Kinetic Monte Carlo simulations with minimal searching

T. P. Schulze*
Department of Mathematics, University of Tennessee, Knoxville, Tennessee 37996-1300
(Received 25 June 2001; published 13 February 2002)

Kinetic Monte Carlo (KMC) simulations are used to simulate epitaxial crystal growth. Presently, the fastest reported methods use binary trees to search through a list of rates in $O(\log_2 M)$ time, where M is the number of rates. These methods are applicable to an arbitrary set of rates, but typical KMC bond-counting schemes involve only a finite set of distinct rates. This allows one to construct a faster list-based algorithm with a computation time that is essentially independent of M. It is found that this algorithm typically reduces computation time by between 30% and 50% for typical simulations, with this factor increasing for larger simulations.

DOI: 10.1103/PhysRevE.65.036704 PACS number(s): 02.70.Rr, 02.50.Ga

The kinetic Monte Carlo (KMC) method is a stochastic model that simulates epitaxial film growth on an atom-by-atom basis using probabilistic rules to govern deposition, diffusion, and other growth processes. This technique was first adopted in the early 1970s [1,2] and has bifurcated in numerous directions. In a typical implementation, one assumes what is called a ''solid-on-solid'' or, more descriptively, ''cube-on-cube'' epitaxy, where the crystalline films grow with an orthorhombic structure. The bulk of the work focuses on single-species, monatomic crystals. See Levi and Kotrla [3] for a review of KMC studies and algorithms.

The principal KMC algorithm is based on the method of Bortz, Kalos, and Lebowitz (BKL) [4], with the implementation of an efficient binary search described by Blue, Beichl, and Sullivan (BBS) [5]. The BKL algorithm is built on the assumption that the model features M independent Poisson processes with rates r_m that sum to give an overall rate R which can be used to (1) decide which event to execute, and (2) randomly select the time it takes for that event to occur from a Poisson distribution.

After generating a random number $r \in [0,R)$, a linear search requires $O(M)$ operations, whereas the binary search requires $O(\log M)$. An intermediate scheme due to Maksym [6] groups the M rates into N subsets, performing a linear search on the smaller sets. The binary-search algorithm is a generalization of Maksym's method which repeatedly subdivides the subsets. Both Levi and Kotrla [3] and Blue *et al.* [5] give the computation time for Maksym's algorithm as $O(M^{1/2})$. As previously implemented, this is correct, but the present paper shows that Maksym's method can be further adapted, reducing this to a fixed cost per simulated event.

The basic algorithm must be combined with a bond-counting scheme to complete the model. Typical schemes are based on nearest-neighbor interactions and, as a result, feature only a relatively small number of distinct rates. This feature can be used to improve the basic algorithm by initially sorting the possible events according to their rates and then doing some efficient bookkeeping as the simulation proceeds. The important point is that the computation time for this book keeping is essentially independent of M. In practice, increasing memory requirements slow it somewhat, but the method easily outperforms the binary search, reducing typical computation times by 30% to 50% as memory-induced limitations are reached.

First, the algorithm is described in a general context; then illustrated with a specific bond-counting scheme. In KMC simulations the rates r_m are a function of the surface configuration which consists of a set of integer height values $\{h_{ij}\}$. The nearest-neighbor models can assume only a relatively small number of local configurations that affect a given rate. Let this number, which is a constant determined by the specific bond-counting scheme, be N, relabeling the much smaller set ($N \ll M$) of distinct rates as R_n. During the simulation we maintain N lists in an array L_{nk} that contain the event indices m of events which occur with rate R_n. We also maintain an address list A_m which tells us where event m is currently listed in the array L_{nk} and a count C_n of the number of events in each of the N lists.

After the initial construction of these arrays, the algorithm proceeds as follows.

(1) Computer the overall rate $R=\Sigma_{n=1}^{N} R_n C_n$; retain the partial sums S_n.

(2) Select a random number $r \in [0,R)$.

