Journal of Computational Physics **153,** 488–508 (1999)
Article ID jcph.1999.6291, available online at http://www.idealibrary.com on IDEAL®

Parallelization of a Dynamic Monte Carlo Algorithm: A Partially Rejection-Free Conservative Approach

G. Korniss,* M. A. Novotny,* and P. A. Rikvold*,†

*Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida 32306-4130; and †Center for Materials Research and Technology and Department of Physics, Florida State University, Tallahassee, Florida 32306-4350
E-mail: korniss@scri.fsu.edu, novotny@scri.fsu.edu, and rikvold@scri.fsu.edu

Received December 28, 1998; revised April 28, 1999

We experiment with a massively parallel implementation of an algorithm for simulating the dynamics of metastable decay in kinetic Ising models. The parallel scheme is directly applicable to a wide range of stochastic cellular automata where the discrete events (updates) are Poisson arrivals. For high performance, we utilize a continuous-time, asynchronous parallel version of the n-fold way rejection-free algorithm. Each processing element carries an $l \times l$ block of spins, and we employ fast one-sided communication routines on a distributed-memory parallel architecture. Different processing elements have different *local* simulated times. To ensure causality, the algorithm handles the asynchrony in a conservative fashion. Despite relatively low utilization and an intricate relationship between the average time increment and the size of the spin blocks, we find that the algorithm is scalable and for sufficiently large l it outperforms its corresponding parallel Metropolis (non-rejection-free) counterpart. As a sample application, we present results for metastable decay in a model ferromagnetic or ferroelectric film, observed with a probe of area smaller than the total system.

Key Words: Monte Carlo methods; parallel discrete-event simulations; rejection-free algorithms; kinetic Ising model.

1. INTRODUCTION

Fast and efficient algorithms are invaluable ingredients for large-scale simulations in physical sciences and engineering. The implementation of efficient, massively parallel algorithms for Monte Carlo simulations is not only an interesting and challenging problem, but also one of the most complex problems in parallel computing. It belongs to the class of parallel discrete-event simulations (sometimes referred to as distributed simulations) which

0021-9991/99 $30.00

has numerous applications in engineering, computer science, and economics, as well as in physics [1]. For example, in lattice Ising models the discrete events are spin updates, while in queueing networks they are job arrivals. The dynamics of these systems, which obviously contain a substantial amount of parallelism, were traditionally simulated on serial computers. Paradoxically, it is fairly difficult in practice to implement an efficient parallel algorithm to simulate these dynamics, mainly due to the fact that the discrete events (updates) are not synchronized by a global clock. Here we present and analyze the performance of a variation [2] of the *n*-fold way algorithm [3, 4] to simulate magnetization switching in a kinetic Ising model on a distributed-memory parallel computer.

Metastability and hysteresis are widespread phenomena in nature. Ferromagnets are common systems that exhibit these behaviors [5], but there are also numerous other examples ranging from ferroelectrics [6] to electrochemical adsorbate layers [7] to liquid crystals [8]. An important potential technological application of nanoscale ferromagnetic particles and ultrathin films is high-density magnetic recording media, and computational experiments and modeling of these materials are integral parts of current research and engineering.

Simulating metastable decay often involves long characteristic time scales (the lifetime of the metastable phase), and several sophisticated algorithms have been developed for serial computers [3, 4, 9] to tackle this problem. Common testbeds for these algorithms are kinetic Ising ferromagnets below their critical temperature T_c, which exhibit slow metastable decay after reversal of the external magnetic field [10]. These models are appropriate for the study of highly anisotropic single-domain nanoparticles and thin films [11].

There are powerful techniques, such as multi-spin coding [12] and cluster algorithms [13, 14], to simulate the *equilibrium* properties of the Ising model, but these methods completely change the original microscopic *dynamics*. Kinetic Ising models, either with integer-time updates or with Glauber's continuous-time interpretation [15], were believed to be inherently serial; i.e., the corresponding algorithm could attempt to update only one spin at a time. Providing a counterexample to that belief, Lubachevsky presented an efficient conservative approach for parallel simulation of these systems [2] *without* changing the dynamics of the underlying model. Applications of his scheme also include modeling of wireless cellular communication networks [16] and ballistic particle deposition [17]. Also, he *proposed* a way to incorporate the *n*-fold way algorithm, possibly further contributing to speedup. Here, we implement this algorithm on the isotropic, two-dimensional Ising model, and we systematically compare its performance to the parallel Metropolis algorithm. To our knowledge, this is the first actual implementation of the parallel *n*-fold way scheme. More importantly, detailed comparison between the two parallel schemes (the parallel *n*-fold way and the parallel Metropolis) has not been given before. As we show in this paper, there is an interesting competition between their performances, and detailed analysis is essential to decide which one to apply in different parameter regimes. We shall see that our algorithm provides an efficient scalable way for simulating *large* systems which would not fit the memory of a single computer or for which the performance would be severely degraded due to excessive (remote) memory usage.

This paper is organized as follows. In Section 2 we define the model and summarize the standard Metropolis and rejection-free serial algorithms. In Section 3 we outline the basic conservative approach for parallel discrete-event simulation, applied to Ising spins, and describe the parallel Metropolis and *n*-fold way algorithms. In Section 4 we give some details of the implementation and analyze its performance. In particular, we compare it to that of the parallel Metropolis update scheme. In Section 5 we give some examples

of ongoing and future applications to model magnetic and ferroelectric systems, and in Section 6 we conclude with a brief summary.

2. THE MODEL

We simulate magnetization switching in the isotropic Ising model on an $L \times L$ square lattice with periodic boundary conditions, which has the Hamiltonian

$$\mathcal{H} = -J \sum_{\langle ij \rangle} s_i s_j - H \sum_{i=1}^{L^2} s_i. \tag{1}$$

Here $J > 0$ is the ferromagnetic nearest-neighbor spin-spin interaction, H is the external field, and the sum $\sum_{\langle ij \rangle}$ runs over all nearest-neighbor bonds. To study metastable decay, all spins are initialized in the $+1$ state, and we apply a negative magnetic field (i.e., a sudden field reversal) at constant $T < T_c$. Here T_c is the critical temperature of the zero-field Ising model, below which the system spontaneously orders. For $T < T_c$ the decay of the metastable phase proceeds through nucleation and growth of one or more compact droplets of the stable phase. For fixed T and H there exists a system size, approximately the typical droplet separation $R_o(T, |H|)$, such that for $L > R_o$, many droplets form and contribute to the decay of the metastable phase. This is called the multidroplet regime [10]. For the simulations presented in this paper the parameters are chosen such that the system is in this regime. In particular, $R_o \approx 275$ at $T = 0.6T_c$ and $|H|/J = 0.2222$, and $R_o \approx 12$ at $T = 0.8T_c$ and $|H|/J = 0.4127$ are the largest and smallest values of R_o for the temperatures and fields used in the following sections.

2.1. *The Serial Metropolis Algorithm*

In the standard serial Monte Carlo (MC) algorithm [15], where time is a discrete variable taking on integer values, we choose the single-spin-flip Metropolis rates, according to which at every MC step (MCS) a spin is picked randomly on the lattice and flipped with the probability

$$p = \min\{1, \exp(-\Delta\mathcal{H}/kT)\}, \tag{2}$$

where $\Delta\mathcal{H}$ is the energy change which would result from the flip.

It is important to note that in this standard algorithm, the choice of fixed integer time increments (i.e., $\Delta t \equiv 1$ MCS between two successive spin-flip trials) is a convenience rather than a necessity: the underlying *dynamics* of the real physical system corresponds to a continuous time evolution, in which spin-flip attempts are Poisson arrivals. Thus, the time increment between two successive events is an exponentially distributed random variable. To exactly mimic the Poisson arrivals one should generate random time increments by

$$\Delta t = -\ln(r) \tag{3}$$

in units of MCS, where r is uniformly distributed in $(0, 1)$. Clearly, this is computationally more expensive than the simple integer–time update and usually yields identical results when averaged over many independent runs. This extra cost, however, can pay off when mapping the system onto a parallel computer, as we shall see in Sections 3 and 4.

An important quantity of interest is the average lifetime $\langle\tau\rangle$ of the system, which is the time needed to exit the metastable phase. A possible way to estimate this quantity is to keep track of the time series of the magnetization

$$m = \frac{1}{L^2}\sum_{i=1}^{L^2} s_i, \tag{4}$$

which approximately equals -1 in the equilibrium phase. An average of its first-passage time to zero (i.e., $m(t=\tau)=0$) over many independent escapes from the metastable state will then yield $\langle\tau\rangle$.

The weakness of the standard Metropolis algorithm (with both integer and continuous time) is the low acceptance rate of spin-flip trials at low temperature and low field, which is a result of the small flipping probability p in Eq. (2). For example, at $T=0.7T_c$ and $|H|/J=0.2857$ the fraction of successful spin-flip attempts is only about 5%. One may overcome this "waste of trials" by using a rejection-free update scheme as summarized in the following subsection.

