Computational Time Optimisation of the Ising Model

Darren D. Q. Ng
Level 7 Advanced Computational Physics, H. H. Wills Physics Laboratory, University of Bristol
(Dated: February 14, 2020)

The 2D ferromagnetic Ising model simulated by the Metropolis algorithm is laid out in this report. The observables of the simulation – energy, magnetisation, specific heat and susceptibility per spin – has been presented as a function of temperature. A variety of optimisation is used and its discussions are presented. By using domain decomposition along with Numba, running on a maximum of 16 cores on University of Bristol's supercomputer BlueCrystal, a maximal speed up of 230× is observed. Potential improvements of the base algorithm are discussed.

I. INTRODUCTION

The Ising model is a model of magnetism in statistical physics. Essentially, the magnetism of an object is comprised of sum of all the magnetic dipole moments of spins given by

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

with each spin confined on a lattice site s_i and can only take the values of $+1$ and -1 representing the spins pointing up and down respectively. The interactions between each nearest neighbour sites are governed by the a coupling constant J. For values of J where either $J > 0$, $J = 0$ and $J < 0$, the type of magnetism that occurs in the material is either ferromagnetic, non-interacting and anti-ferromagnetic respectively. For N spins within the lattice, they system can be in any 2^N particular configuration. The energy of the system for a particular lattice configuration σ is given by the Hamiltonian

$$\mathcal{H}(\sigma) = -J \sum_{i}^{N} \sum_{\substack{j \\ \textbf{n.n.} \\ \text{to } i}} s_i s_j - B \sum_{i}^{N} s_i \tag{2}$$

where i is summed over all N spin sites s_i, s_j represents the nearest-neighbour lattice sites to s_i and B is the external magnetic field. The other interesting observables of the material such as specific heat capacity c and magnetic susceptibility χ can be obtained by [1]

$$c = \frac{1}{(k_B T)^2}(\langle E^2 \rangle - \langle E \rangle^2), \tag{3}$$

$$\chi = \frac{1}{k_B T}(\langle M^2 \rangle - \langle M \rangle^2). \tag{4}$$

This report will be focusing on simulating the Ising model for a simple 2-d system with finite size square lattice, hence each lattice site will only have four neighbouring sites. The simulation will also be performed in zero magnetic field $B = 0$, since that would be sufficient for all the interesting behaviour of the system related to their observables to be investigated. In order to ignore boundary effects and still remain a finite size simulation, periodic boundary conditions are used such that the first spin of a column will take the final spin of the same column as its top neighbour, and the same for left and right neighbours of the first and final spin in a row. This effectively renders the lattice of dimension L into a torus of dimension $L + 1$. This also introduces a translation invariance in the calculations of the observables for domain decomposition as it will be seen later. J is taken to be ferromagnetic and for simplicity the Boltzmann constant k_B is set as 1.

A. Algorithm

A thermal Monte Carlo(MC) method, namely the *Metropolis Algorithm* [2], will be used for this Ising Model simulation.

Metropolis importance sampling Monte Carlo scheme

1 Generate an initial lattice configuration.

2 Compute the energy for the current lattice configuration.

3 Randomly choose a lattice site, compute the energy of the new configuration if the site were to be flipped and also the change in energy, ΔE, between both configurations.

4 Generate a random number r *uniformly* within the interval $[0, 1]$.

5 If $\Delta E < 0$ or $r < \exp(-\Delta E / k_B T)$, update the lattice with the chosen site flipped.

6 Go to back to **2**.

There are several important conditions for which an algorithm needs to obey for it to be equilibrium thermal MC and Metropolis [1]. This report will briefly explain all of them, and elaborate only on the ones which will be used to investigate the validity of the simulation. The main three conditions for MC are detailed balance, importance sampling, and ergodicity.

The detailed balance condition is simply the fact that on average, the amount of times a system starting in a state μ transitioning into state ν should be equivalent to the amount of times the same state ν does a reverse transition back to state μ. The transition from a state to another state is completely random, and this is also known as a Markov process and a repeated transitions of states over time is said to form a Markov chain of states [3]. Importance sampling in MC simulation is used to sample from the states of a system according to the equilibrated Boltzman probability distribution rather than sampling them all with equal probability. A system in equilibrium are dominated by a small proportion of states in which their properties have a narrow and well defined range of values. By sampling them according to the Boltzmann probability

$$p_\mu = \frac{\exp(-\mathcal{H}(\mu)/k_B T)}{\sum_{\mu \in \mathcal{S}} \exp(-\mathcal{H}(\mu)/k_B T)}, \tag{5}$$

, the more "important" states which have a bigger contribution to the value of the observables of the system are being chosen. This allows for estimate of these observables to be obtained quicker without having to sample through all the states with equal probability. The denominator of equation (5) is also know as the partition function which sums over all possible states $\mathcal{S}$, acting as a normalisation constant.

Within this Markov chain of states, the ergodicity condition is said to be fulfilled if it is possible for any state of the system to reach any other state of the system over time. This is an important condition which ensures that the importance sampling in the long run will generate states with the correct Boltzmann weight.

II. SIMULATION

The simulation parameteres from here on wards will be quoted in terms of (NT, L, EQ_STEPS, MC_STEPS, MAX_T, DS). NT is the number of data points within a temperature range, L is the dimension of the lattice, EQ_STEPS is the number of sweeps that is performed for equilibration, MC_STEPS is the number of sweeps that is performed for calculating the observables and DS is the amount of samples to discard within MC_STEPS before an observable is taken.

A. Validity

1. *Equilibration*

An important question to address before quantity of interests are observed is the time for which the simulation needs to be ran before it reaches equilibrium. For a system to be in equilibrium, its average probability of being in some state μ follows equation (5). Equilibrium and time to equilibration depends on the temperature of the system started off with and eventually ends up. At $T = 0$, the initial state of a lattice can have two ground states where its spins are either all pointing up or down, whereas at $T = \infty$ the initial state contain spins which are scrambled and uncorrelated. Since in this version of Metropolis algorithm a random site is chosen and flipped, it may take awhile before every site has been flipped at least once. The act of random choice actually turns out to be a disadvantage in the exact reproducibility of a specific simulation and also other problems, which will be discussed later.