(3) Search through the list of partial sums S_n until $r < S_n$.

(4) Select an event from the set of events that occur at this rate by computing

$$m = \mathrm{Int}\left(\frac{(S_n - r)}{R_n}\right) + 1.$$

(5) Execute that event and update the configuration $\{h_{ij}\}$.

(6) For the (local) events that have their rates changed from R_{n_i} to R_{n_f} (a) move them to the end of list n_f; add 1 to C_{n_f}; update A_m; (b) move the event listed as $L_{n_i C_{n_i}}$ into the vacated position on list n_i; reduce C_{n_i} by 1 update A_m.

Note that the search in step 3 is through a short list of events that does not scale with M. One could perform a binary search on this list, but it is probably better (and simpler)

*Email address: schulze@math.utk.edu

to presort the rates weighted by their typical multiplicities C_n. This can be done to good approximation in an easy way by making use of the bond-energy formula. The savings in this algorithm comes from the fact that the particular event among the chosen subgroup can be determined by calculation (step 4) rather than searching. Alternatively, one can select the event at random from the appropriate list L_{nk} as suggested by Levi and Kotrla [3] (see their algorithm 3), but without the cross listing provided by the address list A_m, one cannot obtain the shorter computation time. The address list is needed because the execution of an event m affects the rates of neighboring events, which must be located within the set of lists L_{nk} before they too can be updated. Without the address list, extensive searching is still necessary.

Next, we demonstrate the algorithm using a popular bond-counting scheme due to Smilauer and Vvdensky [7] (SV). We refer the reader to this article for details and reproduce the essentials here. This model considers random deposition with a uniform rate r_{dep} and nearest-neighbor diffusion based on the local height configuration. We will implement the deposition without the local search for favorable sites that is sometimes performed; this is similar to the procedure used by BBS [5] to test the binary-search version of the BKL algorithm [5] using an earlier bond-counting scheme due to Clarke and Vvedensky [8].

Since the deposition occurs with a uniform rate and the number of possible deposition events is fixed by the number of sites, say $I\times J$, this group rate, call it $S_0=C_0R_0$, never changes and can be handled separately from the diffusion process—i.e., it requires no bookkeeping. This is essentially a simple application of the algorithm presented above which one assumes is routinely implemented in any use of the BKL algorithm. We therefore take this approach in both the binary-tree and minimal-search versions of the BKL algorithm when we make the comparisons described below.

The diffusion process provides us with a nontrivial example of the algorithm at work. In the SV bond-count scheme, the rate r_m is a function of the in-plane, lateral, nearest neighbors $\mu\in\{0,1,2,3,4\}$ and the difference between the remaining lateral, nearest neighbors ν (i.e., in the planes above and below the site being considered) before and after an event where an atom moves one lateral site. This latter number is set to zero if it is not positive; hence $\nu\in\{0,1,2,...,8\}$, giving a total of 45 distinct rates for this particular bond-counting scheme. Of these, ten may be discarded because there are no geometric configurations corresponding to those parameters (i.e., $\nu\in\{0,1,...,4+\mu\}$). Many of the remaining rates are essentially zero and could probably be discarded,but we make no use of such approximations here. These rates are precomputed and stored as $R_{\mu\nu}$ or, if one likes, as R_n with a single rate index $n=1+\mu+5\nu$. Similar array flattening can be used to minimize storage for the names of events $m=l+4(i-1)+4I(j-1)$ in the list L_{nk}, where $l\in\{1,2,3,4\}$ is an index indicating the direction of the hop.

