P_2(\mathrm{AB}) + P_2(\mathrm{BA}) \tag{3}$$

where $P_1(\mathrm{A})$ and $P_1(\mathrm{B})$ are the probabilities for the interaction to occur in just one move by either A or B; $P_2(\mathrm{AA})$ and $P_2(\mathrm{BB})$ are the probabilities for the interaction to occur by two consecutive moves by either A or B; $P_2(\mathrm{AB})$ is the probability for the interaction to occur by the sequential combination of an A move and a B move (or, if viceversa, $P_2(\mathrm{BA})$). Each one of these probabilities includes two distinct contributions,

[2] It is worth noting that the benchmark serial simulation may itself be an approximation, as is the case of BKL for continuum media.

[3] $d_c = d_A + d_B$, where d_A and d_B are the interaction radii of two particles A and B.

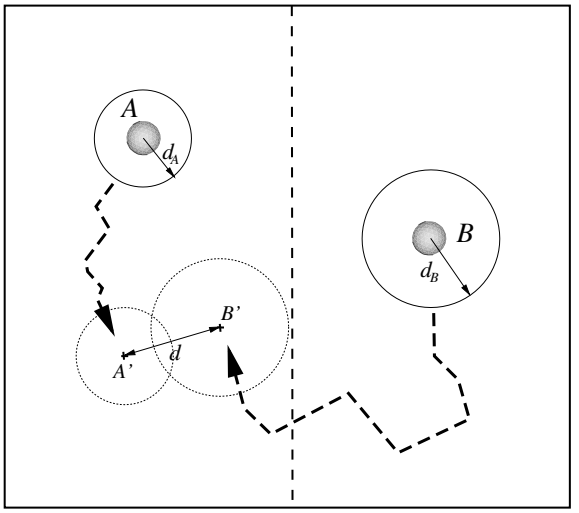

Fig. 1. Schematic diagram of a conflict in parallel kMC. Two particles A and B located on opposite sides of a processor boundary (dashed line) are randomly chosen at the same time to move within the capture radius of one another: $d < d_c = d_A + d_B$.

for example $P_1(\mathrm{A}) = \frac{r_A}{R_{tot}} p_{s1}$, where $\frac{r_A}{R_{tot}}$ is the probability that particle A is selected during a Monte Carlo step, and p_{s1} is the probability that A end up within a distance d_c from particle B in a single jump. Bearing this in mind, and assuming that the particle rates r_A and r_B are time and position independent (*i.e.* $P_2(\mathrm{AB}) = P_2(\mathrm{BA})$), the detailed expression for $P_s(I)$ is:

$$P_s(I) = \frac{r_A}{R_{tot}} p_{s1} + \frac{r_B}{R_{tot}} p_{s1} + \frac{r_A^2}{R_{tot}^2} p_{s2} + \frac{r_B^2}{R_{tot}^2} p_{s2} + 2\frac{r_A r_B}{R_{tot}^2} p_{s2} \tag{4}$$

In parallel, the interaction probability is given by:

$$P_p(I) = P(\mathrm{A})\overline{P(\mathrm{B})} + P(\mathrm{B})\overline{P(\mathrm{A})} + P(\mathrm{AB}) \tag{5}$$

where the first term represents the interaction probability when A is chosen and B is not; the second term is the complementary of the first; and $P(\mathrm{AB})$ represents the probability that both particles react after concurrent jumps, akin to the situation illustrated in Fig. 1. Therefore:

$$P_p(I) = \frac{r_A}{R_{max}}\left[1 - \frac{r_B}{R_{max}}\right] p_{p1} + \frac{r_B}{R_{max}}\left[1 - \frac{r_A}{R_{max}}\right] p_{p1} + \frac{r_A r_B}{R_{max}^2} p_{p2} \tag{6}$$

where, as above, p_{p1} and p_{p2} are, respectively, the probabilities to react in a single jump by either particle, or in concurrent jumps by both particles. Strictly speaking, the jump probabilities p_{s1}, p_{s2}, p_{p1}, and p_{p2}, are overlap integrals of the probability distribution functions solution to the diffusion equation (see Section 3) subject to the condition $d \leqslant d_A + d_B$. For simplicity, however, we have computed all the jump probabilities numerically using specifically-tailored Monte Carlo simulations for the simplified scheme shown in Fig. 1. In Fig. 2 we plot the error between the serial and parallel calculations, defined as $(P_s(I) - P_p(I))$, as a function of the separation distance between the interacting particles. For this simplified case, $r_A = r_B = r_i$, $R_{tot} = 2R_{max}$, and the interaction probabilities in each case are:

$$P_s(I) = \frac{r_i}{R_{max}}\left[p_{s1} + \frac{r_i}{R_{max}} p_{s2}\right] \tag{7}$$

$$P_p(I) = \frac{r_i^2}{R_{max}^2}\left[p_{p2} - 2p_{p1}\right] + \frac{2r_i}{R_{max}} p_{p1} \tag{8}$$

$$\mathrm{Error} = P_s(I) - P_p(I) = \frac{r_i^2}{R_{max}^2}\left[p_{s2} - p_{p2} - 2p_{p1}\right] + \frac{r_i}{R_{max}}\left[p_{s1} - 2p_{p1}\right] \tag{9}$$

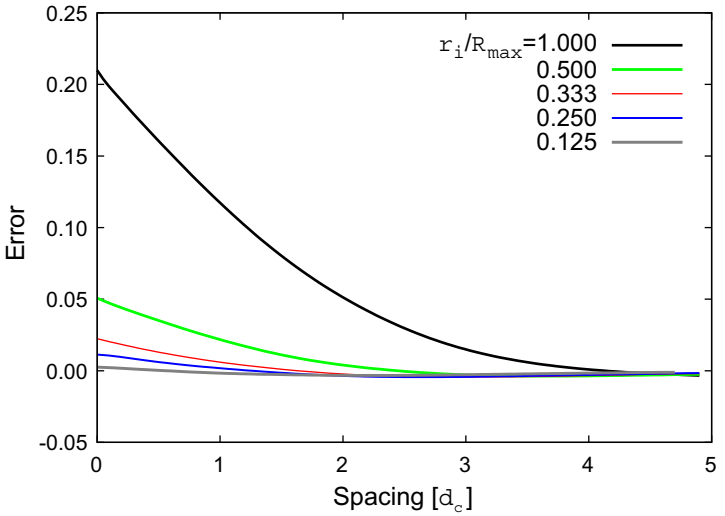

Fig. 2. Interaction probability error between a parallel calculation involving two particles located in different domains, and an equivalent serial calculation. Results are given as a function of the separation distance between the two interacting particles (in units of d_c, the interaction distance) for a family of curves for different r_i/R_{max} ratios (in legend, although several more have been obtained, not shown for clarity). The error converges rapidly to zero as particles become separated beyond $d \gtrsim 3d_c$, except for the worst case scenario of one particle per domain ($r_i/R_{max} = 1$), which converges for $d \gtrsim 4d_c$.

Results in Fig. 2 are given for a family of curves with different $\frac{r_i}{R_{max}}$ ratios (in legend). One can readily see that for large numbers of particles per PE (small $\frac{r_i}{R_{max}}$ ratios) the error rapidly converges to zero. We have also calculated the error varying the value of R_{max} and have seen that convergence is accelerated. These results suggest that conflicts are indeed important only in cases with few particles per processor, in close proximity to one another, which is precisely where our method is expected to underperform. For large systems with typical particle densities (*e.g.* in the examples discussed in Section 3, $\frac{r_i}{R_{max}} \sim 10^{-3}$–$10^{-4}$), neglecting conflicts results in very small errors, as will be showcased in Section 3.2, where our pkMC simulations provide a very accurate kinetic evolution disregarding the treatment of conflicts. It is worth mentioning that a fine-tuned ORB can also help drive the system towards the ideal conditions for the application of our method, namely large numbers of particles per PE, in large spatial domains. Evidently, under such conditions, spatial locality is achieved, and events within one subdomain can be considered independent from events in other Ω_k. Conversely, when the particle separation distance (or the number of walkers per processor) is low, causal dependencies develop among events, which, if not treated properly, can lead to a flawed kinetic evolution. This effects have been noted by Merrick and Fichthorn in their study of thin film growth, where boundary shifting was seen to play an important role in the efficiency and accuracy of the calculations [14].

