Parallelising Spin Models on Different Geometries

Filip Sosnowski

September 9, 2021

1 Abstract

2 Introduction

2.1 The Ising Model

The Ising model is mathematical model used to study the properties of a thermodynamic system. Named after Ernst Ising, it is one of the most important models of statistical mechanics as it helps us to study phase transitions, which are transitions of a physical system into a different state of matter. Classically, the Ising model deals with the ferromagnetic properties of a system, showing how a system's magnetic properties change with temperature. Studying these properties, alongside other observables, is one of the main areas of interest in my project.

In the Ising model the system is comprised of "particles" arranged in a lattice. These particles are then represented by their "spins" σ, which in this case can be either +1 or -1. These spins are either assigned randomly for a "hot start" or the lattice is given a "cold start", where all the spins on the lattice are +1. Neighbouring particles are then allowed to interact with each other, causing spins to get flipped randomly. These interactions are usually limited to the closest neighbours of each particles, which in the case of the standard square lattice are the four particles to the left, right, top and bottom.

Whenever a spin flips the system's observable properties change. The two four main observables in this system are magnetism, energy, magnetic susceptiblity and specific heat capacity. Given a fixed temperature, these properties tend to change constantly, but given enough particle interactions and enough spin flips they tend to settle at a constant value, signifying that the system is in equilibrium. By studying the properties of the system at equilibrium in a range of temperatures we can observe how each observable changes with temperature, allowing us to see phase transitions. These transitions can usually be seen from a sudden and relatively large change in the system's magnetisation and energy, which is also followed by a spike in magnetic susceptibility and specific heat capacity. The temperature at which this phase transition occurs is called the Curie temperature and is denoted by T_C.

2.2 Measuring Observables

The magnetism of the system in the Ising model is the easiest observable to calculate as it is just the sum of all the spins on the lattice. To make the results easier to understand, often the average magnetisation per site is taken. Furthermore, before reaching the critical Temperature, the magnetisation tends to be either +1 or -1 randomly, so to further make the results easier to read often the absolute value of the average magnetisation is taken. This leaves us with the following formula for magnetisation:

$$< M >= |\sum_{i}^{n} \frac{\sigma_i}{n}|$$

The energy of a physical system is usually measured using a Hamiltonian. A Hamiltonian is a function that describes a dynamic system in terms of components of momentum and space-time coordinates. When time isn't an explicit component of the function then the Hamiltonian is equal to the total energy of the system. In the case of the two dimensional Ising model the Hamiltonian is equal to the enrgy of the system as it is given by:

$$H = -2J \sum_{<i,j>} \sigma_i \sigma_j - B \sum_{i} \sigma_i$$

The 2J (where the 2 is usually added on by convention) in the Hamiltonian above is the coupling constant, which determines the strength of the force exerted in an interaction between particles. This is multiplied by the sum of the interactions of all neighbour particles, where the subscript ¡i,j¿ means that i and j are closest neighbours. The second term in the Hamiltonian is the effect of the external magnetic field B on the system. This is usually taken to be zero. This leaves us with the following formula for the average energy per site:

$$< E >= \frac{1}{2} < \sum_{i,j} H_{i,j} >$$

The result above has to multiplied by a half as all the site energies get double counted this way.

At low temperatures, all the spins on the grid tend to have the same spin as they in they are in their lowest energy configurations, ie. the ground state. From the equations above, and from the fact that in the square lattice each site has four nearest neighbours, can then be inferred that in the ground state the system is expected to have an average energy of -2J and an average magnetisation of 1 per site.

The magnetic susceptibility χ is dependent on the magnetisation, while the specific heat capacity C_V is dependent on the average energy of the system. Magnetic susceptibility is a measure of how much a system will become magnetised in a given magnetic field. Specific heat capacity is a measure of the amount of energy that needs to be added to one unit of mass in order to raise the temperature by one unit. The formulae for these obserables are:

$$\chi = \beta(< M^2 > - < M >^2)$$

$$C_V = \frac{\beta}{T}(< E^2 > - < E >^2)$$

2.3 Analytic solutions of the Ising model

In the one dimensional case, the Ising model is analytically solvable and experiences no phase transitions. Although difficult to calculate, there also exists an analytic solution for the two dimensional case for an infinitely large system. This solution predicts that the system will experience a phase transition at approximately $T_C = 2.27$. The analytic solution of this model predicts spontaneous magnetisation of the system at $T < T_C$, with an average magnetisation of 1. The solution also predicts that the phase transition shows a sudden continous drop towards 0 at T_C. Furthermore, magnetic susceptibiliy of this system shows divergence at T_C. This is a second order phase transition, meaning that the average magnetisation vanishes continuously. This kind of divergence only happens for an infinitely sized system however, and in the case of a finite system the phase transition is represented by a sudden spike.

