Distributed Event-driven Simulation- Scheduling Strategies and Resource Management

Overeinder, B.J.

Publication date
2000

Chapter 5

Parallel Asynchronous Cellular Automata

Monte Carlo Method [Origin: after Count Montgomery de Carlo, Italian gambler and random-number generator (1792–1838).] A method of jazzing up the action in certain statistical and number-analytic environments by setting up a book and inviting bets on the outcome of a computation.

—S. Kelly-Bootle, The Computer Contradictionary

5.1 Introduction

Many fundamental problems from natural sciences can be modeled as a *dynamic complex system*. A dynamic complex system is defined to be a set of unique elements with well defined microscopic attributes and interactions, showing emerging macroscopic behavior. This emergent behavior can, in general, not be predicted from the individual elements and their interactions. In many cases, dynamic complex systems cannot be solved by analytical methods, but require explicit simulation of the system dynamics to obtain insight into the system. A distinguished computational solving method for a large class of dynamic complex systems are (synchronous) *cellular automata* (CA). The CA model is in itself a set of dynamic systems where space, time, and variables are discrete. Cellular automata exhibit remarkable self-organization that can be used in models for real-world systems. For instance, the CA technique has proven to be useful for direct simulation of fluid flow experiments in both two and three dimensions. Other applications of CA in natural sciences can be found in lattice spin models such as the Ising model, or, in biology for example, in immune deficiency in cancer tissue simulations.

A commonly made assumption is that the update strategy of a CA is synchronous, i.e., the CA system evolves in discrete time. In the *asynchronous cellular automata* (ACA) model the synchronous cell update is relaxed to allow for independent, asynchronous cell updates. The asynchronous update strategy results in a more generic approach to CA, where also continuous-time models can be described conveniently (Bersini and Detours 1994; Lumer and Nicolis

1994).

The APSIS simulation environment is designed to effectively support the parallel execution of dynamic complex systems, and in particular spatially decomposed asynchronous cellular automata. The use of PDES to solve this class of problems is a challenge as the problem sizes in terms of memory and computation time can fill any parallel supercomputer. To validate the design and assess the practical usability of optimistic simulation methods in ACA models, we designed and implemented a well-defined and well-understood problem, namely the Ising spin model. The fairly simple Ising spin model shows a complex behavior that is parameterized by essentially one degree of freedom. The different behavior characterization of the Ising spin model put also different requirements to the PDES simulation kernel, and is as such an ideal vehicle for the validation and assessment of the APSIS environment. For example, the computational load of the Ising spin model is easily adjusted by the problem size, and the communication intensity of the simulation application is determined by a single model parameter. Besides the characteristic computational load and communication intensity, the Ising spin model also exhibits long-range correlations for certain model parameter ranges, where these long-range correlations induce time-dependent (evolution of the) computational behavior. One of the first questions that is raised is how the PDES protocol behaves with respect to this class of asynchronous systems. What are the limitations of the application of Time Warp to spatial decomposed regular problems, and in what way is the execution behavior influenced by the application parameters that determine the spatial interaction (synchronization) and computational load over the parallel processes?

The asynchronous cellular automata model will be introduced in Section 5.2, including the transition from synchronous to asynchronous cellular automata update strategy. In Section 5.3 the Ising spin model is presented and the Monte Carlo simulation method is briefly explained. Further, the asynchronous, continuous-time Ising spin model is introduced. The design and parallel implementation of the continuous-time Ising spin model is presented in Section 5.4, and the results of performance and scalability experiments are reported in Section 5.5.

5.2 Asynchronous Cellular Automata

5.2.1 Cellular Automata

Cellular automata are *discrete*, *decentralized*, and *spatially extended systems* consisting of large numbers of simple identical components with local connectivity. The meaning of discrete here is, that space, time, and features of an automaton can have only a finite number of states. The rational of cellular automata is not to try to describe a dynamic complex system from a global point of view as it is described using for instance differential equations, but modeling this system starting from the elementary dynamics of its interacting parts.

In other words, not to describe a complex system with complex equations, but let the complexity emerge by interaction of simple individuals following simple rules. In this way, a physical process may be naturally represented as a computational process and directly simulated on a computer. The original concept of cellular automata was introduced by von Neumann and Ulam to model biological reproduction and crystal growth respectively (von Neumann 1966; Ulam 1970). Since then it has been applied to model a wide variety of (complex) systems, in particular physical systems containing many discrete elements with local interactions. Cellular automata have been used to model fluid flow, galaxy formation, biological pattern formation, avalanches, traffic jams, parallel computers, earthquakes, and many more. In these examples, simple microscopic rules lead to macroscopic emergent behavior.

The locality in the cellular automata rules facilitate parallel implementations based on domain decomposition. The locality combined with the inherent parallelism of cellular automata make the design and development of high-performance software environments possible on parallel architectures. These environments exploit the inherent parallelism of the CA model for efficient simulation of complex systems modeled by a large number of simple elements with local interactions. By means of these environments, cellular automata have been used recently to solve complex problems in many fields of science, engineering, computer science, and economy. In particular, *parallel* cellular automata models are successfully used in fluid dynamics, molecular dynamics, biology, genetics, chemistry, road traffic flow, cryptography, image processing, environmental modeling, and finance (Talia and Sloot 1999).

5.2.2 Asynchronous Cellular Automata

In the previous section, we have seen the dualistic functionality of cellular automata in modeling and simulation. From a *modelers perspective*, a CA model allows the formulation of a dynamic complex system (DCS) application in simple rules. From a *computer simulation perspective*, a CA model provides an execution mechanism that evaluates the temporal dynamic behavior of a DCS given these simple rules. An important characteristic of the CA execution mechanism is the particular *update scheme* that applies the rules iteratively to the individual cells of the CA. The different update schemes impose a distinct temporal behavior on the model. Thus we must select the proper update mechanism that aligns with the dynamics of the model.

The update mechanism of CAs is described as being synchronously in parallel. However, for certain classes of DCS, the temporal dynamic behavior is asynchronous. In particular, systems with heterogeneous spatial and temporal behavior are, in general, most exactly mapped to asynchronous models (Bersini and Detours 1994; Lumer and Nicolis 1994). In case asynchronous models are solved by CA, the asynchronous temporal behavior must be captured by the update mechanism. This class of CA is called asynchronous cellular automata (ACA) (Ingerson and Buvel 1984; Lubachevsky 1987; Overeinder et al. 1992; Overeinder and Sloot 1993; Sloot and Overeinder 1999). The ACA model incor-

porates asynchronous cell updates, which are independent of the other cells, and allow for a more general approach to CA. With these qualifications, the ACA is able to solve more complicated problems, closer to reality.

Dynamic systems with asynchronous updates can be forced to behave in a highly inhomogeneous fashion. For instance in a random iteration model it is assumed that each cell has a certain probability of obtaining a new state and that cells iterate independently. As an example one can think of the continuous-time probabilistic dynamic model for an Ising spin system (Lubachevsky 1988).

5.2.3 The Asynchronous Cellular Automata Model

The model for an asynchronous cellular automata is given by its constituents: the cellular automata, the definition of the neighborhood, the transition rules, and—this is exclusively for asynchronous cellular automata—a time evolution function. We define a deterministic asynchronous cellular automata by

$$Z = (I^d, N, V, v_0, f, F, T),$$

where:

- I^d is the set of d-tuple integers, called an array of the cellular space. An element of I^d represents the coordinate of a cell or a cell at that coordinate. The positive integer d is called the dimension of the cellular space.
- N is an n-tuple of different elements of I^d, called the neighborhood index. With $N = (n_1, \ldots, n_n)$ and given a cell a, an element of the set $N(a) = \{(a+n_1), \ldots, (a+n_n)\}$ is called a neighborhood cell of a, and a is the center cell of those neighborhood cells. The set consisting of elements of N is simply called the neighborhood.
- The cellular space is homogeneous. Thus for all cells, V is a nonempty finite set, called a state set.
- The state set V has an element v_0 in which the cell is at rest, called the quiescent state.
- The local function f is a mapping from V^n to V and satisfies the specific property $f(v_0, \ldots, v_0) = v_0$.
- The next state of a is given by $s_t(a) = F(a, N, f, t)$, where f is instantaneous applied to the neighborhood N of a at time t.
- The time of the next state change evaluation of a is described by $t' = T(a, N, t)$, where $t' > t$.

A *nondeterministic* asynchronous cellular automata can be obtained by introducing a random experiment and defining a random variable on the sample space of the experiment. The nondeterministic local function f is augmented

with ξ, which is a realization of the random variable. The local function can be written as f_ξ, where one can think of f_ξ as a tabulated function depending on ξ, or if the random variable is discrete, ξ can be considered as an argument to the function.