Fig. 2 can be taken as the basis from which to redefine $P_p(I)$ so that the interaction error is zero. This is a necessary condition to make our method *rigorous* from a mathematical standpoint. All the error curves in the Figure are seen to follow the same qualitative trends (rapid decay), which could be used to fit a general error function from which to extrapolate to correct $P_p(I)$ during a simulation. Another possibility is to compute the p_s and p_p analytically, and bias p_p so as to comply with the requirement of zero error. Whichever the case, at present we simply outline the conditions under which our pkMC simulations reach satisfactory levels of accuracy depending upon the problem under study, with no explicit treatment of these issues. However, we want to emphasize that, although not rigorous, these features make our method a controlled approximation, defined as one where (i) the error can be calculated intrinsically, *i.e.* during the course of a simulation; and (ii) this error can be decreased arbitrarily in another simulation of the same type by modifying a few problem-dependent parameters (*e.g.* in Fig. 2 the number of processors and the domain decomposition chosen) that can be identified *a priori*.

3. Applications

3.1. Diffusion of independent particles

We have studied diffusion of independent particles as a simple test of the basic ideas, as a validation of the time-scaling of the method, and as a first vehicle for assessing the parallel efficiency. Pure diffusion without volumetric terms satisfies the following master equation:

$$\left[-D\nabla^2 + \frac{\partial}{\partial t}\right]\rho(\mathbf{x};t) = 0 \tag{10}$$

where $\rho(\mathbf{x};t)$ is the time and space-dependent particle number density and D is the diffusion coefficient. In this case the method is exact, as there is no possibility of conflicts. We consider two cases with known analytical solution involving diffusion in an n-dimensional square domain of side a, Ω_a, subject to the following boundary conditions:

(i) Absorbing ('black box') boundary conditions:

$$\rho(\mathbf{x};0) = \rho_0 \prod_{\alpha=1}^{n} \cos\left(\frac{\pi x_\alpha}{a}\right), \quad \mathbf{x} \in \Omega_a$$
$$\rho(\mathbf{x};t) = 0, \quad \mathbf{x} \in \partial\Omega_a$$

where ρ_0 is a constant.

(ii) Periodic boundary conditions (PBC):

$$\rho(\mathbf{x};0) = \rho_0 \prod_{\alpha=1}^{n} \left[\frac{1}{a} - \cos\left(\frac{2\pi x_\alpha}{a}\right)\right], \quad \mathbf{x} \in \Omega_a$$
$$\rho(x_1, x_2, \ldots, x_\alpha, \ldots, x_n; t) = \rho(x_1, x_2, \ldots, x_\alpha \pm a, \ldots, x_n; t), \quad \forall\alpha$$

Here we focus on the two-dimensional (2D) case. In both cases (i) and (ii) the solution of the diffusion equation is given by the time dependent Green's function for an infinite medium with diffusion constant D:

$$G(\mathbf{x}, \mathbf{x}_0; t) = \frac{\mathrm{e}^{-\frac{(\mathbf{x}-\mathbf{x}_0)^2}{2\sigma^2}}}{\sqrt{4\pi D t}} \tag{11}$$

where $\mathbf{x}$ and $\mathbf{x}_0$ are the initial and final position of each walker, t is the time, and $\sigma^2 = 2Dt$ is the mean square displacement. For a fixed D, the mean square displacement must be conserved in all Ω_k, from which it follows that, for each walker i, $t_i = \delta t_{\mathrm{p}} R_{\max}/r_i$. Both of these cases are eigenvalue problems [22] with known eigenvalues of (i) $\lambda_{\mathrm{abs}} = \frac{\pi}{a}\sqrt{2D}$; and (ii) $\lambda_{\mathrm{PBC}} = \frac{\pi}{a}\sqrt{4D}$. Fig. 3 shows the comparison between the analytical solution and our parallel algorithm for case (i) with $a = 1$ cm, $D_i = 1$ cm^2 s^{-1}, and $\rho_0 = 131,072$ walkers. For these values, $\lambda_{\mathrm{abs}} = 4.443\ \mathrm{s}^{-\frac{1}{2}}$, while in a series of runs with $K = 2^n$ ($n = 1, \ldots, 7$) processors we obtained an average value of $4.410 \pm 0.042\ \mathrm{s}^{-\frac{1}{2}}$. For case (ii) with the same parameters, we show in Fig. 4 the time evolution with eigenvalue $\lambda_{\mathrm{PBC}} = 6.28\ \mathrm{s}^{-\frac{1}{2}}$ of the spatial particle distribution (ii). The plot contains the projection of $\rho(\mathbf{x};t)$ on one of the box dimensions at four different times. Note that, for $a = 1$ cm and $D = 1$ cm^2 s^{-1}, the exponent of the time dependent terms is $\sim$40 so that the convergence to ρ_0 is very fast. We obtain $\langle\lambda_{\mathrm{PBC}}\rangle = 6.27$ and an average error of $\pm 4.4\%$ with respect to the analytical value of $6.28\ \mathrm{s}^{-\frac{1}{2}}$. For all other parallel runs performed the values were of the same order of magnitude. In this particular case (ii) the initial particle density evolves with time towards a more flattened profile. Thus, the utilization ratio derived from the initial ORB will gradually worsen as walkers diffuse and the mapping between the spatial particle distribution and the initial domain decomposition degrades. For the specific simulation shown in Fig. 4, the UR decreases from its initial value of 97.2% to a steady state value of $\sim$77.5% after homogenization has completed. Fig. 5 illustrates

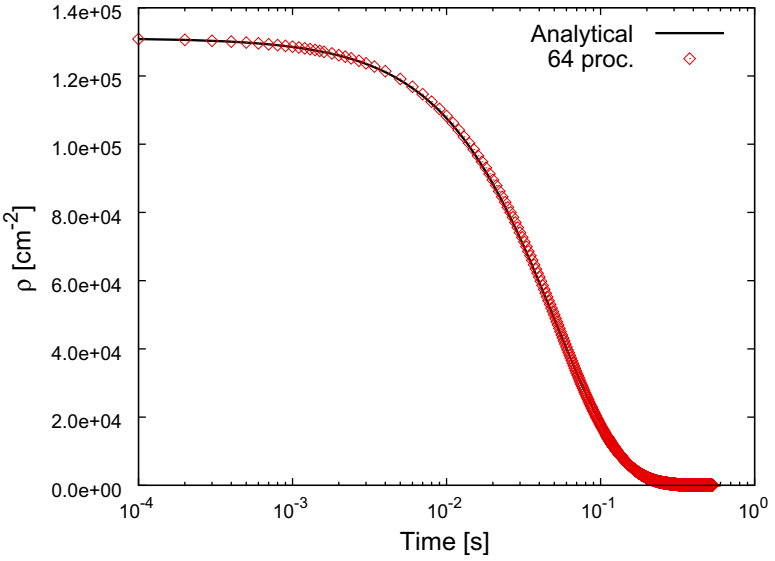

Fig. 3. Comparison between the analytical solution (continuous line) and a parallel run with 64 processors (open diamonds) for the problem of simple diffusion with no particle interactions and absorbing boundaries.