It will not be immediately obvious to know if a system is in equilibrium by just plotting snapshots of the lattice's configuration at certain different times. An ideal way of visualising equilibrium and approximate the equilibration time would be to simply plot some quantity of interests whose behaviour are known at certain temperatures, such as the magnetisation per spin of the system. For example, for a system equilibrating from an initial state at $T = \infty$ to a very low temperature, in equilibrium the state must be in one of the two aforementioned ground states. By equation (1) it can be inferred that the system's magnetisation per spin should equilibrate to around unity. With such an approach, running this test several times can sometimes seem to give a different equilibration time each time. This is because sometimes it is possible for the equilibrating system to be stuck in a metastable region of local energy minimum as opposed to global energy minimum. To get around this, one can simply do the test on more than one distinct lattice starting configuration and see at what time their magnetisation per spin converge. This is illustrated in FIG. 1. and FIG. 2 for different temperatures. For the subsequent discussions, a test run of a simulation is first done in order to obtain the quilibration time, in order to run the same simulation again for a certain set of parameters.

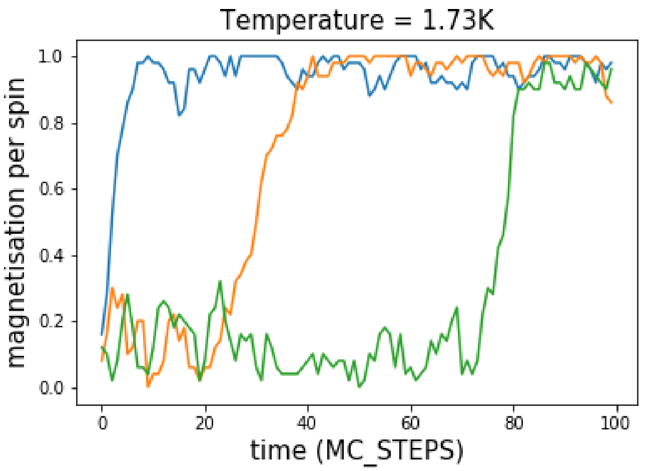

FIG. 1. The absolute value of magnetisation per spin as a function of time for three distinct states starting at $T = \infty$ for a size 10×10 lattice. At a low equilibrium temperature $T = 1.73K$, the lattices converging on an equilibration time of 90 MC_STEPS is observed.

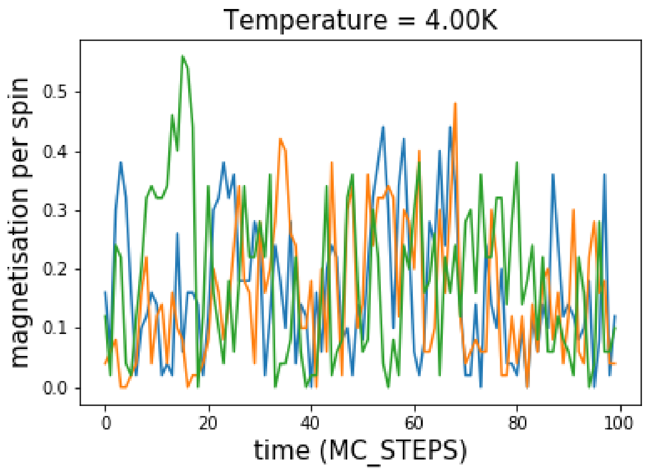

FIG. 2. Same as FIG. 1. but with a higher equilibrium temperature $T = 4K$. It is not immediately obvious where the equilibration time is at since the systems each started off and ended up in a disordered state.

2. *Ergodicity*

It is possible to check if the ergodicity condition of a MC simulation is being fulfilled by comparing the sampled distribution to the true distribution, however only for a small lattice dimension. Since for every $L \times L$ lattice size, the number of its every possible configuration is 2^{L^2}. For $L = 5$, this amounts to $2^{25} = 33554432$, which makes such test undesirable since it will be very slow. For $L = 4$ however, it is computationally feasible to still compute the probability of being in each state for all possible states, thereby obtain the true equilibrium distribution. In `ising_numba_investigate.py`, this is done by the function `permutation()` where it first calculates all the permutation of a bitstring. By reshaping the bitstrings into square lattices and replacing the all zeros with minus ones, all possible configurations of a lattice can be found and hence their corresponding p_μ from equation (5) can be computed and plotted as a histogram which is the true stationary distribution. During the calculation of observables within the `MC_STEPS` loop, which occurs after the the system has reached equilibrium, all sampled energy values can then be used to also compute each of their corresponding p_μ, with its plotted histogram representing the sampled distribution. In the limit that the sampling is done infinite many times, the sampled distribution will tend towards the true equilibrium distribution, as shown in FIG.4. Since a single sweep is defined to have L^2 random spin flips, which means that not every single site would have necessarily been flipped at least once per sweep. This can introduce a bias in the sampling done from the equilibrium distribution in the algorithm. Therefore any MC Metropolis algorithm where single flip spins are done by choosing random sites necessarily have to discard some samples over couple of sweeps in its observations. With enough sweeps ergodicity can be thought to have fulfilled by the averaging of the random L^2 spin flips per sweep. Both the equilibration and ergodicity of the simulation are investigated in `ising_base_investigate.py`.

B. Results

The results for the energy, magnetisation, specific heat and susceptibility per spin is obtained as a function of 100 temperature points for different lattice dimensions are illustrated in FIG. 5, FIG. 6, FIG. 7. and FIG. 8 respectively. The systems are allowed to equilibrate for 10000 MC_STEPS with observations made over 10000 EQ_STEPS, discarding 10 samples for every observation.

