What is the best way to simulate an equilibrium classical spin model?

J. Machta* and L. E. Chayes†

*Department of Physics,University of Massachusetts, Amherst, Massachusetts 01003-3720
†Department of Mathematics,University of California, Los Angeles, California 90095-1555

Abstract. Several cluster algorithms for simulating classical spin models are reviewed and their advantages near critical points discussed. These algorithms are then examined from the general perspective of computational complexity theory. Several extant definitions of the "dynamic exponent" of an algorithm are discussed from this perspective and a robust definition based on an idealized model of parallel computation is proposed.

INTRODUCTION

The 50^{th} anniversary of the Metropolis algorithm is a good time to take stock of some general issues in Monte Carlo simulations in statistical physics. The Metropolis algorithm remains the most robust and widely used simulation method but the last two decades have seen the flowering of new algorithms that have superseded the Metropolis algorithm for special situations. One of the most important classes of new algorithms are cluster methods such as the Swendsen-Wang [1] and Wolff [2] algorithms. These algorithms, when available, are generally much faster than the Metropolis algorithm for the simulation of spin systems at a critical point or in a low temperature phase. The sense in which cluster algorithms are better is usually quantified in terms of a dynamic exponent.

In this paper we will discuss succinctly the concept of the dynamic exponent (which requires a brief foray into theoretical computer science) and propose the definition of a new dynamical exponent based on parallel computing. The new definition agrees with the conventional definition for the Metropolis and Swendsen-Wang algorithms. For the Wolff algorithm the relation between the two definitions is unclear and parallelization of the Wolff algorithm raises interesting questions.

One algorithm can be better than another in practical ways or fundamental ways. The practical question is how to most easily answer your problem given the resources available to you. The architecture, operating system, compiler, speed and memory of the available machines all play a role as do softer considerations such as how easy it is to code competing methods. Fundamental measures of performance ignore many of these real world limitations. Why would we want to consider such measures? First, we may come up with general results that can be usefully refined for answering specific real world questions. But there is another reason for investigating the performance of algorithms in an idealized, fundamental setting that is less well appreciated. An algorithm that performs well for a given system often encodes a deep understanding of the system and elucidating why a "good" algorithm works yields insights into the

system. Cluster algorithms illustrate this point. As we will discuss in greater detail below, cluster algorithms work better than the Metropolis algorithm near critical points because the basic Monte Carlo move of a cluster algorithm coincides with the critical fluctuations of the spin system.

To compare Monte Carlo algorithms from a fundamental perspective we need to choose a standard idealized model of computation and a standard task to perform. We will examine various idealized models of computation and argue that two appropriate though inequivalent choices are the "random access machine" and the "parallel random access machine." The standard task is to sample system configurations from the probability distribution defining the model, usually one of the equilibrium ensembles. [1] This is, of course, exactly what the Metropolis or Swendsen-Wang algorithms are designed to do.

CRITICAL SLOWING AND CLUSTER ALGORITHMS

The Metropolis algorithm performs poorly at critical points because its elementary move is local while critical correlation are long ranged. The result is *critical slowing*—at the critical point, the time τ to reach equilibrium diverges as a power of L, the linear size of the system, according to,

$$\tau \sim L^z, \tag{1}$$

where z is the *dynamic exponent*. The conventional unit of time for measuring τ is a *sweep*, the time it takes to attempt to flip every spin in the lattice once. A sweep is chosen as the unit of time to conform to the time as measured in physical systems where dynamics is parallel. The dynamic universality hypothesis asserts that the Metropolis algorithm simulating a critical spin system has the same dynamic exponent as a physical system with the same order parameter symmetry, short range interactions in the same dimensionality and non-conserved dynamics.

For the Metropolis algorithm simulating the Ising critical point $z \approx 2$. A simple, non-rigorous argument explaining why z is near 2 is that, at criticality there are partially ordered domains comparable to the system size. The local dynamics moves domain walls by a diffusive process so that reorganizing critical domains requires the time for diffusing a distance L, which is L^2. In Ising systems the values of z appear to be somewhat larger than 2.