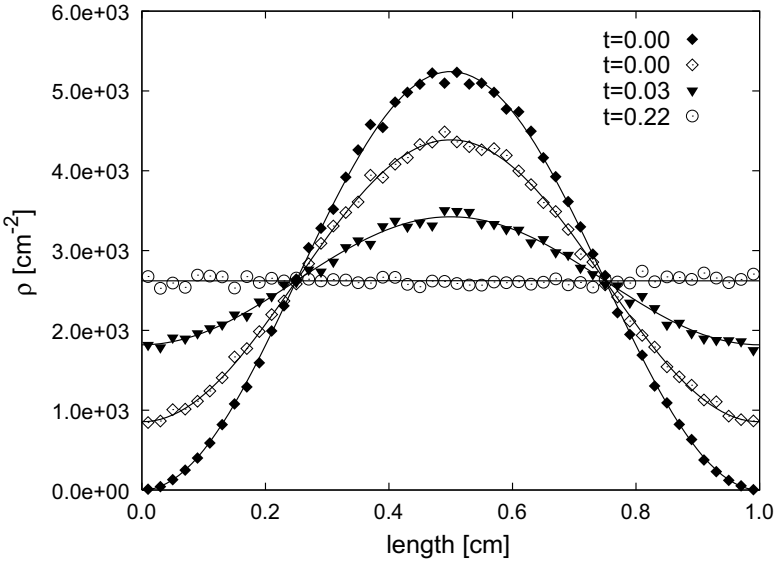

Fig. 4. Comparison between pkMC for 16 processors and the analytical solution of the time evolution of the spatial density profile for the case (ii) of diffusion without particle interactions and periodic boundary conditions.

the temporal variation of the UR for this case, compared with the case of a flat particle density profile using the same number of processors. While both cases start out at UR $\approx$ 97%, the domain decomposition that maps the initial sinusoidal particle distribution in (ii) becomes gradually unsatisfactory, resulting in a steady-state UR of $\sim$77.5%, compared to a value of 96.8% for the flat density profile. Although these results, which are perfectly satisfactory, have been obtained for a single ORB, as noted above, nothing precludes carrying out subsequent ORBs to improve the efficiency when the value of UR drops below some problem-specific (arbitrary) tolerance.

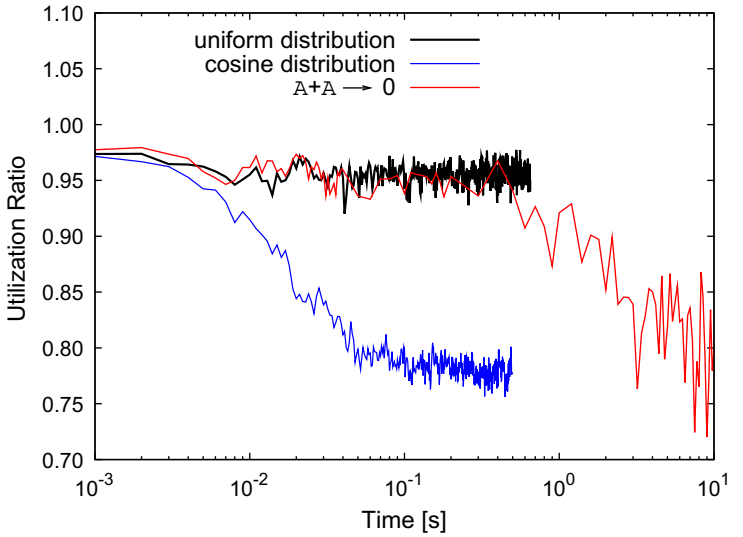

Fig. 5. Time evolution of the utilization ratio (UR) for three different 32-processor cases with periodic boundary conditions: (i) homogeneous particle distribution with no interactions; (ii) cosinusoidal particle distribution with no interactions; (iii) homogeneous distribution with $A + A \rightarrow 0$ particle annihilations.

3.2. Diffusion with particle interactions

Next we turn to the study of cases where particles interact. In particular, we consider the multiparticle reaction $NA \rightarrow 0$ (where N is the number of reacting particles) and the two-species annihilation $A + B \rightarrow 0$. In both cases, boundary conflicts as defined in Section 2.3 may arise, although the chosen simulation conditions are such that the errors, as given by Fig. 2, are almost negligible. Fig. 6 shows the time evolution in 2D of an ensemble of 32,768 type walkers with $a = 1$ cm, $D = 1$ cm^2 s^{-1}, and d_A, the particle interaction radius, equal to 10^{-5} cm. For an arbitrary value of N the appropriate asymptotic decay is $t^{-\frac{1}{N-1}}$ [23]. Here we have chosen $d_c = 2d_A$ small enough to minimize the number of interactions for which $N > 2$, *i.e.* $\rho(t)$ is expected to scale approximately as $1/t$, which is equivalent to the biparticle ($A + A \rightarrow 0$) annihilation time decay. The figure

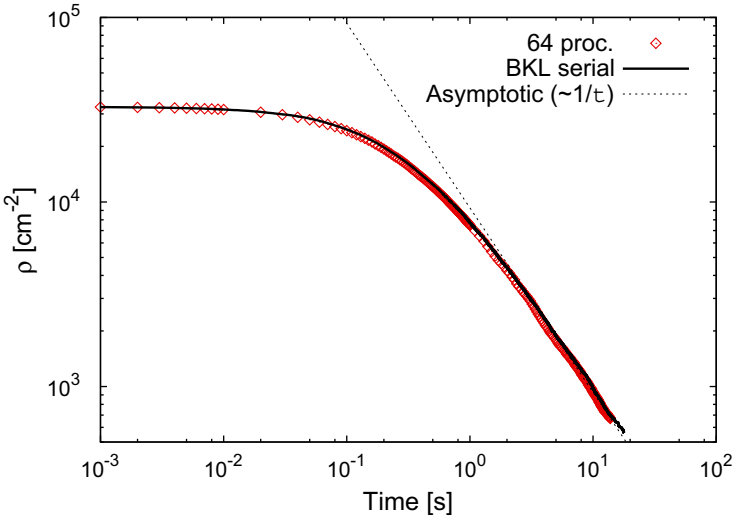

Fig. 6. Comparison between a serial BKL run (continuous line) and a parallel run with 64 processors (open diamonds) for the problem of multiparticle $NA \rightarrow 0$ annihilation with periodic boundary conditions. The asymptotic behavior $\sim 1/t$ expected for this reaction is also shown.

shows results for 64 processors and a single-CPU BKL run, with excellent agreement between both calculations. Also shown is the $1/t$ asymptotic trend characteristic of the $A + A \rightarrow 0$ reaction. The time evolution of the UR in this particular case varies with the number of processors K, ranging from 98% to 89% for $K = 4$, and from 97% to about 70% for $K = 32$ (shown in Fig. 5).

