Technical Report CSTN-104

Cluster and Fast-Update Simulations of Regular and Rewired Lattice Ising Models Using CUDA and Graphical Processing Units

K.A. Hawick[*], A.Leist and D.P.Playne
Institute of Information and Mathematical Sciences
Massey University – Albany, North Shore 102-904, Auckland, New Zealand
Email: { k.a.hawick, a.leist, d.p.playne }@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

August 2010

Abstract

Models such as the Ising system in computational physics are still important tools for analysing phase transitions and universal behaviours for new irregular and distorted lattice networks. Data-parallelism can be exploited to speed up such simulations as well as their analysis using general purpose graphical processing units (GPU) and other accelerating devices. We report on the use of various GPU performance optimisation techniques for both local update schemes like checkerboard Metropolis as well as non-local cluster-update algorithms such as that of Wolff for the Ising model. We present some data-parallel algorithms and code fragments along with performance analysis and discussion of optimal GPU data memory models for this work including bit packed spin-coding techniques. We report some results on changes to the critical temperature arising from probabilistic small-world network rewiring as well as giving algorithms and performance data on parallelising irregular Ising model systems. We employ NVIDIA's compute unified device architecture (CUDA) programming language and report on performance data achieved using modern many-core GPUs to optimise the "wall clock cost per decorrelated measurement" on both regular and irregular Ising models.

Keywords: Ising model; Metropolis Monte Carlo; cluster update; Wolff algorithm; CUDA, GPU.

[*]Author for Correspondence

1 Introduction

The Ising model for a magnet of spins on a lattice is still one of the most referenced models that exhibits a phase transition and which can be used as a base of comparison for other less well studied systems. The Ising model is parameterised in terms of a temperature $T = \frac{1}{Jk_B} = 0$, where J is the spin-spin coupling and k_B is Boltzmann's constant. Originally studied by Ising and Lenz in one dimension, where there is no finite temperature phase transition, the model exhibits a transition at critical temperature $T_c^{2-D} \approx 2.2692$ for its critical dimensionality of two, a result that was determined analytically by Onsager [1]. The three dimensional system – which is obviously the most relevant to real physical systems – has no known analytic solution, but a number of numerical simulation experiments over the last several decades [2, 3] have yielded the critical temperature as $T_c^{3-D} \approx 4.5115$. In systems of four dimensions and higher a finite temperature phase transition still occurs, but the nature of the transition is well described by mean-field theory. Values for the transition temperature in these higher dimensional systems are also known.

The critical temperature location of the transition is not the only numerical quantity of interest for comparing to other phase transitional systems, models and theories. The critical exponents $\beta, \nu, ...$ - which describe how the magnetic susceptibility, heat-capacity and other properties of the model behave near the transition [4] are also valuable outcomes of numerical simulations. Figure 13 shows how some measured properties of the model diverge at the critical temperature.

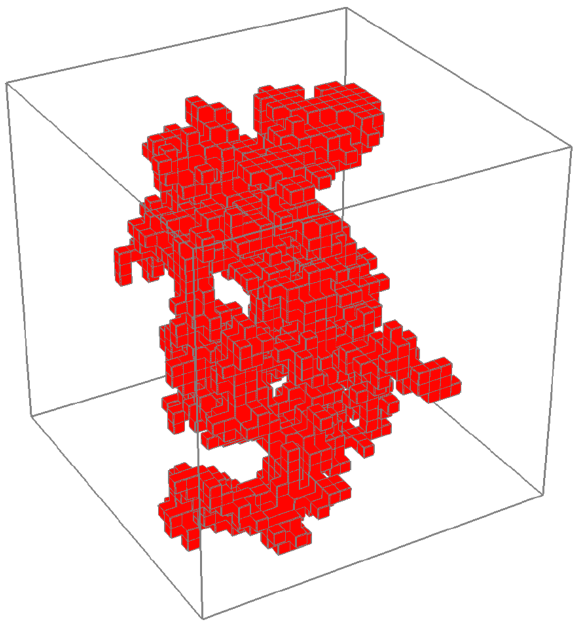

Figure 1: A typical Wolff cluster generated in a critical 3D Ising system of size $32 \times 32 \times 32$ sites.

Numerical approaches to the Ising model take the form of quenching a model system of "hot" i.e. randomized spins down to equilibrium for a particular temperature of interest. The Ising model does not have a physical dynamics of its own so various artificial time evolutionary schemes are employed. These Monte Carlo updating procedures allow an exploration of the phase space of the possible configurations of the model. Generally Monte Carlo procedures for this sort of model fall into two categories. Schemes like Metropolis or Glauber give a dynamics that bears some resemblance to a physical time and can be used to study growth dynamical properties. Other algorithms such as the cluster update schemes of Swendsen and Wang and of Wolff are used to find appropriately weighted but independent samples of the model phase space, as rapidly and efficiently as possible and therefore do not necessarily produce a physically meaningful dynamics. Figure 1 shows a typical large cluster constructed by the Wolff algorithm in a 3D Ising system.

Considerable work has been reported in the literature on the effectiveness of these various algorithms at lowering the dynamical exponent z which gives a measure of how decorrelated successive model configurations are [5]. The Wolff cluster update algorithm is known to have the best (lowest) dynamical exponent for the three dimensional Ising model but it is not a trivial algorithm to implement efficiently on fast parallel computers. These algorithms involve constructing a cluster of spins with appropriate Boltzmann thermal probability and flipping or updating the entire cluster at once. This solves the problem of critical slowing that drastically affects conventional Monte Carlo Ising model update algorithms near the critical temperature.

Parallelising cluster update algorithms has been partly addressed for massively parallel supercomputers and in this present paper we demonstrate how Wolff can be implemented on data parallel computer systems such as Graphical Processing Units (GPUs). Many of the implementation reports in the literature simply quote spin updates per second as a measure of algorithm success. However in this paper we propose and explore the rather more important metric of independent (decorrelated) measurements per second. We give some detailed discussion of the operational issues behind a numerical simulation experiment of a model like the Ising system, in multiple dimensions, with various lattice sizes. We discuss the physical measurements and the associated data parallel and operational algorithms necessary to obtain them.

Our particular interest in the Ising model is as a reference system to explore the effect of small-world network [6, 7] rewiring phenomena. This involves probabilistically rewiring the Ising lattice connections with a small probability p which gives a means of interpolating between perfectly regular cubic or square lattice graphs ($p = 0$) and random graphs ($p = 1$) in a quantitatively controlled way. Some work on this problem has been reported by Herrero but the difficulty in exploring very small p values is that it requires large numbers of simulations to be carried out on quite large network sizes. Recent developments in commodity priced parallel computing systems such as GPUs is therefore opening up new low-p regimes of study.

We report on some of our preliminary low-p rewired Ising studies in Section 8 as motivation for our work, but the main contribution of this paper is to show that: it is overwhelmingly worthwhile to implement cluster algorithms such as Wolff for Ising systems near the critical temperature and that this is feasible on a GPU. We also show that a hybrid mix of cluster and non-cluster updates is optimal both for model system equilibration and for obtaining independent decorrelated measurements. The hybrid algorithm mix ratio to choose is not however trivial as it depends upon both the temperature T regime being explored and also on the rewiring probability p and of course what is interesting is that the critical temperature T_c is itself dependent on p so finding the right choices of update algorithm is an iterative process.

The Ising model has a rich scientific history [8] but has also been the motivation behind a lot of parallel simulation code performance optimisation work on conventional supercomputers in the 1970s and 1980. It is also proving an important target for accelerator processing devices such as Graphical Processing Units [9, 10]. Recent work by Block *et al.* reports on how multi-spin coding can improve memory usage and hence CPU/GPU memory latency effects for conventional Metropolis/Glauber update algorithms for the Ising model [11]. In our present paper we also discuss use of multi-spin bit packing techniques for regular un-rewired ($p = 0$) lattices and show how they compare with Wolff cluster updates. We therefore report on both conventional and cluster updates algorithms; on regular and rewired networks, as well as on operational and measurement decorrelation issues.

There is continued interest in the physical meaning and interpretation of the clusters formed in Ising systems as well as in the use of cluster update algorithms [12] for other models with longer range coupling interactions [13]. We are however unaware of any report in the literature giving a detailed analysis of operational issues concerning hybrid algorithms, and measurement concerns and we hope our performance observations will be of use to researchers in choosing the most efficient hybrid algorithmic mix for further Ising experiments.

In section 2 we describe both the conventional Ising model as well as the "small-world" rewired version that we are studying. In section 3 we describe spin cluster update methods such as that of Wolff, that are used to speed up exploration of the Ising model phase space near the critical temperature. We provide a brief summary of important features of GPU memory and processing capabilities in section 5 and in section 4 we give an outline of our operational simulation method. This explains some of the general methods that were applied both for speeding up regular lattice Ising systems described in section 6 and irregular graph and network-based systems with rewired lattice structures described in section 7. For discuss the general hybrid algorithms that we believe are optimal for this class of problem in section 8 and summarise and suggest some areas for further work in section 9.

2 Ising Small-Worlds

The Ising model is usually formulated on a regular d-dimensional hyper-cubic lattice where each magnetic spin variable is connected to $2 \times d$ nearest neighbouring sites. The model Hamiltonian (or energy function) is usually written in the form:

$$H = -\sum_{i \neq j} J_{ij} S_i S_j \tag{1}$$

where $S_i = \pm 1$, $i = 1, 2, ...N$ sites, and J_{ij} is $|J| = 1/k_B T$ is the ferromagnetic coupling over neighbouring sites i,j on the network.

Many physics models such as the Ising [14], Potts [15], and Heisenberg [16] systems are formulated on a regular mesh or on a topologically regular system. For example the familiar square, cubic or hyper-cubic geometries have sites with 4, 6 or 8 nearest neighbouring sites. If the systems have periodic boundary conditions then it is possible to assign unique "ownership" of half of the bonds or links to a particular site. So each site of the square lattice has 2 bonds that can be uniquely associated with it. The lattices can be considered as graphs with twice as many edges as there are nodes.

It seems likely that many of the familiar properties of models such as the Ising Potts or Heisenberg models depend on the geometry of the underpinning lattice. An interesting question is to ask how strong this dependence is upon local properties such as number of neighbours as against the actual geometry itself. For instance, we can construct a lattice which has the same topology as the conventional square mesh and where each site will still have 4 neighbouring sites, but where the layout is no longer trivial. Such a lattice can be constructed by making pairwise perturbations to a conventional square mesh. A question of interest is, at what point, if any, the behaviour of the Ising model changes as a function of the number of changed bonds?

The effect of these connectivity "worm-holes" will allow information to propagate across long ranges or an alternative interpretation is the speed of propagation of information is increased. Intuition suggests that this long range communication mechanism will encourage long range order and thus raise the critical temperature of the system. Interpretation of the geometry of such a model is non-trivial. What does it mean for the model's physical space to be interconnected in such a manner? The axes and dimensions of the model are effectively inter-folded or mixed.

The bond topology distortions described above provide a means for information to propagate faster across the lattice during a simulation. A conventional Markov Chain Monte Carlo (MCMC) update approach will apply approximately one "hit" per site in a complete time step. Generally, on average, it will therefore take $O(L/2)$ steps for information to propagate across a lattice with edge length L. The factor of two is from likely periodic or wraparound boundary conditions. One or more "worm-holes" will allow changes in one site to propagate across long distances in considerably shorter times. The effect of the swapped bond pairs can therefore be interpreted as either making sites closer to one another in a distorted space metric, or as providing special "fast conduits" or "worm-holes" across which information can propagate more quickly than elsewhere on the lattice.

Some work has already been done on the behaviour of the Ising and related models on small-world systems, albeit mostly on 1-dimensional lattices. [17], [18], [19], [20], [21], [22]. The major work to date on higher dimensional Ising systems is [23].

Although the Ising model does not have a dynamics of its own, an artificial dynamics in the form of one of the variations of the MCMC update algorithms can be interpreted in terms of a simulation time. Investigations of the MCMC method in various contexts have shown that under certain conditions, MCMC simulation time does correlate and scale appropriately in a manner consistent with real physical systems' time [24]. The various algorithms we discuss have associated dynamical exponents describing how each progresses the Ising system through state space. Some good explanatory works include [25–27]. The key issue is that near the critical temperature the dynamics of local update schemes slows down. This makes it computationally very expensive to investigate changes in the critical temperature from a lattice rewiring or small worlds effect. It is necessary to repeat the (already) expensive calculations to obtain T_c with each different scanned value of rewiring probability p to obtain $T_c(p)$.