2.2. *The Serial n-Fold Way Algorithm*

In the *n*-fold way update scheme [3], a spin flip is always performed, and the simulated time is incremented appropriately. To implement the scheme, one must introduce the notion of spin classes which carry the state of the spin itself and its neighbors. In the above model there are 10 such classes, characterized by the number of spins in class i, n_i, and the flipping probability, p_i, which is the same for all spins in a class. Since the classes are disjoint, $\sum_{i=1}^{10} n_i = L^2$. When an update is performed a class is first chosen according to the relative weights $\{n_i p_i\}_{i=1}^{10}$; then one of the spins in the class is picked with equal probability, $1/n_i$. Once the information on the classes has been updated, in particular the n_i's, the time of the next update is determined. As in the standard (non-rejection-free) update scheme, the *n*-fold way algorithm can be performed in either integer or continuous time [4]. In both cases the time increment is a *random* variable. For the integer–time case, it is a geometrically distributed random number,

$$\Delta t = \mathrm{INT}\left[\frac{\ln(r)}{\ln(1-\Gamma)}\right] + 1, \tag{5}$$

while for continuous time it is exponentially distributed,

$$\Delta t = -\frac{\ln(r)}{\Gamma}. \tag{6}$$

Here,

$$\Gamma = \frac{\sum_{i=1}^{10} n_i p_i}{L^2} \tag{7}$$

is the inverse of the average time needed to exit the configuration specified by $\{n_i\}$, and r is a uniformly distributed random number on $(0, 1)$ [3, 4]. For both cases, Δt is given in units of MCS. At low temperatures the p_i's of the dominant classes (those with high populations) can be very small, resulting in large typical time increments. For example, at $T=0.7T_c$ and $|H|/J=0.2857$ the mean time increment is approximately 20 MCS. This is how the algorithm "bypasses" a large number of unsuccessful flip attempts and can increase the efficiency by several orders of magnitude [4].

Finally, it is important to note that the standard Metropolis and the n-fold way algorithms yield identical physical results. They are simply two different implementations for simulating the same dynamics.

3. PARALLELIZATION

3.1. *The Parallel Metropolis Algorithm*

First, we review the conservative scheme for parallelization of the standard Metropolis algorithm [2]. An obvious way to parallelize the serial Metropolis algorithm is to spatially decompose the $L \times L$ lattice into $l \times l$ blocks. When it is mapped onto a parallel computer, each processing element (PE) carries an $l \times l$ block of spins. The number of PEs, N_{PE}, and the block size, l, are simply related through $N_{\mathrm{PE}} = (L/l)^2$ (Fig. 1). Here we outline the algorithmic steps for the continuous-time case. Each PE carries its own *local* time t. Initially a spin configuration, corresponding to $t = 0$, is chosen, and the time of the first update is determined by $t = -\ln(r)$, independently on each PE. In our case, the initial configuration is $s_i = +1$ for all spins. For clarity, our time unit will be one MCS per PE (MCSP), during which each PE attempts to update one spin on average. Each PE is responsible for updating the $N = l^2$ spins that it carries by iterating the following steps:

Step 1. Select a spin from the block with equal probabilities.

Step 2. (a) If the chosen spin is in the kernel of the block, go to Step 3.

(b) If the chosen spin is on the boundary of the block, *wait until* the *local* simulated time t of this update becomes less than or equal to the same quantity for the corresponding

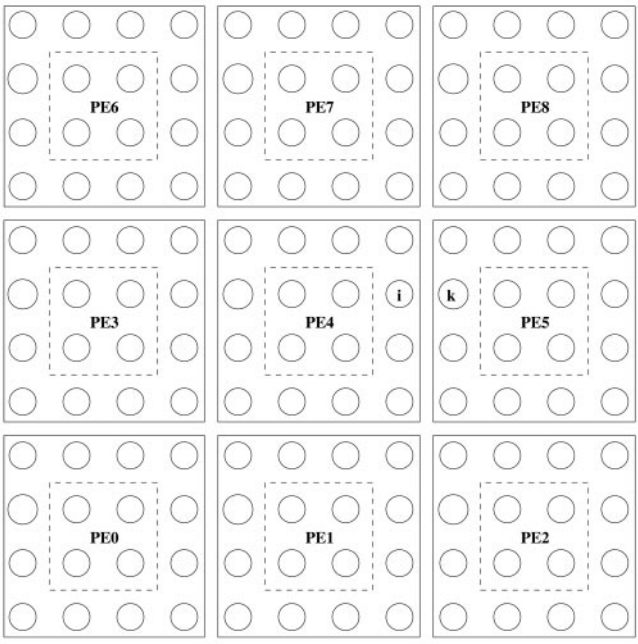

FIG. 1. Schematic diagram of the spatial decomposition of the system and its mapping onto a parallel machine. Here $L = 12$ and $l = 4$. Each of the $N_{\mathrm{PE}} = (L/l)^2 = 9$ processing elements (PEs) carries $l^2 = 16$ spins, confined by solid lines. The spins on the boundary are separated from those in the kernel by dashed lines.

neighboring PE(s) (for our model on a square lattice, 2 PEs for corner spins, one PE otherwise), then proceed to Step 3.

Step 3. Update the state of the spin using the same probabilities as those in the standard serial algorithm [Eq. (2)].

Step 4. Determine the time of the next update by using the *local* time increment $\Delta t = -\ln(r)$, where r is a uniformly distributed random number on (0, 1).

With reliable random number generators (an independent one on each PE), and use of a continuous probability density (exponential) for the time increments, the probability of equal-time nearest-neighbor updates is of measure zero. Thus, no *global* barrier synchronization is necessary among the PEs to preserve the uniqueness of the simulated trajectory, provided the same set of random seeds is used initially. The algorithm is obviously free from deadlock, since in the worst-case scenario the PE with the minimum local time is able to make progress. It is clear from the above *asynchronous* algorithm that at any given (wall clock) moment, different PEs have different local simulated times. The "wait until" directive in Step 2., however, ensures that the information passed between neighboring PEs does not violate causality. This can be seen from the following example. Suppose that PE4 is currently updating spin s_i, which is on the boundary of its block (Fig. 1). This update is possible if the local update time on PE4, t_4, is not larger than that on PE5, t_5. The update is also necessarily correct since s_k is guaranteed to be in the same state that it was in at t_4. This is so because updates for PE5 are forbidden by the "wait until" directive once t_5 becomes larger than t_4.

The above asynchronous algorithm is suitable for a continuous-time update scheme, but it can cause inconsistency when integer time is used. Then, explicit barrier synchronization should be incorporated (synchronous algorithm) to treat equal-time spin-flip trials of nearest-neighbor spins without ambiguity, and to ensure the reproducibility of a simulated path, provided the same set of random seeds is used. However, these (independent) nearest-neighbor, equal-time events violate detailed balance [with respect to the Hamiltonian (1)], and are unprecedented in the corresponding serial algorithm. Thus, we conclude that to be faithful to the original kinetic Ising model, one must employ the *continuous-time* update scheme. Other, more general cellular automata may tolerate nearest-neighbor, equal-time updates, since in such cases the microscopic update rates are not necessarily related to an *a priori* known equilibrium probability distribution, and the corresponding equal-time events may very well represent the real physical behavior. Also note that frequent barrier synchronization can be very "expensive" on a distributed-memory architecture, such as the T3E, where each PE executes the code completely independently of all other PEs. This is especially true for Monte Carlo algorithms, in which the core of the update routine is extremely simple and the time a PE spends at a barrier waiting for the others can easily exceed the time it is active.

3.2. *The "Shielded" Parallel n-Fold Way Algorithm*

A natural idea for parallelizing the n-fold way algorithm is to employ the same spatial decomposition as that for the parallel Metropolis algorithm, and to apply the serial rejection-free update scheme directly on each block. However, one cannot simply run a copy of the serial n-fold way algorithm on each PE: due to the nearest-neighbor interaction, the local time increments and the class populations depend not only on the spin values in the block

carried by that PE, but also on the states of the spins located on the adjacent boundaries of the neighboring PEs. By updating a spin on the boundary of a block, the updating PE could corrupt the simulated history of a neighboring block, if the local time of the latter is already ahead of that of the former. Thus, the neighboring PE would need to perform a rollback procedure to recover from the simulation of a series of "false" events. It is easy to see that this rollback might generate a cascade of rollbacks in other PEs, making the implementation rather difficult. Moreover, it is unclear how frequently such rollbacks would occur, and how far they would go back in time and propagate through space.

Lubachevsky proposed a way to avoid rollbacks by modifying the original algorithm [2]. In each block, an additional class which contains the spins on the boundary is defined. The weight of this class is the number of boundary spins, $N_b = 4(l-1)$, which does not change during the simulation. The original n-fold way tabulation of spin classes is only used in the kernel of the block. Hence, $N_b + \sum_{i=1}^{10} n_i = N$, where $\sum_{i=1}^{10} n_i = N_k$ is the number of spins in the kernel, and $N = l^2$ is the total number of spins in a block. Note that the population of the classes, $\{N_b, \{n_i\}_{i=1}^{10}\}$, is maintained separately for each block by the corresponding PE. Once the initial configuration is set and the corresponding local time of first update is determined, the asynchronous algorithm (with continuous time) consists of the following iterated steps performed by each PE:

Step 1. Select a class according to the relative weights $\{N_b, \{n_i p_i\}_{i=1}^{10}\}$, and select a spin from the chosen class with equal probabilities.