The two-species reaction $A + B \rightarrow 0$ is important in many physical and chemical processes, and has been studied in detail in the literature (*e.g.* Refs. [23,24]). In principle, the kinetics of a random homogeneous bimolecular system with cross-annihilations and equal initial populations $\rho_A(0) = \rho_B(0)$ is governed by two parameters, namely, the capture radius d_c, and the typical diffusion length, $\ell = \sqrt{4D\delta t}$. The relative values of ℓ and d_c give rise to two well-differentiated regimes. In the so-called reaction-limited regime (RLR), $\ell \gg d_c$, and the system obeys an asymptotic decay law of the type $1/kt$, where k is a rate constant. However, in the diffusion-limited regime (DLR), $\ell \lesssim d_c$, spatial fluctuations asymptotically result in the separation of A and B particles into A and B-rich domains. In this case, the kinetics is considerably decelerated and the system evolves as $t^{-\frac{1}{2}}$. Fig. 7 shows pkMC calculations for both the reaction- and the diffusion-limited regimes and their corresponding asymptotic decay laws. For the DLR case we have used $\ell = 5.0 \times 10^{-4}$ and $d_c = d_A + d_B = 2.0 \times 10^{-2}$ cm, whereas, for RLR, we used values for ℓ and d_c of 10^{-2} and 2.0×10^{-5} cm respectively. It is quite clear from the figure that the parallel kMC calculations capture the correct asymptotic kinetics in each case. To further analyze the separation kinetics (or lack thereof) in the RLR and DLR, we show in Fig. 8 the A–B pair correlation function, $g_{AB}(r)$, for both cases.[4] $g_{AB}(r)$ measures the probability of finding a B-type particle from an A particle, averaged over the entire simulation area. These probabilities are given relative to the overall background particle density ($\langle\rho\rangle$) in each case, *i.e.* a probability higher than unity at a distance r simply means that, at that distance, the pair density of particles is higher than $\langle\rho_B\rangle$. In the DLR, where particles separate into A and B-rich domains, $g_{AB}(r)$ is initially very low, corresponding to a B-depleted, A-type domain. As the distance is increased, the pair correlation function gradually reaches its background value of 1.0. On the contrary, in the RLR, where homogenization is expected, $g_{AB}(r)$ resembles the pair distribution for an ideal gas. Different amounts of roughness can be appreciated in both curves, presumably indicating short and medium range order. In summary, our parallel calculations satisfactorily capture the time and spatial correlations of a particle population subject to the $A + B \rightarrow 0$ kinetics.

In both of these cases, the simulated kinetics follows the expected asymptotic behavior due to the scarcity of boundary conflicts of the type specified in Section 2.3. In general, one can obtain a first-order estimate of the error incurred by neglecting boundary conflicts by entering Fig. 2 with a characteristic particle separation distance and a r_i/R_{max} ratio. Particularly, for the multiparticle annihilation case ($NA \rightarrow 0$) above, with *a priori* homogeneous particle distributions, the average inter-particle separation is $d \sim \rho^{-\frac{1}{2}} = 5.5 \times 10^{-3}$ cm, which in units of interaction radius ($d_c = 2.0 \times 10^{-5}$ cm is $d \approx 275 d_c$. For 64 processors, the initial ratio $r_i/R_{max} \approx 0.002$. Entering the corresponding error curve with these two values yields an error of 1.3×10^{-8} (virtually zero). Indeed, for the simulation shown in Fig. 6, we have explicitly counted the number of times a conflict such as that depicted in Fig. 1 takes place and have found zero occurrences, in agreement with the computed value. As the simulation proceeds and the particle concentration diminishes, the kinetics is characterized by increasing separation distances and r_i/R_{max} ratios, which have opposite effects on the total error. In all the cases considered in this work we did not observe a significant deviation of the error with respect to the starting estimation as given by the initial d and r_i/R_{max}. The dynamic behavior of the error, governed by the interplay between these two parameters, will nonetheless vary for each physical problem. Here, we take these results as a partial validation of the analyses performed in Section 2.3.

3.3. Discussion on the generality of the method

The time scale intrinsic to any physical process governed by diffusion is $\tau \approx l^2/D$, where l is a characteristic length scale. As explained in the previous Section, in a 2D system with a homogenous particle density ρ, $l \sim \rho^{-\frac{1}{2}}$, which gives $\tau \approx (\rho D)^{-1}$. In rejection-free kMC algorithms such as BKL, this time scale acts as an upper bound on the value of δt that can be simulated. The time scale inherent to a physical process is of course

[4] Here r is a generic *radial* distance, not to be confused with the rates of the diffusing species r_i, r_{ik}.

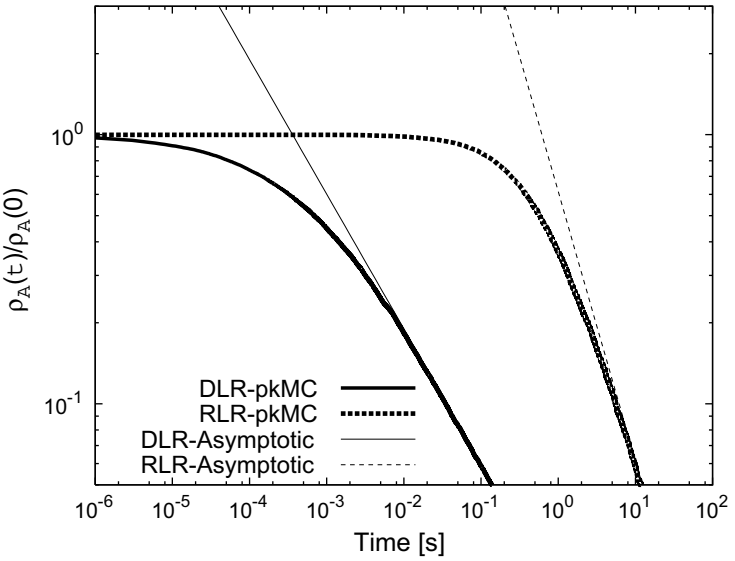

Fig. 7. Two-species annihilation kinetics (only the A-type normalized density is shown) for the reaction ($\ell \gg d_c$) and diffusion ($\ell \lesssim d_c$) limited regimes as obtained with our parallel kMC algorithm. The expected asymptotic decay law in each case is also shown for reference.

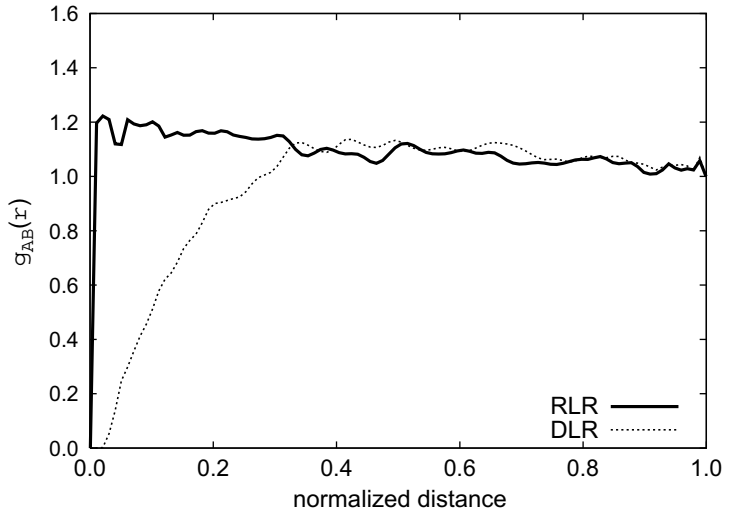

Fig. 8. A − B pair correlation function for the diffusion (DLR) and reaction (RLR) limited regimes. The function measures the probability to find a B particle from an A-type particle, averaged over the entire simulation cell. The particle distance is normalized to half of the box length.

independent of the method employed to simulate it, which means that, in cases where τ may be relatively small, *e.g.* in concentrated systems, or in systems with particles with a large interaction radius, there exists an effective cap on the time step gain that our algorithm can provide. In other words, one must impose $\delta t_p \leqslant \tau$ to ensure the correct kinetics in the 'physical' sense. This inequality must be satisfied dynamically over the course of a pkMC simulation, which can be readily attained *e.g.* by enforcing:

$$R_{\max} \geqslant \frac{1}{\tau} \tag{12}$$

In practice, this requires that $R_{\max}$ be adjusted by increasing the value of the $\{r_{k0}\}$ as much as necessary. Evidently, in cases where δt is reasonably close to τ from the outset, this results in a loss of efficiency (given by Eq.