Cluster algorithms partially overcome critical slowing by flipping large clusters of spins in a single step. The first cluster algorithm was due to Swendsen and Wang (SW) [1]. The original SW algorithm was applied to Ising-Potts models but the method has since been generalized to a number of other classical spin systems [3–6]. As applied to the Ising model, the algorithm is quite simple to describe and consists of two kinds of moves, a *spin move* and a *bond move*. In the bond move a bond configuration ω is created from the spin configuration σ. The spin configuration, σ is a list of the spin

---

[1] In many cases it is sufficient to generate configurations from a good approximation to an equilibrium ensemble.

values, s_i at each lattice site i where each spin can take the value $+1$ or -1 representing up and down spins. A bond configuration ω is a list of bond occupation numbers, n_{ij} on each nearest neighbor bond ij with n_{ij} taking values 0 and 1 representing *occupied* and *unoccupied* bonds, respectively. Given a spin configuration, a bond ij is *satisfied* if $s_i = s_j$. The bond move consists of occupying satisfied bonds with probability

$$p = 1 - e^{-2\beta J}, \tag{2}$$

where $2J$ is the energy difference between a satisfied and unsatisfied bond and β is the inverse temperature in energy units. Unsatisfied bonds are never occupied.

The occupied bonds generated during the bond move define a subgraph of the lattice and this subgraph consists of a number of connected components (including single sites not touching any occupied bonds) or *clusters*. Note that the rule forbidding the occupation of unsatisfied bonds means that each cluster contains only one spin type. The spin move of the algorithm consists of identifying the occupied bond clusters and flipping each cluster with probability $1/2$. Flipping a cluster means flipping every spin in the cluster, if i is in the cluster, $s_i \to -s_i$. It is straightforward to prove that the SW algorithm satisfies detailed balance and is ergodic so that it converges to the canonical ensemble for inverse temperature β. It is something of a miracle that the clusters defined by this procedure encapsulate the critical degrees of freedom of the Ising system. Presumably this is the underlying reason for the success of this algorithm as discussed below.

In an SW algorithm, a sweep is defined as a bond move followed by a spin move. In units of sweeps, the dynamic exponent for the SW–type algorithms, employed on a variety of systems, have consistently proved to be smaller than the corresponding Metropolis exponent. For example in the two-dimensional Ising model the SW exponent it is sufficiently small that it is difficult to distinghish logarithmic growth from a power near 0.2. For the three-dimensional Ising model z is near 0.5 and for the three state, two-dimensional Potts model, where the most accurate measurements have been made, $z = 0.52$ [7].

Wolff [2] described a cluster method closely related to the SW algorithm but with two innovations. First, well worth mentioning but not of any particular relevance to the present discussion, Wolff introduced the idea of an Ising embedding to extend cluster methods to spin systems with continuous symmetries. Second, Wolff proposed growing and then flipping only a single cluster in an elementary Monte Carlo step. It is the second innovation that we will consider here in some detail. In the context of the Ising system, the single cluster Wolff algorithm works as follows. A site is chosen at random on the lattice and a cluster is grown from that site. The growth process consists of occupying satisfied bonds on the perimeter of the growing cluster with probability p, given in Eq. (2), until there are no further bonds to be considered. The cluster formed in this way is then flipped with probability one half. The Wolff algorithm is both easier to code and in some sense it is more efficient than the SW algorithm. The conventional way to define a sweep for Wolff dynamics is to grow and flip a sufficient number of clusters that each spin is flipped on average once. Since the average Wolff cluster has a size that scales as $L^{\gamma/\nu}$ with γ the susceptibility exponent, one sweep of Wolff algorithm requires $L^{d-\gamma/\nu}$

TABLE 1. The sequential dynamic exponent for several critical spins models and algorithms. The dynamic exponent is for the energy integrated autocorrelation time. SW dynamic exponent for the 3-state Potts model is taken from [7]. Other dynamic exponents are from [9, 10].

Model	z_s^{Wolff}	z_s^{SW}	α/ν
2D Ising	0.25	0.25	0
2D 3-state Potts	0.57	0.52	0.4
3D Ising	0.33	0.54	0.2

cluster flips. When measured in this way, the dynamic exponent of the Wolff algorithm is less than or about equal to that of the SW algorithm as shown in Table 1.

It is instructive to cast the Wolff algorithm in a form that is similar to the SW algorithm. Given a spin and bond configuration we can choose a site at random, identify the cluster attached to that site and flip it. After the cluster is flipped, only those bonds inside the cluster and on its boundary are replenished according to usual rule of occupying satisfied bonds with probability given by Eq. (2). This procedure is statistically identical to growing a cluster from the chosen site.