Similarly, the time evolution function can be written as T_ξ, where T_ξ can be a tabulated function depending on ξ, or, for both continuous and discrete random variables, ξ can be considered as an argument to function T_ξ. The use of ξ for continuous random variables is valid since time is continuous in asynchronous cellular automata.

A more general system can be modeled with an asynchronous cellular automata where the transition rule operates on the active cell *and* its neighborhood (Priese 1978; Lubachevsky 1988). We can write the local transition function f as a mapping from V^n to V^n and the next state is given by $s_t(N(a)) = F(a, N, f, t)$.

The concurrent update of the neighborhood $N(a)$ of cellular automaton a introduces ambiguity if two distinct neighbors a and a' decide to change their overlapping neighborhoods $N(a)$ and $N(a')$ at the same simulation time. If two events with the same timestamp are scheduled for the same cellular automaton, the feature of *instantaneous update* that is characteristic to discrete event simulation, does not preclude this ambiguity in update order. Priese indicates how one may devise a computation and construction universal, concurrent, asynchronous cellular automaton where no overlapping can possibly occur. Another practical solution is to prohibit coincidence of update time in pairs of different cells, which is part of the predictability requirements as stated by Misra (1986), or to provide tie-breaking rules for simultaneous event updates (Wieland 1997).

5.2.4 Parallel Simulation of Cellular Automata Models

The parallelization of a CA, both for synchronous and asynchronous models, is realized by spatial decomposition. That is, the individual cells of the CA are aggregated into sub-lattices, which are mapped to the parallel processors. As we will see, parallel synchronization between the sub-lattices is very different for synchronous and asynchronous CA models.

Parallel Synchronous Cellular Automata Simulation

Similar to the sequential execution of synchronous CA, the cells in a parallel synchronous CA simulation undergo simultaneous state transitions under direction of a global clock. All cells must finish their state transition computations before any cell can start simulating the next clock tick.

The parallelization of the discrete-time simulation is achieved by imitating the synchronous behavior of the simulation. The simulation is arranged into a sequence of rounds, with one round corresponding to one clock tick. Between successive rounds, a global synchronization of all cells indicates that the cells

have finished their state change at time step t and the new time step $t + \Delta t$ can be started.

Generally, the simulation proceeds in two phases, a computation and state update phase, and a communication phase. The progress of time in this time-driven simulation is illustrated in Fig. 5.1.

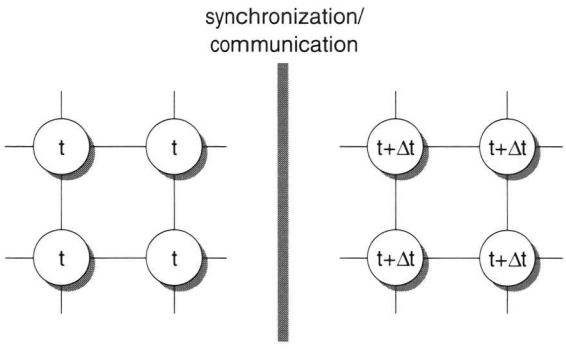


Figure 5.1: Time-driven simulation of a synchronous CA model, where computation and communication phases succeed each other.

Parallel Asynchronous Cellular Automata Simulation

In parallel ACA simulation, state transitions (further called events) are not synchronized by a global clock, but rather occur at irregular time intervals. In these simulations few events occur at any single point in simulated time and therefore parallelization techniques based on synchronous execution using a global simulation clock perform poorly. Concurrent execution of events at different points in simulated time is required, but this introduces severe synchronization problems. The progress of time in event-driven simulation is illustrated in Fig. 5.2.

The absence of a global clock in asynchronous execution mechanisms necessitates parallel discrete event simulation algorithms to ensure that cause-and-effect relationships are correctly reproduced by the simulator.

To summarize, the parallel *synchronous* execution mechanism for discrete-time models mimics the sequential synchronous execution mechanism by interleaving a computation and state update phase with a synchronization and communication phase. The parallel execution mechanism is fairly simple and induces a minimum of overhead on the computation. The parallel *asynchronous* execution mechanism for discrete event models, in our discussion the optimistic Time Warp simulation method, is more expensive than its sequential counterpart. The synchronization mechanism in optimistic simulation requires extra administration, such as state saving and rollback. Despite this overhead, op-

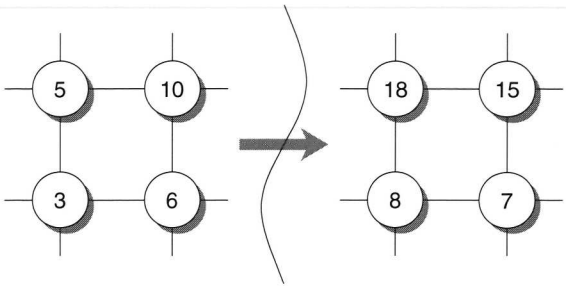


Figure 5.2: Progress of simulation time in event-driven simulation. As the cells evolve asynchronously in time, the simulation time of the individual cells are different.

timistic simulation is an efficient parallel execution mechanism for discrete event models.

5.3 Ising Spin Systems

The Ising spin model is a model of a system of interacting variables in statistical physics. The model was proposed by Wilhelm Lenz and investigated by his graduate student, Ernst Ising, to study the phase transition from a paramagnet to a ferromagnet (Brush 1967). A variant of the Ising spin model that incorporates the time evolution of the physical system is a prototypical example how asynchronous cellular automata can be used to simulate asynchronous temporal behavior.

A key feature in the theory of magnetism is the electron's spin and the associated magnetic moment. Ferromagnetism arises when a collection of such spins conspire so that all of their magnetic moments align in the same direction, yielding a total magnetic moment that is macroscopic in size. As we are interested how macroscopic ferromagnetism arises, we need to understand how the microscopic interaction between spins gives rise to this overall alignment. Furthermore, we would like to study how the magnetic properties depend on temperature, as systems generally loose their magnetism at high temperatures.

5.3.1 The Ising Spin Model

To introduce the Ising model, consider a lattice containing N sites and assume that each lattice site i has associated with it a number s_i, where $s_i = +1$ for an "up" spin and $s_i = -1$ for a "down" spin. A particular configuration or microstate of the lattice is specified by the set of variables $\{s_1, s_2, \ldots, s_N\}$ for all lattice sites (see Fig. 5.3).

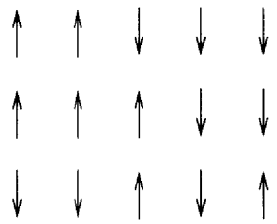


Figure 5.3: Schematic spin model for an Ising spin system.

The macroscopic properties of a system are determined by the nature of the accessible microstates. Hence, it is necessary to know the dependence of the energy on the configuration of spins. The total energy of the Ising spin model is given by

$$E = -J \sum_{i,j=\mathrm{nn}(i)}^{N} s_i s_j - \mu_0 H \sum_{i=1}^{N} s_i \,, \tag{5.1}$$

where $s_i = \pm 1$, J is the measure of the strength of the interaction between spins, and the first sum is over all pairs of spins that are nearest neighbors (see Fig. 5.4). The second term in Eq. 5.1 is the energy of interaction of the magnetic moment, μ_0, with an external magnetic field, H.

Figure 5.4: The interaction energy between nearest neighbor spins in the absence of an external magnetic field.

If $J > 0$, then the states $\uparrow\uparrow$ and $\downarrow\downarrow$ are energetically favored in comparison to the states $\uparrow\downarrow$ and $\downarrow\uparrow$. Hence for $J > 0$, we expect that the state of the lowest total energy is *ferromagnetic*, i.e., the spins all point to the same direction. If $J < 0$, the states $\uparrow\downarrow$ and $\downarrow\uparrow$ are favored and the state of the lowest energy is expected to be *paramagnetic*, i.e., alternate spins are aligned. If we add a magnetic field to the system, the spins will tend to orient themselves parallel to H, since this lowers the energy.

The average of the physical quantities in the system, such as energy E or magnetization M, can be computed in two ways: the time average and the statistical average. The time average of physical quantities are measured over a time interval sufficiently long to allow the system to sample a large number of microstates. Although time average is conceptually simple, it is convenient to formulate statistical averages at a given instant of time. In this interpretation,

all realizable system configurations describe an ensemble of identical systems. Then the ensemble average of the mean energy E is given by

$$\langle E \rangle = \sum_{s=1}^{m} E_s P_s ,$$

where P_s is the probability to find the system in microstate s, and m is the number of microstates.

Another physical quantity of interest is the magnetization of the system. The total magnetization M for a system of N spins is given by

$$M = \sum_{i=1}^{N} s_i .$$

In our study of the Ising spin system, we are interested in the equilibrium quantity $\langle M \rangle$, i.e., the ensemble average of the mean magnetization M.