A fast update scheme is therefore highly desirable. Modern works studying critical systems like the Ising model use the Monte Carlo Renormalisation Group method [2,3,28]. This method unfortunately does not obviously extend to rewired lattice model systems. We therefore employ the non-local cluster update method developed by Wolff, which does have vastly improved dynamical exponent and therefore gives fast updates - near the critical temperature. The best hybrid mix of Wolff and Metropolis is one of the outcomes of our work in this paper.

3 Cluster Update Algorithms

The Metropolis Monte Carlo algorithm is but one of various possible Markov Chain approaches to the exploration of the phase space of a model like the Ising system. Various authors have proposed alternatives and there is still discussion on the validity and usefulness of various Monte Carlo algorithms, including the vexatious question of the significance of detailed-balance and sequential updates. See for example [29], [30], [31] [32], [33]. These works have given rise to specialised cluster identification methods [34], [35] and discussions of the effectiveness of the Wolff algorithm [36].

Cluster update algorithms for Monte Carlo simulations of models like the Ising system are well known and have been reported in the literature since the seminal work by Swendsen and Wang [37]. There are a number of variations possible, reviewed by various authors [38, 39]. Wolff's cluster update algorithm [40] seems to have received less attention than perhaps it deserves. While it is not trivial to program – especially for parallel versions – it does have the best dynamical exponent reported and therefore we thought it worthwhile investigating the trade-off between a local algorithm that is easy to program and which parallelises well and an algorithm like Wolff's which is slower in wall clock time but achieves more per invocation. To set this work in context, it is worthwhile to give a brief derivation of Monte Carlo updating that sets up a dynamical scheme for the Ising model.

Each microstate of the Ising system is completely specified by the set of spin variables $\{s_i\}$ given the coupling (reciprocal temperature). Let the probability functional $P(\mathbf{X}, t)$ be associated with the microstate $\mathbf{X} \equiv \{s_i\}$ at time t and consider the transition probability $W_{\mathbf{X}\to\mathbf{X}'}$ giving the likelihood of a change of microstate $\mathbf{X}$ to $\mathbf{X}'$. The following master equation can immediately be set up, requiring that the rate of change of probability of microstate $\mathbf{X}$ at time t be given by considering all transitions from $\mathbf{X}$ and all transitions to $\mathbf{X}$:

$$\frac{dP(\mathbf{X})}{dt} = -\sum_{\mathbf{X}'} W_{\mathbf{X}\to\mathbf{X}'} P(\mathbf{X}) + \sum_{\mathbf{X}'} W_{\mathbf{X}'\to\mathbf{X}} P(\mathbf{X}') \tag{2}$$

By requiring the algorithm to yield $P(\mathbf{X}) \to P_{eq}(\mathbf{X})$, the thermodynamic equilibrium probability of microstate $\mathbf{X}$ as $t \to \infty$ a solution of 2 with $\frac{dP(\mathbf{X})}{dt} = 0$ so that:

$$\sum_{\mathbf{X}'} W_{\mathbf{X}\to\mathbf{X}'} P(\mathbf{X}) = \sum_{\mathbf{X}'} W_{\mathbf{X}'\to\mathbf{X}} P(\mathbf{X}') \tag{3}$$

A proper derivation of the Monte Carlo update algorithm is based upon the condition of detailed balance [41]. It is common to use the stronger (but tractable) condition that:

$$\frac{W_{\mathbf{X}'\to\mathbf{X}}}{W_{\mathbf{X}\to\mathbf{X}'}} = \frac{P(\mathbf{X})}{P(\mathbf{X}')} \tag{4}$$

So that the probability of the system moving to microstate $\mathbf{X}$ is increased for highly probable microstates $\mathbf{X}$, and decreased for unlikely ones. It is then necessary to recognise that for a Boltzmann statistical weighting of the microstates, the probabilities $P(\mathbf{X})$ can be expressed in terms of the Hamiltonians $\mathcal{H}(\mathbf{X})$.

$$P(\mathbf{X}) = Ae^{-\frac{\mathcal{H}(\mathbf{X})}{k_b T}} \tag{5}$$

Where A is a normalising constant, k_b is Boltzmann's constant and T the temperature. Substituting 5 in 4 gives:

$$\frac{W_{\mathbf{X}'\to\mathbf{X}}}{W_{\mathbf{X}\to\mathbf{X}'}} = e^{-\frac{\{\mathcal{H}(\mathbf{X})-\mathcal{H}(\mathbf{X}')\}}{k_b T}} \tag{6}$$

This does not have a unique solution. We seek a practical microstate updating algorithm expressed in terms of:

$$\triangle\mathcal{H} = \mathcal{H}_{X'} - \mathcal{H}_X \tag{7}$$

which allows us a practical way of computing the transition probability from X to X'. The two most commonly used are [42,43]:

$$W_{\mathbf{X}\to\mathbf{X}'} = \begin{array}{ll} \frac{1}{\tau}e^{-\frac{\triangle\mathcal{H}}{k_b T}}, & \triangle\mathcal{H} > 0 \\ \frac{1}{\tau}, & \triangle\mathcal{H} \leq 0 \end{array} \tag{8}$$

$$W_{\mathbf{X}\to\mathbf{X}'} = \frac{1}{\tau}\left(\frac{e^{-\frac{\triangle\mathcal{H}}{k_b T}}}{1+e^{-\frac{\triangle\mathcal{H}}{k_b T}}}\right) \tag{9}$$

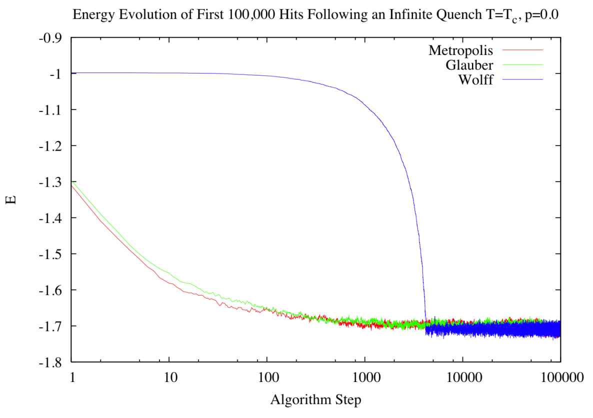

Figure 2: Different equilibration rates for Metropolis, Glauber and Wolf algorithms on a 128 × 128 Ising system, showing the Energy converging, to within its fluctuation range.

The equilibration rates of these algorithms are illustrated in figure 2 where the local Metropolis and Glauber algorithm both gradually converge to within the thermal fluctuation range of a mean value. The Wolff algorithm takes some time to build up big enough clusters within the system, after which it very rapidly takes the system close to its mean value for the quench temperature. This data was generated with a reference serial code implementation that used a very conservative (and computationally wasteful) update procedure for Metropolis and Glauber using N randomly picked sites to hit each step.

It is open to argument which of the schemes produces the more realistic dynamics. Older works in the literature such as [43] have used the Glauber scheme for dynamical work, relegating the Metropolis algorithm to work where a thermodynamically equilibrated configuration is required and the dynamics is unimportant. It appears that for computational purposes, the Metropolis algorithm is better, since it involves one less floating point operation – a division, it produces a faster dynamics but similar equilibrium results, and a qualitatively similar trajectory through phase space. Figure 2 does indicate that Glauber is slower to take the system to thermal equilibrium.

The simulation time can then be defined in units of "an average of one Metropolis update attempt per site." It is important to include every site in this definition of the time-step. Without such a universal definition, it becomes impossible to compare other works in the literature and on slightly different models, in terms of their dynamical properties. The problem remains that this is only an artificially constructed time. It is also noticeable that for deep quench experiments, there is little thermal activity below the phase transition temperature and the system evolves very slowly. This effect is known as hydro-dynamic slowing down. This is a separate effect from the critical slowing down that occurs for a system close to its critical temperature. The latter is due to interactions across all length scales present in the system and is therefore an inevitable consequence of a local update algorithm like Metropolis or Glauber.

A number of alternate Monte Carlo algorithms have appeared in the literature over the years, all offering various speed ups in the dynamics and reduce the effect of the critical slowing down. Some of these are [44], [45], [37], and are reviewed in [46] and [47]. In general, they are all short-cuts to finding the most thermodynamically probable region of phase space, and thus are excellent for obtaining thermally equilibrated configurations although as figure 2 shows a pure Wolff cluster update starts off constructing very small clusters and therefore takes some time to "get going" after the initial quench.

The Wolff algorithm [47] involves constructing a "cluster" of like spins and potentially flipping the spin sign of this entire cluster. In consequence, this algorithm traverses micro-state space in larger "jumps" than either the Metropolis or Glauber update algorithms. This is particularly important in models of critical phenomena like the Ising model whereupon the Wolff algorithm does not suffer from the critical slowing down phenomenon that typically plagues other simulation algorithms near the critical temperature.

The key feature of the Wolff algorithm's ability to meet the detailed balance condition in equation 3 is the precise way that clusters are constructed. The Wolff algorithm is essentially:

- Pick a random lattice site, i, as the first point of a cluster to be built
- Flip spin s_i and remember i
- Visit all links connecting site i to its nearest neighbours j
  - The bond $i - j$ is activated with probability $1 - e^{\min(0,2\beta)}$
  - Where this happens s_j is flipped and site j is marked as part of the cluster
  - Continue this process for all bonds leading to unmarked neighbours until the process stops
- Ergodicity is guaranteed since there is always a non-vanishing probability that the cluster consists of only one site.

where $\beta = J/kT$ is the reciprocal temperature.

The implications of this algorithm are that a Wolff cluster being constructed as part of the time-evolution of the system is bounded in size by the extent of the physical cluster in which the initial random site is chosen. We believe the size distribution of Wolff clusters is closely related to the physical cluster size distribution for systems in equilibrium. Figure 1 shows that these can be very large and even percolated clusters, so flipping them constitutes a highly non-local and multi-scale update algorithm. The downside being that even using advanced cluster component labelling techniques [48] the Wolff algorithm is computationally expensive to apply compared with even multiple Metropolis local update hits.

In [3] Baillie et al. found that a hybrid algorithm mixing Metropolis and Wolff updates was best both in quality and performance trade-offs on a data-parallel architecture such as the DAP [49] although the optimal ratio was not investigated in detail. In our present paper we consider how well a data-parallel device such as a GPU using the CUDA language can perform these hybrid algorithms but we also try to find the best (in terms of minimising wall-clock time) hybrid ratio. Due to the computational time and effort needed to do these calculations – especially when a scan in an additional parameter such as rewiring probability p is involved, a detailed investigation of the optimal algorithm ratio for use in operational experiments seems worthwhile.

4 Operational Simulations

There are several important time-scales we must consider when planning a simulation of the Ising model. Generally speaking, the work we discuss here involves infinite quenching experiments. We initialise the system from a "hot" or completely random mixture of "up" and "down" spins. The system is then evolved to thermal equilibrium for a finite temperature T. Once the system has reached equilibrium it is valid to measure macroscopic properties from individually sampled microstate configurations. Realistic statistical averages are obtained providing these samples truly represent the equilibrium properties of the system and also that they are sufficiently decorrelated so as not to introduce a systematic bias.

Consequently it is important that simulations be run for long enough that they are truly in thermal equilibrium for the quench temperature T. Figure 2 shows the approach to equilibrium achieved by the Metropolis, Glauber and Wolff algorithms on the Ising model at critical temperature T_c. The curves show the behaviour of the system's total energy E as it converges to an equilibrium value. Subsequently variables such as E will fluctuate around a mean thermal average value. It is important to distinguish between the equilibration period and the equilibrated regime. Since measurements from the equilibration phase would bias or contaminate measurements, most researchers choose to err on the side of caution and discard potentially useful measurements to be completely sure of the system being equilibrated. Similarly, if critical temperatures or exponents are being measured to precision, biases and hence systematic errors would be obtained from using measurements from configurations that were correlated. To avoid this most researchers will again err on the side of caution and do over many updates between measurements to avoid bias.

