Performance Comparison of CPU and GPU Algorithms for the 2D Ising Model

Thomas Mullett
Department of Physics, University of Warwick, Coventry CV4 7AL, United Kingdom.
(Dated: November 29, 2025)

A range of Monte Carlo algorithms were tested for the two-dimensional Ising model (discrete spin lattice), including Metropolis (Python and Numba), Wolff (Numba), and OpenCL GPU implementations using single and double checkerboard tiling. The double checkerboard kernel achieved the highest performance, outperforming CPU methods by several orders of magnitude. Validation through magnetisation and Binder cumulant analysis confirmed correct thermodynamic behaviour. Checkerboard-based methods exhibited critical slowing down near the critical temperature, where the Wolff algorithm remained superior. Away from criticality, the double checkerboard approach provided the best balance of speed and accuracy for Ising-like studies.

I. INTRODUCTION

The Ising model [1] remains one of the most studied systems in statistical physics, despite its age and conceptual simplicity. The model describes a set of discrete spins $s_i = \pm 1$, arranged on a lattice and interacting with nearest neighbours s_j, according to the Hamiltonian (in zero field)

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j, \tag{1}$$

where J is the coupling constant and the summation $\langle i, j \rangle$ runs over all neighbouring spin pairs. For $J > 0$, the model describes a ferromagnet, in which aligned spins lower the energy and spontaneous magnetisation can emerge below a critical temperature, T_c. Although the Ising model can be solved analytically in two dimensions, its finite-size behaviour and generalisations remain an important topic of computational research. Modern applications include magnetic material simulations [2], biological systems [3], percolation models, and phase separation studies [4]. In all of these contexts, large-scale Monte Carlo (MC) simulations, such as shown in Figure 1, are essential for obtaining accurate results. However, the computational cost of simulating large lattices scales with the number of spins L^2 and the autocorrelation time near T_c, making efficient algorithms crucial for feasible computation.

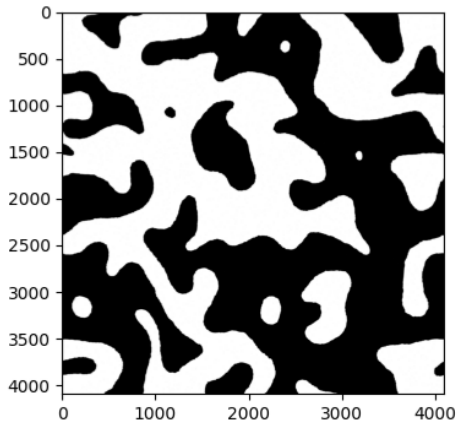

FIG. 1. Large scale magnetic domains on a 4096×4096 lattice after $10,000$ sweeps with $T = 1$.

This study compares CPU and GPU implementations of Metropolis and Wolff algorithms for the 2D Ising model. We evaluate performance in terms of time per spin update and autocorrelation scaling with temperature, and validate statistical accuracy through magnetization and Binder cumulant analysis. Together, these benchmarks provide guidance for choosing the best MC strategies in large-scale spin simulations and related grid-based problems.

II. ALGORITHMS

A. Metropolis Algorithm

The Metropolis algorithm [5] is a MC method in statistical physics that can be used to approximate a distribution. For the Ising model, the Metropolis algorithm starts by selecting a random spin and proposing a spin flip. This flip is then accepted with the probability P_{acc},

$$P_{acc} = \min(1, \exp(-\beta \Delta E)), \tag{2}$$

where ΔE is the energy change from the spin flip and β is the inverse temperature $\frac{1}{k_B T}$. Repeating this procedure L^2 times forms one Monte Carlo sweep.

Because each update affects the acceptance probability of its neighbours, not all spin sites can be updated in parallel as trial moves have to be non-interacting. To enable parallel execution the lattice is divided into two sublattices in a checkerboard pattern [6]. Since no two spins of the same colour are neighbours of each other, all spins of one colour can be updated in parallel without data dependency and preserve detailed balance. The checkerboard algorithm works by alternating between the two sublattices, applying the Metropolis algorithm at each spin site. This approach can be easily implemented on GPUs and should also improve performance on CPUs due to better use of the memory cache and branch prediction over the random spins.