Besides the mean energy, another thermal quantity of interest is specific heat or heat capacity C_v. The heat capacity C_v can be determined by the statistical fluctuation of the total energy in the ensemble:

$$C_v = \frac{1}{kT^2} \left(\langle E^2 \rangle - \langle E \rangle^2 \right) .$$

And in analogy to the heat capacity, the magnetic susceptibility χ is related to the fluctuations of the magnetization:

$$\chi = \frac{1}{kT} \left(\langle M^2 \rangle - \langle M \rangle^2 \right) .$$

For the Ising model the dependence of the energy on the spin configuration (Eq. 5.1) is not sufficient to determine the time-dependent properties of the system. That is, the relation Eq. 5.1 does not tell us how the system changes from one spin configuration to another, therefore we have to introduce the dynamics separately.

5.3.2 The Dynamics in the Ising Spin Model

Physical systems are generally not isolated, but are part of a larger environment. In this respect, the systems exchange energy with their environment. As the system is relatively small compared to the environment, any change in the energy of the smaller system does not have an effect on the temperature of the environment. The environment acts as a heat reservoir or heat bath at a fixed temperature T. From the perspective of the small system under study, it is placed in a heat bath and it reaches thermal equilibrium by exchanging energy with the environment until the system attains the temperature of the bath.

A fundamental result from statistical mechanics is that for a system in equilibrium with a heat bath, the probability of finding the system in a particular microstate is proportional to the Boltzmann distribution (Reif 1965)

$$P_s \sim e^{-\beta E_s},$$

where $\beta = 1/k_B T$, k_B is Boltzmann's constant, E_s is the energy of microstate s, and P_s is the probability of finding the system in microstate s.

The Metropolis Algorithm

To introduce the dynamics that describe the system changes from one configuration to another, we need an efficient method to obtain a representative sample of the total number of microstates, while the temperature T of the system is fixed. The determination of the equilibrium quantities is time independent, that is the computation of these quantities does not depend on simulation time. As a result, we can apply Monte Carlo simulation methods to solve the dynamics of the system. The well-known Metropolis algorithm uses the Boltzmann distribution to effectively explore the set of possible configurations at a fixed temperature T, see for instance Binder and Heermann (1992). The Metropolis algorithm samples a representative set of microstates by using an *importance sampling method* to generate microstates according a probability function

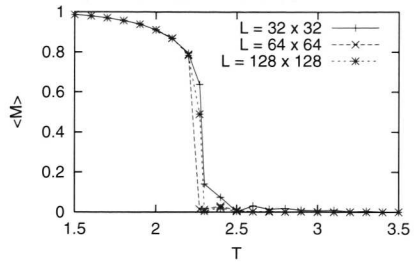
$$\pi_s = \frac{e^{-\beta E_s}}{\sum_{s=1}^{m} e^{-\beta E_s}}.$$

This choice of π_s implies that the ensemble average for the mean energy and mean magnetization can be written as

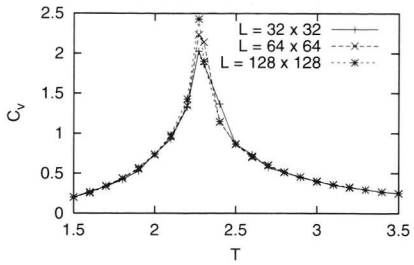
$$\langle E \rangle = \frac{1}{m} \sum_{s=1}^{m} E_s \quad \text{and} \quad \langle M \rangle = \frac{1}{m} \sum_{s=1}^{m} M_s .$$

The resulting Metropolis algorithm samples the microstates according to the Boltzmann probability. First, the algorithm makes a random trial change (a spin flip) in the microstate. Then the energy difference ΔE is computed. The trial is accepted with probability $e^{-\beta \Delta E}$ (note that for $\Delta E \leq 0$ the probability is equal to or larger than one and the trial is always accepted). After the trial, accepted or not accepted, the physical quantities are determined, and the next iteration of the Metropolis algorithm can be started.

The number of Monte Carlo steps per spin (or in general per particle) plays an important role in Monte Carlo simulations. On the average, the simulation attempts to change the state of each particle once during each Monte Carlo step per particle. We will refer to the number of Monte Carlo steps per particle as the "time," even though this time has no obvious direct relation to physical time. We can view each Monte Carlo time step as one interaction with the heat bath. The effect of this interaction varies according to the temperature T, since T enters through the Boltzmann probability for flipping a spin.



(a) Temperature dependence of the mean magnetization per spin for lattice size 32×32, 64×64, and 128×128.



(b) Temperature dependence of the specific heat for lattice size 32×32, 64×64, and 128×128.

Figure 5.5: Ising spin temperature dependence of the mean magnetization and specific heat.

The temperature dependency of the physical quantities $\langle M \rangle$ and C_v are shown in figures Fig. 5.5(a) and Fig. 5.5(b) respectively. For temperature $T = 0$, we know that the spins are perfectly aligned in either direction, thus the mean magnetization per spin is ± 1. As T increases, we see in Fig. 5.5(a) that $\langle M \rangle$ decreases continuously until $T = T_c$, at which $\langle M \rangle$ drops to 0. This T_c is known as the *critical temperature* and separates the ferromagnetic phase $T < T_c$ from the the paramagnetic phase $T > T_c$. The singularity associated with the critical temperature T_c is also apparent in Fig. 5.5(b). The heat capacity at the transition is related with the large energy fluctuations found near the critical temperature. The peak becomes sharper for larger systems but does not diverge because the lattice has finite sizes (singularities are only found in an infinite system).

Continuous-Time Ising Spin System

The standard Ising spin model represents a certain *discrete-time* model, as Monte Carlo steps are regarded to be time steps. However, the discrete evolution of the Ising spin configurations is considered an artifact. Glauber (1963) introduced *continuous-time* probabilistic dynamics for the Ising system to represent the time evolution of the physical system.

The Ising spin model with continuous-time probabilistic dynamics cannot be solved by Monte Carlo simulation, since time has no explicit implication on the evolution of the system in the Monte Carlo execution model. To capture the asynchronous continuous-time dynamics correctly, the problem is mapped to the ACA model and is executed by event-driven simulation.

In the continuous-time Ising spin model, a spin is allowed to change the state, a so-called spin flip, at random times. The attempted state change arrivals for a particular spin form a Poisson process. The Poisson arrival pro-

cesses for different spins are independent, however, the arrival rate is the same for each spin. Similar to the Monte Carlo simulation, the attempted spin flip, or trial, is realized by calculating the energy difference ΔE between the new configuration and the old configuration. The spin flip is accepted with the Boltzmann probability $e^{-\beta \Delta E}$.

The discrete-time and continuous-time models are similar. They have the same distribution of the physical equilibrium quantities and both produce the same random sequences of configurations. The difference between the two models is the time scale at which the configurations are produced: in discrete-time, the time interval between trials is equal, and in continuous-time, the time intervals are random exponentially distributed.

5.4 Optimistic Simulation of Continuous-Time Ising Spin Systems

A parallel model of the continuous-time Ising spin system is designed and implemented with use of the APSIS simulation environment. For the design and implementation of the parallel model a number of issues are important: Metropolis algorithm, simulation time advancement, random number generation, and the spatial decomposition parallelization strategy. Both the Metropolis algorithm and the time advancement (Poisson arrival process) require a pseudo-random number generator, which has to meet certain requirements to preclude undesired correlations if the pseudo-random generator is used in parallel processing. Spatial decomposition has consequences for the embedding of the parallel model in the APSIS simulation environment.

The algorithm that computes the dynamics of the Ising spin system at a temperature is the Metropolis algorithm, as discussed in the previous section. For each successful attempt, thus actual spin flip, the energy quantities E, $\langle E \rangle$, and $\langle E^2 \rangle$, and magnetic quantities M, $\langle M \rangle$, and $\langle M^2 \rangle$ are recomputed. Note that $\langle E \rangle$ and $\langle M \rangle$ are the *ensemble means*, thus not the mean energy of magnetization of the system at that simulation time, but rather the mean value of all ensemble configurations probed by the Metropolis algorithm up to the current simulation time. From the energy and magnetization values we can compute the specific heat C_v and magnetic susceptibility χ. The values E, $\langle E \rangle$, $\langle E^2 \rangle$, M, $\langle M \rangle$, and $\langle M^2 \rangle$ are typically values that are state saved by the optimistic simulation protocol.

The trials to update a spin in the continuous-time Ising spin system occur at random times. The interarrival times of the trials for a particular spin are independent and exponentially distributed, thus forming a Poisson arrival process with rate λ. In the discrete event simulation, upon the execution of a trial event at simulation time t, the spin flip is accepted according to the Metropolis algorithm, and the next trial event is scheduled at simulation time $t' = t - 1/\lambda \log U$, where U is a uniform distributed random variable in the interval $(0, 1]$.