III. COMPUTATIONAL TIME OPTIMISATION

A. Mathematical and Algorithmic Simplification

In step **3** of the algorithm, it is required that the energy difference between an old state μ and its new state ν to be computed. Although this can be simply done by subbing μ and ν into Equation (2), summing over all spins s_i^μ and s_i^ν of both states and obtain $\Delta E = E_\nu - E_\mu$, but this way is nevertheless very inefficient. Since for a lattice dimension of $L \times L$, the sum in Equation (2) has $\frac{1}{2}L^2\gamma$ terms, where γ is the number of neighbouring sites each site s_i has which is four. This can be mathematically simplified since most the spins that are not the neighbours of the chosen site actually cancel out. With respect to only the particular chosen spin before, s_k^μ, and after, s_k^μ, both energies E_ν and E_μ are

$$E_\nu = -Js_k^\nu(s_1^\nu + s_2^\nu + s_3^\nu + s_4^\nu) \tag{6}$$

$$= -Js_k^\nu \sum_{\substack{i \\ \textbf{n.n.} \\ \text{to } k}} s_i^\nu \tag{7}$$

$$E_\mu = -Js_k^\mu(s_1^\mu + s_2^\mu + s_3^\mu + s_4^\mu) \tag{8}$$

$$= -Js_k^\mu \sum_{\substack{i \\ \textbf{n.n.} \\ \text{to } k}} s_i^\mu \tag{9}$$

But the sum of nearest neighbour spins s_i^ν and s_i^μ are never flipped, hence the sum of their values will always be the same i.e. $\sum s_i^\nu = \sum s_i^\mu$. Hence the energy difference $E_\nu - E_\mu$ is now $-J\sum s_i^\nu(s_k^\nu - s_k^\mu)$. An additional trick here which can be done is by considering the value of $(s_k^\nu - s_k^\mu)$. If the spin before, s_k^μ, is $+1$, then the spin after, s_k^ν, is necessarily -1 and vice versa. This means their difference in value can either be -2 or $+2$ respectively and hence their value difference can be written as $(s_k^\nu - s_k^\mu) = -2s_k^\mu$. ΔE can now be expressed in terms of the spins before

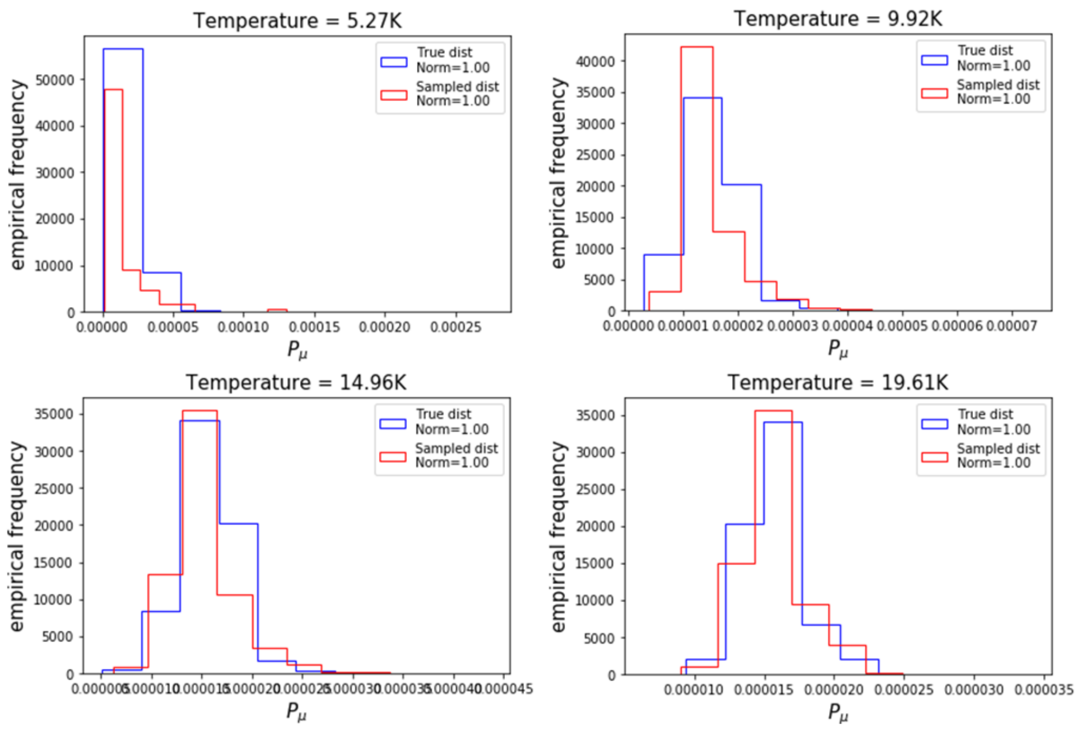

FIG. 3. Sampled and stationary distribution comparison for a run of $(50, 4, 2^{4^2}, 100\times 2^{4^2}, 20, 100)$ for four different temperatures. The energies of the system are only every 100 sweeps, and are otherwise discarded. Both distributions are normalised to one. It can be seen that the sampled distribution is tending towards the true distribution.