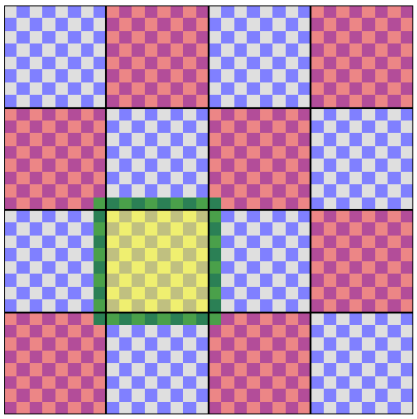

FIG. 2. Double checkerboard decomposition of a square lattice with side length $L = 32$. There are 4×4 big tiles, each one allocated to a work-group. The work-items update the 8×8 spins of each sub-lattice in the tiles in parallel. The green spins represent the halo for the yellow tile.

On the GPU the checkerboard algorithm is limited by global memory bandwidth and latency. To address this, the checkerboard scheme can be extended using a tiling strategy where the lattice is first decomposed into a larger checkerboard of tiles [7], shown in Figure 2. Here each work-group loads one tile of spins, plus a halo of surrounding spins, into local memory. Each work-item then consecutively updates both colour spins within the tile and lastly, writes the updated spins back into global memory. By updating opposite tile colours sequentially the whole lattice can be updated simultaneously without dependency. This approach reduces the global memory traffic, avoids bank-conflicts and removes the need for costly periodic boundary condition checks. It also allows for a multi-hit technique where each spin within a tile is updated multiple times before being written back. This improves the calculation to global memory access ratio but increases autocorrelation times near T_c. Optimal tile sizes are dependant on the devices local memory capacity, global memory latency and maximum wave-group size.

B. Wolff Algorithm

The Wolff algorithm [8] is designed to address critical slowing down by performing cluster updates. It starts by selecting a random spin, s_i, and adds neighbouring spins, s_j, of the same orientation to the cluster with probability $P_{bond} = 1 - exp(-2\beta J)$. This cluster grows recursively until there are no more spins that satisfy the criterion, after which the entire cluster is flipped. Because the algorithm flips large clusters of spins it reduces the autocorrelation time near T_c. Unfortunately these cluster sizes are unpredictable and memory accesses are irregular, making the algorithm very challenging to parallelise on a GPU. In this study, the Wolff algorithm serves as a physical reference to validate the thermodynamic results and compare autocorrelation times of the faster but local Metropolis variants.

III. IMPLEMENTATION

In this study, simulations were executed on a system equipped with an AMD Ryzen 5 5600X (6 cores, 12 threads) CPU and an AMD Radeon RX 7900 XT GPU (20 GB GDDR6, RDNA 3 architecture). All CPU-based codes are written in Python 3.11, while GPU implementations are written in OpenCL C and executed via PyOpenCL. OpenCL was chosen for its portability across GPU vendors and explicit control over memory hierarchy. Pseudo-random numbers are generated with PCG64 on the CPU and MWC64X on the GPU.

A. CPU Implementations

A baseline implementation is written in pure Python and serves as a benchmark for other CPU implementations. The main sequential Metropolis updater is accelerated using Numba [9], providing just-in-time (JIT) compilation to replace the Python loops with compiled kernels. A serial sequential checkerboard algorithm is implemented using Numba. A multi-core version using Numba's `@njit(parallel=True)` to parallelise the loop over spins is also tested, giving scaling up to 12 threads on the 5600X. The Wolff algorithm relies heavily on dynamic memory and recursive operations during cluster formation. To avoid Python recursion overhead, a custom stack-based cluster builder is implemented in Numba.

B. GPU Implementations

The first GPU implementation adopts the checkerboard scheme described earlier and runs one kernel for both colours per sweep. After each kernel call, a global synchronisation ensures consistency between sublattices. The double checkerboard algorithm is implemented with a $L \times L$ lattice decomposed into $B \times B$ tiles of size $t \times t$, where each work-group loads one tile. Once loaded, each tile undergoes k local sub-sweeps before writing back to global memory. Two kernel variants were tested:

1. *Spin-pair kernel*: each work-item updates a pair of adjacent spins, achieving excellent memory coalescing but limited by the 256 work-item per group limit.

2. *Row kernel*: each work-item updates an entire row within the tile. This increases the spin calculations per work-item but allows larger tile sizes.

The OpenCL kernels use single-precision (32 bits) arithmetic and rely on explicit local barriers for intra-group synchronisation.