Operationally, all these wasted computations add up and can add very significantly to the wall clock computational time used up by an experiment. Since work like this can take months if not years of processing time, even with parallel computations, any savings by more accurately estimating the minimum safe number of equilibration steps to avoid measurement steps to discard is worthwhile.

To obtain statistically independent or decorrelated samples it is important that the update algorithm provide an efficient means for the system to reach equilibrium conditions quickly but also to traverse its micro-state space at equilibrium with ideally no correlations at all between subsequent measurement configuration snapshots.

4.1 Operational Procedure Summary

The operational procedure for investigating the Ising model can be summarised as follows:

1. Choose dimension d, lattice side length L and rewiring probability p
2. initialise static lattice/network structure
3. quench from random spin start state
4. use appropriate update algorithm to reach equilibrium temperature
5. use appropriate update algorithm to obtain decorrelated independent measurements of M, E, Cv, χ, U

Where M is the magnetisation and is measured from a single configuration as:

$$M = \frac{1}{N} |\sum_i S_i| \tag{10}$$

The total energy E is obtained from the Hamiltonian in equation 1 but the heat capacity (at constant Volume) $C_v \equiv \frac{\partial <E>}{\partial T}$ is more usefully computed using the fluctuation dissipation theorem from the energy variance using:

$$C_v = \frac{\beta}{T} \left[\langle E^2 \rangle - \langle E \rangle^2 \right] \tag{11}$$

where β is the reciprocal temperature in units of the couple J. Similarly the magnetic susceptibility $\chi \equiv \frac{\partial \langle M \rangle}{\partial H}$ in terms of the magnetic field, but is more usefully calculated in terms of the variance relationship:

$$\chi = \beta \left[\langle M^2 \rangle - \langle M \rangle^2 \right] \tag{12}$$

We employ Binder's finite scaling procedure to obtain the location of the critical temperature since this calculation still works even with non-zero lattice rewiring p unlike procedures like Monte-Carlo Renormalisation Group which work only for regular unperturbed lattices. Finite size scaling is based on Binders observation that the cumulant U forms an s-shaped curve as it varies with temperature and that the curve becomes sharper as the system size is increased. Two scans in temperature for two different system sizes will intercept one another at the critical temperature T_c. For our work, we typically use *three* different system sizes which gives us an error estimate as well as a value for $T_c(p)$. We subsequently compute the temperature shift $\tau = |T_c^0 - T_c^p|$ caused by the rewiring and report that - as shown in Figure 12.

The Binder cumulant for a system of size $N = L^d$ is computed from individual configuration measurements of the Magnetisation M by:

$$U_N = 1 - \frac{< M^4 >_N}{3 < M^2 >_N^2} \tag{13}$$

Plots of heat capacity and magnetic susceptibility (see Figure 13) are useful as a quick visual indicator of the location and shift in the critical temperature, but the cumulant method allows more precise numerical regression fitting and intercept calculations that yield the necessary precision.

Operationally there are a number of ways to manage these calculations. We typically log independent measurements of M and E to file during a "run" and subsequently combine and analyse these to obtain U and thence $T_c(p)$ values.

The important summary point to note is that the computational cost of all these is dominated by the cost of obtaining suitably independent or decorrelated snapshot configurations of the Ising spin system that yield statistically unbiased measurements. This process is governed by the critical slowing-down phenomena near the critical temperature. Algorithms based upon hybrids with the Wolff procedure alleviate this.

4.2 Equilibration & Decorrelation

In addition to the need to equilibrate the system so that its properties are representative of the quench temperature, it is also important to consider how much it changes between measurement steps. In analysing the independence of the configurations measurements and the associated lack of bias in the measurement statistics, it is useful to consider the population correlation coefficient ρ, defined as:

$$\rho_{X,Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y} \tag{14}$$

where E is the expected value operator, μ_X and μ_Y are the expected values and σ_X and σ_Y the standard deviations respectively.

The correlation or overlap should ideally be zero between measurement spin configurations. In figures 6,9 and 14 we show how the correlation changes with both algorithmic steps and with wall clock time, and this assists in determining the ideal hybrid algorithmic ratio to maximise the number of independent measured configurations computed per second.

5 Graphical Processing Units

The advancement of the games industry has driven the development of Graphical Processing Units (GPUs) into highly parallel and powerful architectures. The latest generation Fermi-architecture GPUs from NVIDIA contain between 11-15 multiprocessors, each of which contain 32 scalar processors (SPs). Previous generation GPUs contained 8 scalar processors per multiprocessor.

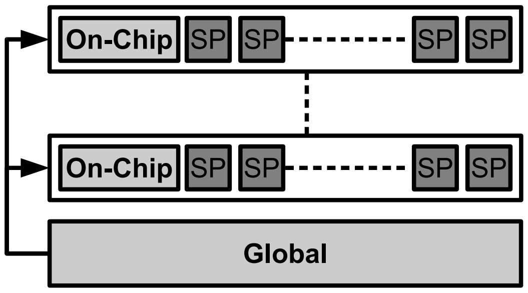

Figure 3: General architecture of NVIDIA GPUs.

Programs can be executed on the GPU by splitting the computation into thousands or even millions of threads. These threads (called kernels) are managed by the GPU hardware and scheduled to execute on the multiprocessors. To allow some degree of control over how these threads are scheduled, they must be organised into blocks. Each block can have a maximum size of 1024 threads for a Fermi architecture GPU and 512 threads for previous generations.

GPUs contain several types of memory, using the best memory type and accessing it in the correct way can have a large impact on performance. The main type of memory on the GPU is **global** memory. This is the only memory type that can be accessed by the CPU host and any input or output must be placed in this memory type. In addition to this are the specialised memory types stored on the multiprocessor. These memory types allow threads within the same block (and thus executing on the same multiprocessor) to reduce access to global memory and share information.

6 Regular Lattice Simulation

While we are most interested in investigating update algorithms for rewired lattices, it is also of importance to investigate how the algorithms perform on regular lattices. Regular lattice algorithms can be more highly optimised as the number of neighbours each site has for a hyper-cubic d dimensional lattice is known ($2d$ neighbours). The memory location of these neighbours can also be easily calculated, eliminating the need to store the addresses. In this section we discuss and compare optimised implementations of three regular lattice update methods: Checkerboard, Wolff and a Hybrid of both.

6.1 CUDA Metropolis Implementation

The Metropolis algorithm is well suited to computation on the GPU, the entire mesh can be updated with two kernels (red/black). In our previous work [10] we discussed the implementation of this algorithm on CUDA GPUs. In this work, substantial performance benefits were gained by packing the spins into unsigned integers. Each unsigned integer in the lattice contains 32 spin values. The Ising model is updated by using logical bit operations to compute the model on all 32 spins simultaneously. On a GTX 260 graphics card this implementation could apply a checkerboard update approximately 125x faster than a traditional single-core CPU implementation.

6.2 CUDA Wolff Implementation

Unfortunately the method of packing spins into unsigned integers does not transfer to the Wolff algorithm. The main computational portion of the Wolff algorithm is to build the Wolff cluster in the system. In a system with N spins, the maximum label value will also be N. Thus storage requirements for each site is now no longer just one bit, but enough bits to store it's unique label.

As discussed in Section 3, Wolff clusters are not like normal connected component clusters. Some sites are excluded from the cluster according to the temperature. To save information about this exclusion, we also store information about links to a site's neighbours. Each cell contains information about the links to the cells left and above it. All of this information is stored in an unsigned integer, as seen in Figure 4.

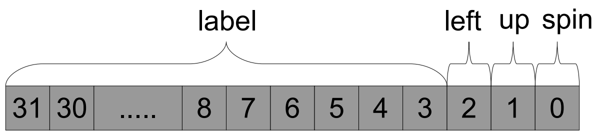

Figure 4: Structure of cell information packed into a unsigned integer.

Using this structure we can provide each kernel with the information for each cell from one global memory read. Each kernel must still access neighbouring values but again this structure is beneficial as the spin values, connection information and label must be read for each neighbouring cell anyway.

Building a Wolff cluster around a randomly selected site is inefficient on a GPU, we instead perform a modified Connected Component Labelling (CCL) algorithm. In previous work we have described how a CCL algorithm can be implemented efficiently on CUDA enabled GPUs [48]. To build a Wolff cluster rather than a normal connected component labelling cluster we modify the algorithm slightly.

Before we perform the CCL algorithm, the bits representing the links between cells are randomly set according to the temperature of the system. This determines how cells are randomly excluded from the cluster. When the CCL algorithm builds the labels, two cells are only considered to be in the same cluster if they have the same value and the link between them is set to 1. In this way we can make use of our CCL algorithm to build Wolff clusters instead of normal connected-component clusters.

Once the clusters have been labelled, we pick a random site and fetch that site's label. Then all the sites with that same label will be flipped. This process performs one Wolff update.

6.3 CUDA Hybrid Implementation

Both the Checkerboard and the Wolff update algorithms have their individual merits, a hybrid of both can provide better performance than both. The Checkerboard algorithm updates each cell every step but takes a long time for change to propagate through the system while the Wolff algorithm changes the field very quickly but is generally slower and can take a long time to update each cell. By mixing calls of each algorithm together, the performance in terms of both time to reach equilibrium and produce de-correlated results can be improved.

This Hybrid algorithm uses the same data structure as the Wolff algorithm. This means that the checkerboard update steps are not as efficient as pure Checkerboard algorithm but the additional information is necessary for the Wolff updates. The higher performance of the more highly optimised Checkerboard implementation does not outweigh the time taken to convert between the two data structures.

The real question faced by this Hybrid algorithm is what ratio the algorithm calls should be mixed together. As Checkerboard update steps are generally a lot faster than Wolff updates, we test ratios with more Checkerboard steps than Wolff steps. In the following section we present the results on a regular lattice to identify the optimal ratio for the hybrid method.

6.4 Regular Lattice Performance

To investigate the effectiveness of the Hybrid update algorithm and compare it to the Metropolis and Wolff methods, we measure two properties. First we test how well the update methods can move the system to a state of equilibrium from an initial random configuration. Second we test how quickly the methods can produce de-correlated results once the system is in equilibrium.

In our initial quenching phase we compare three different update algorithms: Metropolis, Wolff and our Hybrid. We present performance data on our best performing implementation on the GPU of each algorithm. First we look at how the algorithms reach equilibrium from initial random starting conditions, we show how the algorithms compare in terms of energy over time. The results of this experiments are shown for 2D and 3D in Figure 5.

We can see from the two plots that the Hybrid algorithm does not provide any benefit in helping the system reach equilibrium from an initial random state. Because the purely Metropolis algorithm can make use of the packed-bit implementation, it can perform more algorithm steps in one second than the hybrid method can.

Secondly we examine how the algorithms decorrelate a system already at equilibrium. The algorithm com-

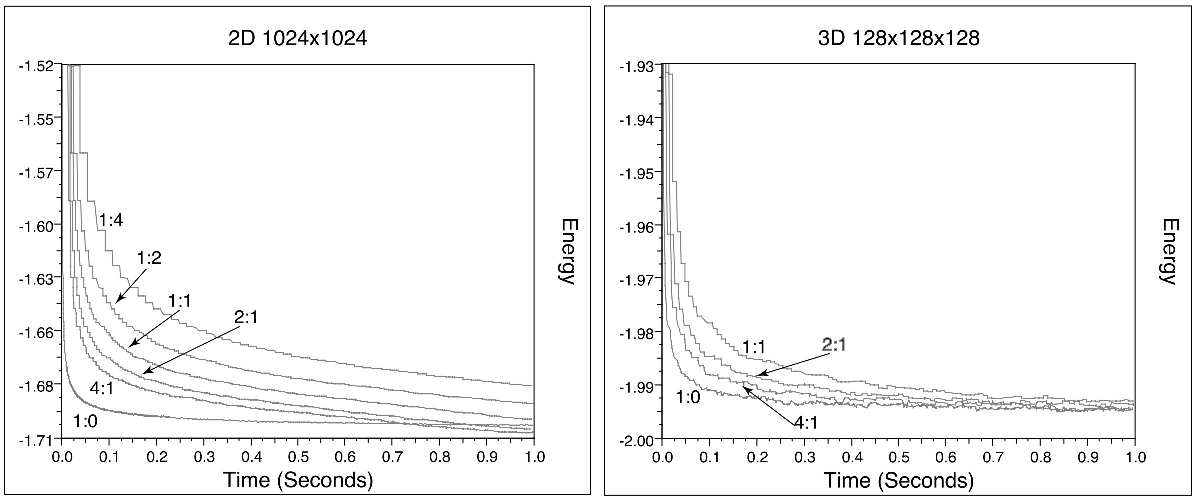

Figure 5: A comparison of update methods showing the initial phase of the Ising model as it reaches equilibrium in 2D (left) and 3D (right) with several ratios of Metropolis and Wolff (M:W). System sizes are $N = 1024^2$ in 2D and $N = 128^3$ in 3D. The Hybrid algorithms 1:2 and 1:4 behave almost identically to 1:1 in 3D and have been removed to avoid cluttering the plot.

parison results are shown as correlation (in log_10 scale) vs time. These are shown in Figure 6.