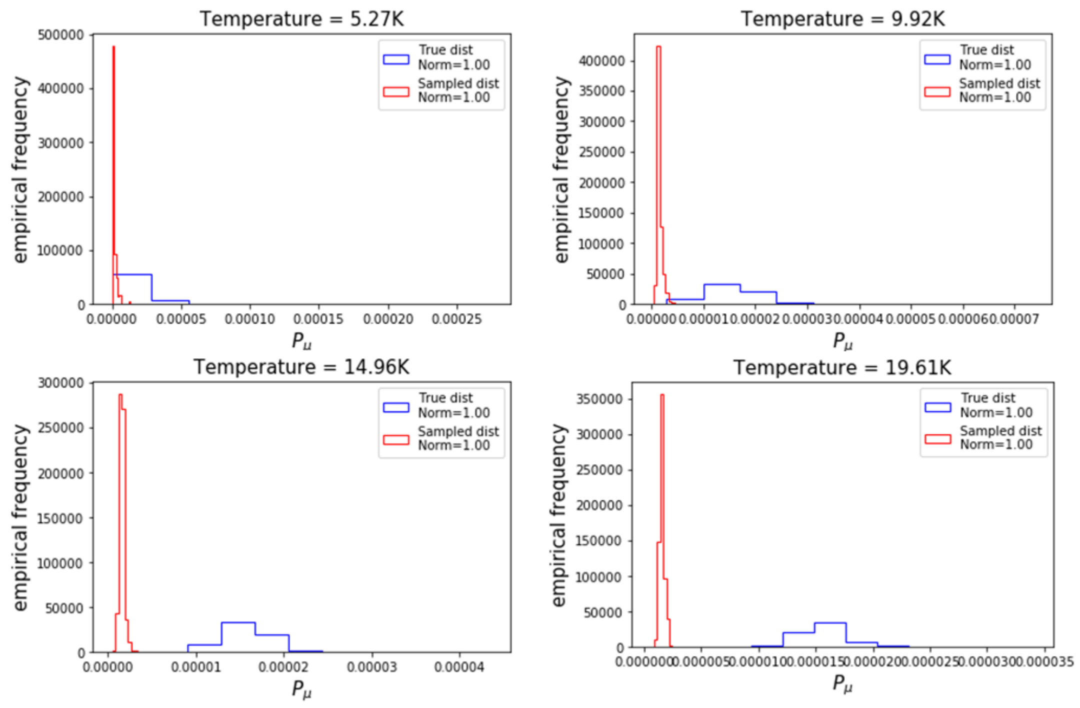

FIG. 4. A similar graph to FIG. 4. with the same simulation parameters. An important distinction here is that energies are being observed for every sweep instead of every 100 sweeps i.e. no samples are discarded. It can be seen that the true distribution deviates further and further away from the sampled distribution as a result of ergodicity not being fulfilled.

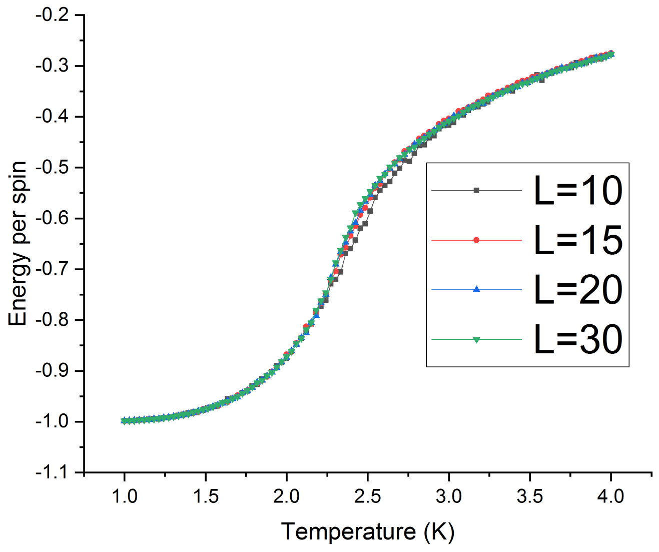

FIG. 5. The energy per spin as a function of temperature for four different lattice sizes. At very low temperature it can be seen that all systems are in their ground states and have near identical energy per spin values. As temperature increases the spins become more and more disordered so their total contribution of energy will tend to 0.

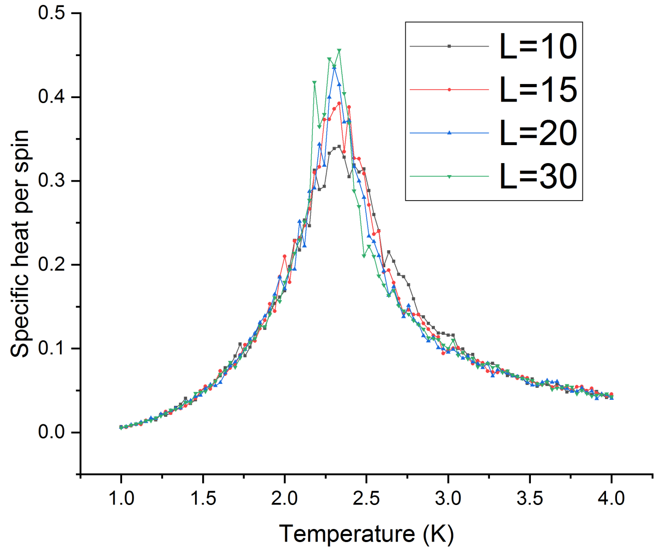

FIG. 7. The specific heat per spin as a function of temperature for four different lattice sizes. It is known to peak around Curie temperature and the height of its peak is proportional to lattice site. The fluctuations around Curie temperature is a phenomenon known as "critical slowing down" [5], which is one of the shortcomings of Metropolis algorithm around critical points where phase transitions happen.

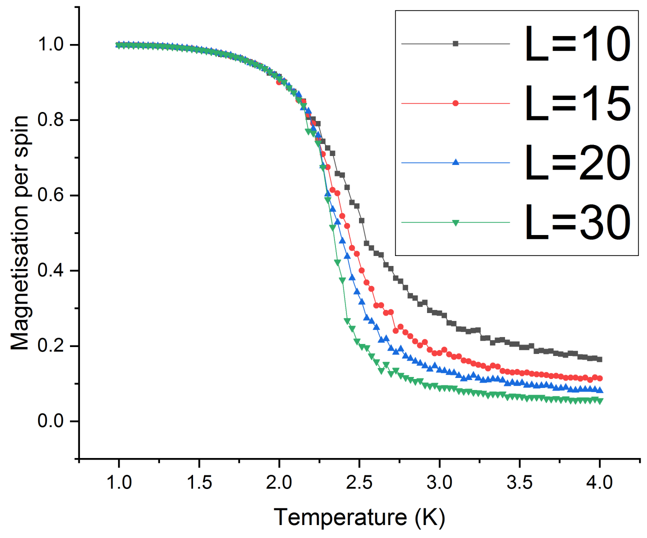

FIG. 6. The absolute value of magnetisation per spin as a function of temperature for four different lattice sizes. At very low temperature, the spins in their ground states could either all point up or down, hence the absolute magnetisation per spin is expected to be 1. The disordering of spins in the lattice as a result of increasing temperature destroys the magnetisation, and hence will tend to 0. The turnover point at which a system loses its permanent magnetisation is commonly known as the Curie temperature, having an analytical value of 2.266K in 2D with zero field [4].