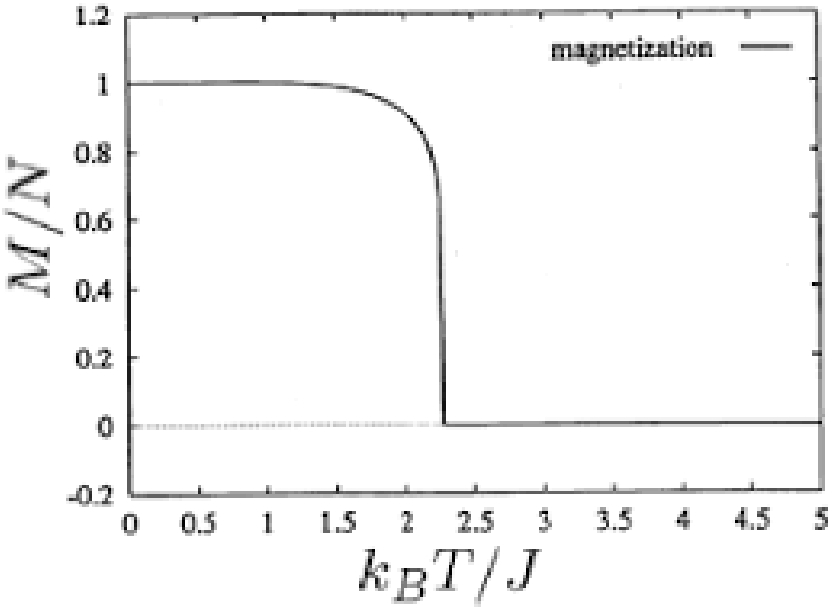


Figure 1: [1] Phase transition for the square grid Ising model

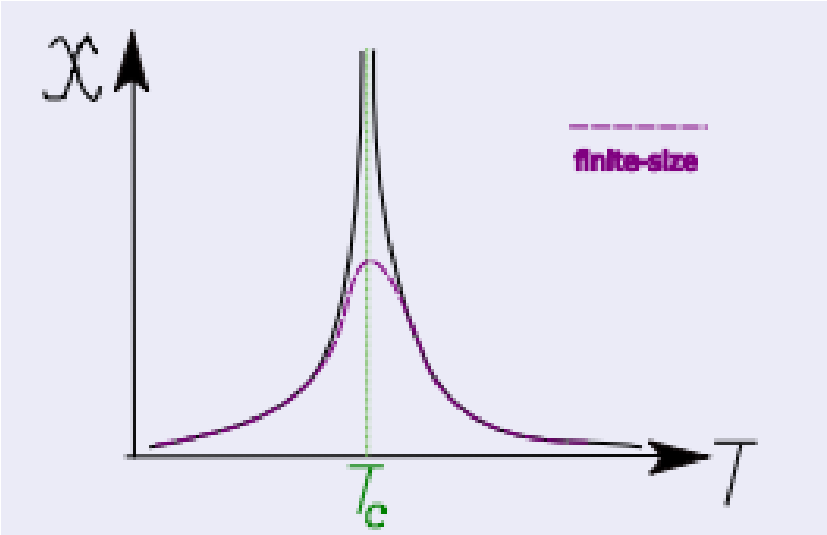


Figure 2: Phase transition in the Ising model shown by divergence of mangetic susceptibility

The average energy of the system experiences a similiar phase transition to the average magnetisation, rising from -2J to 0 at the Curie temperature. The specific heat also diverges at this point.

No analytic solutions exist for the three dimensional Ising model, or even for different geometries. This means that in order to study these systems it is necessary to perform Monte Carlo simulations.