Step 2. (a) If the chosen spin is in the kernel, flip it with probability one and go to Step 3.

(b) If the chosen spin belongs to the boundary class, *wait until* the *local* simulated time of this update becomes less than or equal to the same quantity for the corresponding neighboring PE(s). Then the state of this spin *may* or *may not* change: flip it with the usual Metropolis probability [Eq. (2)] and proceed to Step 3.

Step 3. Update the tabulation of the spin classes $\{n_i\}_{i=1}^{10}$ in the kernel.

Step 4. Determine the time of the next update (in units of MCSP) by using the *local* time increment, $\Delta t = -\ln(r)/\Gamma_s$, where

$$\Gamma_s = \frac{N_b + \sum_{i=1}^{10} n_i p_i}{N}, \tag{8}$$

and r is a uniformly distributed random number on $(0, 1)$.

Thanks to the introduction of the class of boundary spins (which has *fixed* weight), the neighboring PEs are shielded from each other, since a spin flip on the boundary of a block does not affect the tabulation of spin classes in an adjacent block. Hence, just as in the standard parallel scheme, the same conservative approach (the "wait until" directive in Step 2) ensures that the information passed between PEs is valid and the updates are correct.

Again (employing proper synchronization), one can experiment with integer–time updates. Although in this case time increments are random integers and the probability of picking two nearest-neighbor spins residing on two adjacent PEs with equal update times is small, it is nevertheless nonzero. We have already argued that these (perhaps rare) events have no corresponding analogues in the serial algorithm. Furthermore, the use of barrier synchronization severely degrades performance. Although, for "experimentation" purposes

on the architecture we implemented the integer–time, synchronous algorithm as well [18], we will not discuss its performance in detail here.

4. PERFORMANCE

We implemented both the parallel Metropolis and the parallel n-fold way algorithms (as described in the previous section) on the Cray T3E parallel architecture at the National Energy Research Scientific Computing Center (NERSC). For message passing, we employ the Cray-specific, logically shared, distributed memory access (SHMEM) routines. The fast SHMEM library supports communication initiated by *one* PE, together with remote atomic memory operations. Without these features, it would be extremely inconvenient to code an algorithm for stochastic simulation on a distributed memory machine, where the pattern of communication is completely unpredictable. These characteristics clearly outweigh the loss of portability of our code.

One-sided communications, which are the essential ingredient for efficient coding of the stochastic, massively parallel algorithm, have also been implemented in the MPI-2 library. This library is widely available and supported on various platforms, providing a high degree of portability. The reason for our choice of the proprietary SHMEM library is simply (and paradoxically) that MPI-2 was not yet implemented on the T3E at the time of our code development.

To better understand the performance of our implementation we monitored the following quantities:

Utilization. This is defined as the fraction of "non-idling" PEs, i.e., the PEs which either pick a spin in the kernel or successfully pass the "wait until" directive in Step 2 of the algorithms. Since the routines are asynchronous (the main simulation cycles on each PE are not executed in lock-step) it is fairly difficult to measure this ratio. To obtain an estimate, we placed explicit barrier synchronization in our code and performed *separate* runs. The performance was irrelevant for these runs; the only information that we aimed for was the fraction of non-idling PEs in each (now artificially lock-step) main simulation cycle. We emphasize that the utilization only measures the fraction of PEs that are not idle. It does *not* say anything about whether the active PEs are doing anything "useful" in the sense of performing successful updates.

The mean local time increment, $\overline{\Delta t}$. Although Δt in the n-fold way algorithm is not stationary over the course of the metastable decay, its mean carries important information about the degree of success by which the algorithm "bypasses" those spin-flip attempts which would be rejected if the Metropolis algorithm were used. For the parallel Metropolis algorithm $\overline{\Delta t} = 1$ trivially ($\overline{\ln(r)} = 1$).

PE update rate. This quantity is literally the simulation speed of a PE in units of standard MCS per PE per second, i.e., MCSP/s. This is an "absolute" measure and in fact determines which parallel algorithm should be used for optimal simulation speed. The full update rate of the parallel algorithm is simply (PE update rate) $\times N_{\mathrm{PE}}$. In order to compare directly the performances of the parallel algorithms to those of their serial counterparts, when the full lattice size was small enough to fit on one PE, we also ran the corresponding serial routines on one node of the T3E and determined the following measures:

Efficiency. This is the ratio of the PE update rate of the parallel algorithm to the update rate of the corresponding serial algorithm using the same full system size L. For both parallel

algorithms it is proportional to the utilization, and it is related to the communication speed of the architecture through the fraction of spin-flip attempts in which message passing is needed. For our algorithms, due to the fast communication hardware of the architecture, the communication overhead is a small effect compared to the inherent low utilization, which is a common drawback of conservative parallel methods. For the parallel n-fold way algorithm the efficiency is also proportional to the ratio of the typical time increments of the parallel and the corresponding serial routine, $\overline{\Delta t}_{\rm par}/\overline{\Delta t}_{\rm ser}$.

Speedup. This is the ratio of the (full) update rate of the parallel algorithm to the update rate of its corresponding serial counterpart, i.e., efficiency $\times\, N_{\rm PE}$.

Both efficiency and speedup are *relative* measures, and they merely indicate how successfully the parallelization of the *corresponding* serial algorithm is accomplished. As we shall see, a parallel code with lower efficiency can outperform one with higher efficiency, due to the faster "serial" core of the former. For very large systems, direct simulation using the serial algorithms is not possible, due to memory limits. The largest system sizes we could allocate were $L = 2048$ for the serial Metropolis and $L = 1280$ for the serial n-fold way algorithm. To obtain speedup and efficiency estimates for larger systems we extrapolated the smaller-system update rates of the serial routines.

Before discussing the performance of the "shielded" n-fold way algorithm, we note two inherently weak features, which are not related to the otherwise fast communication hardware of the T3E parallel architecture.

First, as a general guideline, the fewer communications one must execute, the better the performance of the parallel code. In our case, the probability of picking a spin on the boundary, which will be followed by some kind of communication between neighboring PEs, is greater than the surface-to-volume ratio, $N_{\rm b}/N = 4(l-1)/l^2 \approx 4/l$. In particular, it is determined by the relative weights in the modified n-fold way algorithm, which are given by $N_{\rm b}/(N_{\rm b} + \sum_{i=1}^{10} n_i p_i)$. With very small p_i's this ratio can take unfavorably large values, close to unity, leading to more frequent message passing and, more importantly, idling if required by the "wait until" directive.

Second, the typical time increment of the parallel algorithm is smaller than that for the serial algorithm. As mentioned in Section 2, the advantage of the serial n-fold way routine lies in the large typical time increments that correspond to the very small flipping probabilities at low temperatures. However, for the same sets of class-specific flipping probabilities p_i, the mean time increments of the "shielded" parallel and serial n-fold way routines are related as

$$\frac{1}{\overline{\Delta t}_{\rm par}} \approx \frac{N_{\rm b}}{N} + \frac{\overline{\sum_{i=1}^{10} n_i p_i}}{N_{\rm k}} \frac{N_{\rm k}}{N} \approx \frac{N_{\rm b}}{N} + \frac{1}{\overline{\Delta t}_{\rm ser}} \frac{N_{\rm k}}{N} = \frac{1}{\overline{\Delta t}_{\rm ser}} + \frac{N_{\rm b}}{N}\left(1 - \frac{1}{\overline{\Delta t}_{\rm ser}}\right). \qquad (9)$$

This follows from Eq. (8) and implies that $\overline{\Delta t}_{\rm par} < \overline{\Delta t}_{\rm ser}$ always. Thus at very low temperatures, where $\overline{\Delta t}_{\rm ser} \gg l/4$, $\overline{\Delta t}_{\rm par}$ is determined almost completely by the block size, rather than by the p_i's and the populations of the corresponding classes:

$$\overline{\Delta t}_{\rm par} \lesssim l/4. \qquad (10)$$

We test the scaling of the parallel algorithms with up to 400 PEs. First, the system size is kept constant, and we divide it into smaller and smaller blocks (Fig. 2). Then, alternatively, we keep the block size fixed and study the scaling for larger and larger systems by increasing