Cluster algorithms do not completely eliminate critical slowing. For Ising and Potts systems, Li and Sokal [8] proved a bound on the dynamic exponent for SW,

$$z \geq \alpha/\nu, \tag{3}$$

where α is the specific heat exponent and ν is the correlation length exponent. This bound has been generalized to all algorithms of the SW–type [4]. In practice it turns out that the Li-Sokal bound is nearly satisfied as an equality for many systems and, since α/ν is small for many models, SW type algorithms are frequently very effective at reducing critical slowing. For the Wolff algorithm, no bound has been proved but $z \geq \alpha/\nu$ appears to hold.

MODELS OF COMPUTATION AND THE DYNAMIC EXPONENT

A fundamental measure of the performance of Monte Carlo algorithms requires a well-defined model of computation. Possible computational models include Turing machines, cellular automata, random access machines (RAM) or parallel random access machines (PRAM) [11]. These devices are shown in Fig. 1 on a two-dimensional diagram that classifies them according to number of processors and constraints on communication. The original model for fundamental investigations in the theory of computing was the Turing machine. It has one processor that moves along a one-dimensional tape. Because data is stored on a one-dimensional tape and is accessed by the motion of the head, time on a Turing machine does not agree with the intuitive notion of Monte Carlo time. For example, one sweep of the Metropolis algorithm for a two or higher dimensional system

takes time that is more than linear in the number of spins for the obvious reason that neighboring spins in 3D cannot be stored in neighboring positions along the tape.

Cellular automata are an attractive candidate for measuring Monte Carlo time because their parallelism is similar to that of the physical world. Time on a cellular automaton whose dimensionality and number of processors agrees with the model to be simulated yields a dynamic exponent for the Metropolis algorithm that captures the intuitive notion of measuring time in sweeps. Unfortunately, cellular automata time does not agree with our intuition about how to measure time for cluster algorithms. Efficient cluster identification involves long range transmission of information and a single sweep of a cluster algorithm cannot be carried out in constant or logarithmic time on a cellular automata.

The RAM and PRAM both allow for global communication. The RAM is an idealized and simplified version of the ubiquitous microprocessor. It consists of a single processor with a simple instruction set that communicates with a global random access memory. The notion of time on a RAM presumes, unphysically, that any memory cell can be accessed in unit time. The RAM is the customary way of thinking about *computational work* or the number of elementary operations needed to carry out a computation. A formal definition of the conventional notion of Monte Carlo time is computational work per spin, that is RAM time divided by the number of spins (or other degrees of freedom). Since a RAM is reasonable approximation to a single processor workstation, the conventional definition of Monte Carlo time often provides a practical way of deciding among algorithms. However, RAM time divided by number of spins is not always a good measure of time on a parallel computer because some algorithms cannot be efficiently parallelized even if one is given an essentially unlimited number of processors with unconstrained communication between processors.

The PRAM is an idealized model of parallel computation. The PRAM consists of many identical, except for distinct integer labels, processors all connected to a global random access memory. The processors are the same as the processor of a RAM, each is a stripped down microprocessor. The number of processors is allowed to grow polynomially (as a power) of the problem size, in our case the number of degrees of freedom to be simulated. The PRAM is a synchronous machine and each processor runs the same program though they carry out different computations depending on their integer labels. As in the case of the RAM, it is assumed that the each processor can communicate with any memory cell in unit time. Because of the ability of every processor to communicate with any memory cell, PRAM time is, like RAM time, unphysical. Nonetheless, PRAM time is a fundamental computational resource that measures the number of logical steps needed to complete a task. This idea is made more perspicuous by an equivalent computational model, *Boolean circuit families*. A Boolean circuit is a feedforward (e.g. loopless so in a computation each gate evaluates once) network of AND, OR and NOT gates with a fixed number of inputs and outputs. The *depth* of a Boolean circuit is the maximum over all paths from inputs to outputs of the number of gates along the path. Given the assumption that logical evaluations take much longer than communication between gates, the time it takes for the circuit to evaluate is proportional to its depth. A PRAM running a specific program is equivalent to a family of Boolean circuits, one circuit required for each problem size. The program is embodied in the wiring of the circuit and the time on a PRAM is equivalent to the

depth of the circuit. Computational complexity theory can be equivalently formulated in terms of PRAMs or Boolean circuit families. Thus PRAM time measures the number of parallel logical steps to go from inputs to outputs.