(1)) that may limit the usefulness of our algorithm. Furthermore, in Green's functions Monte Carlo [25], such as in the present work for continuum systems, enforcing Eq. (12) does not necessarily guarantee that $\delta r = |\mathbf{r} - \mathbf{r}_0| < l$, as there exists a small probability, from sampling Eq. (11), that violates this premise. This is a limitation intrinsic to BKL that carries over to our method, which, nonetheless, by setting an upper limit on δt_p, gives a more accurate kinetic evolution. In discrete systems, where l is quantized, this is no longer a concern, and Eq. (12) suffices to give the exact kinetics (provided, of course, that boundary conflicts are correctly resolved). Additionally, this measure is necessary to treat problems where the time scales associated with different events differ considerably, as is the case in the thin film growth problem studied by several researchers [13,14], characterized by monomer deposition and diffusion rate relative ratios of the order of $10^{-3} - 10^{-7}$, or in irradiation damage accumulation, where vacancies and interstitials diffuse with rates that differ by three to four orders of magnitude [26].

We acknowledge this limitation in our method, which effectively sets a limit on the maximum time step gain that can be achieved. Note, however, that, subject to the qualification relating to boundary conflicts (here very small), our method provides very accurate results. Nevertheless, as with other parallel methods published in the literature, a careful analysis of the timescale characteristics of each kinetic problem is advised prior to the use of our algorithm.

4. Performance analysis

Next we turn to the study of the parallel efficiency of our algorithm as implemented on a distributed-memory Linux cluster with 2.4-GHz AMD Opteron processors with version 1 of the MPI libraries [27]. We define two metrics for our scalability analysis, namely 'weak' and 'strong' scalability. For simplicity, we study these metrics on a PBC system in 2D with a uniform particle distribution with no interactions. The PBC case is an extreme one in the sense that the parallel capabilities are minimally exploited. Recalling that walkers are reassigned to processors after every move, relative to other diffusion problems, the communication-to-calculation ratio is disproportionately high in PBC simulations, where no particle interactions exist and all that a calculation cycle entails is the trivial diffusion of a given walker. Any relative increase in the calculation component of the overall computational cost will result in an improved parallel performance. A natural way to achieve this is to subdivide each Ω_k into several cells – denoted by the subindex j – so as to break the one-to-one correspondence between processors and Ω_k (cf. Refs. [5,15]). In this manner, j calculations are performed per each communication during every pkMC iteration. The results shown below are for $j = 64$, found to be the optimum value of j for the PBC problem.[5] Nevertheless, comparisons between $j = 1, 64$ are provided to give an idea of the relative improvement achieved in each case.

The PBC case used for the scalability studies is special in that it is error-free – there are no particle interactions – and, therefore, there is no need to insert an extra step in the algorithm to tally the number of boundary conflicts that occur during a simulation. The insertion of this step, as it was done in Section 3.2, will affect the efficiency negatively, just as solving for boundary errors would. The definition of our method as a controlled approximation implies that we must be capable of computing the error during the course of a simulation, although the choice of whether to do it or not depends on each problem and the desired compromise between efficiency and accuracy. These consideration must be kept in mind when comparing the efficiency results shown in the next sections with those of other methods, rigorous or not, where conflicts are more actively dealt with.

4.1. Weak scalability

Weak scalability (WS) measures the performance of a parallel algorithm using K processing units when the problem size is increased K-fold. From steps (5) and (7) of our algorithm, it is clear that, although not strictly necessary for this computation, our program incurs a communications overhead when particles that move

[5] The optimum number of Ω_{kj} ascribed to each PE is problem and machine dependent, but the parallel efficiency is expected to worsen again for large values of j due to the extra cost associated with vectorized nested loops.

across domain boundaries are reassigned to the corresponding PE's. As we show in Appendix B, this communications cost shows a dependence with the number of processors of the type (Eq. (B.7)):

$$\mathrm{WS} = 1 + a(\log K)^b \tag{13}$$

The weak scalability is plotted in Fig. 9, which shows a family of curves for three different numbers of walkers per processor (see legend), all for $j = 64$. It is common in the literature to find the inverse of WS, the parallel efficiency η_{ws}, as the metric of choice to study the parallel performance, so in the figure we give the corresponding values of η_{ws} for comparison (in the right-hand axis). Essentially, WS, or, equivalently, η_{ws}, estimates the cost of parallel communications when all other factors are kept invariant. Ideal weak scaling is represented by a horizontal line at $\mathrm{WS} = 1$, and, thus, the deviation from horizontality illustrates the relative parallel performance. As the figure shows, the agreement between Eq. (13) and the data points is excellent. The coefficients a and b of the non-linear fits are given in Fig. 9's legend, and their ranges are $(0.12 \leqslant a \leqslant 0.21)$ and $(1.31 \leqslant b \leqslant 1.45)$, respectively. The latter are on the higher end of those computed by Shim and Amar [13] and Merrick and Fichthorn [14] using synchronous relaxation algorithms with similar efficiency behavior. However, a and b are strongly problem and machine dependent, both of which were different in Refs. [13,14], which diminishes the meaningfulness of the comparison. By way of example, the efficiency for the 2^{16}-particle case is ~86% for $K = 4$, and ~52% for 64 processors. As expected, the method scales better (significantly) when the number of particles per processor is increased. Since our aim is to use large parallel architectures to study large problems, this is the most relevant regime, and weak scalability the more meaningful metric.

In the inset to Fig. 9 we show the comparison between the standard case ($j = 1$) and a case for which each Ω_K is subdivided into 64 cells ($j = 64$), both for 2^{15} particles per processor. As expected, WS is slightly better when j is increased. The curves, especially that for $j = 1$ display marked increases from one value of K to the next. This is because the communications overhead has been shown to correlate directly with the perimeter-to-surface area ratio [28]. The communication-to-calculation ratio in 2D is known to improve as the aspect ratio of the subdomains tends toward perfect quadrature [29]. Of course, a perfectly quadratic decomposition can only be achieved when K is an even power of 2, *e.g.* 4, 16, 64, etc., which explains why the curves in the inset to Fig. 9 display abrupt increases at, for example, $K = 2$ and $K = 8$.

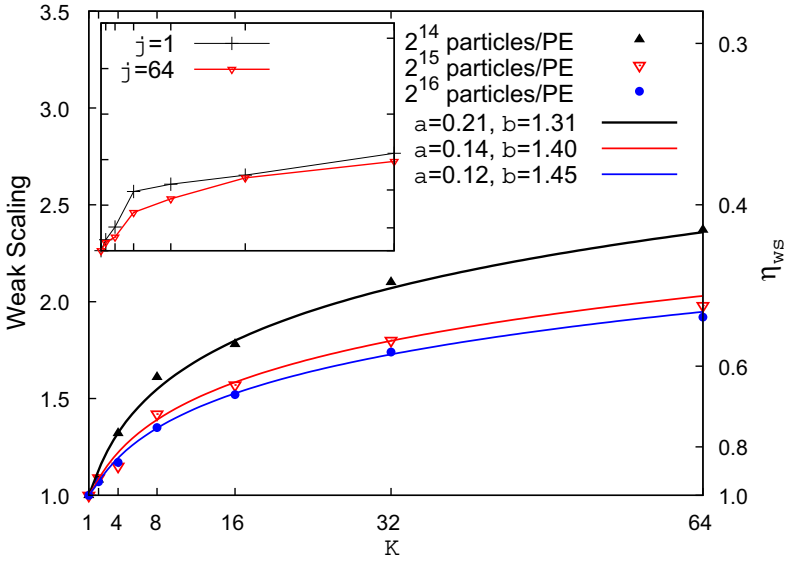

Fig. 9. Weak scalability of our algorithm for PBC cases with different numbers of particles per processor (in legend) with $j = 64$ (see text). The points are the data computed in our runs, whereas the lines are fits using Eq. (13). The coefficients a and b for each case are given in the legend. Inset: comparison between the $j = 1$ and $j = 64$ cases for the 2^{15}-particle problem. The K and WS axes have the same scale as the main graph.