2.4 Extending the Ising Model to Other Lattices

The Ising model can be extended to other geometries, retaining the same formulae for the observables. The only change needed in this case is the calculation of the closest neighbours. As this is a very simplified model that is not reflected by reality, the different angles between closest neighbours can be ignored. My main geometries of interest are be in triangular, hexagonal and cubic lattices. Over the course of this project I will be discussing the methods for finding closest neighbours in each geometry as well as the actual results of the Ising model simulations in each case.

2.5 Potts Model

The Potts model is a generalisation of the Ising model. Like the Ising model, it does not model real physical systems well, however it is helpful in ganing insight into the behaviour of ferromagnets as well as other areas of solid state physics.

The Potts model consits of spins arranged on a lattice, which are then allowed to interact with each other just like in the Ising model. The crucial difference

in this case is that there can now be more than two different spins. These spins take positive values in the set $\{1, ..., q\}$, where q is the highest possible spin. This is often called the q-state Potts model.

Another important difference between the Ising model and the Potts model is how the obervables are calculated. The Hamiltonian for this system is now given by:

$$H = -J \sum_{,i,j.} \delta(\sigma_i, \sigma_j)$$

Where the δ stands for the delta Kronecker function.

The formulae for measuring the other observables remains unchanged.

It is easy to see that the 2-state Potts model is equivalent to the classic Ising model - although it has different numerical values, it still experiences phase transitions at the same temperatures.

I will be applying the Potts model to all the geometries above and studying the difference in behaviours for a number of different spins.

2.6 Autocorrelation

In mathematics, correlation is a statistical relationship between two random variables. Autocorrelation is a "mathematical representation of the degree of similarity between a given time series and a lagged version of itself over successive time intervals" [].

In Monte Carlo Markov Chain simulations it is uaullt necessary to obtain a large number of samples in order to obtain accurate results, so it is crucial the the correlations between the samples is as small as possible. Autocorrelation is important for these kind of MCMC simulations because it provides the correlations between the consectutive steps.

For a discrete random variable X_t with N measurements labelled from 0 to N-1 the correlation function is given by[]:

$$c(t) = \frac{1}{N-t} \sum_{i=0}^{N-t-1} X_{i+t}(X_i - \frac{1}{N-t} \sum_{j=0}^{N-t-1} X_j)$$

3 Procedure

3.1 The Metropolis Algorithm

The Metropolis Algorithm is one of the most widely used Monte Carlo simulations for simulating the Ising model. The metropolis algorithm generates a collection of states according to some desired distribution P(x), which in this case is the distribution of equilibrium states in a range of given temperatures. To accomplish this, the Metropolis algorithm employs a Markov process which asymptotically reaches a unique stationary distribution $\pi(x)$, which satisfies $\pi(x) = P(x)$.

To ensure that this distribution is reached by an algorithm, the criterion of detailed balance is sufficient but not necessary. This criterion requires that a state transition from state x to stater x' must be reversible, ie. for two states x and x' the probability of the transition from x to x' is the same as the probability of the transition from x' to x. Given a transition distribution $T(x'|x)$, detailed balance is formally given as:

$$T(x'|x)P(x) = T(x|x')P(x')$$

The Metropolis Algorithm fulfills this criterion.

The uniqueness of the stationary distribution is ensured by the ergodicity of the Markov process. Informally, ergodicity means that eventually the system will take on all possible states, in a uniform and random sense. Ergodicity of a Markov process also requires that every state occurs aperiodically, but is expected to occur again in a finite number of steps.

As the Metropolis Algorithm fulfills the two criterions above, it is a good algorithm to employ in simulating the Ising model. A step by step implementation of this algorithm is as follows:

1. Initiate an array to represent the chosen lattice and assign each point with either +1 or -1. I tested my algorithm for both a hot start and a cold start.

2. The two dimensional Ising model is assumed to be on an infinitely sized grid. As this is not feasible, and using a very large matrix is inefficient, it is a better idea to employ periodic boundary conditions. This means using the modulo operator for calculating each particle's closest neighbours so that the opposite ends of the matrix interact with each other, mimmicking an infinitely sized system. For example, on an NxN matrix a spin on the point (N,2) would have the bottom neighbour ((N+1)mod(N),2) = (1,2).

