Parallel Kinetic Monte Carlo Simulations on a Shared Memory Multiprocessor System

Travis Smith*

Advisor: Dr. Jacques G. Amar

Department of Physics & Astronomy

University of Toledo, Toledo, OH 43606

(Dated: August 7, 2003)

Abstract

In order to simulate non-equilibrium processes over larger time scales and for realistic size systems, it is desirable to use parallel computing. Unfortunately, the standard algorithm for simulating activated processes, kinetic Monte Carlo (KMC), is inherently serial and thus only suitable for use with a single processor. Recently, however, our group has developed parallel (KMC) algorithms which have been successfully tested on a Beowulf cluster using Message Passing Interface (MPI) Application Program Interface (API). Due to the lack of communication overhead such algorithms should be even more efficient on shared-memory machines. As a first step in investigating this possibility, I have been developing a parallel KMC code to simulate one-dimensional irreversible epitaxial growth on a shared memory machine using OpenMP. We have verified and tested this code on the Sunfire and Origin 2000 computers at the Ohio Supercomputer Center (OSC) using multiple processors. Unfortunately, so far we have not obtained a significant speed increase using this method. However, we believe that this is not due to a fundamental limitation in the algorithm but rather to computational and/or compiler limitations. In the near future, we hope to increase the parallel efficiency of our code so that it can then be applied to more complex and realistic problems.

*Electronic address: tsmith2@physics.utoledo.edu

I. INTRODUCTION

The simulation of non-equilibrium processes such as irreversible epitaxial thin film growth is a useful tool in understanding the process of crystal growth. There is a computational limit on the complexity and size of a system that can be simulated in a reasonable amount of time. To overcome this limit and explore the area of thin film growth in more detail it is useful to examine multiprocessor computing. To reduce the computational time required to run these simulations two major types of multiprocessor systems are being explored by my group: shared memory and distributed memory machines. Shared memory machines are also known as symmetric multiprocessor (SMP) systems. Distributed memory machines are mostly called beowulf clusters because of the cost effectiveness of such implementations (due to being made completely of off-the-shelf parts). Beowulf clusters are a large bunch of computers all networked together and the addition of any computer, or node, to a cluster only requires a network interface and the installation of the required software. This leads to a very versatile and cost effective system. While they are much more scalable than SMP machines but require extra time to communicate. Scalability is something SMP machines lack because all of the processors must be identical and are not made from off-the-shelf parts. The use of beowulf clusters for these simulations is outside the scope of this paper but has been researched by my group. However, the use of SMP machines will be discussed in greater detail.

II. KINETIC MONTE CARLO

Kinetic Monte Carlo (KMC) is used for Markov processes or activated events. KMC is faster than Metropolis Monte Carlo (MMC, which was developed in 1949 by Metropolis and Ulam) since MMC events are only accepted with a finite probability based upon the rate of the event. In contrast KMC accepts the next event with an acceptance probability of 1. Monte Carlo simulations in general are an excellent way to solve computational problems and have a wide range of uses outside of the regime of physics and mathematics.

Thin film growth has some inherent randomness to the particle movements and so using KMC makes a sensible choice in this simulation. With KMC there is no waiting around for the next event to happen, the event is selected and the system time is updated accordingly.

The time interval is $\Delta t = -\frac{\ln r}{R_{total}}$ where r is a uniform random number on the interval $[0, 1)$ and R_{total} is the total rate of the local system. We get this equation from Markov processes. After updating the system time and completing the event the next event is then selected, the system time updated, and this continues until the ending conditions are met. Often this is until a given number of monolayers have been deposited in the system.

III. MODEL

We have used OpenMP to simulate a simple one-dimensional model of irreversible growth on a square lattice. In our model, particles are deposited randomly with rate F per site per unit time while monomers diffuse with hopping rate D. Whenever a diffusing monomer encounters another monomer as nearest-neighbor, the two particles are irreversibly bonded and form a stable dimer. Similarly, a monomer will irreversibly attach to an island if any particle of that island is its nearest-neighbor. Monomers deposited on top of islands also diffuse with rate D.

At each point of a KMC simulation, the type of event (i.e. deposition or diffusion) is chosen by calculating the ratio P of the total diffusion rate (corresponding to the number of monomers times the hopping rate D) to the total rate for all events including deposition. A uniform random number r between 0 and 1 is then generated. If $r \leq P$, then a diffusion move is selected, and conversely if $r > P$, then a deposition move is selected.

IV. OPENMP

OpenMP [http://www.openmp.org] is a standard that is used to run programs in parallel on shared memory, or symmetric multiprocessing, machines. Directives are added into the source code that tells the special compiler how to make the code run in such a manner that allows use of more than one processor. These changes only describe when the program starting running in parallel. By using OpenMP a program runs normally until it is told to break off into separate threads owned by different processors.

We created a test program to verify that OpenMP on SMP machines did scale properly. Our results (as seen in Table I) show that OpenMP does in fact scale. The total CPU time stays the same while the wall time or length of time you wait for the simulation to complete

scales as $\frac{1}{N_{CPUs}}$.

TABLE I: OpenMP Run Times for Test Program

Number of Processors	1	2	4
Wall Time	29 s	15 s	7.5 s
Total CPU Time	29 s	30 s	30 s

V. SYNCHRONOUS KMC ALGORITHM

For our model of irreversible one-dimensional growth, our Synchronous KMC algorithm is not difficult to implement. For a system with N processors, the system to be simulated is broken up into N equal parts with each part being controlled by one of the processors. Each part of the lattice that is controlled by a processor is also broken up further into two sublattices. Each sublattice is labeled with a 0 or a 1 and and is randomly selected at the beginning of each cycle to be the same for all procesors. The length of each cycle corresponds to a period T_{max} which is determined by the rate of the fastest event in the system.