In this experiment we see how the Hybrid method can provide an improvement over both the Metropolis and Wolff algorithm. For the sake of clarity we have only shown the optimal Hybrid ratio on this plot. We found that in two-dimensions, a Hybrid ratio of 1:1 decorrelated the system fastest while in 3D, 4:1 was the optimal ratio. In both cases the optimal Hybrid method provides an important performance improvement over both the Metropolis and the Wolff update algorithms. These algorithms are further discussed in Section 8.

7 Small-World Rewired Lattice Simulation

In this section we describe an implementation of the Wolff cluster update algorithm for the Ising model on an irregular graph. The system is constructed starting with a regular lattice, but then a number of edges are rewired randomly to create the shortcuts common to small-world networks.

Either end of an edge, which connects two cells, is independently considered for rewiring with probability $\frac{1}{2}P$, thus rewiring a fraction P of all edges. In order to make this algorithm compatible with the Metropolis Ising implementation for arbitrary graphs described in [10], the same restriction to selecting a new neighbour applies. That is, the neighbour is selected randomly from all cells of the same colour as the previous neighbour, where each cell is assigned one of two colours using a checkerboard pattern. This means that no cells of the same colour are directly connected.

7.1 Rewired Irregular Data Structure

For 2D and 3D Ising systems, the cell spin states are stored in an `unsigned char` array S of the same dimensionality. In the 2D case, the spin array is bound to a texture reference, which allows the CUDA kernels to utilise the on-chip texture cache when reading the spin values of neighbouring cells. Texture fetches are optimised for spatial locality, which makes them very suitable for the task. This is not done in the 3D case, because CUDA does not support binding 3D textures to linear memory. 3D textures must be bound to read-only CUDA arrays (the new surface references introduced in CUDA 3.1 can currently only be used to write to 1D and 2D CUDA arrays), which would make it necessary to copy the spin values to the CUDA array every time the spins are modified, which is slower than simply accessing the 3D spin array directly in global memory. This is much less of a concern with Fermi-based graphics cards, as global memory accesses are cached in the on-chip L1/L2 caches.

Cells that are affected by rewiring are considered as modified. Every time an edge is rewired, 3 cells are affected: the source cell replaces a neighbour, the original neighbour loses a connection and the new neigh-

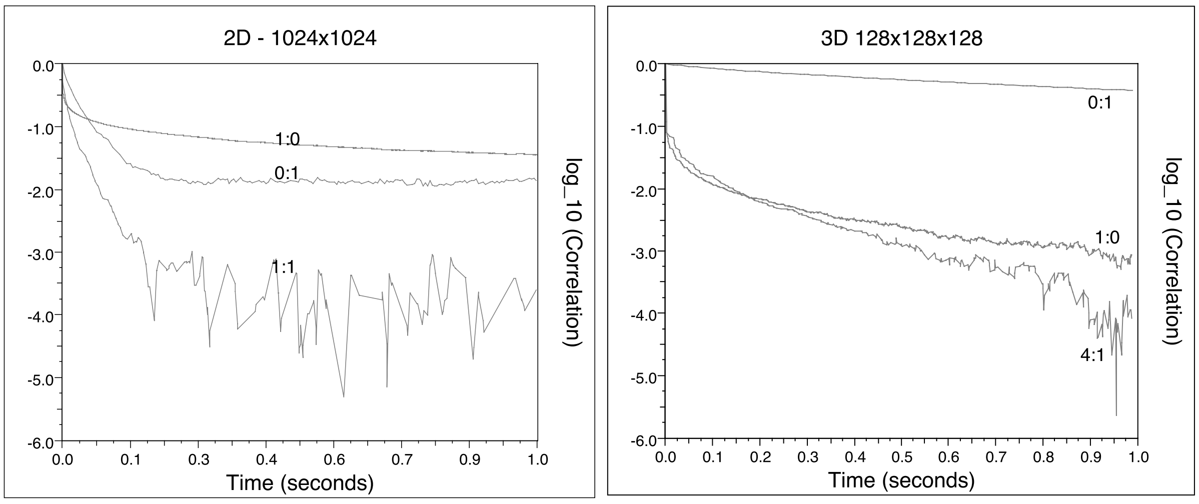

Figure 6: A comparison of the different update methods showing how the system correlation (log_10) reduces with time in 2D (left) and 3D (right). Only the Metropolis (1:0), Wolff (0:1) and even ratio Hybrid (1:1) algorithms are shown.

bour gains one. Unmodified cells, i.e. cells not affected by rewiring, do not need to store their neighbour structure, as the neighbour coordinates can simply be calculated. Modified cells, on the other hand, do not have a fixed number of neighbours and may be connected to any cell in the graph. Therefore, the neighbour information for these cells needs to be stored explicitly. Two arrays are used to do this. Vertex array V stores the length of the cell's adjacency-list, and arc array A stores the actual neighbour IDs. A thread with thread ID tid processes the cell whose adjacency-list length is stored at index tid in V. Respectively, the first neighbour ID is stored at index tid in array A. The second neighbour ID is stored at position $tid + |V_m| +$ padding and so on, where $|V_m|$ is the number of modified cells and padding increases $|V_m|$ to the next multiple of 32 (i.e. 128 bytes for 4 byte integers). Storing the neighbours in this order means that memory transactions from the vertex and arc arrays can be fully coalesced, as sequential threads can access sequential elements.

The ID of the cell processed by a thread with a particular thread ID (tid) is stored in arrays M_u and M_m for unmodified and modified cells respectively. This is necessary because different CUDA kernels are used to process the two types of cells to avoid conditionals that would cause warp divergence and thus hurt performance. The following section describes the kernel implementations in detail.

7.2 CUDA Implementation Of The Rewired Lattice Wolff Algorithm

The implementation of the Wolff cluster update algorithm for rewired Ising models uses a number of CUDA kernels to perform the updates in parallel. Algorithm 1 describes the tasks performed by the host system that are necessary to advance the simulation. Some of the kernels are executed by T threads, where T depends on the graphics hardware used. It should be a multiple of 512, the maximum thread block size used for these kernels, as well as a multiple of the physical number of scalar processors on the device, and finally it must be large enough to allow the device to switch between threads to hide memory latencies. We use $30,720$ for the GeForce GTX480 and GTX295 and $27,648$ for the GTX260-216.

The implementation uses 2D or 3D thread blocks depending on the number of system dimensions. The thread block dimension lengths used depend on the graphics hardware. Fermi based devices work well with blocks of size 32×16 and $32 \times 4 \times 4$ respectively, whereas 16×16 and $16 \times 4 \times 4$ work best on GT200 series devices.

The total number of cells $|V|$ is the sum of the number of unmodified cells $|V_u|$ and the number of modified cells $|V_m|$. It is equal to the system size N. Note that $<<$ and $>>$ are used to denote left and right binary shift operations respectively. The base 2 logarithm of the dimension length L used by the CUDA kernels is calculated once by the host and passed as parameter to the kernels.

Algorithm 1 Evolve the model by $STEPS$ simulation steps. This is the host code that manages the graphics processing unit.

```
allocate device memory for arrays $R, V, A, S, M_u$,
        $M_m, M2, X, F, P$
copy initial spin values to the device
copy RNG seeds to the device
do in parallel on the device using $T$ threads:
        initialise the RNGs
for $i \leftarrow 1$ to $STEPS$ do
  $k \leftarrow$ pick a cell randomly
  $s_k \leftarrow$ copy $k$'s spin to the host and flip it
  do in parallel on the device using $T$ threads:
          generate $|V_m|$ packed random
          results and store them in $R$
  do in parallel on the device using $T$ threads:
          call Kernel 1 for unmodified cells
  do in parallel on the device using $|V_m|$ threads:
          call Kernel 1 for modified cells
  $F \leftarrow 0$ //initialise frontier array $F$ to 0
  $F[k] \leftarrow 1$ //set cell $k$ as active in the frontier
  $P \leftarrow 0$ //initialise processed array $P$ to 0
  repeat
    $c \leftarrow 0$ //flag indicating changes
    if local Kernel 2 is enabled then
      do in parallel on the device using $|V|$
              threads: call local Kernel 2
    end if
    do in parallel on the device using $|V_u|$ threads:
            call Kernel 2 for unmodified
    cells
    do in parallel on the device using $|V_m|$
    threads:
            call Kernel 2 for modified cells
  until $c = 0$
end for
copy final spins back to the host
```

The random numbers generated on the device use Marsaglia's lagged-Fibonacci random number generator (RNG) [50] implementation for CUDA as described in [51]. Every CUDA thread uses its own lag-table to produce an independent stream of random numbers, thus avoiding read-write race conditions. However, for the implementation to perform well, it is necessary that all threads in a half-warp collectively generate new random deviates, even if some of them are not actually used. This enables global memory reads and writes to be coalesced.

When a Wolff cluster is formed around a source cell k, neighbouring cells with the same spin value are only included in the cluster if the arc (i.e. the edge in one direction) used to reach them is active in this simulation step. This is determined by generating a uniform random number and comparing it to the probability pWolff. Kernel 1 for unmodified cells, described in Algorithm 2 generates the random deviates itself, whereas the random numbers for Kernel 1 for modified cells, see Algorithm 3, are generated beforehand by a dedicated kernel. The reason for this distinction is that one thread is used for every modified cell, while the kernel for unmodified cells is executed by T threads. Every RNG instance requires 400 bytes of device memory, which can potentially add up to a lot of memory if the rewiring probability P is high and a sep-

Algorithm 2 This is CUDA Kernel 1 for unmodified cells in 2-dimensions. It is executed by T threads, each of which processes a number of cells until all $|V_u|$ unmodified cells have been processed. S is the spin-array, M the thread ID to cell ID mapping array, X the traversable array, L the dimension length and D the number of dimensions.

```
$t \leftarrow$ global thread ID queried from the runtime
load variables for the RNG instance used by this
thread from global memory to registers
for all $tid \in V_t := \{t, t+T, t+2T, \ldots; t < |V_u|\}$ do
  //compare random numbers to probability
  pWolff and store the result as a bit-value in $r$
  $r \leftarrow 0$
  for $i \leftarrow 1$ to $D$ do
    $r_{tmp} \leftarrow$ generate uniform random number
    $r \leftarrow (r << 1)$ OR $(r_{tmp} <$ pWolff $)$
  end for
  $id \leftarrow M[tid]$ //load the cell ID
  $ix \leftarrow id$ AND $(L-1)$ //the x-coordinate
  $iy \leftarrow id >> log2(L)$ //the y-coordinate
  $s \leftarrow S(ix, iy)$ //load the spin value
  //calculate the neighbour coordinates and mark
  traversable edges with bit-value 1 in $x$
  $x \leftarrow 0$ //bits marking active edges
  $n\_ix \leftarrow ix = 0\ ?\ L-1 : ix-1$
  $n\_s \leftarrow S(n\_ix, iy)$ //spin value for left neighbour
  $x \leftarrow (s = n\_s)$ and $(r$ AND $(1 << 0))$ ?
      $(x$ OR $(1 << 0)) : x$
  $n\_ix \leftarrow ix = (L-1)\ ?\ 0 : ix+1$
  $n\_s \leftarrow S(n\_ix, iy)$ //spin value for right neigh-
  bour
  $x \leftarrow (s = n\_s)$ and $(r$ AND $(1 << 1))$ ?
      $(x$ OR $(1 << 1)) : x$
  and so on for all other dimensions ...
  $X[tid] \leftarrow x$ //write to global memory
end for
write the updated local variables for the RNG in-
stance back to global memory
```

arate RNG instance was used for each thread. Therefore, a dedicated RNG kernel instance is used to generate the random numbers for the modified cells using T threads. This kernel not only generates the deviates, but also performs the comparison to the value pWolff and only stores a bit value for every arc, 1 indicating that it is active and 0 that it is inactive. These bits are packed into an `unsigned int` value per cell, thus allowing a maximum degree of 32. The packed results are stored in array R.