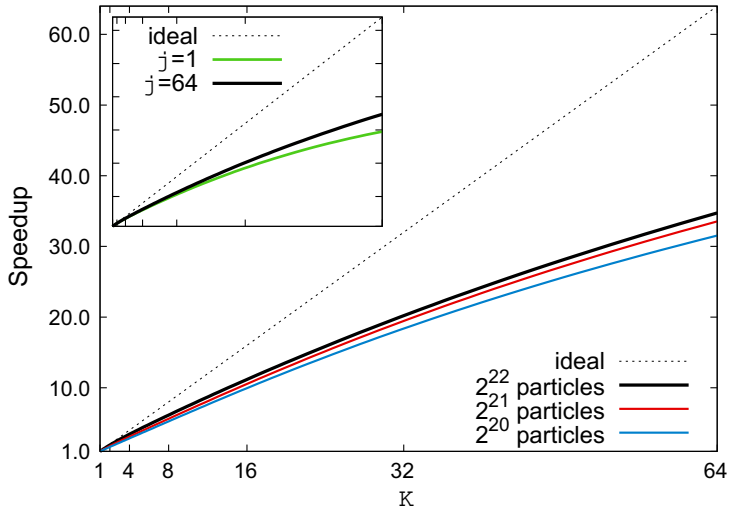

Fig. 10. Strong scalability for our parallel kMC algorithm as implemented in our prototype code with $j = 64$. Three PBC cases are considered with 2^{20}, 2^{21}, and 2^{22} walkers. The curves confirm the tendency outlined in the text that our method behaves better for large problems with many particles. Inset: comparison between the $j = 1$ and $j = 64$ cases for the 2^{22}-particle problem. The K and Speedup axes have the same scale as the main graph.

4.2. Strong scalability

In contrast, strong scalability measures the computational speedup when an increasing number of processors is applied to a problem of fixed size. Results for up to $K = 64$ processors for three PBC cases with, approximately, one, two and four million (2^{20}, 2^{21}, and 2^{22}) walkers are presented in Fig. 10. Results in Fig. 10 have also been obtained using $j = 64$ cells per processor. Results for $j = 1$ and 2^{22} particles are given in the inset to the Figure, where a slightly worse scaling is measured with respect to the $j = 64$ case (also shown in the inset). For the results shown in Fig. 10 it is clear that, as in the previous section, the larger the problem, the better the scalability, although by a modest margin. As above, we have fitted the data shown in the figure to Eq. (B.4):

$$\eta_{\mathrm{st}} = \frac{\mathrm{UR}}{1/K + a(\log K)^b} \tag{14}$$

for which we have obtained coefficients in the range of 0.66–0.92 for a and 0.50–0.68 for b (assuming, from Fig. 5, that $\mathrm{UR} \approx 1$). This translates into efficiencies, η_{st}, of the order of almost 100% for $K = 2$ to $\eta_{\mathrm{st}} \approx 0.50 \sim 0.55$ for 64 processors. In the ($K \leqslant 64$)-range explored, all three curves increase monotonically, albeit concavely, so that the parallel performance is eventually expected to follow Amdahl's Law for problem size bound scaling [30], and gradually saturate.

The PBC results effectively represent a lower-bound estimate of the general scalability behavior of our algorithm, which will presumably be enhanced in problems where the communication-to-calculation ratio is not as unfavorable. Also, as discussed in Section 2.2, the total speedup benefits from two distinct contributions, namely (i) the time step gain derived from decreasing the length of R_{tot}, and (ii) a contribution associated with the ORB decomposition implemented here. As implemented, the computational cost associated with selecting an event out of the frequency line is $\mathcal{O}(N)$.[6] After performing our ORB, each processor must now perform a search with cost $\mathcal{O}(\frac{N}{K})$. In other words, ignoring the overhead, there is a factor of K speedup related simply to

[6] A more efficient way to do this is by using a binary search tree, which carries an associated computational cost that ideally scales as $\log(N)$ [31,32], or more optimized variations thereof [33]. However, the speedup generally achieved with a binary search may be severely curtailed in parallel computations, where the overhead associated with updating the frequency lines after communication among PE's scales no better than $\mathcal{O}(N)$ in any case.

the cost of carrying out smaller search operations in parallel. These considerations are also formulated and analyzed in Appendix B.

5. Summary

In summary, we have developed a novel parallel kinetic Monte Carlo algorithm that promises to access time and length scales as-of-yet unexplored in kMC simulations. Our algorithm is based on a perfectly-synchronous parallel decomposition of the master equation, to which it provides an exact solution for the case of independent walkers. The correct simulation of interacting systems is contingent on the rigorous treatment of boundary conflicts, which we will address in future publications. Regardless, in the limiting case of large problems (large number of particles, many PE's), our algorithm provides fairly accurate solutions for interacting systems. The efficiency of the method is dependant on the characteristics of the problem at hand and the optimization facilities of the decomposition chosen. We have shown the validity and performance of our algorithm in a few well understood diffusion problems, with reasonable scaling and excellent agreement between our computational results and analytical and serial cases. Due to its trivial implementation in parallel architectures, our algorithm suggests itself as a practical alternative to other previously published parallel methods.

Acknowledgement

This work has been performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-ENG-48 and the Laboratory-directed Research and Development Office under Programs 06-ERD-005 and 06-LW-028. The authors gratefully acknowledge Dr. J. Knap for support and help with the implementation.

Appendix A. General proof of correctness: master equation

Here we prove that the synchronous decomposition of a serial master equation into K subdomains results in the same master equation when the null events are introduced. Let us start by considering an infinite system with n particles characterized by an internal state $X(n)$, and represented by a homogeneous medium with periodic boundary conditions. In this ideal system, particles can either diffuse, with rate constant d, or annihilate (two by two) with rate constant a. The total rate of the system R_{tot} is:

$$R_{\text{tot}} = d(X) + a(X) \tag{A.1}$$

Then, the probability that in step $s+1$ the system is in state $X(n)$ is (for simplicity, we will assume that state X is characterized solely by the number of particles n, *i.e.* $X \equiv X(n)$):

$$P(X;s+1) = P(X+2;s)\frac{a(X)}{R_{\text{tot}}} + P(X;s)\frac{d(X)}{R_{\text{tot}}} \tag{A.2}$$

In other words, the probability that the system is in state X at step $s+1$ is the probability that the system is in state $X+2$ at step s times the probability that, in this state, two particles will annihilate $(a(X)/R_{\text{tot}})$, plus the probability that the system is in state X at step s times the probability that, in this state, a particle will diffuse $(d(X)/R_{\text{tot}})$. Making use of the fact that $d(X) = R_{\text{tot}} - a(X)$, we have:

$$P(X;s+1) = P(X+2;s)\frac{a(X)}{R_{\text{tot}}} + P(X;s)\frac{R_{\text{tot}} - a(X)}{R_{\text{tot}}} \tag{A.3}$$

$$(P(X;s+1) - P(X;s))R_{\text{tot}} = (P(X+2;s) - P(X;s))a(X) \tag{A.4}$$

Taking increments and assuming that $\Delta t = 1/R_{\text{tot}}$:

$$\frac{\Delta P(X)}{\Delta t} = (P(X+2;s) - P(X;s))a(X) \tag{A.5}$$

In the limit of infinitesimal Δt, Eq. (A.5) is reduced to:

$$\frac{\partial P(X)}{\partial t} = (P(X+2;s) - P(X;s))a(X) \tag{A.6}$$

This is the master equation that represents our idealized system, where the time step to go from s to $s+1$ is $1/R_{\text{tot}}$. Now let us decompose our initial computational box into K domains, Ω_i, according to our parallel algorithm. In this case, for each subdomain i, we have to add three new transitions, namely the probability that a particle will escape each Ω_i by diffusion, $e(X_i)$, the (complementary) probability that a particle will come from other subdomains, $c(X_i)$, and, to ensure synchronicity, the null event, r_{0i}. In this case, we have that the total rate in each Ω_i is:

$$R_{\max} = d(X_i) + a(X_i) + e(X_i) + c(X_i) + r_{0i} \tag{A.7}$$

Now, the probability that in step $s+1$, in each subdomain i, the system will be in state X_i is:

$$\begin{aligned} P_i(X_i;s+1) = {} & P_i(X_i+2;s)\frac{a(X_i)}{R_{\max}} + P_i(X_i;s)\frac{d(X_i)}{R_{min}} + P_i(X_i+1;s)\frac{e(X_i)}{R_{\max}} + P_i(X_i-1;s)\frac{c(X_i)}{R_{\max}} \\ & + P_i(X_i;s)\frac{r_{0i}}{R_{\max}} \end{aligned} \tag{A.8}$$