During each synchronous cycle each processor figures out the rates for each type of event in its own sublattice. In our case there were only two types of events: diffusion and deposition. All of the rates R_i are summed together as R_{total}. A time for the next event is generated and added to the current local time: $\Delta t = \frac{-\ln r}{R_{total}}$ where r is a random number from zero to one. If t is greater than T_{max} then the event is rejected and we randomly select a sub-lattice again. If t is less than T_{max} then the probability of the first type of event happening is $\frac{R_1}{R_{total}}$ and the probability of the second type of event occurring is $\frac{R_1+R_2}{R_{total}}$ and so on for each type of event left.

Depositing is carried out by randomly selecting a site of the selected sublattice and incrementing the height of that site by one. A routine to update the neighbors and make sure that monomers are kept track of is then called. Diffusion works a bit differently. A list of monomers, or walkers is kept. If a diffusion move is selected, then a walker is randomly selected from the walker list and assigned a random direction. In more complex simulations in two-dimensions with corner or edge diffusion a direction is often stored with the location of the walker. For this simple one dimensional case this was not done. Again, once the walker moves the area around the event is updated to ensure that the list of walkers is

complete and correct.

VI. RESULTS

We first wrote a serial code to emulate our parallel KMC algorithm for our 1D model of irreversible growth. Excellent agreement was found between the usual serial KMC results and our parallel emulation results. We then implemented a parallel version of our code by using OpenMP on the Origin 2000 and Sunfire at the Ohio Supercomputer Center (OSC). As can be seen in Fig. 1 using our OpenMP code we also found excellent agreement with serial KMC results. However, unfortunately the code did not scale properly for the SMP machines that we used (see Table II). In particular, the "wall time" did not decrease for fixed system size with an increasing number of processors. The cause of this poor scaling is not understood, but does not appear to be due to a defect in the algorithm, since other members of our group have obtained good scaling using similar algorithms on Beowulf clusters using MPI. Instead, we believe that it is due to some limitation of the implementation of OpenMP or SMP machines that we do not yet understand.

TABLE II: OpenMP Run Times for Parallel KMC

Number of Processors	1	2	4
Wall Time	31.2 s	30.8 s	36.2 s
Total CPU Time	31.2 s	61.6 s	144.8 s

VII. FUTURE GOALS

In the near future, we hope to be able to understand and correct the scaling problem with our OpenMP implementation so that we can carry out parallel KMC simulations of this model as well as more complicated models. We also plan to use OpenMP to carry out simulations using asynchronous KMC.

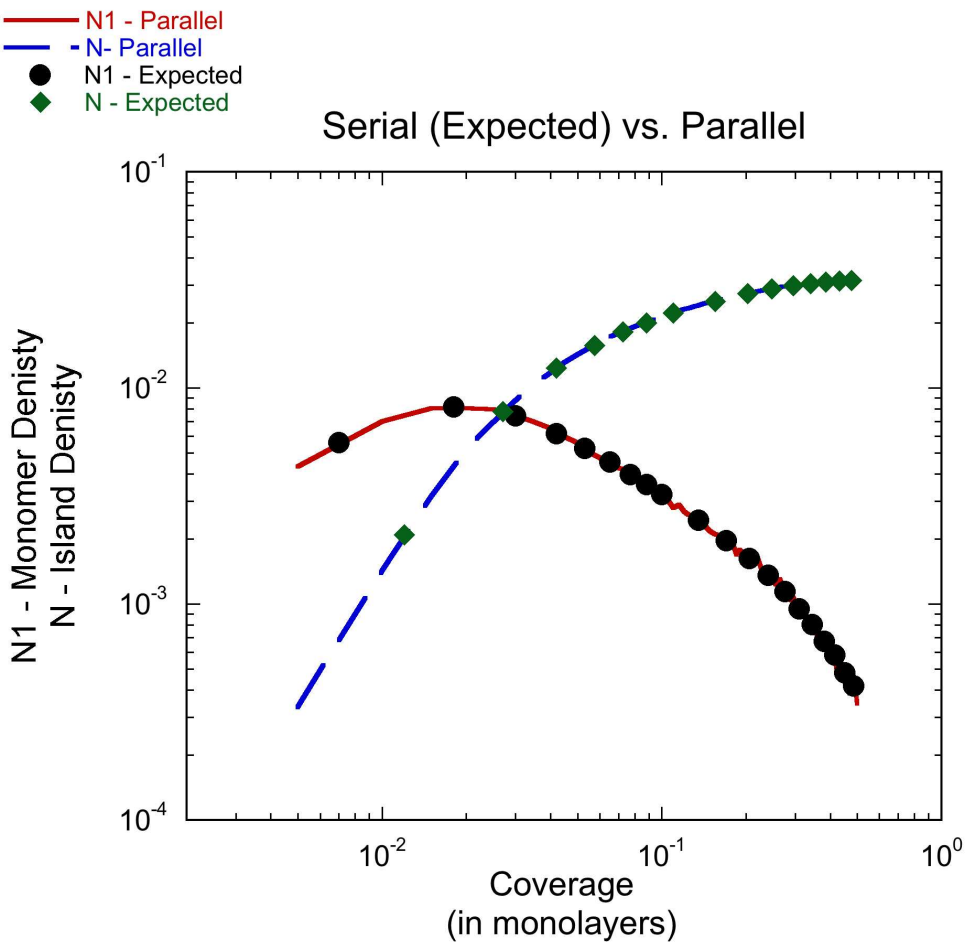

FIG. 1: Comparison of Parallel Code to known values for KMC Simluation

VIII. CREDITS AND THANKS

This work was supported by a grant from the National Science Foundation. We would also like to thank the Ohio Supercomputer Center for a grant of computer time.