As an example of the power of parallel computation, consider the addition of n numbers. On a RAM this requires $\mathcal{O}(n)$ time but on a PRAM with $n/2$ processors it requires $\mathcal{O}(\log n)$ time. The procedure is to have each processor add a pair of numbers and send the result back to memory. After each step the number of summands is reduced by half so that logarithmically many steps are needed to carry out the sum. The corresponding Boolean circuit is a binary tree. Surprisingly difficult task can be carried out quickly in parallel. Relevant to the discussion of cluster algorithms is the problem of identifying the connected components of a graph. Given a graph with n nodes, the connected components can be identified in $\mathcal{O}(\log^2 n)$ time using n^2 processors on a PRAM. The implication of this is that the bond move of the SW algorithm can be carried out in *polylogarithmic*[2] time on a PRAM.

Finally, for purposes of measuring the performance of Monte Carlo algorithms, it is useful to imagine that the RAM and PRAM are equipped with ideal sources of random bits. The standard task for a Monte Carlo algorithm in statistical physics then consists of converting uncorrelated randomness to typical system configurations.

Let us now state two possible ways of measuring the sampling or equilibration time τ for a Monte Carlo algorithm. The conventional definition is formalized as the time on a RAM divided by the number of degrees of freedom, L^d and will be indicated by a subscript s for "sequential" or "sweep", e.g. τ_s or z_s. The proposed new definition is the time on a PRAM and will be indicated with a subscript p, e.g. τ_p or z_p. Related ideas for using PRAM time to define "critical slowing" were discussed in [12]. The sequential definition of Monte Carlo time is a measure of computational work while the parallel definition measures logical steps. In both cases communication time is ignored. A rough analogy that might help illuminate the difference concerns (extreme) strategies for erecting an office building, the building playing the role of the typical system state. One strategy is to minimize total cost not worrying about elapsed time. If a dollar spent corresponds to an elementary logical operation this strategy is the analog of minimizing computational work. A second strategy minimizes the time to completion without regards to cost or number of workers and machines on the job, analogous to minimizing parallel time. Note that the building cannot be put up arbitrarily quickly even with lavish resources because the interior work cannot be done until the foundation and superstructure are finished.

RESULTS

For the Metropolis algorithm the sequential and parallel definitions of Monte Carlo time are in agreement. The Metropolis algorithm is easy to parallelize and the full power of the PRAM is not needed. Similarly, for the SW algorithm the two definitions agree, at

---

[2] A function grows polylogarithmically in n if it is bounded by some power of the logarithm of n.

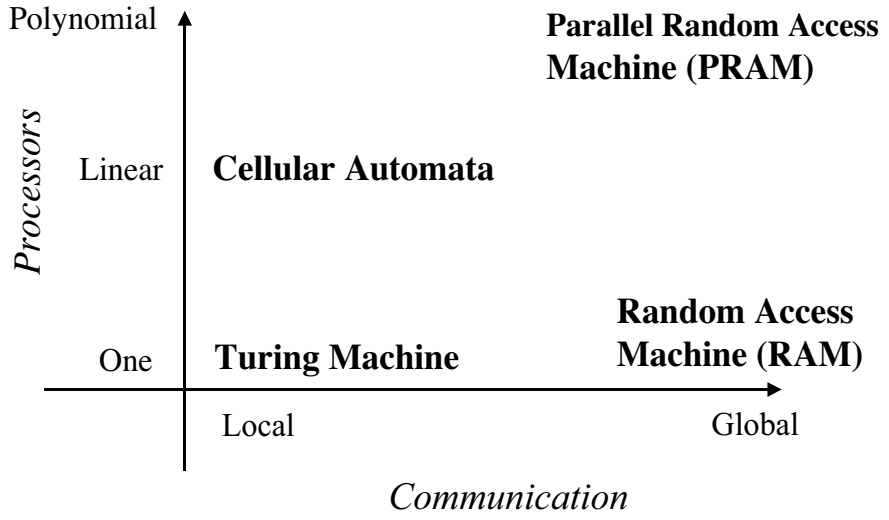

FIGURE 1. Several fundamental models of computation distributed in the horizontal direction according to whether they have local or global communication and in the vertical direction according to the degree of parallelism.

least up to logarithmic factors. The most complex part of carrying out a single SW step is identifying the clusters of bonds, which requires $\mathcal{O}(\log^2 L)$ time on a PRAM. Note that non-local communication and L^{2d} processors are essential for carrying out a SW step in polylog time. Thus, for both Metropolis and Swendsen-Wang, $z_p \leq z_s$. It seems unlikely that many sweeps of either algorithm can be carried out in one parallel step [3] so we conjecture that $z_p = z_s$ for Metropolis or SW.