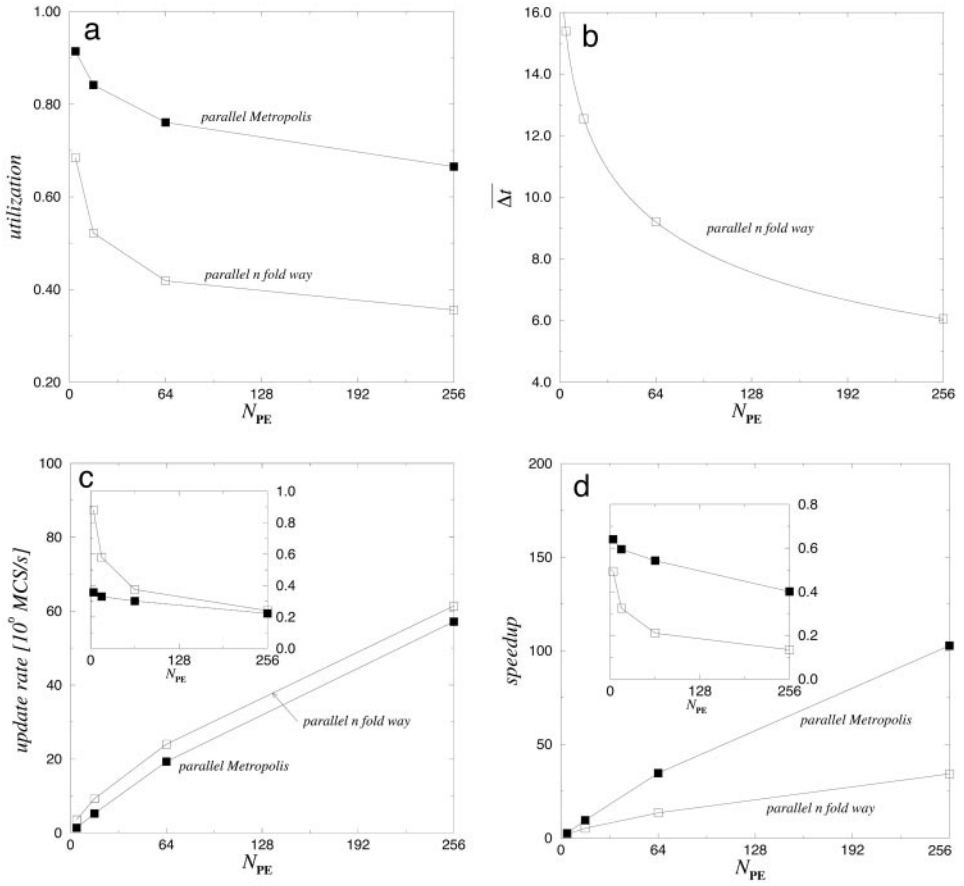

FIG. 2. Scaling and performance analysis for fixed system size, $L = 512$, at $T = 0.7T_c$ and $|H|/J = 0.2857$, comparing the parallel n-fold way algorithm (open squares) and the parallel Metropolis algorithm (filled squares). The lines connecting the data points are merely guides to the eye except in (b), where they represent the theoretical prediction of Eq. (9). (a) Utilization. (b) Mean time increment, $\overline{\Delta t}$, for the parallel n-fold way algorithm. (c) Update rate (inset: PE update rate in units of 10^6 MCSP/s). (d) Speedup (inset: efficiency).

the number of blocks (Fig. 3). We also carry out experiments with fixed number of PEs and varying block size (Fig. 4) to study in detail the effect of increasing the surface-to-volume ratio of the blocks. Each of these studies was performed at $T = 0.7T_c$ and $|H|/J = 0.2857$, where the typical droplet separation, R_o, is approximately 32 lattice constants. Finally, with fixed number of PEs and fixed block size we study the effect of the temperature and magnetic field on the performance (Fig. 5), to determine the regime of efficient applications to our particular model system. The results reflect the features discussed in the previous paragraph.

4.1. *Scaling with N_{PE} for Fixed System Size*

For this series of runs we choose $L = 512$ for the total system size and we employ $N_{\mathrm{PE}} = 4, 16, 64$, and 256 (corresponding to $l = 256, 128, 64$, and 32, respectively). For both the parallel n-fold way and the parallel Metropolis algorithm the utilization drops with decreasing block size (Fig. 2a). The utilization for the n-fold way routine is significantly lower than that for Metropolis: the probability of choosing a spin on the boundary is greater than the surface-to-volume ratio and then it is ultimately followed by an inquiry of the

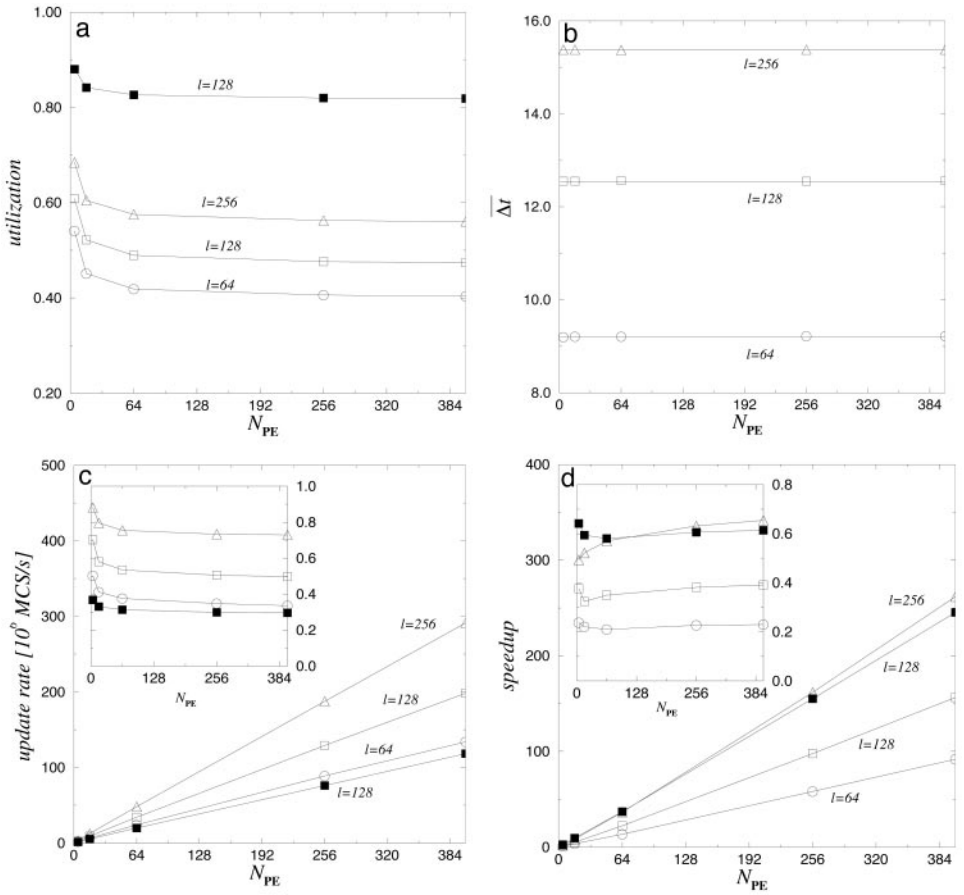

FIG. 3. Scaling and performance analysis for three different block sizes, $l = 64$, 128, and 256 for the parallel n-fold way algorithm (open circles, squares, and triangles, respectively), and $l = 128$ for the parallel Metropolis algorithm (filled squares) at $T = 0.7T_c$ and $|H|/J = 0.2857$. The linear system size is $L = l\sqrt{N_{PE}}$. The lines connecting the data points are merely guides to the eye except in (b), where they represent the theoretical prediction of Eq. (9). (a) Utilization. (b) Mean time increment, $\overline{\Delta t}$, for the parallel n-fold way algorithm. (c) Update rate (inset: PE update rate in units of 10^6 MCSP/s). (d) Speedup (inset: efficiency).

local time of the neighbor(s) and possible idling. Further, the typical time increments are decreasing as well (Fig. 2b). Note that for the *serial* n-fold way routine the mean time increment is $\overline{\Delta t}_{ser} = 19.9$ (Table I). This is the only parameter in Eq. (9) which gives complete agreement with the measured values of $\overline{\Delta t}_{par}$, as shown by the solid curve in

TABLE I

Mean Time Increments (in MCSP) for the Serial and the Parallel n-Fold Way Algorithms with Different Block Sizes l at $T = 0.7T_c$, $|H|/J = 0.2857$[a]

	Parallel n-fold way with block size l							Serial
l	16	32	64	128	256	512	1024	[b]
$\overline{\Delta t}$	3.7	6.1	9.2	12.6	15.4	17.4	18.5	19.9

[a] They are approximately independent of the full system size L and N_{PE}.

[b] The mean time increment for the serial algorithm is approximately independent of L.