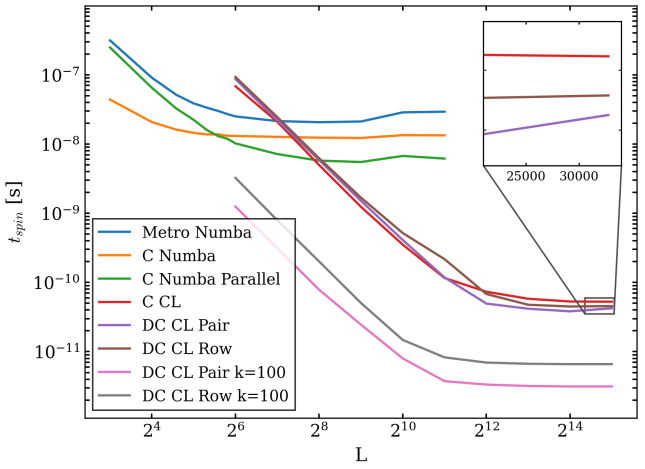

FIG. 3. Time per spin-flip with CPU and GPU methods as a function of lattice size. All data is at $T = 2$. The spin pairs use a tile size of $t = 16$ and rows use $t = 32$. Here C means checkerboard, DC means double checkerboard and CL means the OpenCL implementation.

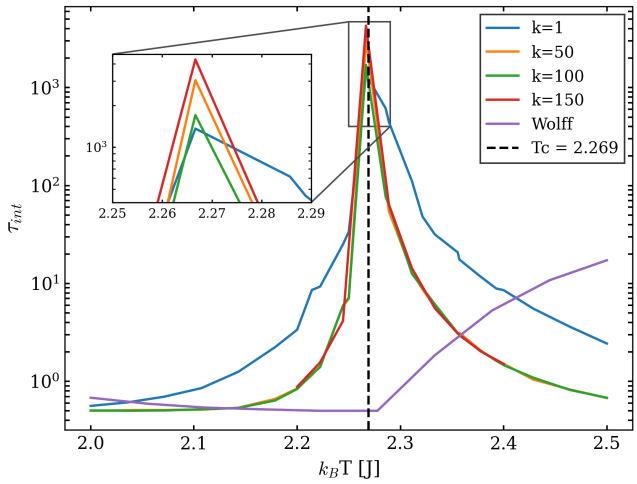

FIG. 4. Integrated autocorrelation times of magnetisation on a 256×256 lattice for the Wolff and double checkerboard algorithm as a function of temperature. The GPU updates were performed with different multi-hit updates, k, but with a constant total number of updates.

IV. RESULTS

The time per spin update, t_{spin}, for $L = 8$ to 32768 is shown in Figure 3. It was calculated from the average t_{spin}/L^2 over $10,000$ sweeps. The unoptimised python version is not shown but has a $t_{spin} \approx 1 \times 10^{-5} s$ at all L. t_{spin} can exhibit a weak temperature dependence from the varying proportion of accepted flips and resulting global-memory write-backs. $T = 2$ was chosen for measurements due to proximity to the critical point while being within the ferromagnetic regime.

The integrated autocorrelation time of magnetisation is shown in Figure 4 for the double checkerboard and Wolff implementations. Both algorithms had $500,000$ measurements, each 10 sweeps apart, after $100,000$ sweeps used for equilibration.

Figure 5 shows the phase diagram for magnetism, m, of the 2D Ising model, which was produced by the double checkerboard implementation over the same sweeps alongside the exact solution [10].

The binder cumulant, U_L [11], at various L for the Wolff implementation is shown in Figure 6 and is defined as

$$U_L = 1 - \frac{\langle m^4 \rangle}{3\langle m^2 \rangle^2}, \tag{3}$$

where $\langle\rangle$ denotes the average.