3. Once an initial configuration of the matrix is chosen the Metropolis sweep can be performed. The sweep can be performed by succesively checking each matrix entry or by selecting a site at random. Once a site is chosen, the difference in energy difference ΔE if the spin was to be flipped is calculated. if $\Delta E \leq 0$, the spin is flipped. If $\Delta > 0$, a uniformly distributed random variable r in the interval [0,1] is generated and the spin is flipped only if $r < exp(-\beta \Delta E)$ where $beta = \frac{1}{K_b T}$, with K_b being the Boltzmann cosntant and T the temperature. As the Boltzmann constant is extremely small, to make this step more computationally friendly I assimilated the Boltzmann constant into T so that the it is checked that $r < exp(-\frac{\Delta E}{T})$, and T is not measured in terms of the constant.

4. After each sweep the matrix is updated with the new spins at each point in the lattice. The sweeps are repeated until a stopping criterion is reached, eg. after a required number of iterations.

5. Finally, once all the Metropolis sweeps are done the observable properties are calculated. The matrix is then restarted to an initial configuration so that the process can be repeated for another temperature.

3.2 Serial Ising Square Grid Model

In developing my code I began by writing a simple simulation of the two dimensional square lattice Ising model. For this I created a 50x50 statically allocated square array and randomly assigned each site with either +1 or -1. I initially decided to use statically allocated arrays as the array sizes are relatively small so storing them on the stack should make the code run faster, because it is very likely to always sit in the cache.

Next, I had to modify the metropolis algorithm so that it is more suited to computation. I removed the Boltzmann constant from the step which involves generating a random number, as it was causing the probability of a flip occuring to be almost impossible which in turn made the results wrong. Another issue I encountered was that the start of the grid was not communicating properly with the end of the grid during the sweeps. This issue appears to have been caused by the % operator, as it operator does not function like the usual mathematical modulo operator, in the sense that $-1\%(m)$ does not return m-1 but rather -1. To fix this issue I wrote my own modulo function, which adds +m to the % operation if the result is negative. Of course this wouldn't work for any numbers smaller than -m, but in this case it is of no concern as I only needed the function to deal with $-1mod(m)$ in terms of negative modulos.

In order to make each sweep of the lattice more computationally efficient, I precalculated the inverse of the temperature at the each temperature change. This is because division is 3-6 times slower to compute than multiplication, and this would have a significant effect when performing 1000 sweeps for each temperature.

I ran the simulation 10 times to obtain an average for each observable at each temperature, with the temperature ranging from 0 to 5 in increments of 0.05.

To measure the average magnetisation per site I decided to make my results be give the absolute value of the average magnetisation, as this made the graph of the results easier to read. I also computed the average energy per site, as well as the magnetic susceptibility and specific heat. In order to calculate the latter two I had to write additional functions that computed the magnetisation squared and the enrgy squared per site.

3.3 Parallelising the Ising Model

3.4 Ghost Cell Exchange - 1D Decomposition of The 2D Ising Model

In order to parallelise the code I decided to employ ghost cell exchange. In this procedure the matrix is divided between the processes, and the sweeps are performed by each process on their respctive part of the grid. The relevant information is then shared between the processes ebtween each sweep.

As reading arrays along the columns is faster than along the rows I decided to divide the lattice along the rows - this is due to contiguous memory being

read faster, and statically allocated arrays are stored row-wise. To do this I wrote a function that divides the number of rows between the processes so that they differ by at most one row, which ensures that the grid is divided as evenly as possible. Next, I reworked my Metropolis algorithm so that each process only performs the sweep on its assigned part of the grid.

In each Metropolis sweep every site must know the state of all of its nearest neighbours. This implies that the sites at the top and bottom of each partition of the grid must have a way of knowing the states of the closest sites in the neighbouring processses, which is where ghost row exchange comes in. To allow the exchange of the necessary information between neighbouring processes additional ghost or "halo" rows are assigned to the each boundary between the processes. These ghost rows store the states of each site in the closest row of each neighbouring process. This provides the information required to perform a Metropolis sweep along each grid partition. However, as the state of each site can change between each consecutive sweep, the ghost rows must be exchanged between every sweep so that the sweeps are performed with up to date information.

Open MPI provides the best library for the above task. Using MPI_Cart, I was first able to compute which processes were neighbouring which. Then, once the grid was divided between the processes, I was able to use MPI_Isend and MPI_Irecv to send the ghost rows between the processes. I opted to use non-blocking calls as they are generally faster than their blocking counterparts.