In the PDES implementation of the Ising spin model we have two choices to schedule the trial events for the spin updates. With spatial decomposition of the Ising spin model, we aggregate a large number of spins into one sub-lattice and assign this sub-lattice to one LP. For example, the two-dimensional Ising spin system of size 128×128 is partitioned in eight sub-lattices of size 32×64. With this decomposition, each LP simulates the dynamics of $32 \cdot 64 = 2048$ spins. As each spin generates its own Poisson stream with rate λ, the LP can schedule for each individual spin a trial event. The execution of a trial event at a spin schedules a new future trial event for that particular spin. In this approach the number of events pending for execution at an LP is equal to the size of the sub-lattice, e.g., with the 32×64 sub-lattice the LP has 2048 events scheduled for future execution. However, we can do better since the Poisson asynchrony in the aggregated algorithm is a special case: the sum of k independent Poisson streams with rate λ each, is a Poisson stream with rate λk. In the event scheduling algorithm, k is the size of the sub-lattices. In the second approach, we neither maintain individual Poisson streams, nor future trial events for individual spins. Instead, a single cumulative Poisson stream is simulated, and spins are delegated randomly to meet these trial events.

In parallel simulation, and in particular parallel Monte Carlo simulation, special care should be taken with the generation of random numbers. Both the Metropolis algorithm and the Poisson stream need a uniform random variable for their operation. For sequential architectures, good random number generators exist. However, it is not at all trivial to find high-quality random number generators for parallel architectures. It should be noted that highly correlated and statistically dependent parallel random number generators originating from bad parallelization or distributed strategies may destroy or dramatically forge simulation results.

There are two basic parallelization techniques to produce random numbers. The first approach assigns different random number generators to different processors, and the second approach assigns different substreams of one large random number generator to different processors. The first approach suffers form intrinsically bad scalability (this approach requires thousands of different high-quality random number generators on massively parallel architectures such as the ASCI Option Red or ASCI Option Blue systems; Dyadkin and Hamilton (2000) presented approximately 2100 good 128-bit multipliers for congruential pseudo-random number generators). Additionally, there might also be unknown correlations between the different random number generators we use. The second approach can be controlled better, although its risks should not be forgotten.

There are two methods for splitting a given stream of random numbers into suitable parallel streams (Hellekalek 1998). The first method, the "leap-frog technique", assigns the substream $(x_{kn+i})_{n\geq 0}$ to the ith processor, $0 \leq i \leq k-1$. In other words, we use substreams of lag k of the original sequence $(x_n)_{n\geq 0}$, see Table. 5.1(a). The second method, the "splitting technique", partitions the original sequence into k (very long) consecutive blocks, see Table. 5.1(b). Each of the k processors is assigned a different block, where every block is defined by

a unique seed. This approach is very efficient way to assign different streams of random numbers to different processors.

p_0	p_1	p_2	p_3
x_0	x_1	x_2	x_3
x_4	x_5	x_6	x_7
x_8	x_9	x_{10}	x_{11}
$\vdots$	$\vdots$	$\vdots$	$\vdots$

(a) The leap-frog technique.

p_0	p_1	p_2	p_3
x_0	x_L	x_{2L}	x_{3L}
x_1	x_{L+1}	x_{2L+1}	x_{3L+1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_{L-1}	x_{2L-1}	x_{3L-1}	x_{4L-1}

(b) The splitting technique.

Table 5.1: Parallel random number generation: leap-frog and splitting.

The Mersenne Twister MT19937 random number generator has an extremely long period of $2^{19937} - 1$ and an extensive theoretical background (Matsumoto and Nishimura 1998). Due to its long period, we can choose the initial value, the seed, randomly and obtain as many substreams as we need. It is highly improbable that two substreams will overlap. Other solutions are offered by parallel random number generator libraries such as the PRNG library (Entacher et al. 1998) or the SPRNG library (Ceperley et al. 1999). These libraries provide implementations of various parallel random number generators. The user can initialize the parallel random number generator by specifying all the parameters to the parallel random number generator, including the splitting method and the number of parallel streams.

An additional complicating factor in optimistic PDES is that due to the rollback synchronization, the sample path generated according to the desired statistics can be altered, unless some precautions are taken (Tsitsiklis 1989). In particular, if part of the simulation is performed for a second time, due to a rollback, one should use the same random numbers that were used the first time. Suppose that the dynamics of an LP has been formulated so that the statistics of the random variable x_i corresponding to the ith event has a prescribed distribution depending only on i. We can then generate random variables $x_0, x_1, \ldots$ and the the value x_i will be the one to be used for the simulation of the ith event, no matter how many times the ith event has to be simulated (due to rollbacks) and even if different simulations of the ith event corresponds to different simulation times.

To make random number generators rollback proof, the APSIS environment encapsulates the random number generators into the simulation kernel. For each ith event, the corresponding random variable x_i is generated once and stored in a data structure by the simulation kernel. Upon rollback from the kth event to the jth event, the next random variables x_i, $j \leq i \leq k$, are retrieved from the data structure before new random variables are generated. This can be efficiently implemented in a circular buffer (see Section 3.5.3, Fig. 3.5). Besides, the reuse of random variables is profitable as random number generation is relative expensive.

The resulting continuous-time Ising spin model is parallelized by spatial decomposition. The Ising spin lattice is partitioned into sub-lattices, and the sub-lattices are mapped onto parallel processors. To minimize the communication between sub-lattices, local copies of the neighbor boundaries are stored locally (see Fig. 5.6). By maintaining local copies of neighbor boundaries, spin values are only communicated when they are actually changed, rather than when they are only referenced. A spin flip along the boundary is communicated to the neighbors by an event message. The causal order of the event messages, and thus the spin updates, are guaranteed by the optimistic simulation mechanism.

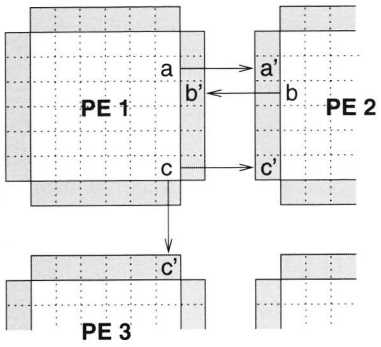


Figure 5.6: Spatial decomposition of the Ising spin lattice. The grey areas are local copies of neighbor boundary strips. For example, processor PE 2 has a local copy of spin "a" owned by processor PE 1. Processors PE 2 and PE 3 both own a copy of spin "c". The arrows in the figure indicate the event messages sent upon a spin flip.

Asynchronous cellular automata, and thus also the Ising spin model, put efficient memory management requirements on the original formulation of the Time Warp method. The state vector of a spatial decomposed ACA can be arbitrarily large, that is, all the cells in the sub-lattice are part of the state vector. Hence, the incremental state saving method of the APSIS environment is used during the simulation of the Ising spin model. Although incremental state saving requires less state saving time and memory, there is an increased cost of state reconstruction. In general, the number of rolled back events is a fraction of the number of events executed during forward simulation. In this respect, the state recovery overhead of incremental state saving and copy state saving are in the ratio of 10^2 bytes to an order of 10^6 bytes, therefore incremental state saving is favorable in spatial decomposed ACA applications.

5.5 Parallel Performance and Scalability

A series of experiments were executed, for different problem sizes and Ising spin parameter settings, to get insight into the efficiency and scalability behavior of Time Warp. The experiments with the parallel Ising spin simulation were performed on the Distributed ASCI Supercomputer (DAS)*. (Note that ASCI stands for Advanced School for Computing and Imaging—a Dutch research school.) The DAS consists of four wide-area distributed clusters of total 200 Pentium Pro nodes. ATM is used to realize the wide-area interconnection between the clusters, while the Pentium Pro nodes within a cluster are connected with Myrinet system area network technology. The experiments are performed within one single cluster, thus all communication is via the 1.28 Gbit/s Myrinet. The communication layer is an efficient implementation of PVM on top of Panda (Ruhl et al. 1996). Panda is a virtual machine designed to support portable implementations of parallel programming systems. The efficient communication primitives and thread support in Panda allows for low latency, high throughput communication performance over the Myrinet network. The measured Panda/PVM (null message) latency is $\pm 17\,\mu$sec, and the throughput ± 60 MB/s.† For random number generation, we make use of the Mersenne Twister MT19937 random number generator, where each LP initializes the random number generator with a different seed to get parallel random number substreams.