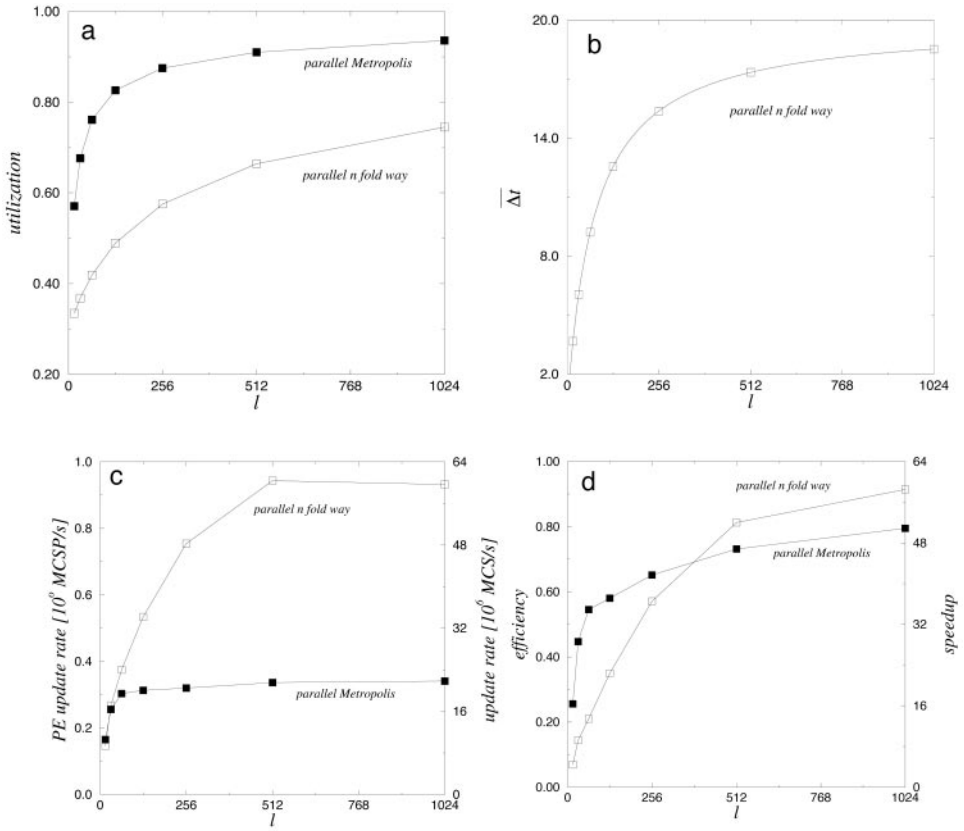

FIG. 4. Performance analysis for fixed number of PEs, $N_{\rm PE} = 64$, as a function of the block size, l, for the parallel n-fold way algorithm (open squares), and for the parallel Metropolis algorithm (filled squares) at $T = 0.7T_c$ and $|H|/J = 0.2857$. The lines connecting the data points are merely guides to the eye except in (b), where they represent the theoretical prediction of Eq. (9) (a) Utilization. (b) Mean time increment, $\overline{\Delta t}$, for the parallel n-fold way algorithm. (c) Update rate (right scale) and PE update rate (left scale). (d) Speedup (right scale) and efficiency (left scale).

Fig. 2b. As a result of these factors, the PE update rate of the parallel n-fold algorithm drops faster than that of the parallel Metropolis algorithm (Fig. 2c inset). For sufficiently small blocks, the performance of the n-fold routine actually becomes poorer than that for the Metropolis routine, as we shall see in the experiments with fixed number of PEs and varying block size. For both algorithms the scaling is systematically *worse* than linear, for the reasons explained above (Fig. 2c). In particular, the parallel n-fold way for large number of PEs (small block size l) cannot scale better than $\sqrt{N_{\rm PE}}$, since for fixed L the typical time increment scales as $\overline{\Delta t}_{\rm par} \lesssim l/4 \sim 1/\sqrt{N_{\rm PE}}$. Although employing many *small* blocks does result in speedup, the efficiency clearly becomes poorer at the same time (Fig. 2d).

For completeness, we mention that the performance of the integer–time, synchronous n-fold way routine becomes progressively weaker than that of the asynchronous version with continuous time for an increasing number of PEs. For example, with 256 PEs it runs 2.3 times slower than the continuous-time routine, due to the necessity of explicit barrier synchronizations. It is even slower than the asynchronous Metropolis routine by a factor of 2.1.

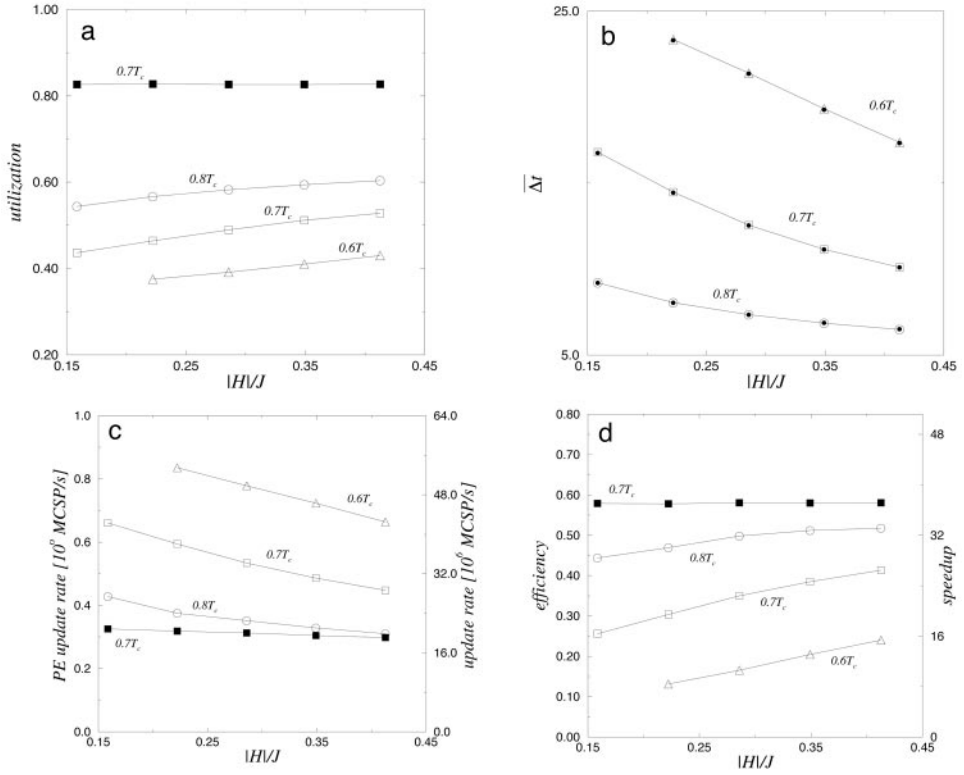

FIG. 5. Performance analysis for three different temperatures, $T = 0.8T_c$, $0.7T_c$, and $0.6T_c$ for the parallel n-fold way algorithm (open circles, squares, and triangles, respectively), and $T = 0.7T_c$ for the parallel Metropolis algorithm (filled squares) as functions of the magnetic field. We employ $N_{\rm PE} = 64$ and $l = 128$ ($L = 1024$). The lines connecting the data points are merely guides to the eye. (a) Utilization. (b) Mean time increment, $\overline{\Delta t}$, for the parallel n-fold way algorithm. Filled circles indicate the theoretical predictions of Eq. (9). (c) Update rate (right scale) and PE update rate (left scale). (d) Speedup (right scale) and efficiency (left scale).

4.2. *Scaling with $N_{\rm PE}$ for Fixed Block Size*

Here we keep the size of the blocks fixed (using three different values: $l = 64$, 128, 256), while increasing the system size by using larger numbers of PEs ($N_{\rm PE} = 4$, 16, 64, 256, and 400). Clearly, the larger the block size, the higher the utilization. Even with fixed l, the utilization monotonically decreases with increasing $N_{\rm PE}$. However, it appears to approach a *nonzero* value for large numbers of PEs (Fig. 3a). As was pointed out in Ref. [2], it is a highly non-trivial mathematical problem to prove that the utilization tends to a nonzero value as $N_{\rm PE} \to \infty$. For practical purposes, given our architecture with 512 nodes this question might seem academic, but it truly lies at the heart of this conservative approach to discrete-event simulation. Again, the utilization for the parallel Metropolis algorithm significantly exceeds that for the n-fold way: for the Metropolis routine with $l = 128$ it is 82% with 400 PEs, while for the n-fold way it is only 56%, even for $l = 256$. With fixed l, the mean time increments are independent of $N_{\rm PE}$ in the parallel n-fold way routine (Fig. 3b). However, increasing l systematically improves $\overline{\Delta t}$ and thus the performance. We find almost *linear* scaling of the update rate with $N_{\rm PE}$ for both parallel algorithms (Figs. 3c, d). Despite its relatively low utilization, the n-fold way routine clearly outperforms the Metropolis routine, due to its large time increments (partly rejection-free nature).

Again we note that our implementation of the integer–time synchronous n-fold way algorithm performs rather poorly compared to its asynchronous counterpart: it runs 2.3 times slower than the asynchronous n-fold way and 1.3 times slower than the asynchronous Metropolis routine with $l = 128$ and 256 PEs.

4.3. *Simulation with Fixed Number of PEs and Varying Block Size*

Here $N_{\mathrm{PE}} = 64$, which can be regarded as a reasonable number for production runs. We increase the block size ($l = 16$, 32, 64, 128, 256, 512, and 1024) and "stretch" the routine close to its memory limits. For both algorithms the utilization increases with increasing block size. This happens faster for the parallel Metropolis routine, for which the utilization is more directly related to the surface-to-volume ratio of the blocks (Fig. 4a): for $l = 1024$ it is 93% while it is 74% for the n-fold way routine. Further, the mean time increment for the parallel n-fold way algorithm systematically approaches that of its serial counterpart, $\overline{\Delta t}_{\mathrm{ser}} = 19.9$ (Fig. 4b and Table I). For large l such that $l/4 \gg \overline{\Delta t}_{\mathrm{ser}}$, $\overline{\Delta t}_{\mathrm{par}} \lesssim \overline{\Delta t}_{\mathrm{ser}}$, as expected from Eq. (9). The data are in complete agreement with our theoretical result, given by the solid curve. Here the temperature and field are moderate, so that the increasing block size l allows $\overline{\Delta t}_{\mathrm{par}}$ to "catch up" with $\overline{\Delta t}_{\mathrm{ser}}$. The drawback is that employing large blocks can lead to excessive memory usage. The PE update rate of the n-fold way routine with $l = 1024$ slightly decreases compared to $l = 512$ (Fig. 4c), even though the utilization and the mean time increment slightly increase. In fact, $l = 1024$ is fairly close to the largest block size we could allocate in the memory of one node. For very small l ($l \leq 16$), the utilization and $\overline{\Delta t}$ of the n-fold way routine are so low that the routine is actually outperformed by the parallel Metropolis routine (Fig. 4c).