The task of Kernel 1 is to determine which connections are both active and connect cells with the same spin value. Arcs that pass this test are called traversable and are marked in array X using the bit value 1, arcs that fail the test are marked with value 0. Separate traversable arrays of length $|V_u|$ and $|V_m|$ are used for Kernel 1 for unmodified and modified cells respectively to avoid uncoalesced memory accesses.

Algorithm 3 This is CUDA Kernel 1 for modified cells in 2-dimensions. Every CUDA thread processes one cell, thus $|V_m|$ threads are executed to process all modified cells. S is the spin-array, V the vertex-array, A the arc-array, M the thread ID to cell ID mapping array, R the random results array, X the traversable array and L the dimension length.

```
$tid \leftarrow$ global thread ID queried from the runtime
$id \leftarrow M[tid]$ //load the cell ID
$ix \leftarrow id$ AND $(L-1)$ //the x-coordinate
$iy \leftarrow id >> log2(L)$ //the y-coordinate
$s \leftarrow S(ix, iy)$ //load the spin value
$r \leftarrow R[tid]$ //load the random results for this cell
//look-up the neighbours and mark traversable edges with bit-value 1 in $x$
$r_{shift} \leftarrow 0$ //shift $r$ by $r_{shift}$ bits
$x \leftarrow 0$ //bits marking active edges
$v \leftarrow V[tid]$ //load the adjacency-list length
for all $n\_id \in \{A[tid], A[tid+|V_m|], \ldots, A[tid+(v-1)|V_m|]\}$ do
    $n\_ix \leftarrow n\_id$ AND $(L-1)$ //the x-coordinate
    $n\_iy \leftarrow n\_id >> log2(L)$ //the y-coordinate
    $n\_s \leftarrow S(n\_ix, n\_iy)$ //load the neighbour's spin
    $x \leftarrow (s = n\_s)$ and $(r$ AND $(1 << r_{shift}))$ ?
          $(x$ OR $(1 << r_{shift})) : x$
    $r_{shift} \leftarrow r_{shift} + 1$ //increment the shift value
end for
$X[tid] \leftarrow x$ //write to global memory
```

Kernel 2 uses the traversable array to spread the new spin value in a breadth-first search manner. Starting from the source cell, it advances a frontier F across eligible connections to other cells, marking visited cells in P, until no more traversable arcs can be reached. At this point, all cells in the current Wolff cluster have been updated with the new spin value.

Kernel 2 for unmodified cells is described in Algorithm 4. The version for modified cells is essentially the same as the one for unmodified cells, only that it loads the neighbour information from global memory as described in Algorithm 2.

Algorithm 4 This is CUDA Kernel 2 for unmodified cells in 2-dimensions. Every CUDA thread processes one cell, thus $|V_u|$ threads are executed to process all unmodified cells. S is the spin array, M the thread ID to cell ID mapping array, X the traversable array, F the frontier array, P the processed array, L the dimension length, c a flag indicating changes in the cluster size and s the new spin value for all cells in the cluster.

```
$tid \leftarrow$ global thread ID queried from the runtime
$id \leftarrow M[tid]$ //load the cell ID
if $F[id]$ and (NOT $P[id]$) then
    //this cell is part of the Wolff cluster
    $P[id] \leftarrow 1$ //mark as processed
    $c \leftarrow 1$ //set the flag to indicate the changes
    $ix \leftarrow id$ AND $(L-1)$ //the x-coordinate
    $iy \leftarrow id >> log2(L)$ //the y-coordinate
    $S(ix, iy) \leftarrow s$ //store the new spin value
    $x \leftarrow X[tid]$ //load the traversable flags
    //set nbrs as active in $F$ if the edge is traversable
    $n\_ix \leftarrow ix = 0$ ? $L-1 : ix-1$
    if $x$ AND $(1 << 0)$ then
        $F[iy * L + n\_ix] \leftarrow 1$ //frontier for left nbr
    end if
    $n\_ix \leftarrow ix = (L-1)$ ? $0 : ix+1$
    if $x$ AND $(1 << 1)$ then
        $F[iy * L + n\_ix] \leftarrow 1$ //frontier for right nbr
    end if
    and so on for all other dimensions ...
end if
```

Since this implementation of Kernel 2 only advances the frontier by one step from the source cell during every kernel call, it does not perform very well once the Wolff clusters increase in size. To speed-up the propagation of the new spin value throughout the cluster, calls to a `local` version of Kernel 2 can be interleaved with the previously described `global` versions. In this `local` version of the kernel, the threads of a thread block use the fast shared memory of the streaming multi-processors to quickly propagate the spin to all cells processed by threads of the same block that are also part of the Wolff cluster. Algorithm 5 describes this process. As the kernel is executed by $|V|$ threads, the traversable array can not be accessed by thread ID as it is done in the `global` implementation. An additional mapping array $M2$ is used to map the thread ID to the correct index into the traversable array X.

The `global` versions for unmodified and modified cells need to be called after one of these optional calls to the `local` kernel in order to advance the frontier across thread block boundaries and rewired edges. The performance analysis in Section 7.3 shows when it is sensible to start using the `local` kernel in addition to the `global` updates.

Algorithm 5 The shared memory propagation of the frontier performed by the `local` version of Kernel 2. The parameters and the local variables id, ix and iy are the same as in Algorithm 4. $M2$ is the thread ID to traversable index mapping and B the block length. $|V|$ CUDA threads are used.

```
$btid \leftarrow$ block thread ID queried from the runtime
$F_s[B * B]$ //frontier array in shared mem.
$X_s[B * B]$ //traversable array in shared mem.
$F_s[bid] \leftarrow F[id]$ //load the frontier into shared mem.
$X_s[btid] \leftarrow 0$
**if** the cell has no rewired edges **then**
    $idx \leftarrow M2[tid]$ //look-up traversable array idx
    $X_s[btid] \leftarrow X[idx]$ //load the traversable flags
**end if**
**repeat**
    $f_{old} \leftarrow F_s[btid]$ //remember the current value
    **if** $btid = 0$ **then**
        $vote_s \leftarrow 0$ //shared mem. variable for votes
    **end if**
    synchronize threads in block
    //calc. the shared mem. index $n_{idx}$ for the neighbours and set the cell as active in the frontier if the neighbour is active and the edge is traversable
    $n_{idx} \leftarrow$ threadIdx.y $* B +$ threadIdx.x $+ 1$ //right
    $F_s[btid] \leftarrow$ (threadIdx.x $< (B - 1)$) and
                ($X_s[n_{idx}]$ AND $(1 << 0)$) and
                $F_s[n_{idx}]$ ? $1 : F_s[btid]$
    $n_{idx} \leftarrow$ threadIdx.y $* B +$ threadIdx.x $- 1$ //left
    $F_s[btid] \leftarrow$ (threadIdx.x $> 0$) and
                ($X_s[n_{idx}]$ AND $(1 << 1)$) and
                $F_s[n_{idx}]$ ? $1 : F_s[btid]$
    and so on for all other dimensions ...
    **if** $X_s[btid] \neq 0$ and $f_{old} \neq F_s[btid]$ **then**
        $vote_s \leftarrow 1$ //the frontier changed
    **end if**
    synchronize threads in block
**until** $vote_s = 0$
**if** $F_s[btid]$ and (NOT $P[id]$) **then**
    $F[id] \leftarrow 1$
    //process unmodified cells as described in Alg. 4
**end if**
```

A small modification to Kernel 2 can substitute the conditional branches that decide if a neighbouring cell is set as active in the frontier array F by evaluating the traversable value stored in x with atomic operations of the form:

atomicOr($F[\ldots], ((x >> \{0, 1, \ldots, D - 1\})$ AND 1))

This gives a comparable performance on the Fermi based GTX480, but is considerably slower on the older GT200 series cards, clearly highlighting the increased performance of atomic operations on Fermi devices. But because it does not provide a performance gain either, we use the implementation based on conditionals that performs well on both hardware generations.

We have also tested an implementation of Kernel 2 that is based on labelling all components in the graph using a variation of the component labelling algorithm for arbitrary graphs described in [48], which works for directed graphs based on the traversable array. This implementation requires an additional CUDA kernel, which flips the spins of the cluster that contains the source cell after all components have been detected. We expected this implementation to perform better than the breadth-first search based implementation when the Wolff clusters are relatively large on average, as it takes a few iterations for the current implementation to extend the frontier far enough to really utilise the parallel computing capabilities of the hardware. However, it turns out that this is not the case, the implementation described in this paper performs better in all cases.

7.3 Rewired Lattice Performance

This section analyses the performance of the Wolff algorithm and compares it to the Metropolis implementation described in [10].

The first plot in Figure 7 compares the performance of the Metropolis and Wolff CUDA implementations for rewired lattice Ising simulations with $P = 10^{-2}$ in terms of wall clock time. The abrupt increase in execution time per simulation step for the Wolff algorithm marks the time when the typical Wolff cluster size drastically increases due to the system reaching an equilibrium value. This state change can also be observed in the sudden drop of the system energy illustrated by the plot on the right. When the temperature is above the critical temperature T_c, as it is the case with $T = 2.723$ in the example, then the system remains chaotic and the average Wolff cluster size stays small. The performance of the Metropolis algorithm

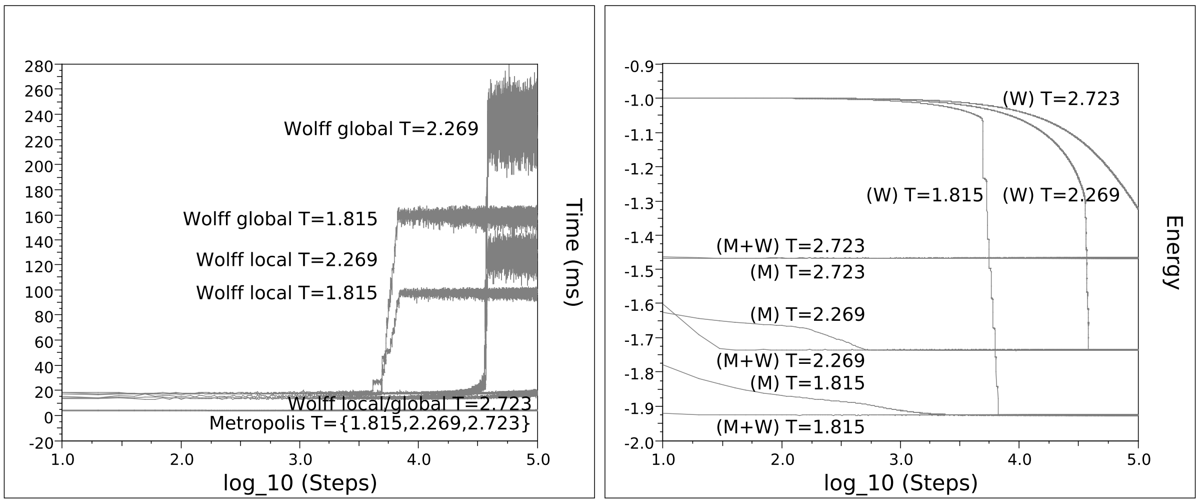

Figure 7: The figure on the left compares the performance of the Metropolis algorithm with the two Wolff cluster detection flavours, the first using only the `global` version of Kernel 2 and the second also using the `local` implementation. The system size $N = 1024^2$, the rewiring probability $P = 10^{-2}$ and the temperature $T = \{1.815, 2.269, 2.723\}$ (i.e. the critical temperature of the regular 2D Ising model and $\pm 20\%$ around it). The time is measured in intervals of 10 simulation steps on an NVIDIA GTX480 and represents the time in milliseconds since the previous measurement. The figure on the right illustrates the changes in system energy over the same period, again with $P = 10^{-2}$, when using only **W**olff, only **M**etropolis or both interleaved at a 1:1 ratio.

does not significantly depend on the system state.