5.5.1 Relative Parallel Performance and Scalability

In the first series of experiments, we study the *relative efficiency* and *scalability* of the parallel Ising spin simulation. This is done by comparing the execution time of the parallel simulation on one processor, $T_p(1)$, with the execution time on different number of processors, $T_p(P)$. The relative efficiency is now defined as

$$E = \frac{T_p(1)}{T_p(P) \cdot P}.$$

The parameters to the Ising spin experiment are the lattice size $L \times L$, the temperature T, the number of simulation steps, and the number of processors. The lattice size $L \times L$ and the number of processors determine the granularity of the computation, or the computation to communication ratio. Given the decomposition shown in Fig. 5.6, the boundary lattice points are potentially communicated to the neighboring LPs. The ratio of boundary lattice points to the total number of local lattice points is $4/M$, where $M \times M$ is the sub-lattice size after decomposition. The temperature T of the Ising spin system determines also in part the granularity of the LPs: as the temperature increases, the behavior of the system becomes more dynamic and hence more communication occurs between the nodes. The temperature of the system also influences

*http://www.asci.tudelft.nl/das/das.shtml or http://www.cs.vu.nl/das/

†Performance experiments with MPI are presented by al Mourabit (2000).

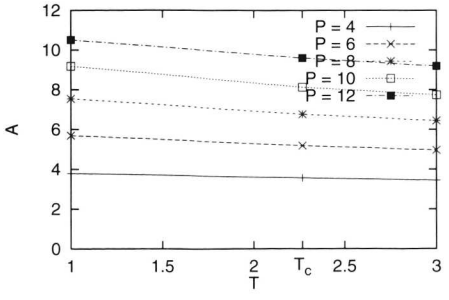
the computational behavior of the simulation in a more subtle way, which is presented in Chapter 6.

The simulation time is denoted by a derivative of Monte Carlo time steps. A Monte Carlo time step embodies $L \times L$ spin update attempts such that all the lattice points in the system have potentially got the opportunity to change their state. In the continuous-time Ising spin simulation we still use the notion of Monte Carlo steps to specify the duration of the simulation, as it conveniently indicates both the statistical evolution of the system and the expected amount of computational work. Of course, it has no (direct) relation to the simulated time, as simulation time progress is determined by the aggregated Poisson arrival process.

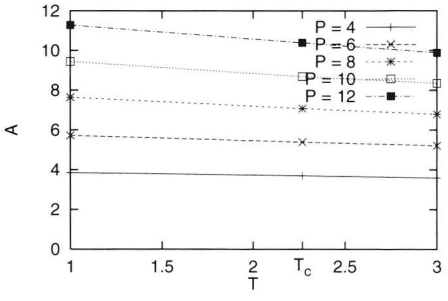
APSE Analysis

The influence of temperature and lattice size on the average parallelism inherent to the Ising spin simulation is studied by use of the APSE analysis framework (see Chapter 4). The APSE analysis allows one to study the scalability behavior of the simulation application without any assumption on the simulation protocol, or stated differently, with an ideal, omniscient simulation protocol. The results from the APSE analysis supports the interpretation of the experimental results in the next section. For example, if the APSE analysis of a simulation application results in a limited average parallelism, this should be attributed to the bounded inherent parallelism of the simulation application software, rather than the inability of the simulation protocol to exploit the available parallelism.

The dependency of the average parallelism on the Ising spin temperature is shown in Fig 5.7 for lattice sizes $L = 32$ and $L = 64$. From the figures we learn that the average parallelism decreases for increasing temperature.



(a) Average parallelism versus temperature for $L = 32$.



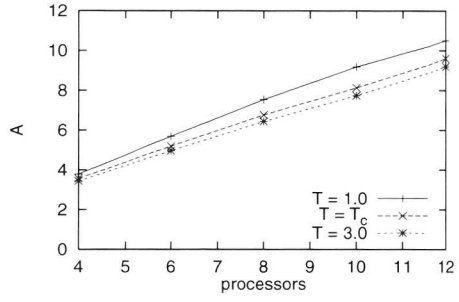
(b) Average parallelism versus temperature for $L = 64$.

Figure 5.7: APSE average parallelism versus temperature analysis.

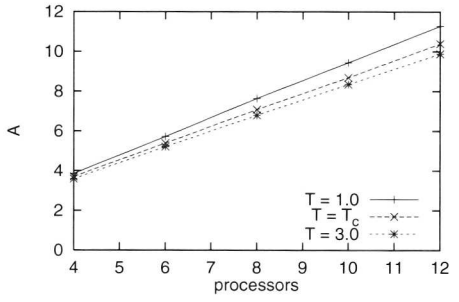
This can be explained by the higher rate of successful spin flips for higher temperatures. Spin flips on the sub-lattice boundaries must be synchronized with the neighboring sub-lattices, resulting in sequentialization of (part of) the spin flip trials.

The impact of lattice size on the sequentialization of boundary spin flips appears also from the figures for lattice sizes $L = 32$ and $L = 64$. The average parallelism is smaller for smaller lattice size L, and vice versa. For constant temperature, the ratio of boundary spin flips is larger for small lattice sizes than for large lattice sizes. Hence, for small lattice sizes the ratio of sequentialized spin flips is larger, which depresses the average parallelism.

Figure 5.8 depicts how the average parallelism scales with the number of processors for the temperatures $T = 1.0$, T_c, and 3.0. The average parallelism figures show the temperature and lattice size dependency as the number of processors increase. Again, low temperature and large lattice size enhance the average parallelism in the Ising spin simulation.



(a) Average parallelism versus number of processors for $L = 32$.

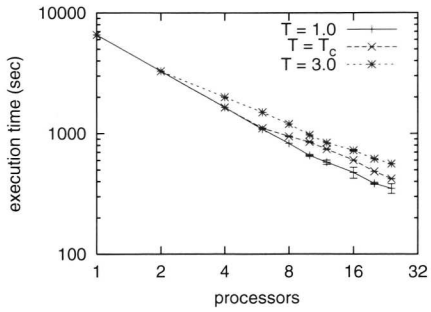


(b) Average parallelism versus number of processors for $L = 64$.

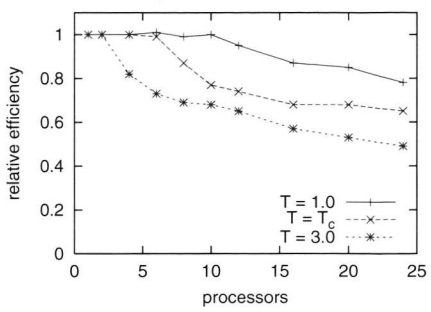
Figure 5.8: APSE average parallelism versus number of processors analysis.

Experiments

Figures 5.9 and 5.10 show the relation between execution time and the number of processors for fixed problem sizes 128×128 and 256×256. The number of Monte Carlo steps for each experiment is 12000 for lattice size 128×128, and 3000 steps for lattice size 256×256. In this way approximately the same number of events are executed for both system sizes. From these figures we can see that the parallel Ising spin simulation for $T = 1.0$ scales almost linearly up to 10 processors, but eventually drops to a relative efficiency of 0.78 for 24 processors with lattice size 128×128, and to a relative efficiency of 0.72 for 24 processors with lattice size 256×256. For temperature $T = 3.0$ the relative efficiency decreases gradually to 0.49 for 24 processors with lattice size 128×128, and to 0.62

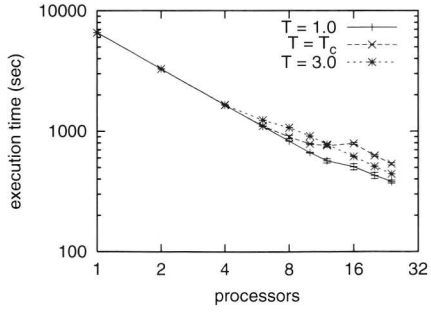


(a) Log-log plot of the execution time of parallel Ising spin for lattice size 128 × 128. The measure points and error boxes indicate the mean and standard deviation of 6 measurements.

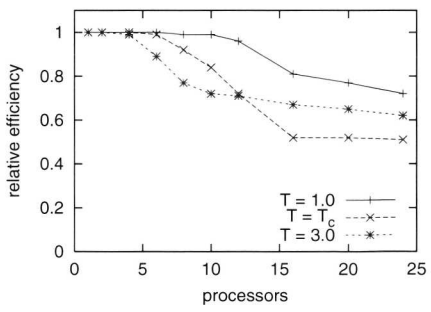


(b) Relative efficiency of parallel Ising spin for lattice size 128 × 128.

Figure 5.9: Scalability and relative performance of parallel Ising spin simulation.



(a) Log-log plot of the execution time of parallel Ising spin for lattice size 256 × 256. The measure points and error boxes indicate the mean and standard deviation of 6 measurements.



(b) Relative efficiency of parallel Ising spin for lattice size 256 × 256.

Figure 5.10: Scalability and relative performance of parallel Ising spin simulation.