Once all the sweeps were complete I had each process compute the relevant observables on their portion of the grid, then used MPI_Reduce to bring the sums from each process to the master process in order to compute the global averages.

An issue that comes with the approach above is that it is no longer possible to generate the same random matrix for each process between each temperature change as each process uses a different random number generator seed. However an easy workaround to this is to just generate a different random matrix on each process, then exchange the ghost rows. The other parts of the matrix are not relevant in this case as they will not be sweeped, and the information needed between the processes for the sweeps to be correct is already provided.

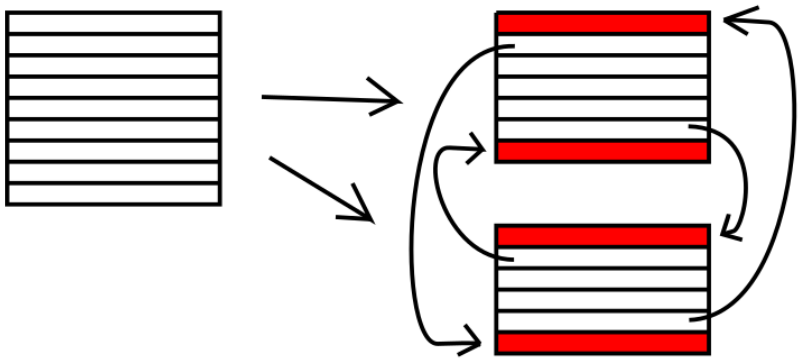


Figure 3: Example of one dimensional decomposition between two processes. An array of 9 rows is split is into 5 rows for process 1 and 4 rows for process 2, with the red columns signifying the ghost columns received from the other process. The arrows show the direction and destination of the row exchanges.

3.5 Two Dimensional Decomposition of The 2D Ising Model

Another way to parallelise this model is to use both ghost row and ghost column exchange, decomposing the array along both dimensions. In this kind of decomposition the array is no longer divided up into conitguous memory slices, but rather into rectangular blocks that are potentially spaced out in memory.

The first issue that I had to take into account for this kind of decomposition was how the memory is arranged in my matrix. As my array is statically allocated, it consits of m rows of n elements contiguously allocated side by side. This means that each column in this array has its elements spaced n numbers apart. As this forms a vector of equally spaced blocks of numbers, the easiest way to access information in this format is to use the MPI function MPI_Type_vector. This function is perfect as it creates a vector of blocks of a datatype which are spaced apart by some length of a particular datatype. In this case the blocks are made up of 1 integer spaced apart by N integers. Creating this new MPI datatype allowed for simple sending and receiving of data.

Another issue that I had to take into consideration was how processes are split up along the two dimensions of the array. I wanted to ensure that the processes are split up efficiently among the available dimensions. A very useful function in this endevour was MPI_Dims_create as it creates a division of processors in a cartesian grid. Unfortunately this function comes with the constraint that it only creates a two dimensional decomposition among processors when the number of processors is even. In the case that the number of processors is odd the decomposition is only one dimensional. Using this function also required me to adjust the neighbours of the processes along the row edges of the grid, to ensure that the left and right neighbours processes were in the same row.

Finally, I had to ensure that the the starting positions of each subarray as well as the end points were assigned correctly. To do this I used the same one dimensional decomposition function that I used for row decomposition, but this time I used it once along the rows and once along the columns. However, this came with the issue of calculating each processes' position along the rows and columns of the decomposition. To do this I used the following formulae:

$$\text{row position} = (\text{process rank})(mod)(\text{number of porcesses along each row})$$

$$\text{column position} = \frac{\text{(process rank) - (row position)}}{\text{(number of processors along each row)}}$$

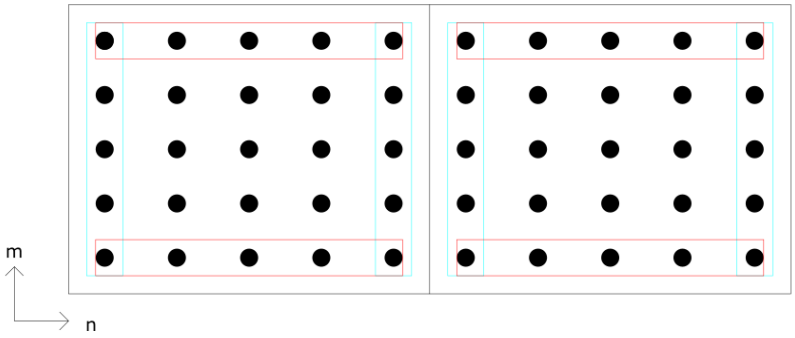


Figure 4: Two dimensional decomposition between two neighbouring processes. The rows in the red box show signify the rows to the neighbouring processes to the top and bottom of the process, while the columns in the blue boxes show the columns shared with neighbouring processes to the left and right of the process.