The results show that local updates in addition to global updates in the implementation of the Wolff algorithm, as described in the previous section, perform significantly better than just global updates after the system reaches an equilibrium value. The local updates slow the algorithm down slightly before the system equilibrates or if T is significantly larger than T_c, because the Wolff clusters tend to be very small and therefore the overhead introduced by the local updates dominates their benefit. However, as the second plot of Figure 7 as well as Figure 8 show, the system can be equilibrated very quickly if the "right" combination of update algorithms is used. Almost all the simulation time is therefore spent after the system has reached an equilibrium value, which is why we always use the local updates in the following measurements.

As already mentioned, the plot on the right of Figure 7 illustrates how the system energy changes throughout the duration of the simulation. The Wolff algorithm by itself takes many more simulation steps than the Metropolis algorithm to reach an equilibrium value. But using the two algorithms together equilibrates the system in even less simulation steps than Metropolis by itself. We have tested ratios of up to 10:1 and vice versa and the system equilibrates in less simulation steps than just Metropolis in all cases. The differences between all tested ratios that use both algorithms are within the margin of error, with ratios that are very imbalanced tending to take more steps than those that are more balanced.

What is usually more important than the number of simulation steps is the wall clock time that it takes to equilibrate the system. And as the results have shown, the Metropolis algorithm parallelises more nicely on the GPU architecture and therefore achieves a much higher performance. This poses the question if the higher performance of the Metropolis algorithm makes up for the increased number of simulation steps needed to equilibrate the system or if a combination of Metropolis and Wolff algorithm performs better. Figure 8 shows the measurement results for a number of algorithmic ratios for a particular system configuration in both 2D and 3D running on a GeForce GTX480. The measurements show that the Metropolis algorithm by itself or a combination with the Wolff algorithm that heavily favours Metropolis (e.g. an 8:1 ratio) equilibrates the system more quickly than ratios that are more in favour of Wolff. Wolff by itself takes so long to equilibrate the system that it is left out in the plots and should never be used for this task.

If we want to compute a metric like the heat capacity, susceptibility, or Binder cumulant, then it is important to take decorrelated samples from the simulation after it has been equilibrated. But being overly cautious means wasting precious time. We therefore

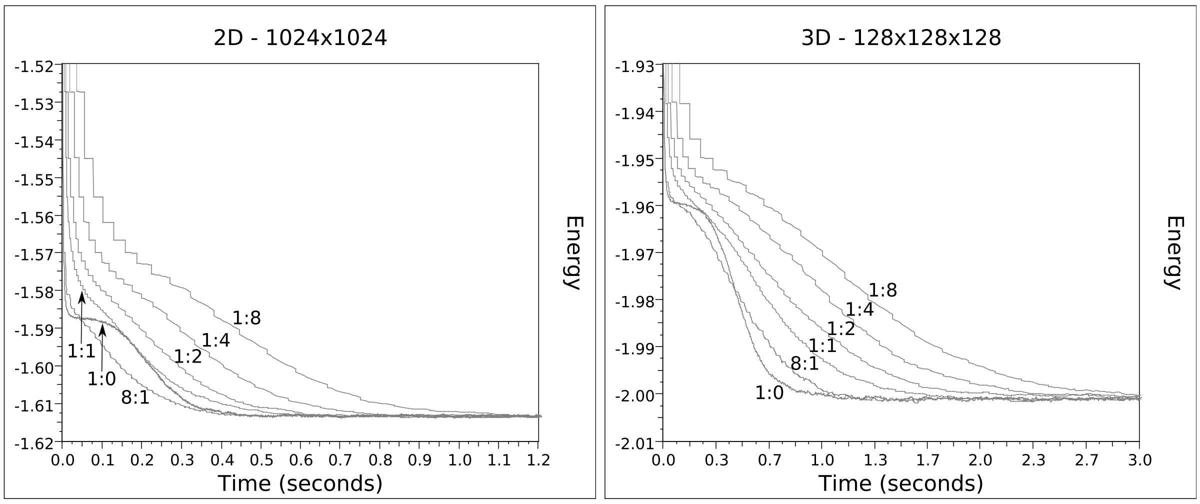

Figure 8: The plots show the execution time that it takes until the system reaches an equilibrium value on an NVIDIA GTX480 for 2D (left) and 3D (right) systems with different ratios of Metropolis and Wolff (M:W). The system size is $N = 1024^2$ and $N = 128^3$ respectively. $P = 10^{-2}$ for both systems; $T = 2.4$ for the 2D simulations and $T = 4.55$ for the 3D simulations (both are as close to $T_c(P)$ as we have determined it using the heat capacity (see Figure 13)). Each data point is the mean value of 100 simulation runs.

investigate the time that it takes to decorrelate the system with different ratios of Metropolis and Wolff updates. The results are given in Figure 9. The results for some of the tested ratios are not shown in the plots to make them easier to read. These results lie in between the other ratios as expected.

Every data point is the mean value of 5,000 measurements from 50 independent simulation runs. Each simulation run is initialised with a different seed value for the random number generator and therefore uses a different rewired lattice structure. The standard deviations are not displayed to keep the plots readable, as the logarithmic scale causes the error bars to appear inflated. However, on a non-logarithmic scale, the error is insignificant for the pure Metropolis results. All other ratios have a larger standard deviation due to the Wolff algorithm (up to ≈ 0.29 in 2D and ≈ 0.24 in 3D for pure Wolff), which gradually shrinks until the system is decorrelated at which point the error again becomes insignificant. The results should therefore be taken as trends, no one ratio gives the best results in every single measurement.

The Metropolis algorithm again shows that its superior execution speed makes up for the increased number of steps that it requires to decorrelate 2D systems (see Figure 14). The more Wolff steps are added to the ratio, the longer it takes to decorrelate the system, even though every individual Wolff step decorrelates it more than a single Metropolis update step. Surprisingly, this is not always true for the 3D case, especially when more Wolff than Metropolis steps are performed. Here the Metropolis step decorrelates the system more than the Wolff step, which is particularly visible in the 1:8 or 1:4 cases. This may indicate that the chosen temperature $T = 4.55$ is higher than the critical temperature for the given configuration and the Wolff clusters are therefore rather small.

Figure 10 gives a different view on the decorrelation issue. It illustrates how decorrelated a 2D system is after a set of simulation steps for a particular temperature T, where the number of algorithmic steps depends on the ratio. The respective timing results for each set of steps are given in the second plot. If the temperature is much smaller than $T_c(P)$, i.e. $T = 1.815$ in the example, then all ratios decorrelate the system quickly and the fastest algorithm should be used, which is Metropolis by itself. When the temperature gets close to or is larger than $T_c(P)$, then one of the sets with a higher number of total steps is needed to get good results. Which one is a trade-off between the level of decorrelation and the time needed to execute the set of simulation steps. For example, the 20:1 ratio may be a good choice for $T = 2.723$, while 20:10 may be better for $T = 2.269$.

8 Discussion

As discussed above the Wolff algorithm assembles clusters of spins to flip. These clusters are assem-

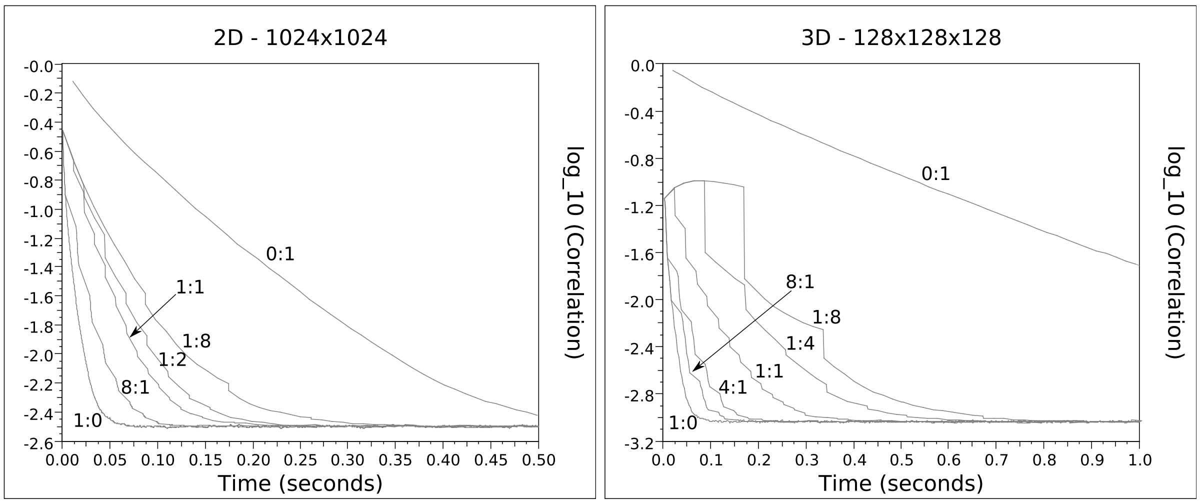

Figure 9: Decorrelation over half a second of execution time for the 2D system (left) and over one second for the 3D system (right) on an NVIDIA GTX480 for different ratios of Metropolis and Wolff (M:W). The system size $N = 1024^2$ and $N = 128^3$ respectively. $P = 10^{-2}$ for both systems; $T = 2.4$ for the 2D simulations and $T = 4.55$ for the 3D simulations (i.e. $\approx T_c(P)$). Every data point is the mean value of 5,000 measurements.

bled with the correct thermal probability and so the distribution of Wolff cluster sizes is not directly representative of the distribution of physical clusters. Nevertheless, we can use the Wolff cluster size distribution as an indication of what is going on inside the system.

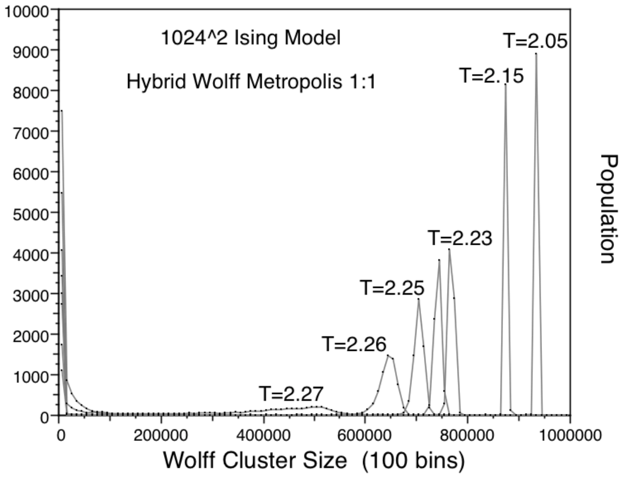

Figure 11: A typical distribution of Wolff cluster sizes from an equilibrated 2D Ising system.

Figure 11 illustrates what the Wolff cluster update algorithm is doing in practise for a range of temperatures. At low temperatures very large clusters of nearly all N spins dominate the distribution and a pure Wolff algorithm essentially "thrashes" the spin configuration back and forth. At very high temperatures the Wolff algorithm only forms very small clusters and is therefore not significantly different from a local algorithm like Metropolis. The distribution is seen to be most interesting near criticality where it broadens out. This reflects the importance of all length scales contained in a critical system model such as the Ising system at its critical temperature.

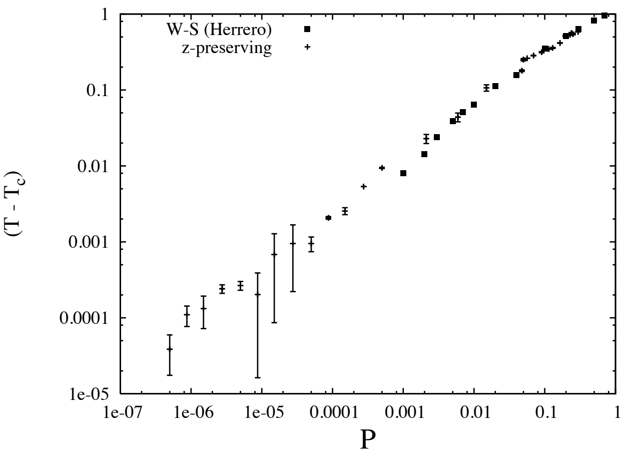

Figure 12: Shift in critical temperature with rewiring probability p, after [52].