This brings us to a discussion of storage requirements. The principal storage requirement of basic BKL method is the $M=4IJ$ real numbers corresponding to the rates r_m. Alternatively, one can store integers indicating where these rates are in the array $R_{\mu\nu}$, which is useful in any version of the BKL algorithm. A binary tree requires storing approximately $\log_2(M)$ partial sums of rates. These can be efficiently packed into $2M$ real storage locations if I and J are powers of 2. The principal storage requirement of the present algorithm is NM for the set of lists $\{L_{nk}\}$ with an additional set of M integers for the address list A_m. In the case of the SV bond-counting scheme this is a total requirement of about $36M$; well above the other methods. Several things can be done to reduce this. First, note that only M of the $35M$ storage locations in L_{nk} are being used at any one time. To avoid searching, one cannot store the array efficiently, but one can anticipate that many of the higher storage locations will never be used. One way to take advantage of this is to declare some of the N lists to be shorter than others, although this might not be convenient in most programming languages. Another way, which depends on the bond-counting scheme, is to recognize that the vast majority of events in most simulations will correspond to flat (i.e., $\mu=4$, $\nu=1$) sites. Since these events are rarely executed, one can avoid listing them by simply choosing a site (i,j) and hop direction $\in\{1,2,3,4\}$ at random, checking to see if it is of the desired type, and choosing again if necessary. Neither of these methods has been implemented here, however, as it appears that computation speed will be more limiting than storage for most applications.

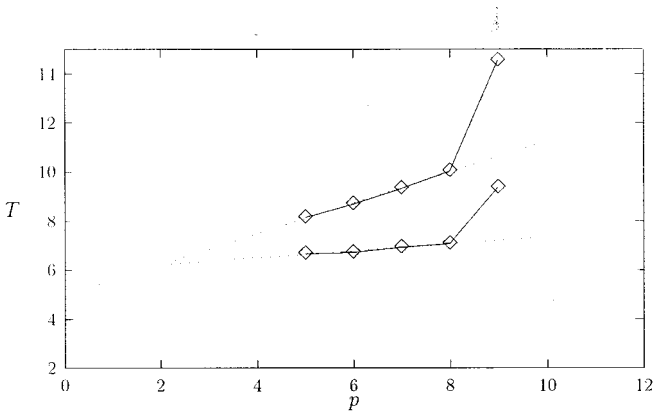

FIG. 1. The computation time in seconds for 10^5 events of a $2^p\times 2^p$ simulation using the binary-search algorithm and the minimal-search algorithm presented in the text.

The computation times graphed in Fig. 1 were the result of simulations done on a Pentium III PC with one gigabyte of RAM, the LINUX operating system, and the G77 FORTRAN compiler. The storage for the binary-tree BKL method was slightly suboptimal in that an $M\log_2(M)$ array was used rather than an efficient array of size $2M$. With this qualification, it was found that the two methods exceeded the available memory on subsequent powers of 2 for $2^p\times 2^p$ square-lattice simulations. Both methods reached a memory-swap limitation at the same value of p. Figure 1 confirms the expected linear growth of computation time per simulation event with the exponent p for the binary-search algorithm. While theoretically flat, the present algorithm is also found

to increase linearly with p—but with a much smaller slope.This is attributable to the increasing cost of memory access associated with both methods as the size of the system increases. Although the difference in computation time for the two methods grows only logarithmically with the overall simulation size M, it starts to take its toll even with the 512×512 simulations that are typical in the current literature. Extrapolating this growth, one anticipates that the method described here will be of increasing value as hardware capabilities improve.

[1] F. F. Abraham and G. W. White, J. Appl. Phys. **41**, 1841 (1970).
[2] G. H. Gilmer and P. Bennema, J. Appl. Phys. **43**, 1347 (1972).
[3] A. C. Levi and M. Kotrla, J. Phys.: Condens. Matter **9**, 299 (1997).
[4] A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, J. Comput. Phys. **17**, 10 (1975).
[5] J. L. Blue, I. Beichl, and F. Sullivan, Phys. Rev. E **51**, 867 (1995).
[6] P. A. Maksym, Semicond. Sci. Technol. **3**, 594 (1988).
[7] P. Smilauer and D. D. Vvedensky, Phys. Rev. B **52**, 14 263 (1995).
[8] S. Clarke and D. D. Vvedensky, J. Appl. Phys. **63**, 2272 (1988).