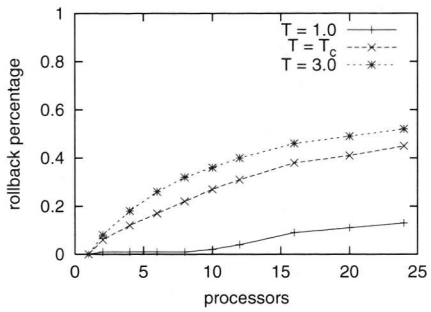
for 24 processors with lattice size 256 × 256. The decreasing efficiency is mainly due to the increased costs to synchronize the parallel processes. With the increase in the number of processors, the time period necessary to synchronize the parallel simulation processes also increases. A minimal memory manage-

ment requirement in optimistic simulation is the limitation of optimism, and hence the maximum rollback length (this is described further on in this section). For lattice size 256 × 256, the decrease in efficiency flattens at $P = 16$ for $T = 1.0$ and T_c, and at $P = 10$ for $T = 3.0$, since the rollback lengths in these regimes approach the maximum rollback length. Additional increase of the number of processors does not further increase the rollback lengths.

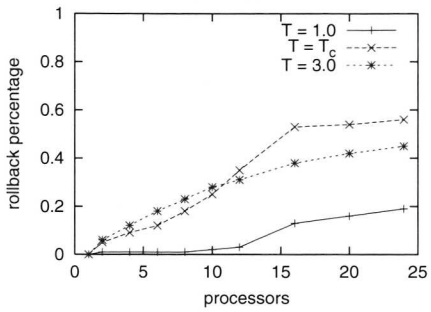
Temperature $T = T_c \approx 2.269$ is a special case. If we consider Fig. 5.9 and Fig. 5.10 separately, we see different scaling behavior for $T = T_c$ compared with $T = 1.0$ and $T = 3.0$. For lattice size 128 × 128 (Fig. 5.9), $T = T_c$ scales more or less within the bounds of $T = 1.0$ and $T = 3.0$, which is expected as synchronization costs increases with the temperature and the number of processors. However, for lattice size 256 × 256 (Fig. 5.10), the scalability behavior (i.e., the execution time and efficiency versus the number of processors) for $T = T_c$ is not bounded by $T = 1.0$ and $T = 3.0$. Up to 10 processors, the Ising spin scaling behavior can be explained by increased communication and synchronization costs due to the dynamics (or temperature) of the Ising spin system, but for 12 processors and more, another factor determines the execution time behavior—which is discussed further on. In Fig. 5.10(a) the execution time increases from 12 to 16 processors for $T = T_c$, resulting in execution times larger than for $T = 3.0$.

The influence of the temperature, the lattice size, and the number of processors on the execution behavior of the parallel simulation processes is further investigated. Figure 5.11 shows the rollback percentage of the total amount of executed events for the lattice size 128 × 128 and 256 × 256. The rollback percentage is an expression of the amount of synchronization errors due to optimistic execution of events. As such, it is a relative indication of the increased execution time due to event execution order dependencies and simulation protocol overhead. If we consider Fig. 5.11(a), the rollback percentages for lattice size 128 × 128, we find a correspondence between the increase of the rollback percentage and the decrease in relative performance as shown in Fig. 5.9(b). Moreover, in Fig. 5.11(b) we see the same anomalous rollback percentage behavior for $T = T_c$ as in the relative efficiency in Fig. 5.10(b). The increase in execution time for $T = T_c$ in the trajectory from 12 to 16 processors in Fig. 5.10(a) can also be found in the strong increase of rollback percentage for $T = T_c$ in the trajectory from 12 to 16 processors in Fig. 5.11(b). Hence, there is a strong relation between the rollback behavior and the execution time (and the derived relative performance).

To understand the anomalous performance behavior of the Ising spin simulation with lattice size 256 × 256 at temperature $T = T_c$, a detailed execution trace of the event rate is monitored. The event rate is the number of events that are committed per second, and in this respect a measure for progress. During normal operation, the Ising spin model simulation reaches an event rate of around 19 000 events per second (see Fig. 5.12(a)). If we look in more detail to Fig. 5.12(a), we can identify four serious glitches in the event rate, around execution time 20, 48, 58, and 88, which indicate periods of resynchronization of the parallel simulation. In these periods, the event rate drops to

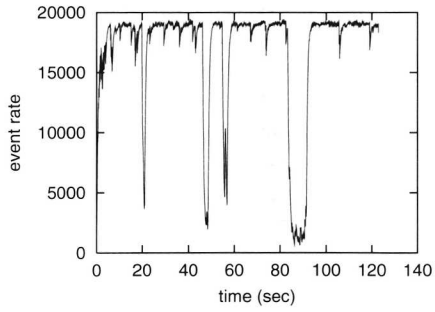


(a) Rollback percentages for lattice size 128×128.

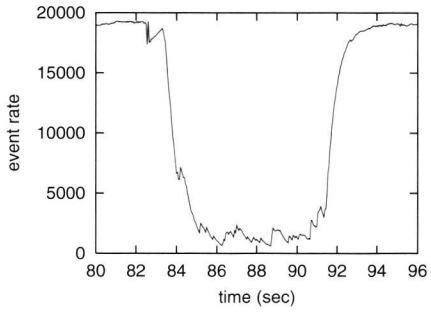


(b) Rollback percentages for lattice size 256×256.

Figure 5.11: Event rollback percentages for lattice sizes 128×128 and 256×256.



(a) Simulation progress (event rate) during execution. Parallel Ising spin with lattice size 256×256 on 6 processors.



(b) Simulation progress (event rate) during execution (detail of Fig. 5.12(a) between 80 and 96).

Figure 5.12: Simulation progress (event rate) during execution. The curve is smoothed by taking the exponential weighted moving average (EWMA), as the EWMA follows the dynamic behavior accurately and can be efficiently computed.

10% of the steady state performance (about 2000 events per second). In particular the period centered around 88 (see Fig. 5.12(b)) takes about 10 seconds to resynchronize and weights heavily upon the parallel performance.

The periods of resynchronization are a typical example of *thrashing*, where most of the time is spent on simulation rollback instead of forward simulation. While one simulation process rolls back, another process advances in simulation time. When the rollback is completed, the simulation process restarts

with event execution and as a result sends event messages to neighboring processes. These event messages arrive in the simulation past of the neighboring processes, and trigger a rollback, etc., etc., until the simulation processes are in synchrony. The thrashing behavior is a combination of a number of factors: number of processors, lattice size, event granularity, and temperature (synchronization frequency).

To shorten these periods of resynchronization, the optimism of the protocol must be throttled, that is, the simulation execution mechanism should not execute events that lie in the remote future as it is likely that these events have to be rolled back eventually. The progress of the individual simulation process should be bound to a limited simulation time window (see Section 2.4.5). In this way, the processes are forced to synchronize with each other in a short time frame, after which the simulation can continue as before. A key problem with Bounded or Moving Time Window optimism control is the determination of the appropriate size of the virtual time window. A narrow time window limits the rollbacks, but also the amount of parallelism. A time window that is too large, can potentially exploit more parallelism, but the rollbacks increase as well.

Another performance consideration in the determination of the appropriate virtual time window size is the sequential simulation performance and its relation to the GVT computation frequency (or progress rate). A narrow time window does not only limit the amount of parallelism, but can also limit the sequential event rate due to a slow GVT progress rate. As the sequential simulation process proceeds faster in simulation time (that is the progress of the LVT) than the progress of the GVT, the sequential simulation process will eventually reach the upper time window boundary, and will block until the next GVT progress update. The influence of virtual time window size on the sequential Ising spin simulation performance in APSIS is presented in Table 5.2 and Table 5.3. In the experiments, a new GVT update computation is started every 50 msec. The first columns of Tables 5.2 and 5.3 show the sequential simulation execution times with unbounded time window for the three temperatures $T = 1.0$, $T = T_c$, and $T = 3.0$. As one can see, the execution time of the sequential simulation increases with the temperature. This can be explained by the dynamics of the simulation: the higher the temperature, the higher the dynamics of the Ising spin system, and hence the higher the computational costs.

	VTW = ∞	VTW = 3000	VTW = 2000
T = 1.0	1146	1645 (1.44)	2463 (2.15)
T = 2.269	1232	1645 (1.34)	2464 (2)
T = 3.0	1340	1645 (1.23)	2464 (1.84)

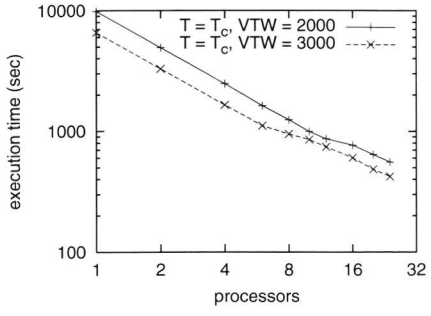
Table 5.2: Single processor execution time (in seconds) for Ising spin lattice size 128×128 and 3000 Monte Carlo steps. The slow down factor due to bounded virtual time window size is in parentheses.