Figure 12 shows some preliminary results on the effect of the small-world rewiring. The data is based on earlier work by Herrero [23] for high p values and Hawick and James [52] across the full p-range shown. At very low p the shift does not seem to flatten out or tail off as Herrero conjectured. We believe any flattening out of the critical temperature shift is likely to be just an

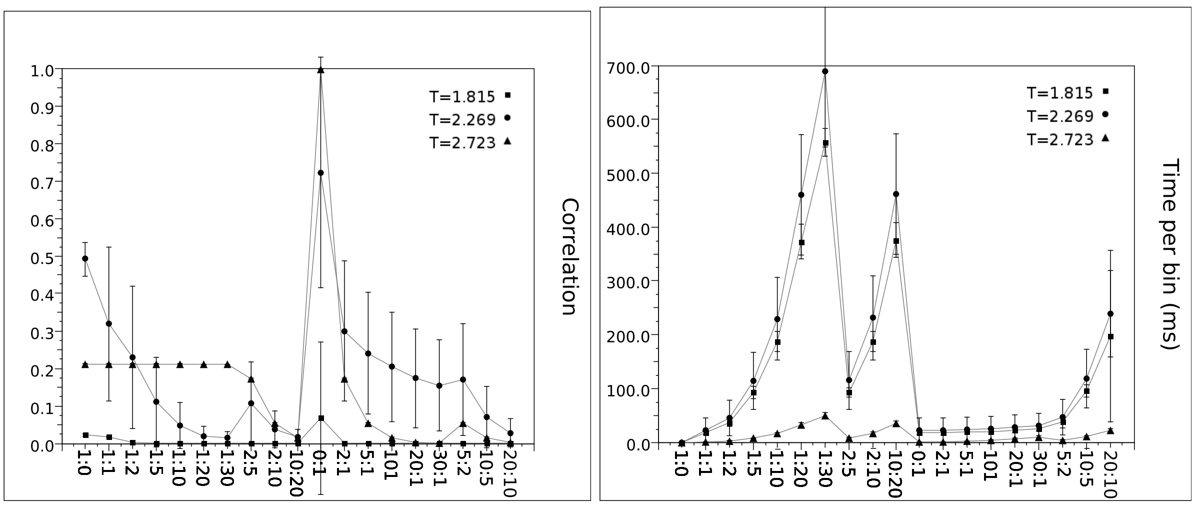

Figure 10: The figure on the left shows the correlation value for different ratios of Metropolis and Wolff (M:W) with 0.0 being decorrelated and 1.0 being (anti-)correlated for a 2D system with $N = 1024^2$. The correlation value is calculated after x simulation steps, where x is the sum of Metropolis and Wolff steps in the given ratio (e.g. $x = 2$ for a 1:1 ratio and $x = 3$ for a 1:2 or 2:1 ratio). The plot on the right shows the respective timing results for x steps on an NVIDIA GTX480.

artifact of size-limited simulation systems which cannot sustain effective small p values. As the error bars in the data indicate there is still considerable uncertainty of the true behaviour of the temperature shift with p although we do believe it is likely to be a power law of the form $(T_c(p) - T_c^{p=0}) \sim p^x$ – as suggested by the likely straight-line trend in this plot of log-log data.

Our present paper is strongly motivated by the need to investigate larger system sizes for more independent runs to improve the statistics of a plot like figure 12 but also to have sufficiently good data to also investigate the critical exponents and thus to investigate if the rewiring changes the nature of the Ising phase transition as well as just shifting it.

Section 4.1 mentions that the heat capacity C_v and the susceptibility χ can be used as a visual indicator of the location and shift in the critical temperature. Figure 13 illustrates both metrics for systems of size $N = 1024^2$ and $N = 128^3$. The sharpness of the phase transition is limited by how large a system size is simulated, but the plots do illustrate nicely the shift in T_c when re-wiring is applied, both from the unperturbed value in 2-D of $2/\log(1+\sqrt{2}) \approx 2.2692$ and the 3-D experimental value of $1/0.221652 \approx 4.5116$ [3]. As rewiring is applied and p increases, the critical temperature shifts upwards. This can be interpreted as the long-range links making it easier for the system to maintain long-range order even at higher temperatures, and the critical "disappearing point" of the long-range behaviour is shifted upwards.

We have already analysed how much time it takes to get statistically independent samples of the Ising system using various combinations of Metropolis and Wolff updates for both regular and rewired lattice Ising systems in the respective performance sections. Those results depend on the implementations and possibly on the hardware used for the experiments as well. A better implementation of either one of the algorithms, or the capabilities of future CUDA hardware, may change the outcomes completely. It may therefore be even more relevant to compare the algorithms both individually and in combination in terms of their ability to decorrelate the Ising system in a certain number of simulation steps. See Figure 14.

From the first two graphs we can see how the update algorithms decorrelate the regular lattice Ising system over 100 simulation steps. The Hybrid ratio 1:1 provides the best results and decorrelates the system faster than the Wolff and the Metropolis algorithms in both 2D and 3D. It should be noted that as the Hybrid algorithms are approaching a correlation of 0.0, minor variations appear amplified on the log_10 scale.

The bottom row of Figure 14, on the other hand, shows how many simulation steps it takes for an irregular lattice Ising system with $P = 10^{-2}$ to decorrelate. We have tested ratios from 1:0 to 1:8 and vice versa. Every data point is the mean value of 5,000

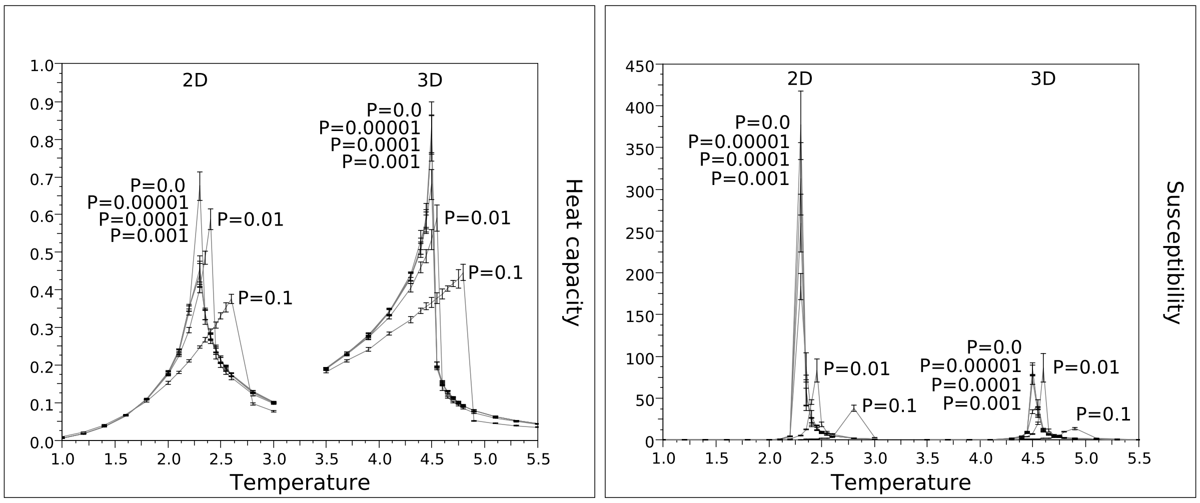

Figure 13: The figures illustrate how the critical temperature shifts with the rewiring probability P using the heat capacity (left) and the susceptibility (right). The system size is $N = 1024^2$ and $N = 128^3$ for the 2D and 3D simulations respectively. The results show the mean and standard deviation for 10 simulation runs.

measurements from 50 independent simulation runs. Each simulation run is initialised with a different seed value for the random number generator and therefore uses a different rewired lattice structure. The standard deviations are not displayed for the reasons already given in Section 7.3, but behave like those of Figure 9.

Ratios 1:2, 1:4 and 1:8 are not shown in the results for the irregular 2D system, because they are almost identical to the 1:1 ratio. The remaining data clearly shows that a combination of both algorithms performs better than either one by itself in this scenario. Ratios 2:1, 4:1 and 8:1 show an interesting step function whenever a Wolff update is performed, which decorrelates the system more drastically than a single Metropolis step. The results for the irregular 3D system look quite different from those of the 2D system. The data sets for ratios 1:1, 1:2, 2:1, 4:1 and 8:1 are very close to the values for 1:0 and therefore are not shown. While they all perform the task of decorrelating the system in approximately the same number of simulation steps, ratios that significantly favour Wolff do not perform so well.

The overall conclusions for both regular and irregular lattice systems are that the 1:1 ratio of Metropolis to Wolff updates quickly decorrelates the system in terms of simulation steps for both 2D and 3D simulations. Other ratios perform good in one situation but not in the other. Wolff by itself is not a good choice in either case.

9 Conclusions

We have reported on the computational performance of various Monte Carlo update algorithms for simulations of the regular and re-wired lattice Ising model using GPUs. We have experimented with various data structures and different GPU device memory layouts (bit-packing, graph structure, cluster component labelling) to optimise both Metropolis and Wolff update algorithms. Over and above the raw performance optimisation and our GPU/CPU speed-ups of over 100, and giving a discussion of implementation approaches, we have also reported on some subtleties of the nature of the Ising phase transition on simulation performance.

We have employed some pragmatic metrics based on the wall-clock time we can obtain from our implementations. We have also considered **correlation times**, both in terms of wall clock time and number of simulation steps. There are also interesting algorithmic classification issues based on the dynamical critical exponent z [53]. This is notoriously difficult to calculate [54] but is based on an analysis of the **correlation length** ξ which diverges at the critical point. This is usually expressed in terms of the system relaxation time $\tau \sim \xi^z$ and near the critical temperature $\tau \sim |T - T_c|^{-z\nu}$ where ν is the correlation length exponent. We hope to be able to investigate the behaviour of z, τ, ξ in the case of $p > 0$.

We obtain the interesting result concerning the effect of non-zero p on the behaviour of the Metropo-