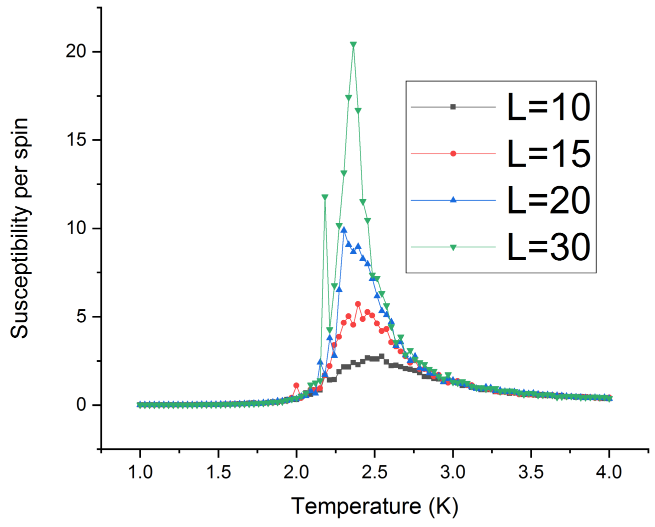

FIG. 8. The susceptibility per spin as a function of temperature for four different lattice sizes. Again the peak is sharper for higher lattice dimensions around Curie temperature.

only, giving

$$E_\nu - E_\mu = -J \sum_{\substack{i \\ \textbf{n.n.} \\ \text{to } k}} s_i^\mu (s_k^\nu - s_k^\mu) \tag{10}$$

$$= 2Js_k^\mu \sum_{\substack{i \\ \textbf{n.n.} \\ \text{to } k}} s_i^\mu \tag{11}$$

$$= 2Js_k^\mu (s_1^\mu + s_2^\mu + s_3^\mu + s_4^\mu), \tag{12}$$

thereby reducing the complexity of the calculation to just γ terms. Additionally, although the neighbouring spins have 2^4 ways of rearranging them in ±1(s), the sum of their values however can only have any five values at a time, i.e. $\sum s_i^\mu \in \{-4, -2, 0, +2, +4\}$, since certain permutations can have the same sum value. This means the computation of $\exp(-\Delta T/k_B T)$ can be precomputed and stored in an array, to be retrieved only when needed. In fact, it only needs to be computed when the sum is negative, due to step **5** of the algorithm. This reduces the number of times the exponential function is computed to just two terms, and is more efficient since they are usually done by series expansion methods which are computationally heavy. With all simplifications above considered, the comparison are shown in FIG. 9, with `ising_base.py` being the optimised version and otherwise. The code `ising_base.py` is adapted from here onwards for further optimisation.

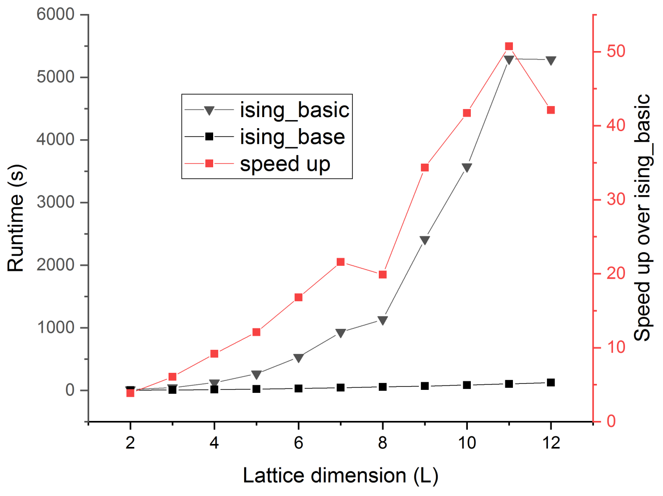

FIG. 9. Lattice dimension against computational run time between both versions of algorithm, for parameters $(10, 5000, 5000, 4, 10)$. A maximal speedup of 50.7 times is observed for $L = 12$. The run time for `ising_basic` scales proportional to $L^2\gamma$ and `ising_basic` scales proportional to γ which is as expected.

B. Numba

Numba [6] is an efficient and open source Just In Time (JIT) compiler which accelerates Python algorithms and works extremely well with the Numpy module. It does so by converting certain Python codes into optimised machine code, allowing for speed up which sometimes levels that of C or FORTRAN. With no C/C++ compiler needed, simply adding a decorator on a Python function is sufficient. Numba also allows for parallellising for loops on multiple CPU cores for operations in functions which have parallel semantics i.e. appending values into an array. The speed up of `ising_base` using Numba is shown in FIG. 10. The speed up of parallelising Numba over normal Numba performed on two different local machines is shown in FIG. 11 and FIG. 12.

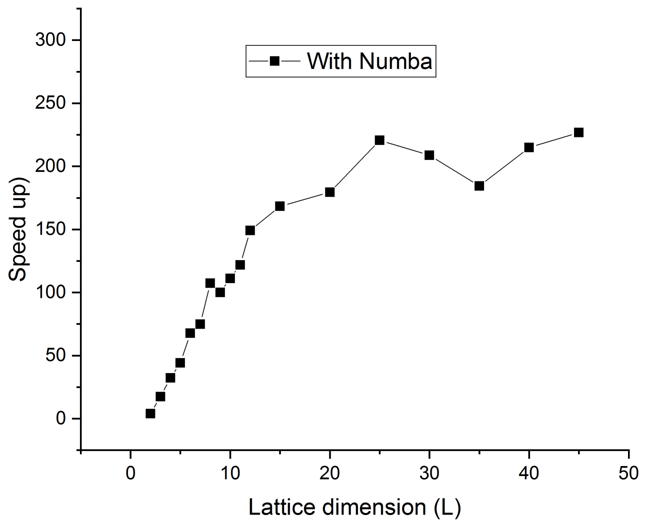

FIG. 10. The speed up of `ising_base.py` using Numba. Lattice dimension against computational run time between both versions of algorithm, for parameters $(10, L, 5000, 5000, 4, 10)$. A maximal speedup of 50.7 times is observed for $L = 12$.

C. Mpi4py Domain Decomposition

Mpi4py is a Message Passing Interface for Python which allows Python programs to execute on multiple processors. It has a distributed memory structure, which means each local memory belongs to each processor, and so messages must be passed between processors in order for data exchange. Ideally, the frequency and amount of small messages sent should be minimise, and more of them should be contained within a larger message that will be sent, so as to minimise the communication latency and increase bandwidth.