3.6 Triangular Ising Model

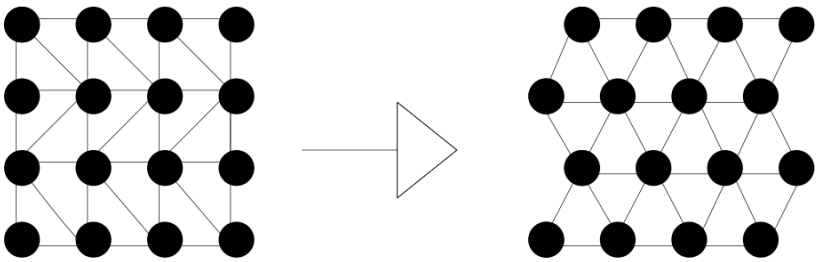


Figure 5: Mapping of a triangular lattice to a square lattice.

To find the nearest neighbours in the triangular Ising model I drew the triangular lattice first then mapped it to a normal square grid. From this I was able to deduce the nearest neigbours. As can be seen, each site on the lattice has six neighbours, the four same ones as on the 2-D square grid and two additional ones. These two additional neighbours differ for even and odd rows, going forwards for even rows and backwards for odd rows. This gives the following neighbours:

g[i-1][j]
g[i+1][j]
g[i][j-1]
g[i][j+1]
g[i+1][j+(-1)^i]
g[i-1][j+(-1)^i]

After finding the nearest neighbours above, extending the model to this geometry was only a matter of changing the Metropolis algorithm function as well as the functions to calculate the average energy of the system. I was then able to run the program as normal.

3.7 Hexagonal Ising Model

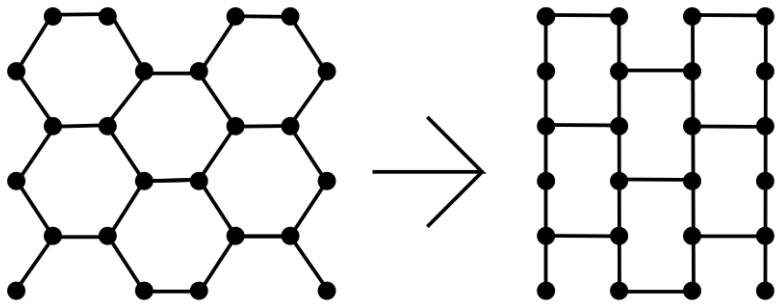


Figure 6: Mapping of a hexagonal lattice to a square lattice.

To find the nearest neighbours for each site in the hexagonal lattice I once again drew it and mapped it to a square lattice manually. This allowed me to easily find the nearest neighbours for this kind of lattice, especially seeing as each site has only three nearest neighbours:

g[i+1][j]
g[i-1][j]
g[i][j+(-1)$^{i+j}$]

The third nearest neighbour can be deduced from the fact that it alternates between left and right not only for every second element in each row, but also along each column.

A challenge that this system proposes is having the right number of elements such that the sites along the edges of the grid communicate with the other edge properly, as to mimmick the behaviour of an infinite lattice properly. If the grid was to have its nearest neighbours assigned as above, then I believe that the number of columns would have to be a multiple of 4 and the number of rows would have to be a multiple of 2. As can be seen from the graph above, this allows the edges of the lattice to sync up perfectly with the opposite edge. However as my test were all done on a 100x100 lattice this is not a problem.

Having once again adjust the Metropolis algorithm and energy functions, I was able to test my code as before.

3.8 Parallelising the Triangular and Hexagonal Ising models

Parallelising the hexagonal model above was no different than parallelising the square lattice. This is because the geometries were already mapped to a square lattice, and the hexagonal model required the same neighbours as the square lattice. This allowed me to use the same techniques and functions that I used to the square lattice case.