4.4. *Simulation with Fixed Number of PEs and Varying Physical Control Parameters*

Here $N_{\mathrm{PE}} = 64$ and $l = 128$ are kept constant while we vary the magnetic field for three different temperatures to study the effects of changing the characteristic length and time scales of the simulated system. As expected, the utilization for the Metropolis algorithm is *unaffected* (Fig. 5a); the slight increase in the PE update rate at lower fields (Fig. 5c) or temperatures is due to the fact that for higher rejection rates fewer variables need to be updated. The parallel n-fold way routine suffers from low utilization at low temperatures and fields: in a high percentage of the execution time of the main simulation cycle, a spin on the boundary is picked. This not only implies necessary communications with its neighboring PE(s), but possible idling if the local times of the neighboring blocks are behind. Although the time increments increase with decreasing temperature and field, they cannot exceed $l/4$, as shown in Eq. (10) (Fig. 5b). Nevertheless, even with the bounded time increments, the parallel n-fold way routine outperforms the parallel Metropolis routine (Fig. 5c). The drawback of the parallel (partially rejection-free) n-fold way scheme is the low efficiency with respect to its *serial* counterpart (Fig. 5d); for example, at $T = 0.6T_c$ and $|H|/J = 0.2222$ it is only 13.2%, corresponding to a speedup of only 8.4. For comparison, the mean time increments of the serial n-fold way routine are also given in Table II.

5. APPLICATIONS

As discussed in Section 2, below the equilibrium critical temperature the kinetic Ising system exhibits metastable decay after an instantaneous magnetic field reversal from

TABLE II

Mean Time Increments (in MCSP) for the Serial and the Parallel *n*-Fold Way Algorithms for Different Temperatures and Magnetic Fields ($N_{PE} = 64$, $l = 128$)

			$\lvert H\rvert/J$				
			0.1587	0.2222	0.2857	0.3492	0.4127
T/T_c	0.6	Serial	—	81.5	61.4	46.4	36.3
		Parallel	—	23.4	21.4	19.3	17.4
	0.7	Serial	33.8	25.4	19.9	16.5	14.3
		Parallel	16.8	14.5	12.6	11.1	10.1
	0.8	Serial	12.5	10.4	9.2	8.5	7.9
		Parallel	9.2	8.0	7.4	6.9	6.5

$|H|$ to $-|H|$. Using standard droplet theory [10], one can show that a thermally nucleated domain of the stable phase must reach a critical droplet size, corresponding to a (temperature and field dependent) critical radius R_c, before its growth becomes energetically favorable. For systems significantly larger than the typical droplet separation R_o (which decreases in a nonlinear fashion with increasing $|H|$ and T), many droplets of the stable phase form and grow until they coalesce and occupy the whole system (multi-droplet (MD) regime) [10, 19]. The parameters we choose correspond to this decay mode. Note that $R_c \ll R_o$, in particular, $R_c/R_o \to 0$ in the $|H| \to 0$ limit. A quantity of interest is the lifetime of the metastable phase, $\langle\tau\rangle$, which is defined as the average first-passage time to zero magnetization. Exploiting the fact that the system is self-averaging in this regime, one may employ the classical Avrami law for homogeneous systems [20] in two dimensions to obtain an analytical form for the time evolution of the system magnetization [10, 19],

$$m(t) \approx (m_{\rm ms} - m_{\rm s})\phi_{\rm ms}(t) + m_{\rm s}, \tag{11}$$

where

$$\phi_{\rm ms}(t) = e^{-(\ln 2)(t/\langle\tau\rangle)^3} \tag{12}$$

is the volume fraction of the metastable phase, and $m_{\rm ms}$ and $m_{\rm s}$ are the metastable and the stable (equilibrium) magnetization, respectively.

For illustration we choose an $L = 1024$ system at $T = 0.7T_c$ and $|H|/J = 0.2857$, and we study metastable decay as observed through a $b \times b$ window. Employing different block sizes, b, mimics the effect of choosing different finite (smaller than the system size) observation windows in a large system. This is clearly relevant to real experiments and observations, such as those using the magnetooptical Kerr effect [21]. We run our application with $N_{\rm PE} = 16, 64$, and 256, directly corresponding to block sizes $b = l = 256, 128$, and 64, respectively. Since the maximum number of PEs available to us is 512, for $b \le 32$ (for the same $L = 1024$ system) we employ 256 PEs with $l = 64$, and monitor observables for a $b \times b$ block within the 64×64 subsystem carried by each PE. Even our smallest block, $b = 8$, is sufficiently large that subcritical fluctuations of size $R < R_c \approx 2$ are not recorded as first-passage times to a block magnetization of zero. Together with the global magnetization,

Eq. (4) we monitor the individual block magnetizations

$$m_b = \frac{1}{b^2} \sum_{i=1}^{b^2} s_i. \tag{13}$$

Typical time series of these quantities are shown in Fig. 6, where time is measured in units of MCS/spin (MCSS). Recording the first-passage times to zero for at least 100 escapes, we calculate the mean and the standard deviation of the lifetime (Fig. 7a) and construct histograms for $P^b_{\rm not}(t)$, the probability that the system or block magnetization has not changed sign by time t (Fig. 7b).

The *average* lifetime of a finite subsystem, as observed through a finite window, *does not* differ significantly from the lifetime of the whole system ($\langle\tau\rangle = 158.5$ MCSS). However, the statistical properties of the lifetime change not only quantitatively, but also qualitatively with b. For b much larger than the typical droplet separation ($R_o \approx 32$ lattice constants at this temperature and field), the time-dependent block magnetization itself is self-averaging, the switching process is Gaussian, and consequently $P^b_{\rm not}(t)$ is an error function [10]. While the average lifetime within a block is unaffected, its standard deviation, σ_b, is proportional to $1/b$ in two dimensions. Once b becomes comparable to or smaller than R_o, the above picture is no longer valid and a crossover to a qualitatively different behavior is observed. For these smaller blocks, the switching within a block is not related to the growth of several droplets nucleated within that block, but rather to a single droplet formed within the same block, or to droplets formed elsewhere in the system which propagate across the observed block. It is a challenging theoretical problem to describe the switching behavior analytically in this crossover regime, and work is in progress to accomplish this task. In the $b/R_o \to 0$ limit the coarse-grained approximation (in which the size of the critical droplet is negligible) yields $P^o_{\rm not}(t) \approx \phi_{\rm ms}(t)$ [Eq. (12)], since the probability that the block magnetization has not switched by time t then becomes equal to the volume fraction of the metastable phase. The standard deviation in this limit can be obtained from the probability density of the first-passage time, $-dP^o_{\rm not}/dt$, yielding $\sigma_o \approx 0.1345\langle\tau\rangle \approx 58.12$ MCSS. Note, however, that for b smaller than the diameter of the critical droplet ($2R_c \approx 4$ lattice constants for our parameters) the above argument (which is based on the coarse-grained picture) is no longer valid, since zero-crossings of the block magnetization are frequently induced by subcritical fluctuations.

Another future application in the MD regime is to study the response to a periodic applied magnetic field, which exhibits highly nontrivial hysteretic behavior [22]. If the half-period of the applied field is less than the metastable lifetime $\langle\tau\rangle$, the system almost always does not switch, resulting in a nonzero period-averaged magnetization ("dynamically ordered phase"). On the other hand, when the half-period exceeds $\langle\tau\rangle$, the magnetization switches in almost every half-period and the period-averaged magnetization is zero ("dynamically disordered phase"). The transition between these two phases is sharp and singular: the system exhibits a *dynamic* phase transition, which fits into the general framework of critical phenomena and continuous phase transitions [23]. Here there is clearly a need to study scaling and universality by obtaining the corresponding critical exponents with high accuracy. Also in this regime, our parallel algorithm appears to be very efficient for large systems. We plan to implement it with a periodic square-wave shaped applied magnetic field and carry out a large-scale finite-size scaling analysis of the dynamic phase transition.