Two similar methods of domain decomposition with slight variation is presented here. Both methods first broadcast a given square lattice with dimension $L \times L$ to all its workers. Upon receiving, the main lattice on each worker will be partitioned equally into equal rectangular sublattices using modular and integer division. The number of columns both any sublattice are always the same, the number of rows for each worker

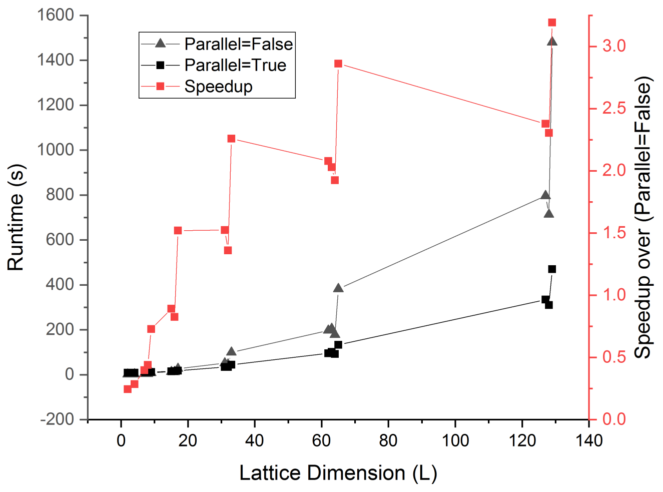

FIG. 11. The speed up of parallelised over non-parellised Numba, on a local machine which has an Intel i3 8100 processor, 4 cores, and base frequency 3.6GHz. The simulation is done with increasing lattice size of powers of two. A maximum speed up of $0.3\times, 0.3\times, 0.4\times, 0.8\times, 1.4\times, 2.3\times$ and $3.2\times$ are observed for $L = 2, 4, 8, 16, 32, 64, 128$ and 129 respectively. Note the strange behaviour where the run time is sped up slightly for $L = 2^n$ and then doubles immediately for $L = 2^n + 1$.

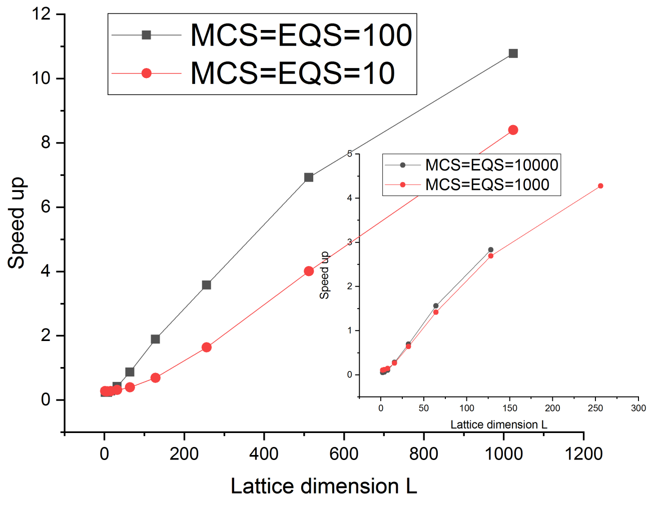

FIG. 12. The speed up of parallelised over non-parellised Numba, this time on a local machine which has an Intel i9 7980XE processor, 18 cores, and base frequency 2.7GHz. The simulation is done with increasing lattice size of powers of two over increasing Monte Carlo time. A maximum speed up of $3\times$, $4\times$, $11\times$ and $10\times$ are observed for $L = 128$, 256, 1024, and 2048 respectively.

is computed by `(L-(size-1))//size` and for master it is `(L-(size-1))//size + (L-(size-1))%size`, where `size` is the number of cores. This implies that for any number of cores, the master will always have more rows than each worker as a result of taking up the remainder rows that were partitioned, and so it will always possess the sublattice at the bottom of the lattice. This imposes the restriction that at any point, `L` is strictly less than `2*size+1`; it is taken cared of in the code by an exception statement. If this restriction is violated, then the lattice dimension is too small to be decomposed into domains by the given amount of cores. Initially, by using their rank numbers, the workers and master will work out the start and end index of their corresponding sublattices for which the sweep will be done on, as shown in FIG. 13. The end index for every sublattice represents a row which borders all sublattices and will not be updated. This is crucial since if the bottom row of master is allowed to be swept, each time it does it will have to inform worker 1 with the updated values. This method is only ideal if the message parsing between processors are synchronised perfectly, otherwise it will introduce unnecessary overhead. Detailed balance is still fulfilled since within the rows that are not swept, the probabilities of a spin being flipped from ± 1 to ∓ 1 are both zero.

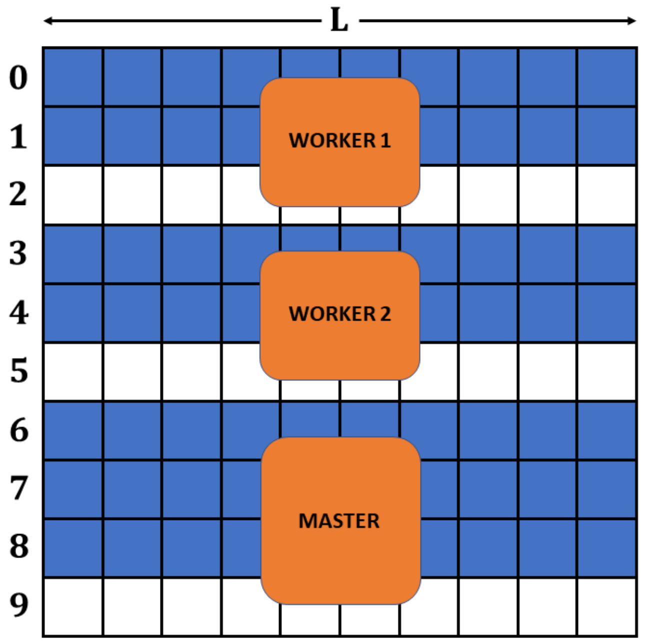

FIG. 13. Domain decomposition on three CPU cores demonstrated with a lattice of size 10×10. The workers always take the sublattices in order from top to bottom. The master always take the most bottom sublattice. The numbers on the left represent the start and end index for each sublattice. They are $(0, 2)$, $(3, 5)$ and $(6, 9)$ for worker 1, worker 2 and master respectively. The final (white) row of each sublattice will not be updated, as opposed to the (blue) rows which will.