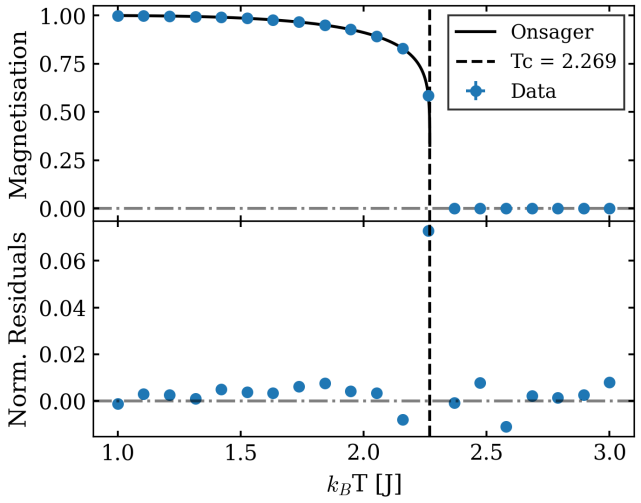

FIG. 5. Phase diagram for the double checkerboard algorithm with $k = 1$.

V. DISCUSSION

Figure 3 shows that on the CPU, both Metropolis and Checkerboard Numba-accelerated variants scale approximately as expected, with the parallel Numba implementation giving the lowest t_{spin}. The parallel checkerboard update achieves modest speedup due to better cache usage and independent site updates, but remains limited by memory bandwidth and the number of CPU threads. In contrast, the OpenCL GPU implementations achieve the largest speedups by exploiting massive data parallelism. The single-layer checkerboard OpenCL kernel outperforms all CPU implementations by two to three orders of magnitude for larger system sizes. This kernel remains fully memory-bound but the GPU's higher aggregate bandwidth provides an immediate advantage over the CPU. Between the two GPU variants, the row-update double-checkerboard kernel is consistently slower than the spin-pair version. As a larger tile size is required to fill the wavefront, the row kernel needs to load

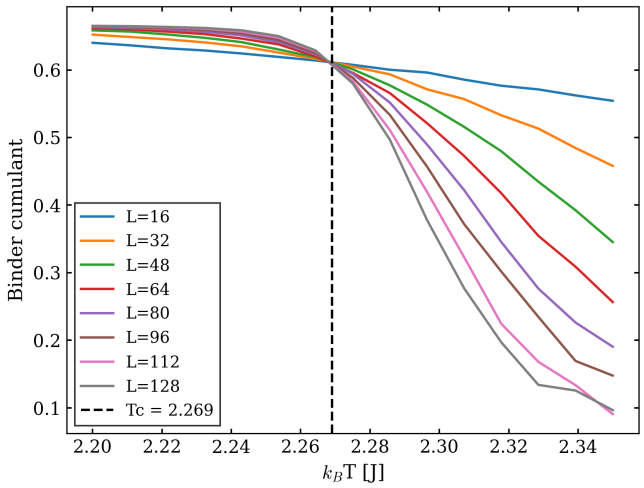

FIG. 6. Binder cumulant as a function of temperature at several L for Wolff algorithm.

more spins into shared memory. This induces register pressure, resulting in register spill into global memory and increased L2 cache traffic, which was observed using AMD's Radeon GPU Profiler. The spin-pair kernel fits comfortably in registers and achieves higher occupancy.

The single-layer checkerboard is faster for small lattices due to reduced overhead, but becomes slower than the double-checkerboard method with $k = 1$ for sufficiently large L. This difference arises as tiling increases the arithmetic intensity and amortises the cost of memory accesses across the spins in the tile. Increasing the tile-update parameter k magnifies this effect by further reducing kernel launch overhead and global-memory traffic across many sweeps. For large k, the double-checkerboard spin-pair variant becomes the fastest algorithm tested, achieving the lowest t_{spin} at large L.

Across all implementations, the simulations reproduce the known equilibrium thermodynamics of the 2D Ising model. Figure 5 shows the magnetisation curve from the double-checkerboard GPU data follows the analytic Onsager solution well, agreeing within roughly 1%. Residuals remain predominantly positive near T_c, consistent with runs that are not fully equilibrated. Near criticality the correlation length $\xi \to \infty$ [12] and the autocorrelation time diverges, requiring significantly longer equilibration. This is evident with the data point closest to T_c, where it remains far from equilibrium despite a constant number of sweeps for all runs. Otherwise, no clear trend is seen.

Figure 6 shows the Binder cumulant exhibits the expected qualitative behaviour, and all system sizes show a transition near the exact critical temperature $Tc = 2.269$. Although finite-size scaling predicts a clear crossing of $U_L(T)$ curves at T_c, statistical noise and the limited number of sweeps in the present dataset somewhat obscure this behaviour. Nevertheless, the Binder curves confirm that the Wolff implementation produces physically meaningful results.