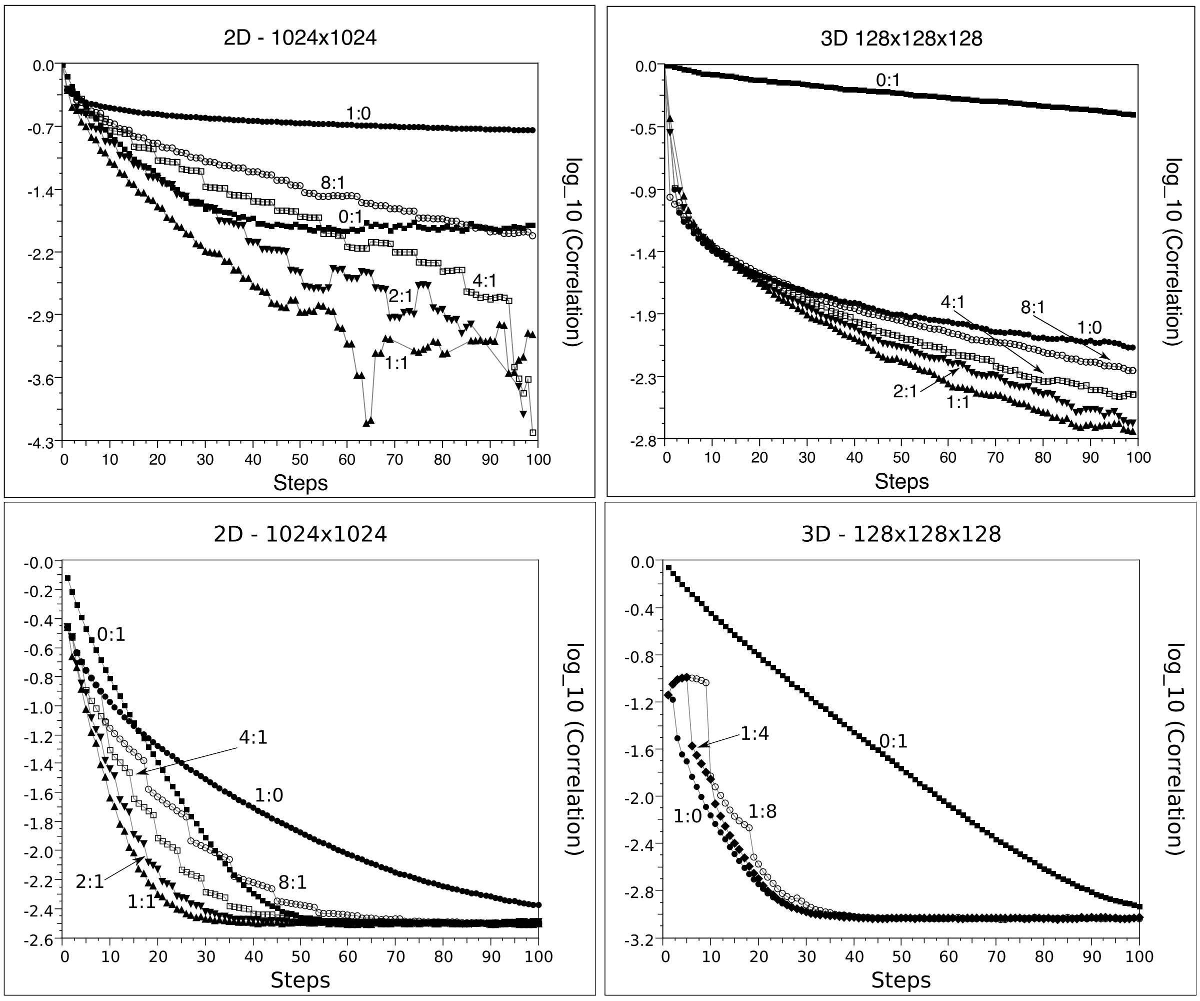

Figure 14: Correlation over number of simulation steps for 2D (left) and 3D (right) systems with different ratios of Metropolis and Wolff updates (M:W) for regular lattice Ising simulations (top row) and irregular lattice Ising simulations with $P = 10^{-2}$ (bottom row). The system sizes are $N = 1024^2$ and $N = 128^3$ respectively. The regular lattice simulations use $T = T_c$, while the irregular lattice simulations use $T = 2.4$ for the 2D simulations and $T = 4.55$ for the 3D simulations (i.e. $\approx T_c(P)$).

lis algorithm. The rewired links change the nature of the Ising system subtly but considerably. In the case of $p \equiv 0$ Metropolis is of necessity a wholly localised update and as we have shown and discussed it fares poorly against the Wolff-hybrid algorithms near criticality. However in the case of finite and significant non-zero p, Metropolis acts as a relatively non-local algorithm through the effect of the non-localised rewirings. This manifests itself in a rather more subtle comparison between Metropolis and Wolff. For very small p then the computational effort of Wolff is still well worth it compared to pure Metropolis or even hybrids. However for $p \gtrsim 0.01$ our implementations mean that simple Metropolis is comparable in efficiency.

One can interpret this in a number of ways. The small-world rewiring is shifting the critical temperature, but even if we still work at temperatures close to the shifted values we still find a change. It is an iterative procedure to refine $T_c(p)$ and this is certainly not known to the precision we know it for the case $p \equiv 0$. Even taking this into account however, it suggests that the nature of the different length scales present in a rewired system might be doing more than just shifting the critical temperature but might be changing the nature of the phase transitions as well.

Using the pragmatical and operational insights we have described in this paper, we therefore plan to investigate the nature of the rewiring-shifted transition as expressed by its critical and dynamical exponents. We believe the data-parallelism offered by individual GPUs and the throughput-parallelism from using GPU-clusters will make investigating this computationally feasible.

References

[1] Onsager, L.: Crystal Statistics I. Two-Dimensional Model with an Order-Disorder Transition. Phys.Rev. **65** (1944) 117–149

[2] Pawley, G.S., Swendsen, R.H., Wallace, D.J., Wilson, K.G.: Monte-Carlo renormalization group calculations of critical behaviour in the simple cubic Ising model. Phys. Rev. B **29** (1984) 4030–4040

[3] Baillie, C., Gupta, R., Hawick, K., Pawley, G.: Monte-Carlo Renormalisation Group Study of the Three-Dimensional Ising Model. Phys.Rev.B **45** (1992) 10438–10453

[4] Binney, J.J., Dowrick, N.J., Fisher, A.J., Newman, M.E.J.: The Theory of Critical Phenomena. Oxford University Press (1992)

[5] Zheng, B.: Monte carlo simulations and numerical solutions of short-time critical dynamics. Physica A: Statistical Mechanics and its Applications **283** (2000) 80 – 85

[6] Milgram, S.: The Small-World Problem. Psychology Today **1** (1967) 61–67

[7] Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature **393** (1998) 440

[8] Niss, M.: History of the Lenz-Ising Model 1920-1950: From Ferromagnetic to Cooperative Phenomena. Arch. Hist. Exact Sci. **59** (2005) 267–318

[9] Preis, T., Virnau, P., Paul, W., Schneider, J.J.: GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. Journal of Computational Physics **228** (2009) 4468 – 4477

[10] Hawick, K., Leist, A., Playne, D.: Regular Lattice and Small-World Spin Model Simulations using CUDA and GPUs. Technical Report CSTN-093, Computer Science, Massey University (2009) To appear in Int. J. Parallel Programming (2010).

[11] Block, B., Virnau, P., Preis, T.: Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D Ising model. Comp. Phys. Comms. **181** (2010) 1549–1556

[12] Dotsenko, V.S., Selke, W., Talapov, A.L.: Cluster monte carlo algorithms for random ising models. Physica A: Stat. and Theor. Phys. **170** (1991) 278–281

[13] Fukui, K., Todo, S.: Order-N Cluster Monte Carlo Method for Spin Systems with Long-Range Interactions. J. Comp. Phys. **228** (2009) 2629–2642

[14] Onsager, L.: Crystal Statistics I. Two-Dimensional Model with an Order-Disorder Transition. Phys.Rev. **65** (1944) 117–149

[15] Baxter, R.J.: Exactly Solved Models in Statistical Mechanics. Number ISBN 0-12-083180-5. Academic Press (1982)

[16] Anderson, P.W.: New approach to the theory of superexchange interactions. Phys. Rev. **115** (1959) 2–13

[17] Boyer, D., Miramontes, O.: Interface motion and pinning in small-world networks. Phys. Rev. E **67** (2003)

[18] Pękalski, A.: Ising model on a small world network. Phys. Rev. E **64** (2001)

[19] Jeong, D., Hong, H., Kim, B.J., Choi, M.Y.: Phase transition in the Ising model on a small-world network with distance-dependent interactions. Phy. Rev. E. **68** (2003)

[20] Kim, B.J., Hong, H., Holme, P., Jeon, G.S., Minnhagen, P., Choi, M.Y.: XY model in small-world networks. Phy. Rev. E. **64** (2001)

[21] Hong, H., Kim, B.J., Choi, M.Y.: Comment on "Ising model on a small world network". Phy. Rev. E. **66** (2002) 018101

[22] Yi, H., Choi, M.S.: Effect of quantum fluctuations in an Ising system on small-world networks. Phy. Rev. E **67** (2003)

[23] Herrero, C.P.: Ising model in small-world networks. Phys. Rev. E **65** (2002)

[24] Hawick, K.A.: Domain Growth in Alloys. Edinburgh University, Ph.D. Thesis (1991)

[25] MacIsaac, K., Jan, N.: On the dynamic exponent of the two-dimensional ising model. J. Phys. A. Maths. Gen (1992) 2139–2145

[26] Caracciolo, S., Edwards, R.G., Pelissetto, A., Sokal, A.D.: Dynamic critical behaviour of wolff's algorithm for rp^n σ-models. Nuclear Physics B (Proc Suppl) (2004) 1–3

[27] Sokal, A.D.: Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms. In: Cours de Troisieme Cycle de la Physique en Suisse Romande, Lausanne (1989)

[28] Blöte, H.W.J., Compagner, A., Croockewit, J.H., Fonk, Y.T.J.C., Heringa, J.R., Hoogland, A., Smit, T.S., van Willigen, A.L.: Monte Carlo Renormalization of the Three-Dimensional Ising Model. Physica A (1989) 1–22

[29] Dellago, C., Geissler, P.L.: Monte Carlo sampling in path space: Calculating tme correlation functions by transforming ensembles of trajectories. In: The Monte Carlo Method in the Physical Sciences: Celebrating the 50th Anniversary of the Metropolis Algorithm. Volume 690., AIP (2003) 192–199

[30] Creutz, M.: Microcanonical Monte Carlo Simulation. Phys. Rev. Lett. **50** (1983)

[31] Manousiouthakis, V.I., Deem, M.W.: Strict Detailed Balance is Unneccessary in Monte Carlo Simulation. J. Chem. Phys. **110** (1999)

[32] Creutz, M.: Microcanonical Monte Carlo. (2003)

[33] Bruce, A.D., Jackson, A.N., Ackland, G.J., Wilding, N.B.: Lattice-switch Monte Carlo. Phys. Rev. E **61** (1999) 906–919

[34] Baillie, C.F., Coddington, P.D.: Cluster identification algorithms for spin models – sequential and parallel. Concurrency: Practice and Experience **3** (1991) 129–144

[35] Bae, S., Ko, S., Coddington, P.: Parallel wolff cluster algorithms. Int. J. Mod. Phys. C **6** (1995) 197–210

[36] Davis, S.W., McCausland, W., McGahagan, H.C., Tanaka, C.T., Widom, M.: Cluster-based monte carlo simulation of ferrofluids. Phys. Rev. E **59** (1999) 2424–2428

[37] Swendsen, R.H., Wang, J.S.: Nonuniversal critical dynamics in Monte-Carlo simulations. Phys. Rev. Lett. **58** (1987) 86–88

[38] Barkema, G., Newman, M.: New monte carlo algorithms for classical spin systems. Advances in Chemical Physics **105** (1998) 1–36 Monte Carlo Methods in Chemical Physics.

[39] Luijten, E.: Introduction to cluster monte carlo algorithms. In: Lect. Notes Phys. Number 703. Springer (2006) 13–38

[40] Wolff, U.: Comparison Between Cluster Monte Carlo Algorithms in the Ising Model. Physics Letters B **228** (1989) 379–382

[41] Binder, K., Heermann, D.W.: Monte Carlo Simulation in Statistical Physics. Springer-Verlag (1997)

[42] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. **21** (1953) 1087–1092

[43] Glauber, R.: Time dependent statistics of the Ising Model. J. Math. Phys. **4** (1963) 294–307

[44] Bortz, A.B., Kalos, M.H., Lebowitz, J.L.: A new algorithm for Monte-Carlo simulation of Ising spin systems. J. Comp. Phys. **17** (1975) 10–18

[45] Creutz, M.: Microcanonical Monte Carlo Simulation. Phys. Rev. Lett. **50** (1983) 1411–1414

[46] Baillie, C.F.: Lattice spin models and new algorithms. Caltech Preprint C3P-777 (1989)

[47] Wolff, U.: Collective Monte Carlo Updating for Spin Systems. Phys. Lett. **228** (1989) 379

[48] Hawick, K.A., Leist, A., Playne, D.P.: Parallel Graph Component Labelling with GPUs and CUDA. Technical Report CSTN-089, Massey University (2009) Accepted (July 2010) and to appear in the Journal Parallel Computing.

[49] Reddaway, S.F.: DAP a Distributed Array Processor. In: Proceedings of the 1st annual symposium on Computer Architecture,(Gainesville, Florida), ACM Press, New York (1973) 61–65

[50] Marsaglia, G., Zaman, A., Tsang, W.W.: Toward a universal random number generator. Statistics and Probability Letters **9** (1987) 35–39 Florida State preprint.

[51] Hawick, K., Leist, A., Playne, D.: Some Parallel Random Number Generators and Implementation Issues on GPUs, Multi-Core and Cell Processors. Technical Report CSTN-103, Computer Science, Massey University (2009)

[52] Hawick, K.A., James, H.A.: Ising model scaling behaviour on z-preserving small-world networks. Technical Report arXiv.org Condensed Matter: cond-mat/0611763, Information and Mathematical Sciences, Massey University (2006)

[53] Coddington, P.D., Baillie, C.F.: Empirical relations between static and dynamic exponents for ising model cluster algorithms. Phys.REv.Lett. **68** (1992) 962–965

[54] Krinitsyn, A., Prudnikov, V., Prudnikov, P.: Calculations of the dynamical critical exponent using the asymptotic series summation method. Theor. and Math. Physics **147** (2006) 561–575