The ergodicity of the simulation however will be violated, since there will be sites that do not get swept. Something will need to be done to eventually include the sweeping of these rows. There are two ways to remedy this, which will now be called method 1 and method 2. For method 1 the domains of each sublattice are fixed in place, and for certain amount of sweeps, the data entries

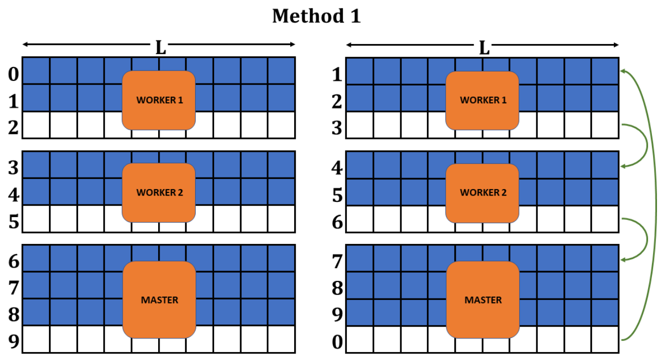

FIG. 14. The numbers on the left represent rows of data, which are shifted down from time to time to fulfill ergodicity. This is possible due to the translational invariance introduced by the periodic boundary condition. Green arrows represent communication between processor i.e. worker 2 updates its own third row by receiving worker 1's third row. It is important to note that each processor possesses the whole lattice but only sweeps its own sublattice.

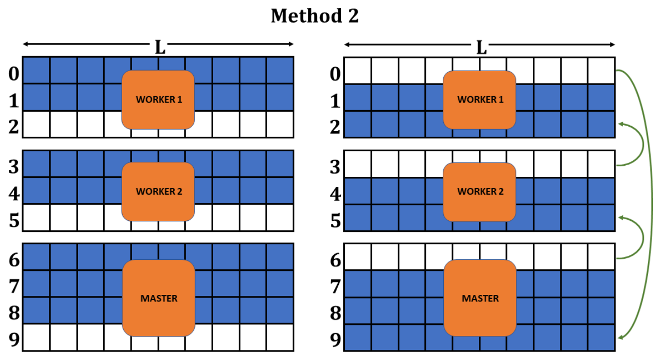

FIG. 15. For method 2 the domains of each sublattice are shifted down rather than the data, by increasing the start and end indices of each domain. This time each processor needs to inform its *previous* processor rather by sending its new row of data, since this can be thought as row data shifting up. This is harder to implement due to the energy and magnetisation functions now have to take into account of wrapping around the whole lattice since at any time any lattice can exist between the top and bottom of the whole lattice.

in every row is being shifted down by one. By doing so, the penultimate row of data in each sublattice that are usually swept is now in the region where it no longer does. Each processor then needs to inform its next processor by sending this new row of data.

The speed up investigation is performed using supercomputer BlueCrystal phase 3 of Bristol University. Numba is used along with MPI since the investigation aimed to look at the maximum amount of speed up possible with the possible number of cores for domain decomposition given a lattice size. The run time and speed up of both method 1 and 2 are illustrated in FIG 16, 17, 18, 19

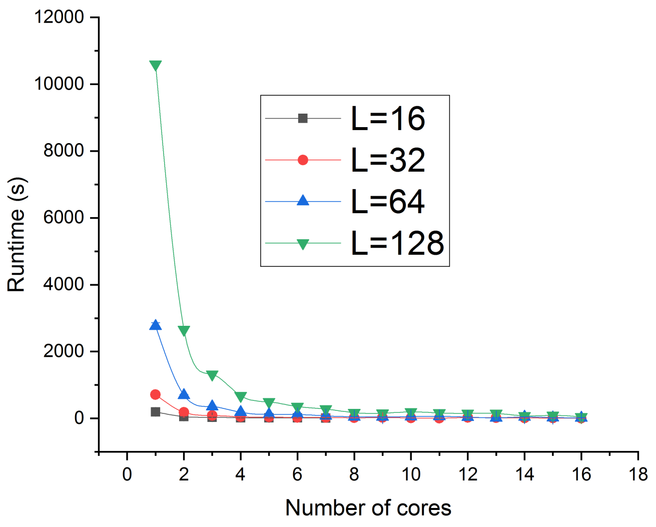

FIG. 16. Runtime against number of cores for various lattice sizes L using method 1 along with Numba, with parameters $(100, 5000, 10000, 4, 10)$. The speed up for smaller L is being overshadowed by speed up for larger L, illustrated more clearly however by FIG. 19.

IV. DISCUSSIONS

A. Ising Model

The obtained values for the energy, magnetisation, specific heat and susceptibility in zero field look reasonable, as fluctuations are expected to occur around the Curie temperature and peaks are expected to be sharper closer to Curie temperature. An improvement to this investigation would be to actually compute the value of Curie temperature and its accuracy, by using equation such as the Binder Cumulant [7]. Moreover, the properties of anti-ferrormagnetic and ferrimagnetic materials can be explored too using the same Ising model and algorithm by changing the value of the coupling constant. The next nearest neighbour interactions with reduced coupling constant could also be considered as an extension of this project. Finally, a 3D Ising model could be investigated instead by rewriting the simulation in a programming language such as C or C++ which are commonly known to have a computational speed superiority over Python.