The Wolff cluster algorithm provides the cleanest thermodynamic behaviour near T_c, owing to its dramatically reduced critical slowing down. However, cluster flips make the method difficult to parallelise efficiently on GPUs, so wall-clock performance scales worse than the local-update GPU kernels.

The autocorrelation shown in Figure 4 produces the expected critical behaviour. Near T_c, τ_{int} increases, demonstrating the critical slowing down seen in local spin flip algorithms. Across the double-checkerboard algorithms the ordering is:

$$\tau(k = 1) < \tau(k = 100) < \tau(k = 50) < \tau(k = 150)$$

Small k (particularly $k = 1$) yields the lowest autocorrelation because each tile is synchronised with its neighbours after every sweep, allowing correlations to propagate across the lattice rapidly. Large k causes each tile to evolve independently for longer before global synchronisation, increasing autocorrelation time. This works to oppose larger k values improving t_{spin}, producing an optimal trade-off around $k \approx 100$.

The Wolff algorithm again exhibits the lowest autocorrelation at T_c due to collective cluster flips that decorrelate the system in a single step. Away from criticality, cluster sizes shrink or grow excessively, and autocorrelation becomes larger relative to its performance at the critical point.

Overall, GPU acceleration combined with shared-memory tiling is over four orders of magnitude faster compared to optimised CPU–Numba implementations, clearly demonstrating the advantage of GPU algorithms. However, near criticality τ_{int} for local methods becomes too large, meaning cluster algorithms are the only viable approach, even with worse wall-clock times.

VI. CONCLUSION

We compared CPU and GPU implementations of several Monte Carlo algorithms for the 2D Ising model. GPU-based checkerboard methods significantly outperform CPU versions, with the double-checkerboard OpenCL kernel offering the best overall performance. All algorithms reproduced correct thermodynamic behaviour and phase transitions, validating the GPU implementations. The Wolff algorithm remains superior near criticality due to reduced autocorrelation, but GPU methods dominate away from T_c. Future work could explore adaptive tile sizes, RNG benchmarking, and extensions to 3D and Potts models.

The code is available at: `https://github.com/ThomasMullett/2D-Ising-Sim`

ACKNOWLEDGMENTS

I would like to thank to Dr. John Back for proof reading this manuscript.

[1] E. Ising, Beitrag zur theorie des ferromagnetismus, Zeitschrift für Physik **31**, 253 (1925).

[2] C. Hang, W. Liu, G. Dobmann, Y. Wu, W. Chen, and P. Wang, Ising model simulation and empirical research of barkhausen noise, Journal of Nondestructive Evaluation **43**, 20 (2024).

[3] M. Ibrahimi, S. Zakany, and M. C. Milinkovitch, Cell-biology effective interpretation of the ising model describing skin color patterning, Physical Review Research **7**, 023093 (2025).

[4] K. Endo, Y. Matsuda, S. Tanaka, and M. Muramatsu, A phase-field model by an ising machine and its application to the phase-separation structure of a diblock polymer, Scientific Reports **12**, 10794 (2022).

[5] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of state calculations by fast computing machines, The Journal of Chemical Physics **21**, 1087 (1953).

[6] T. Preis, P. Virnau, W. Paul, and J. J. Schneider, Gpu accelerated monte carlo simulation of the 2d and 3d ising model, Journal of Computational Physics **228**, 4468 (2009).

[7] M. Weigel, Performance potential for simulating spin models on gpu, Journal of Computational Physics **231**, 3064 (2012).

[8] U. Wolff, Collective monte carlo updating for spin systems, Physical Review Letters **62**, 361 (1989).

[9] S. K. Lam, A. Pitrou, and S. Seibert, Numba: a llvm-based python jit compiler, in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15 (Association for Computing Machinery, New York, NY, USA, 2015).

[10] L. Onsager, Crystal statistics. i. a two-dimensional model with an order-disorder transition, Physical Review **65**, 117 (1944).

[11] K. Binder, Finite size scaling analysis of ising model block distribution functions, Zeitschrift für Physik B Condensed Matter **43**, 119 (1981).

[12] D. W. H. Kurt Binder, *Monte Carlo Simulation in Statistical Physics* (Springer Berlin, Heidelberg, 2010).