Making use of the fact that $r_{0i} = R_{\max} - d(X_i) - a(X_i) - e(X_i) + c(X_i)$, we then have:

$$\begin{aligned} P_i(X_i;s+1)R_{\max} = {} & P_i(X_i+2;s)a(X_i) + P_i(X_i;s)d(X_i) + P_i(X_i+1;s)e(X_i) + P_i(X_i-1;s)c(X_i) \\ & + P_i(X_i;s)(R_{\max} - d(X_i) - a(X_i) - e(X_i) + c(X_i)) \end{aligned} \tag{A.9}$$

which simplifies to:

$$\begin{aligned} (P_i(X_i;s+1) - P_i(X_i;s))R_{\max} = {} & (P_i(X_i+2;s) - P_i(X_i;s))a(X_i) + (P_i(X_i+1;s) - P_i(X_i;s))e(X_i) \\ & + (P_i(X_i-1;s) - P_i(X_i;s))c(X_i) \end{aligned} \tag{A.10}$$

Now, operating as for Eq. (A.5):

$$\begin{aligned} \frac{\Delta P_i(X_i)}{\Delta t_{\text{p}}} = {} & (P_i(X_i+2;s) - P_i(X_i;s))a(X_i) + (P_i(X_i+1;s) - P_i(X_i;s))e(X_i) + (P_i(X_i-1;s) \\ & - P_i(X_i;s))c(X_i) \end{aligned} \tag{A.11}$$

where, in this case, $\Delta t_{\text{p}} = 1/R_{\max}$ is the time needed to go from step s to step $s+1$.

We now sum over all Ω_i:

$$\begin{aligned} \sum_i^K \frac{\Delta P_i(X_i)}{\Delta t_{\text{p}}} = {} & \sum_i^K \{(P_i(X_i+2;s) - P_i(X_i;s))a(X_i) + (P_i(X_i+1;s) - P_i(X_i;s))e(X_i) \\ & + (P_i(X_i-1;s) - P_i(X_i;s))c(X_i)\} \end{aligned} \tag{A.12}$$

Evidently, the sum over all subdomains of the $[(P_i(X_i+1;s) - P_i(X_i;s))e(X_i) + (P_i(X_i-1;s) - P_i(X_i;s))c(X_i)]$ terms must be zero, as all particles coming into any one Ω_i do it after having escaped from other Ω_j. In other words, the detailed particle balance makes these terms vanish. Therefore, the reduced equation is:

$$\frac{\Delta P(X)}{\Delta t_{\text{p}}} = \sum_i^K ((P_i(X_i+2;s) - P_i(X_i;s))a(X_i)) \tag{A.13}$$

Provided that boundary conflicts are solved rigorously, the rates $a(X_i)$ are not affected by the summation, as they are simply a constant acting on the internal variables defining state X_i, *i.e.* $\sum_i^K P_i(X_i)a(X_i) = P(X)a(X)$. Then:

$$\frac{\Delta P(X)}{\Delta t_{\mathrm{p}}} = (P(X+2;s) - P(X;s))a(X) \tag{A.14}$$

where, as in Eq. (A.6), taking infinitesimal values of Δt_{p} we have:

$$\frac{\partial P(X)}{\partial t} = (P(X+2;s) - P(X;s))a(X) \tag{A.15}$$

In other words, Eqs. (A.6) and (A.15) are equivalent, and we prove that a synchronous parallel decomposition of a global master equation for this idealized problem results in the same master equation as formulated in Eq. (A.6). In practice, what this means is that, on average, our pkMC method advances time by an amount equal to that resulting from simulating K events sequentially in serial BKL. In reality, as Eq. (1) shows, Eq. (A.15) is advanced faster in time by a factor of $(K \cdot \mathrm{UR})$.

Appendix B. Parallel efficiency

The parallel efficiency, η_{st}, related to strong scaling, is defined as:

$$\eta_{\mathrm{st}} = \frac{t_{\mathrm{s}}}{K t_{\mathrm{p}}} \tag{B.1}$$

where t_{s} and t_{p} are, respectively, the times expended in identical serial and parallel calculations, and K, as above, is the number of processors. The computational cost of performing a serial calculation is the time it takes to complete a kMC cycle times the number of cycles, n_{s}. The CPU time per cycle can be decomposed into the cost of doing a frequency line search – which, as we have shown in Section 4.2, scales as $\mathcal{O}(N)$, where N is the total number of particles – plus the execution time of an event, t_{exe}. In the same fashion, t_{p} is composed of a 'serial' (calculation) time plus a communications time. The calculation time comprises an execution cost, $n_{\mathrm{p}} t_{\mathrm{exe}}$, where n_{p} is the number of cycles needed to complete the parallel calculation ($n_{\mathrm{p}} < n_{\mathrm{s}}$), and a search cost that scales as $\mathcal{O}(N/K)$. Evidently, t_{exe} is the same for a parallel calculation, and depends only on the characteristics of the processor and the compiler. Taking all these details into account, Eq. (B.1) can be written as:

$$\eta_{\mathrm{st}} = \frac{n_{\mathrm{s}} t_{\mathrm{exe}} + n_{\mathrm{s}} \mathcal{O}(N)}{K\left(\left(n_{\mathrm{p}} t_{\mathrm{exe}} + n_{\mathrm{p}} \mathcal{O}(N/K)\right) + n_{\mathrm{p}} t_{\mathrm{comm}}\right)} = \frac{n_{\mathrm{s}}}{K n_{\mathrm{p}}} \left(\frac{t_{\mathrm{exe}} + \mathcal{O}(N)}{t_{\mathrm{exe}} + \mathcal{O}(N/K) + t_{\mathrm{comm}}}\right) \tag{B.2}$$

The number of serial and parallel kMC cycles needed to complete a simulation of duration t_{sim} is, respectively:

$$n_{\mathrm{s}} = \frac{t_{\mathrm{sim}}}{\delta t_{\mathrm{s}}}$$

$$n_{\mathrm{p}} = \frac{t_{\mathrm{sim}}}{\delta t_{\mathrm{p}}}$$

where δt_{s} and δt_{p} are the average serial and parallel time steps ($\delta t_{\mathrm{s}} = \delta t_{\mathrm{BKL}} = 1/R_{\mathrm{tot}}$, $\delta t_{\mathrm{p}} = 1/R_{\mathrm{max}}$). From Eq. (2) we have that $n_{\mathrm{s}}/n_{\mathrm{p}} = K \cdot \mathrm{UR}$, which means that the first term in the right-hand side of Eq. (B.2) is equivalent to UR. In the spirit of the so-called $\log P$ model for global communications [34,35], we take the MPI communications overhead to be proportional to $(\log K)^b$, where b is a constant. Assuming that the CPU cost of performing linear searches and the communications cost are characterized by two constants c_0 and c_1 (architecture and compiler dependent),[7] Eq. (B.2) becomes:

$$\eta_{\mathrm{st}} = \mathrm{UR}\,\frac{t_{\mathrm{exe}} + c_0 N}{t_{\mathrm{exe}} + c_0(N/K) + c_1(\log K)^b} \tag{B.3}$$

[7] We have not attempted to fit c_0 and c_1. Suffice it to say that, for the two cases shown in the inset to Fig. 10, the relative amount of time spent on MPI functions (obtained via code profiling) for $K = 8$ was 15.7% and 12.2% for $j = 1$ and $j = 64$ respectively.