For the Wolff algorithm, the situation is more complicated. The Wolff algorithm is equivalent to identifying all clusters in a bond configuration, choosing one site at random, flipping the cluster containing the chosen site and then replenishing the bonds inside and on the boundary of the cluster. This straightforward implementation of the Wolff algorithm would require $L^{d-\gamma/\nu}$ parallel steps to perform one sweep and, using the hyperscaling relation $\gamma + 2\beta = d\nu$, it sets a lower bound, $z_p^{\text{Wolff}} \leq z_s^{\text{Wolff}} + 2\beta/\nu$.

It is possible to flip more than one Wolff cluster in a single parallel step since two successive clusters can be flipped in either order so long as they do not overlap or abut one another. Indeed, one can flip all clusters connected to a sequence of randomly chosen sites in a single parallel step so long as no two sites fall in the same cluster or abutting clusters. After a group of clusters is flipped all the bonds interior to and on the boundary of these clusters are replenished. We have simulated this parallelization for the 2D Ising model by carrying out a long run of the conventional Wolff algorithm. During the run, we parse the Wolff clusters into bunches corresponding to parallel steps such that each bunch is as large as possible but without any overlaps or abutments. We found that the number of clusters flipped per parallel step diverges as a small power of L strongly suggesting that the upper bound is strict, $z_p^{\text{Wolff}} < z_s^{\text{Wolff}} + 2\beta/\nu$. There

[3] **P**-completeness results for SW and Metropolis dynamics [12] support this assertion.

may be other parallel implementations that allow even more clusters to be flipped in a single step. Thus, all that can be said with some confidence at the moment is that, $z_s^{\text{Wolff}} \leq z_p^{\text{Wolff}} < z_s^{\text{Wolff}} + 2\beta/\nu$.

DISCUSSION

We have examined the notion of Monte Carlo time and discussed two formal definitions. The conventional definition can be stated in terms of time on an idealized sequential computer, the random access machine. The proposed new definition is formalized in terms of time on the parallel random access machine, an idealized parallel computer. The dynamic exponents resulting from the two definitions of Monte Carlo time agree for the Metropolis and Swendsen-Wang algorithms. Efficient PRAM implementations of the Wolff algorithm are not obvious and, currently, little is known about its parallel dynamic exponent.

Cluster algorithms, when available, are more efficient than local algorithms such as Metropolis for classical spin systems with long range correlations. In the sequential domain, the Wolff single cluster method is typically more efficient than the Swendsen-Wang all cluster method. In the ideal parallel domain, cluster algorithms retain their advantage over local algorithms but it is an open question whether Swendsen-Wang, Wolff or some other parallel cluster algorithm is generally most efficient.

ACKNOWLEDGMENTS

This work was supported by NSF grants DMR-0242402 and DMS-0306167.

REFERENCES

1. Swendsen, R. H., and Wang, J.-S., *Phys. Rev. Lett.*, **58**, 86 (1987).
2. Wolff, U., *Phys. Rev. Lett.*, **62**, 361 (1989).
3. Kandel, D., and Domany, E., *Phys. Rev. B*, **43**, 8539 (1991).
4. Chayes, L., and Machta, J., *Physica A*, **239**, 542–601 (1997).
5. Chayes, L., and Machta, J., *Physica A*, **254**, 477–516 (1998).
6. Redner, O., Machta, J., and Chayes, L. F., *Phys. Rev. E*, **58**, 2749–2752 (1998).
7. Salas, J., and Sokal, A. D., *J. Stat. Phys.*, **87**, 1 (1997).
8. Li, X.-J., and Sokal, A. D., *Phys. Rev. Lett.*, **63**, 827 (1989).
9. Baillie, C. F., and Coddington, P. D., *Phys. Rev. B*, **43**, 10617–10621 (1991).
10. Coddington, P. D., and Baillie, C. F., *Phys. Rev. Lett.*, **68**, 962–965 (1992).
11. Papadimitriou, C. H., *Computational Complexity*, Addison Wesley, 1994.
12. Machta, J., and Greenlaw, R., *J. Stat. Phys.*, **82**, 1299 (1996).