	VTW = ∞	VTW = 3000	VTW = 2000
T = 1.0	4596	6578 (1.43)	9868 (2.15)
T = 2.269	4919	6577 (1.34)	9866 (2.01)
T = 3.0	5339	6579 (1.23)	9868 (1.85)

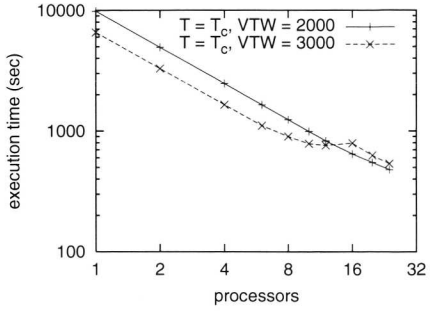
Table 5.3: Single processor execution time (in seconds) for Ising spin lattice size 256 × 256 and 3000 Monte Carlo steps. The slow down factor due to bounded virtual time window size is in parentheses.

The influence of the bounded virtual time window size, and hence the GVT update rate, is apparent from the second and third columns in Table 5.2 and Table 5.3. Ideally, for infinite GVT update rate, the time window size does not limit the sequential simulation performance as long as there is at least one pending event within the boundaries of the time window. In practice however, the GVT update rate is finite, and together with the time window size it limits the potential event execution rate, and hence determines the execution time. As one can see from Tables 5.2 and 5.3, the sequential execution time for virtual time window sizes VTW = 3000 and VTW = 2000 are determined by the time window size and *not* by the temperature of the Ising spin system (as with window size VTW = ∞). The interdependency between virtual time window size and GVT update rate adds another dimension to the determination of the appropriate time window size.

The effect of the bounded virtual time window on the simulation execution time can be clearly seen in Fig. 5.13. All the Ising spin experiments discussed before are performed with a virtual time window of 3000. In the APSIS environment, the virtual time window is also used as a memory management mechanism. If no virtual time window is used, arbitrary long rollbacks can



(a) Ising spin lattice size 128 × 128.



(b) Ising spin lattice size 256 × 256.

Figure 5.13: Log-log plot of the execution times of parallel Ising spin for different virtual time windows.

occur whose history (i.e., event queue, output queue, etc.) consumes all (virtual) memory. Hence, for memory management purposes the time window should be bounded, and experimentally a time window size of 3000 showed to be an appropriate starting value. (For a discussion on optimal virtual time window sizes, see Section 6.4.3.) The execution time figures for the virtual time window size VTW = 3000 in Fig. 5.13 are the same results as presented in Fig. 5.9(a) and Fig. 5.10(a) at temperature $T = T_c$. For virtual time window size VTW = 2000, the sequential simulation execution time (single processor execution) is about 3000 seconds longer than for VTW = 3000 (for both lattice sizes). The relative distance between the execution times for VTW = 2000 and VTW = 3000 remains constant for 2, 4, and 6 processors. In this region the execution time difference is predominantly determined by the virtual time window and its effect on the simulation event rate (see also previous discussion). However, where the execution times for VTW = 3000 start to deviate from linear scaling behavior (at 8, 10, 12, and 16 processors) due to excessive rollback behavior, the execution times for VTW = 2000 continues to scale linearly with the number of processors up to 12 processors for lattice size 128×128, and up to 16 processors for lattice size 256×256. The most prominent property of the figures is the crossover point in Fig. 5.13(b), between the execution times for VTW = 2000 and VTW = 3000. For 12 processors, the execution times for both virtual time window sizes are almost the same, but for 16 processors, the execution time for VTW = 2000 is significantly shorter.

The results presented in Fig. 5.13 show the potential of virtual time window management to control excessive rollback behavior, i.e., thrashing. However, the determination of an appropriate time window size is far from trivial, as it depends on various system parameters (i.e., number of processors) and application parameters (i.e., in Ising spin for example lattice size or temperature).

5.5.2 Absolute Parallel Performance and Scalability

In the second series of experiments, we study the *absolute efficiency* of the parallel Ising spin simulation compared to the best-known sequential Ising spin simulation for different temperatures, problem sizes, and event granularity (that is, the amount of work per event). The sequential continuous-time Ising spin simulation is basically a Monte Carlo simulation extended with a Poisson arrival process to incorporate time evolution into the model. The Monte Carlo simulation execution mechanism is a lightweight process compared to sequential discrete event simulation execution mechanism. With Monte Carlo simulation there is nearly no overhead involved in the execution of the spin flip trials: a random spin is selected and a trial is executed. With discrete event simulation, a trial is an event that must be scheduled for future execution, that is, inserted into the event list (in general a priority queue). Later, if the scheduled trial is the next pending event, the event is dequeued and the trial is executed. Parallel discrete event simulation includes, besides the event list management overhead, also the state saving and rollback overhead as described in the previous section. The absolute efficiency figures include all these

extra overhead costs compared to the sequential Monte Carlo simulation.

The experiments to assess the absolute performance, quantify the overhead induced by the APSIS parallel simulation protocol. An interesting perspective to approach the quantitative overhead evaluation is to relate this overhead with the so-called *event granularity*. The event granularity is defined as the amount of work per event (or trial in this discussion) and is in our study the amount of extra computational work in terms of a sinus and an exponential evaluation. The results for event granularity 0 are for the basic Ising spin system. The results for increasing event granularities give an indication how a similar problem scales as the amount of computational work to evaluate a trial (or state change) increases. Note that extra computational work is assigned to each trial, successful or not successful.

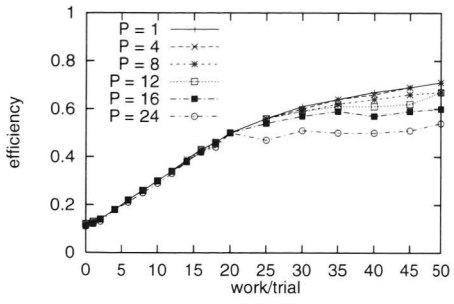
The absolute efficiency is defined as

$$E = \frac{T_s}{T_p(P) \cdot P},$$

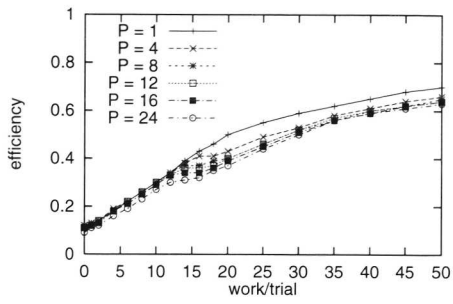
where T_s is the execution time of the sequential Monte Carlo Ising spin simulation, and $T_p(P)$ is the execution time of the parallel Ising spin simulation on P processors.

The absolute efficiency experiment results for Ising spin simulations with temperatures $T = 1.0$, $T = T_c$, and $T = 3.0$ are presented in Fig. 5.14. The absolute efficiency figures show the performance for different number of processors with scaled problem size, i.e., the lattice size is scaled with the number of processors such that the amount of local work at a processor remains constant. The lattice sizes for $P = 4, 8, 12, 16, 24$ are the square of $L = 91, 128, 158, 181, 222$ respectively. The performance figures for $P = 1$, that is the parallel simulation executed on one processor, is included as an upper boundary to the performance figures for other values of P. All results presented in the figures are the means of six experiments. The figures indicate that the parallel performance depends on the event granularity and Ising spin temperature. The event granularity determines the PDES protocol overhead ratio, apart from synchronization errors. The temperature T of the Ising spin system determines the computation/communication ratio: as the temperature increases, the behavior of the system becomes more dynamic and hence more communication occurs between the processors.

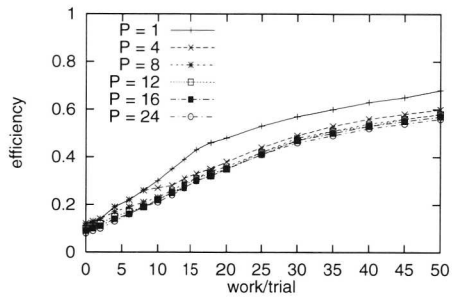
For low temperature $T = 1.0$, the absolute efficiency starts at 0.12 for work/-trial is 0. Around work/trial is 20, the absolute efficiency starts to diverge for the different number of processors, and eventually varies from 0.71 for $P = 4$ to 0.54 for $P = 24$ at work/trial is 50. For critical temperature $T = T_c$, we see in Fig. 5.14(b) that the point of divergence has moved from 20 to 15. However, at work/trial is 30, the performance figures per processor converge and end up in the range of $[0.63 - 0.66]$. Finally for high temperature $T = 3.0$, the point of divergence moved down to the range $[4 - 8]$, see Fig. 5.14(c). Also for high temperature, the performance figures converge and end up in the range of $[0.56 - 0.6]$.



(a) Temperature $T = 1.0$.