Assuming $t_{\text{exe}} \ll c_0 N, c_0(N/K)$:

$$\eta_{\text{st}} \approx \text{UR}\,\frac{c_0 N}{c_0(N/K) + c_1(\log K)^b} = \frac{\text{UR}}{1/K + (c_1/c_0 N)(\log K)^b} = \frac{\text{UR}}{1/K + a(\log K)^b} \tag{B.4}$$

Eq. (B.4) gives an idea of the factors that affect the parallel efficiency. For example, the numerator indicates that η_{st} is directly proportional to the utilization ratio, and, hence, to the time step gain introduced by the parallelization. It also shows that the efficiency is inversely proportional to the factor $a(\log K)^b$, associated with the cost of point-to-point communications in distributed parallel systems. In the limit of large numbers of processors, $1/K \ll \log K$, and $\eta \approx \text{UR}/a(\log K)^b$, *i.e.* the parallel efficiency does not saturate, but converges slowly to zero. On the other hand, when $K = 1$ (no communications), $\eta_{\text{st}} \approx \text{UR}$, which gives the maximum theoretical efficiency, consistent with Eq. (2).

For its part, the parallel efficiency, η_{ws}, associated with weak scaling, is defined as:

$$\eta_{\text{ws}} = \frac{1}{\text{WS}} = \frac{t'_{\text{s}}}{t'_{\text{p}}} \tag{B.5}$$

where t'_{s} is the CPU time required to complete a standard serial simulation of one domain with one PE, and t'_{p} is the time required to simulate a system K times as big as the serial one with K processors. In this case, both approaches require the same number of kMC cycles to complete the simulation, *i.e.* $n_{\text{s}} = n_{\text{p}} = n$. Then:

$$\eta_{\text{ws}} = \frac{n t_{\text{exe}}}{n t_{\text{exe}} + c_0(\log K)^b} \tag{B.6}$$

where, as before, we have assumed that the communications cost is proportional to $a(\log K)^b$. Evidently, with the communications overhead factored out, t_{exe} is the same for both the serial and parallel cases, as the parallel simulation is the equivalent of K replicas of the serial simulation. Therefore, η_{ws} is simply:

$$\eta_{\text{ws}} = \frac{1}{1 + (c_0/n t_{\text{exe}})(\log K)^b} = \frac{1}{1 + a(\log K)^b} \tag{B.7}$$

in agreement with recent works published in the literature [13,14]. Of course, for $K = 1$ (serial case), $\eta_{\text{ws}} = 1/\text{WS} = 1$, which is the ideal limit for weak scaling.

References

[1] M.H. Kalos, P.A. Whitlock, Monte Carlo Methods, John Wiley & Sons, New York, 1986.
[2] D.P. Landau, K. Binder, Monte Carlo Simulations in Statistical Physics, Cambridge University Press, 2000.
[3] R. Friedberg, J.E. Cameron, J. Chem. Phys. 52 (1970) 6049.
[4] R.H. Swendsen, J.S. Wang, Phys. Rev. Lett. 57 (1986) 2607.
[5] B.D. Lubachevsky, Complex Sys. 1 (1987) 1099.
[6] B.D. Lubachevsky, J. Comput. Phys. 75 (1988) 103.
[7] G. Korniss, Z. Toroczkai, M.A. Novotny, P.A. Rikvold, Phys. Rev. Lett. 84 (2000) 1351.
[8] G. Korniss, M.A. Novotny, H. Guclu, Z. Toroczkai, P.A. Rikvold, Science 299 (2003) 677.
[9] Y. Shim, J.G. Amar, J. Comput. Phys. 212 (2006) 305.
[10] D.R. Jefferson, Virtual time, ACM Trans. Prog. Lang. Syst. 7 (1985) 404.
[11] S.G. Eick, A.G. Greenberg, B.D. Lubachevsky, A. Weiss, ACM Trans. Model. Comp. Simul. 3 (1993) 287.
[12] B.D. Lubachevsky, A. Weiss, in: Proceedings of the 15th Workshop on Parallel and Distributed Simulations, Lake Arrowhead, CA, 2001, p. 185.
[13] Y. Shim, J.G. Amar, Phys. Rev. B 71 (2005) 115436.
[14] M. Merrick, K.A. Fichthorn, Phys. Rev. E 75 (2007) 011606.
[15] Y. Shim, J.G. Amar, Phys. Rev. B 71 (2005) 125432.
[16] A.B. Bortz, M.H. Kalos, J.L. Lebowitz, J. Comput. Phys. 17 (1975) 10.
[17] P. Hanusse, A. Blanché, J. Chem. Phys. 74 (1981) 6148.
[18] D. ben-Avraham, J. Chem. Phys. 88 (1987) 941.
[19] I. Beichl, F. Sullivan, IEEE Comput. Sci. Eng. 4 (1997) 91.
[20] J.G. Amar, Comput. Sci. Eng. 8 (2006) 9.
[21] M.J. Berger, S.H. Bokhari, IEEE Trans. Comput. 36 (1987) 570.

[22] i.e. The formal solution to Eq. (10) may be expressed in terms of the eigenfunctions, ϕ_m, of the operator $-\nabla^2$. Then we may express the Green's functions for the operator $[-\nabla^2 + \frac{\partial}{\partial t}]$ as $G(x, x_0; t) = \sum_m e^{-\lambda_m^2 t}\phi_m(\mathbf{x})\phi_m(\mathbf{x}_0)$.
[23] K. Kang, S. Redner, Phys. Rev. A 32 (1985) 435.
[24] F. Leyvraz, S. Redner, Phys. Rev. Lett. 66 (1991) 216.
[25] T. Oppelstrup, V.V. Bulatov, G.H. Gilmer, M.H. Kalos, B. Sadigh, Phys. Rev. Lett. 97 (2006) 230602.
[26] M.J. Caturla, N. Soneda, E. Alonso, B.D. Wirth, T. Díaz de la Rubia, J.M. Perlado, J. Nucl. Mater. 276 (2001) 13.
[27] The LLNL "Zeus" cluster (http://www.llnl.gov/computing/tutorials/lc_resources/#IntelSystems) on which all kMC simulations were performed runs version 1 of the MPI libraries, with two-sided communications.
[28] D.J. Kerbyson, H.J. Alme, A. Hoisie, F. Petrini, H.J. Wasserman, M. Gittings, in: Proceedings of the ACM/IEEE SC2001 Conference, 2001.
[29] S. Goedecker, A. Hoisie, Performance Optimization of Numerically Intensive Codes, SIAM, Philadelphia, PA, 2001.
[30] G. Amdahl, in: AFIPS Conference Proceedings, vol. 30, 1967, pp. 483–485.
[31] C.K. Wong, M.C. Easton, SIAM J. Comput. 9 (1980) 111.
[32] D.E. Knuth, The Art of Computer Programming: Sorting and Searching, second ed., vol. 3, Addison-Wesley Professional, 1998.
[33] I. Beichl, F. Sullivan, IEEE Comput. Sci. Eng. 3 (1996) 13.
[34] D.E. Culler, R. Karp, D.A. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, Commun. ACM 39 (1996) 78.
[35] K.W. Cameron, R. Ge, X.H. Sun, IEEE Trans. Comput. 56 (2007) 314.