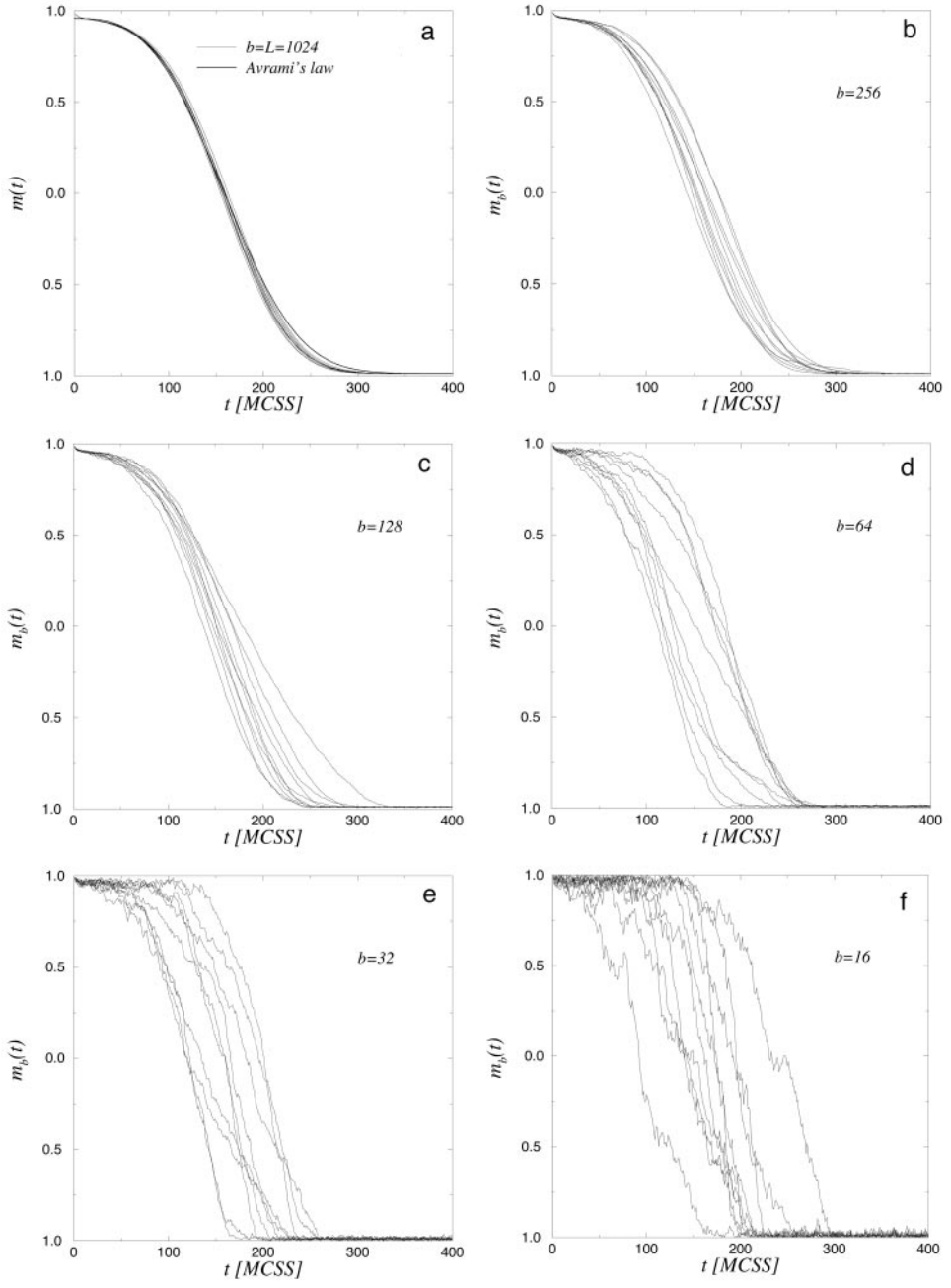

FIG. 6. Global and block magnetization time series for 10 different realizations at $T = 0.7T_c$ and $|H|/J = 0.2857$. (a) For $b = L = 1024$ (global). The thick solid line, which is difficult to distinguish from the simulation results, represents Avrami's law [Eqs. (11) and (12)]. (b) For $b = 256$. (c) For $b = 128$. (d) For $b = 64$. (e) For $b = 32$. (f) For $b = 16$.

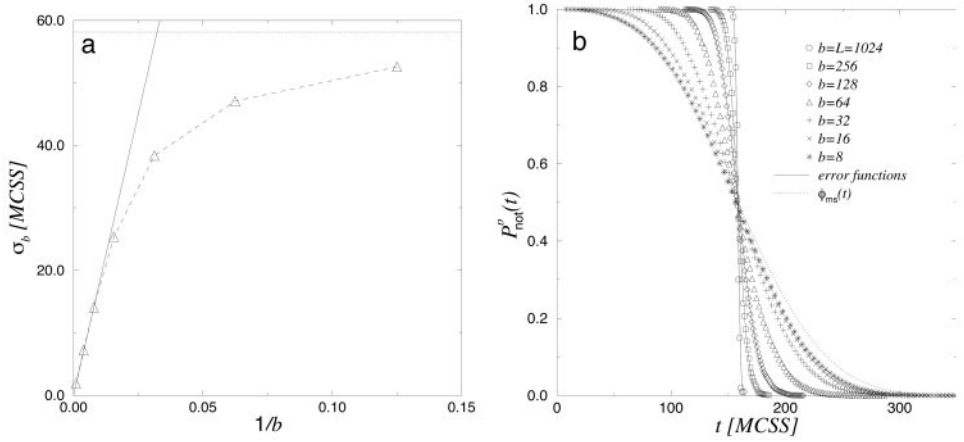

FIG. 7. (a) Standard deviation of the first-passage time of the block magnetization to zero (open triangles) as a function of the inverse block size, $1/b$. The dashed line is merely a guide to the eye. The solid straight line represents the theoretical dependence on b of the standard deviation for large b. The dotted horizontal line is the $b/R_o \to 0$ limit of the standard deviation ($\sigma_o = 58.12$ MCSS) within the coarse-grained approximation. (b) The probability that the block magnetization, observed through a $b \times b$ window, has not changed sign by time t after a sudden magnetic field reversal in a 1024×1024 Ising system. The solid curves for $b = 1024$, 256, and 128 are scaled error functions, given by theory [10] for large b. The dotted curve for $b = 8$ represents the theoretical coarse-grained $b/R_o \to 0$ limit, i.e., the metastable volume fraction, $\phi_{\rm ms}(t)$ [Eq. (12)], given by Avrami's law. As expected, it fits the data very well for $t < \langle\tau\rangle$, where droplet coalescence is unimportant.

6. CONCLUSION

We have studied the performance of the parallel n-fold way dynamic Monte Carlo algorithm proposed by Lubachevsky [2], in which each PE carries an $l \times l$ block of random variables. The algorithm was implemented for a two-dimensional kinetic Ising ferromagnet undergoing metastable decay, but the parallel scheme is generically applicable to a wide range of stochastic cellular automata where discrete events (updates) are Poisson arrivals.

One may clearly ask why not implement "trivial" or "embarrassing" parallelization where one uses the serial algorithm and simply averages over independent parallel runs on various processors at the end. This approach is obviously hard to beat in terms of programming effort and utilization. However, fitting a very large system in the memory of a computer *without* degrading the performance requires special hardware, e.g., extended cache or disks with fast remote memory access. For example, for the serial n-fold way algorithm the largest system we could allocate on one node of the T3E was $L = 1280$, for which the performance (PE update rate) was approximately 57% of that of the $L = 64$ case. Even for $L = 512$, in which case the memory is far from being exhausted, the performance was already degraded to 65%, compared to that of the $L = 64$ case. Thus, massive stochastic parallelization provides a fast alternative to special hardware for simulating very large systems.

To obtain reasonable performance on the T3E distributed-memory parallel architecture and to be faithful to the original dynamics, one must utilize an asynchronous update scheme with continuous time. Then the expensive global barrier synchronizations are avoided and spin-flip attempts are modeled as independent Poisson arrivals. We analyzed the performance of our implementation, which sensitively depends on the block size and the number of PEs, as well as on the characteristic length and time scales of the simulated system. We found that for large enough block size, the routine outperforms the standard parallel

Metropolis algorithm. For moderately low temperatures it yields high speedups with respect to the already fast serial n-fold way algorithm. For example, at $T = 0.7T_c$ and $|H|/J = 0.2857$, employing $l = 256$ we obtained a speedup of 260 with 400 PEs, and for $l = 1024$ a speedup of 58 with 64 PEs. Often the system size (possibly large) is specified, and for fixed L, although significantly worse than linear, the speedup is still a monotonically increasing function of the number of PEs, up to the maximum 256 PEs that we studied. At the same time, the efficiency is monotonically decreasing, which results in larger *total* CPU time usage to execute the same task with a larger number of PEs. If one has unlimited resources (i.e., no allocation limits) this aspect is not relevant. For most, like us, who have limited CPU resources on a certain parallel architecture, "optimization" between speedup and efficiency can be important. Our implementation is obviously best suited to simulating large systems.

On the other hand, for very low temperatures, the algorithm does not provide an efficient way to simulate metastable decay. The reason for the relatively narrow regime of efficient implementation lies in the introduction of a special class in the n-fold way algorithm which "shields" the blocks from each other, but significantly decreases the typical time increments. The algorithm avoids rollbacks, but pays a large price: it loses the arbitrarily large time increments that are the most important feature of the serial n-fold way algorithm, at arbitrarily low temperature and field. To obtain reasonable efficiency compared to the efficiency of the serial n-fold way algorithm, one needs to employ large blocks such that $l/4 \gtrsim \overline{\Delta t}_{\mathrm{ser}}$, and clearly it is impossible to keep up with very large serial time increments by increasing l.