B. Computational Time Optimisation

It has been demonstrated that by removing the considerations for the spin sites that do not change upon every spin flip and only considering the nearest neighbours of the chosen to be flipped site, the change in energy can be

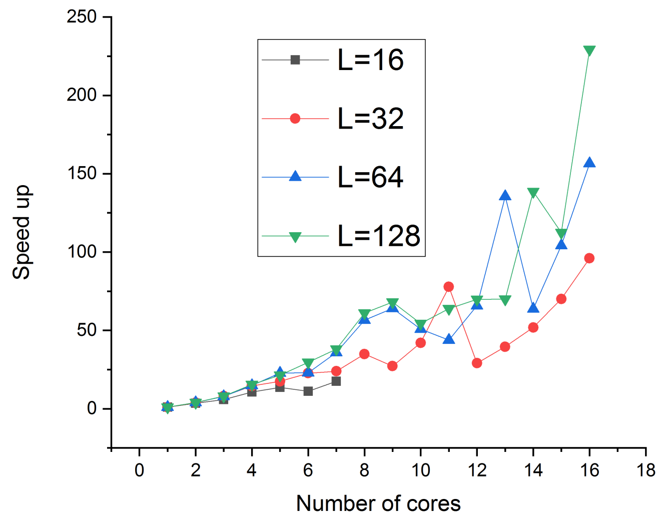

FIG. 17. The speed up of method 1 along with Numba with increasing cores up to 16 relative to single core. For $L =$ 16, 32, 64 and 128, a maximum speed up of 18×, 96×, 156× and 230× are observed respectively.

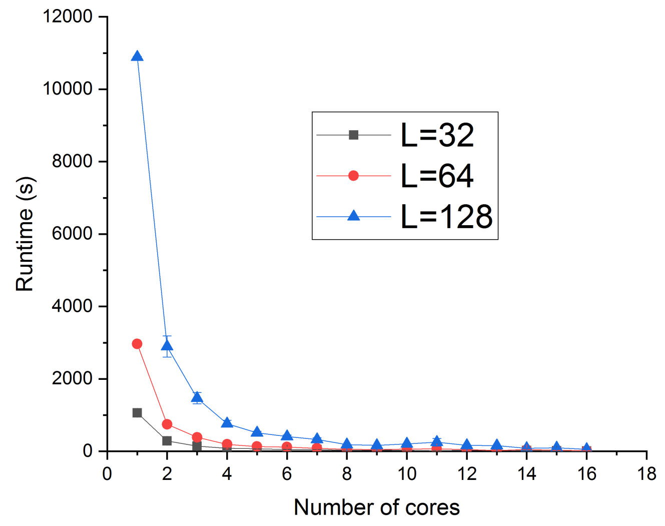

FIG. 18. Runtime against number of cores for various lattice sizes L using method 2 along with Numba, with parameters $(100, 5000, 10000, 4, 10)$. The speed up of multiple cores over a single core is illustrated more clearly however by FIG. 19.

expressed in terms of the spins before only, reducing the much unnecessary complexity of the calculation. It has also been demonstrated that values which do not change often can be precomputed and store in order to quickly retrieve it instead of computing it every time it is needed.

An additional improvement to the underlying algorithm would be to consider picking the sites in series instead of at random for every spin flips. Since flipping each one in order row by row allows for the complete control of fulfilling the ergodicity condition, since one would only have to flip it as many times as there are lattice sites on

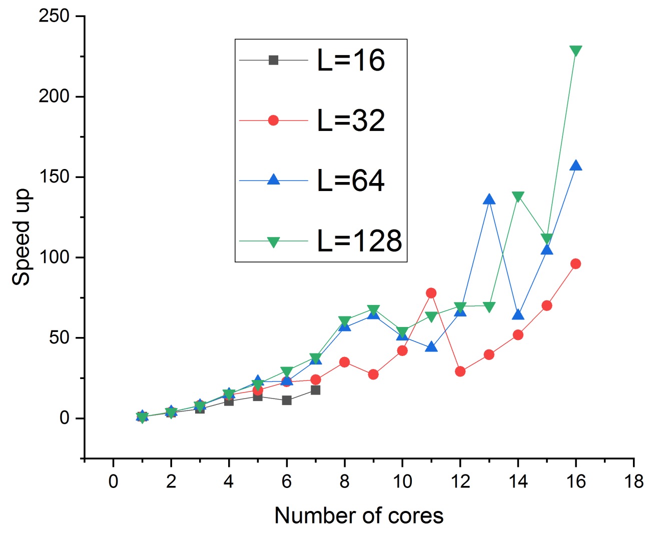

FIG. 19. The speed up of method 2 along with Numba with increasing cores up to 16 relative to single core. For $L =$ 32, 64 and 128, a maximum speed up of 96×, 150× and 220× are observed respectively.

a lattice. Whereas with the choice of randomly choosing spins to flip, ergodicity is only fulfilled over many more sweeps, and the exact amount of sweeps that is needed to be skipped is not an easy task to calculate, and hence any observation for any quantity of interest would be biased if it were done every sweep. This also affects the reproducibility of a specific simulation, since in order to reproduce the result of a particular run, one must set a different seed value for different temperatures and also different seed values for every sweep. Therefore the number of different seeds, which themselves are generated randomly, would scale with the number of temperature points and the number of sweeps that are done in total. A version of the base code with reproducible result is shown in `ising_reproducible.py`.

For domain decomposition, both method 1 and 2 having the same frequency of communications between processors. However, method 2 ended up being slightly faster over method 1 for the same simulation parameters at a larger lattice dimensions. This is because by simply shifting the domains for which each sublattice is swept, it does not require a shift in the values of the lattice, which can potentially act as a computational overhead for large lattice sizes.

[1] M. Newman and G. Barkema, *Monte Carlo Methods in Statistical Physics*. Clarendon Press, 1999.

[2] 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*, vol. 21, no. 6, pp. 1087–1092, 1953.

[3] K. Chan, C. Lenard, and T. Mills, “An introduction to markov chains,” 12 2012.

[4] L. Onsager, “Crystal statistics. i. a two-dimensional model with an order-disorder transition,” *Phys. Rev.*, vol. 65, pp. 117–149, Feb 1944.

[5] K. Binder and D. Heermann, *Monte Carlo Simulation in Statistical Physics : An Introduction*, vol. 80. 01 2010.

[6] 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, (New York, NY, USA), Association for Computing Machinery, 2015.

[7] K. Binder, “Finite size scaling analysis of ising model block distribution functions,” *Zeitschrift für Physik B Condensed Matter*, vol. 43, pp. 119–140, Jun 1981.