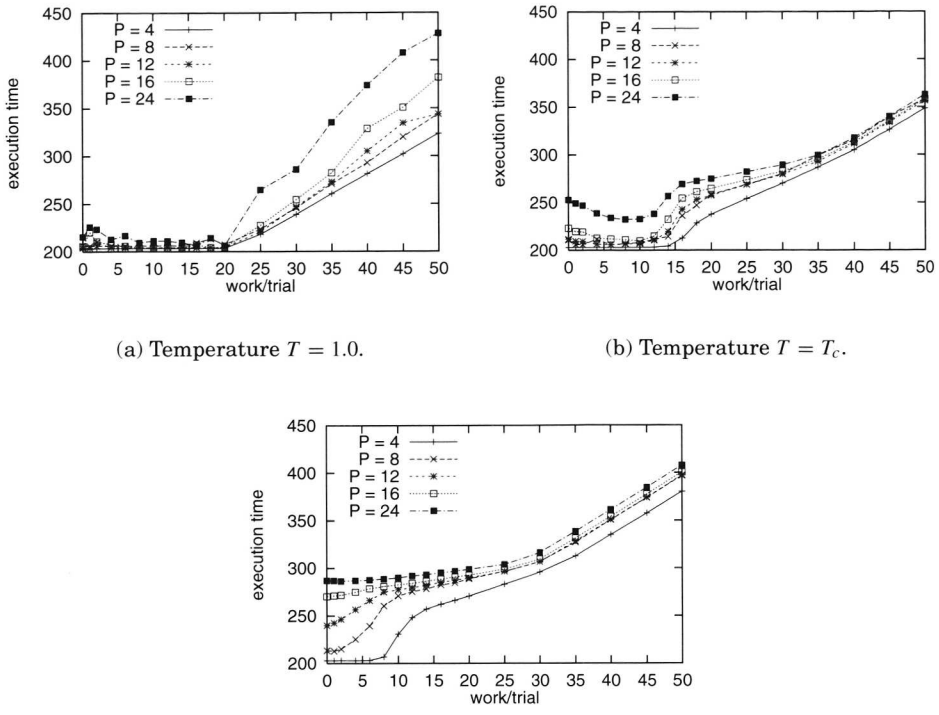
(b) Temperature $T = T_c$.



(c) Temperature $T = 3.0$.

Figure 5.14: Absolute efficiency versus event granularity (work/trial) for parallel Ising spin simulations with temperatures $T = 1.0$, T_c, and 3.0 on 1, 4, 8, 12, 16, and 24 processors, with scaled problem size.

To understand Fig. 5.14, the relation between execution time, temperature and rollback behavior must become clear. Figure 5.15 shows the parallel Ising spin simulation execution times for the three temperatures. In Fig. 5.15(a), the execution times for low temperature $T = 1.0$ are almost constant up to a work/trial of 20, which is to be expected with a scaled problem size. However, after this point, the execution times for larger number of processors starts to increase faster than for smaller number of processors. This is due to increasing synchronization costs of the Time Warp protocol for increasing number of processors. For critical temperature $T = T_c$ and high temperature $T = 3.0$, we see an interesting transient execution time behavior in Fig. 5.15(b) and Fig. 5.15(c). At work/trial is 0, the execution times increase with the number of processors. For the critical and high temperature, there is more communication, and hence more synchronization overhead. As the work/trial increases,



(a) Temperature $T = 1.0$.

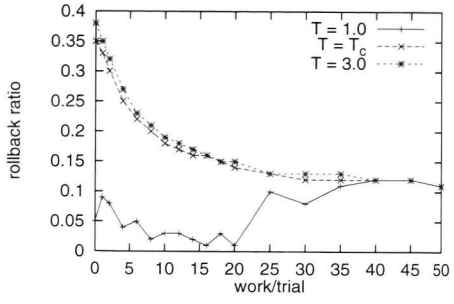
(b) Temperature $T = T_c$.

(c) Temperature $T = 3.0$.

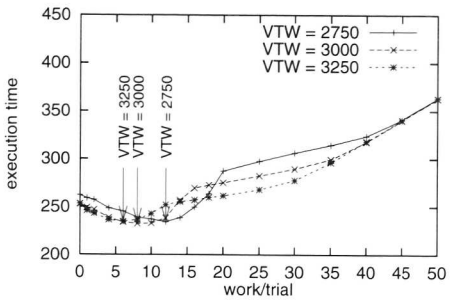
Figure 5.15: Parallel Ising spin simulation execution times for temperatures $T = 1.0$, T_c, and 3.0 on 1, 4, 8, 12, 16, and 24 processors, with scaled problem size.

the execution times converge to each other. For this magnitude of work/trial, the increased synchronization costs for larger number of processors are compensated by the larger event granularity.

The interesting transient execution time behavior is most prominent for intermediate temperatures around $T = T_c$, see Fig. 5.15(b). Here, the execution times for $P = 16$ and $P = 24$ even decrease with increasing work/trial up to 10. Apparently, two factors determine the execution time: one factor increases, and the other factor decreases for larger work/trial values. The factor that increases the execution time is of course the amount of work per trial, so we expect to see an increase in execution time for increasing work/trial. The second factor that decreases with work/trial is the rollback ratio, i.e., the ratio of events that are rolled back to the total number of executed events (rolled back or committed). In Fig. 5.16(a), the rollback ratio versus the work/trial are depicted for the three temperatures $T = 1.0$, T_c, and 3.0. For low temperature



(a) Event rollback ratio versus work/trial for temperatures 1.0, T_c, and 3.0.



(b) Execution time versus work/trial for the virtual time window sizes 2750, 3000, and 3250 with temperature T_c.

Figure 5.16: Influence of event rollback ratio and virtual time window on the execution time for 24 processors.

$T = 1.0$, the rollback ratio fluctuates significantly; the large variance can be explained by the infrequent synchronization which can actuate large cascaded rollbacks. For the intermediate and high temperatures, the rollback ratio falls off smoothly for increasing work/trial. Consequently, the synchronization overhead decreases. Thus, the increasing work per trial and the decreasing rollback ratio compete with each other, where the rollback ratio dominates for small values of work/trial, and the work per trial dominates for larger values of work/trial.

The crossover point, where the rollback ratio and work/trial are in balance, is partly determined by the virtual time window size. As the virtual time window size determines the amount of optimism, it also indirectly determines the rollback length and frequency. In Fig. 5.16(b), we see how the crossover point moves from work/trial is 12 to 6 for VTW = 2750 to 3250.

5.6 Summary and Discussion

An important subclass of dynamic complex systems, namely asynchronous cellular automata, has been used to rigorously evaluate the APSIS simulation environment. The specific asynchronous cellular automata used in our experiments is the continuous-time Ising spin model. The Ising spin model is a well-defined and understood problem, and shows a complex behavior that is essentially parameterized by the Ising spin temperature. The spatial decomposition of the Ising spin model over the parallel processors put severe memory constraints upon the APSIS environment, necessitating the use of incremental state saving.

The average parallelism analysis within the APSE framework exhibits the dependency of the average parallelism on the Ising spin temperature and the lattice size. Increasing temperatures results in (slowly) decreasing average parallelism, and increasing lattice sizes incorporate larger average parallelism. The APSE average parallelism analysis is consistent with our experiments, except around the critical temperature T_c. The parallel Ising spin simulation execution times around the critical temperature are larger than the execution times for temperature $T = 3.0$. Detailed study of the event rate showed that thrashing behavior of the parallel simulation occurs around the critical temperature, resulting in a simulation progress drop of 90%.

The absolute efficiency study compares the performance of the (parallel) discrete event simulation with the sequential Monte Carlo simulation implementation of the Ising spin model. The absolute efficiency is also a measure of the amount of overhead introduced by the (parallel) discrete event simulation compared to the relatively simple Monte Carlo simulation. In this respect, the event granularity is an important quantity as it determines the (parallel) discrete event simulation protocol overhead. The experimental results show a subtle interplay between the increased execution time for increasing event granularity, and decreasing rollback ratio and thus decreasing PDES overhead.

The application of optimistic parallel discrete event simulation methods such as Time Warp to asynchronous cellular automata is in potential a viable approach to parallelize the simulation. However, two essential extensions to the Time Warp method have to be included: incremental state saving and optimism control (throttling). The results show that given a fast communication network such as Myrinet, the Time Warp optimistic simulation method achieves good scalable performance. In particular, low communication latencies are essential to achieve performance, as the event messages are small.

The most promising approach to effective optimism control is the design and implementation of an adaptive mechanism. That is, the parallel simulation kernel determines the optimal virtual time window size using local state variables, such as event rate, rollback ratio, and communication statistics. A future research challenge is to devise a forecast method that exploits the local state variables for adaptive virtual time window control. The formulation of simple though applicable metrics to control the amount of optimism in the Time Warp method determines the success of the mechanism.