One way to preserve the advantage of the original n-fold way algorithm in principle would be to apply it directly on each block (optimistic approach). This would require a complex protocol to correct erroneous computations. Such a rollback procedure would ensure the correct time ordering of simulated events. This mechanism is not unknown in distributed event simulation [24] and it certainly has some potential. The complexity of such an implementation, however, might carry a tremendous overhead with respect to the very simple and fast serial algorithm for the Ising model. Another possible way to improve efficiency, while avoiding a general rollback procedure, is to consider relaxation [25] which might use local speculative computations before scheduling an event [26].

ACKNOWLEDGMENTS

Special thanks to B. D. Lubachevsky and M. Kolesik for invaluable discussions and to R. Gerber of the National Energy Research Scientific Computing Center consulting group for important hints on debugging the parallel code. Helpful discussions with S. W. Sides, S. J. Mitchell, and G. Brown are also gratefully acknowledged. This research was supported in part by the Florida State University Supercomputer Computations Research Institute, which is supported by the U.S. Department of Energy under Contract DE-FC05-85ER25000; by the NSF under Grants DMR-9520325 and DMR-9871455; and by the Florida State University Center for Materials Research and Technology. This research used resources of NERSC, which is supported by the Office of Energy Research of the U.S. DOE under Contract DE-AC03-76SF00098.

REFERENCES

1. R. M. Fujimoto, Parallel discrete event simulation, *Commun. ACM* **33**, 30 (1990).
2. B. D. Lubachevsky, Efficient parallel simulations of asynchronous cellular arrays, *Complex Systems* **1**, 1099 (1987); Efficient parallel simulations of dynamic Ising spin systems, *J. Comput. Phys*. **75**, 103 (1988).

3. A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, A new algorithm for Monte Carlo simulation of Ising spin systems, *J. Comput. Phys*. **17**, 10 (1975).
4. M. A. Novotny, A new approach to an old algorithm for the simulation of Ising-like systems, *Comput. Phys*. **9**, No. 1, 46 (1995); Monte Carlo algorithms with absorbing Markov chains: Fast local algorithms for slow dynamics, *Phys. Rev. Lett*. **74**, 1 (1995); Erratum **75**, 1424 (1995).
5. A. Aharoni, *Introduction to the Theory of Ferromagnetism* (Clarendon, Oxford, 1996).
6. H. M. Duiker and P. D. Beale, Grain-size effects in ferroelectric switching, *Phys. Rev. B* **41**, 490 (1990); P. D. Beale, Comparison of classical nucleation theories with Monte Carlo simulations of Ising models, *Integrated Ferroelectrics* **4**, 107 (1994).
7. P. A. Rikvold *et al.*, Computational lattice–gas modeling of the electrosorption of small molecules and ions, *Surf. Sci*. **335**, 389 (1995).
8. A. Cheng and M. Caffrey, Interlamellar transition mechanism in model membranes, *J. Phys. Chem*. **100**, 5608 (1996).
9. M. Kolesik, M. A. Novotny, P. A. Rikvold, and D. M. Townsley, Projected dynamics for metastable decay in Ising model, in *Computer Simulations in Condensed Matter Physics X*, edited by D. P. Landau, K. K. Mon, and H.-B. Schüttler (Springer-Verlag, Berlin, 1998), p. 246; M. Kolesik, M. A. Novotny, and P. A. Rikvold, Projection method for statics and dynamics of lattice spin systems, *Phys. Rev. Lett*. **80**, 3384 (1998); M. A. Novotny, M. Kolesik, and P. A. Rikvold, Slow forcing in the projected dynamics method, *Comput. Phys. Commun.* (Proceedings, 1998 Conference on Computational Physics (CCP '98), Granada, Spain), to appear; cond-mat/9811039, preprint, 1998.
10. P. A. Rikvold, H. Tomita, S. Miyashita, and S. W. Sides, Metastable lifetimes in a kinetic Ising model: Dependence on field and system size, *Phys. Rev. E* **49**, 5080 (1994); H. L. Richards, S. W. Sides, M. A. Novotny, and P. A. Rikvold, Magnetization switching in nanoscale ferromagnetic grains: Description by a kinetic Ising model, *J. Magn. Magn. Mater*. **150**, 37 (1995); M. A. Novotny, M. Kolesik, and P. A. Rikvold, Magnetization switching in single-domain ferromagnets, *J. Magn. Magn. Mater*. **177**, 917 (1998).
11. P. A. Rikvold, M. A. Novotny, M. Kolesik, and H. L. Richards, Nucleation theory of magnetization reversal in nanoscale ferromagnets, in *Dynamical Properties of Unconventional Magnetic Systems*, edited by A. T. Skjeltorp and D. Sherrington (Kluwer, Dordrecht, 1998), p. 307; M. Kolesik, M. A. Novotny, and P. A. Rikvold, Monte Carlo simulation of magnetization switching in Fe sesquilayers on W(110), *Phys. Rev. B* **56**, 11790 (1997).
12. R. Friedberg and J. E. Cameron, Test of the Monte Carlo method: Fast simulation of a small Ising lattice, *J. Chem. Phys*. **52** 6049 (1970); L. Jacobs and C. Rebbi, Multi spin coding: A very efficient technique for Monte Carlo simulations of spin systems, *J. Comput. Phys*. **41**, 203 (1981).
13. R. H. Swendsen and J.-S. Wang, Nonuniversal critical dynamics in Monte Carlo simulations, *Phys. Rev. Lett*. **58**, 86 (1987); U. Wolff, Collective Monte Carlo updating for spin systems, *Phys. Rev. Lett*. **62**, 361 (1989).
14. L. Chayes and J. Machta, Graphical representations and cluster algorithms. I. *Physica A* **239**, 542 (1997); II. *Physica A* **254**, 477 (1998).
15. K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics*, 3rd ed. (Springer-Verlag, Berlin, 1997).
16. A. G. Greenberg, B. D. Lubachevsky, D. M. Nicol, and P. E. Wright, Efficient massively parallel simulation of dynamic channel assignment schemes for wireless cellular communications, in *Proceedings, 8th Workshop on Parallel and Distributed Simulation (PADS '94), Edinburgh, UK, 1994*, p. 187.
17. B. D. Lubachevsky, V. Privman, and S. C. Roy, Casting pearls ballistically: Efficient massively parallel simulation of particle deposition, *J. Comput. Phys*. **126**, 152 (1996).
18. G. Korniss, G. Brown, M. A. Novotny, and P. A. Rikvold, Hard simulation problems in modeling of magnetic materials: Parallelization and Langevin micromagnetics, in *Computer Simulation Studies in Condensed Matter Physics XI*, edited by D. P. Landau and H.-B. Schüttler (Springer-Verlag, Berlin, in press); cond-mat/9803118, preprint, 1998.
19. R. A. Ramos, P. A. Rikvold, and M. A. Novotny, Test of the Kolmogorov–Johnson–Mehl–Avrami picture of metastable decay in a model with microscopic dynamics, *Phys. Rev. B* **59**, 9053 (1999).
20. A. N. Kolmogorov, *Bull. Acad. Sci. USSR, Phys. Ser*. **1**, 355 (1937); W. A. Johnson and R. F. Mehl, Reaction kinetics in processes of nucleation and growth, *Trans. Am. Inst. Mining Metall. Eng*. **135**, 416 (1939);

M. Avrami, Kinetics of phase change. I. General theory, *J. Chem. Phys*. **7**, 1103 (1939); II. Transformation-time relations for random distribution of nuclei, *J. Chem. Phys*. **8**, 212 (1940); III. Granulation, phase change, and microstructure, *J. Chem. Phys*. **9**, 177 (1941).

21. S. Lemerle, J. Ferré, C. Chappert, V. Mathet, T. Giamarchi, and P. Le Doussal, Domain wall creep in an Ising ultrathin magnetic film, *Phys. Rev. Lett*. **80**, 849 (1998).
22. S. W. Sides, P. A. Rikvold, and M. A. Novotny, Kinetic Ising model in an oscillating field: Finite-size scaling at the dynamic phase transition, *Phys. Rev. Lett*. **81**, 834 (1998); Kinetic Ising model in an oscillating field: Avrami theory for the hysteretic response and finite-size scaling for the dynamic phase transition, *Phys. Rev. E* **59**, 2710 (1999).
23. N. Goldenfeld, *Lectures on Phase Transitions and the Renormalization Group* (Addison–Wesley, Reading, MA, 1992).
24. D. R. Jefferson, Virtual time, *ACM Trans. Prog. Lang. Syst*. **7**, 404 (1985).
25. B. D. Lubachevsky, Relaxation for massively parallel discrete event simulation, in *Proceedings, Performance '93 Conf.*, edited by L. Donatiello and R. Nelson, Lecture Notes in Computer Science (Springer-Verlag, Berlin, 1993), Vol. 729, p. 307.
26. B. D. Lubachevsky, private communication.