Parallelising the triangular lattice required some additional work. This was due to the sites on the triangular lattice requiring the states of its diagonal neighbours. In the four corners of each subarray this information is not passed on by the regular column and row exchanges, as it doesn't belong to any of the neighbour processes to the left, right, top or bottom. These exchanges leave a missing integer in each corner of the halo surround the subarray.

To fix the above issue of missing information, I first had to find the four neighbouring processes that lie diagonally to each process. The simplest and quickest way that I found to do this was to simply find the top and bottom neighbours of the neighbouring processes to the left and to the right. The only thing that remained to do then was to write four trivial exchanges of the cornering integers in each subarray to the neighbouring diagonal neighbours.

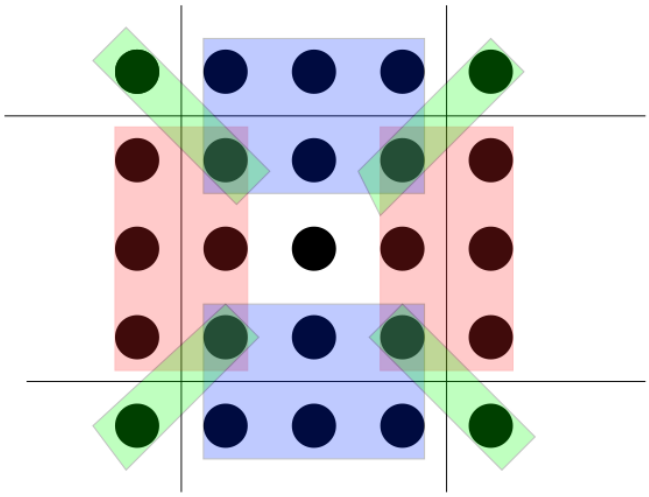


Figure 7: Cell exchanges for a sublattice in the triangular model. The red areas show exchanges of columns, the blue areas show row exchanges, while the green areas show diagonal exchanges.

3.9 Cubic Ising Model

The process of finding the nearest neighbours for each site in the cubic Ising model was quite a trivial one as it very much resembles the square grid model. Nevertheless, the nearest neighbours in this model are given by:

g[i-1][j][k]
g[i+1][j][k]
g[i][j-1][k]
g[i][j+1][k]
g[i][j][k-1]
g[i][j][k+1]

Extending the model to three dimensions required some trivial tweaks to accomodate a three dimensional array. Initially, I ran my simulations on a 10x10x10 grid.

3.10 Parallelising the Cubic Ising Model

In C, a statically allocated three dimensional array g[i][j][k] stores its information so that the k-axis is contiguous along the j-axis, and these two-dimensional slices are then stored contiguously along the i-axis. This means that to parallelise the three dimensional Ising model it is most efficient to decompose the array along the i-axis, and to exchange information using j-k ghost slices.

3.11 2D decomposition of the Cubic Ising Model

Decomposing the array in 3 dimensions was done somewhat similarly to the 2 dimensions, with a few changes due to the different memory layout.

By ignoring the third dimension I was able to divide the cube among the processes the same way that I divided the processes in the 2 dimensional array case. This allowed me to divide the array into rectangular cuboids which all had depth z. The ghost exchange was then employed along four two-dimensional slices along the i-axis and the j-axis.

The achieved decomposition had the added benefit that the slices along the i-axis were still contiguous, however the the slices along the j-axis were not. The consecutive elements along the j-axis different by 1 along the i-axis, which corresponded to the elements by spaced y*z integers apart in the contiguous memory. As this is a reqgular spacing between the elements I was able to once again create and MPI vector, of block size 1 and spaced y*z integers apart, to help send the j-axis slices to the relevant neighbours.

3.12 3D decomposition of the Cubic Ising Model

The three dimensional decomposition of the 3D array is relatively more complicated than the two dimensional decomposition, largely due to the fact that most of the slices are no longer contiguous. The cube in this decomposition consits of smaller cuboids, with potentially none being the whole length of any of the dimensions.

Like last time, the first thing that I considered is the spacing of each ghost slice in the memory. To make the explanations easier lets assume that for a given sub-cuboid the starting indices are given by (s,s2,s3) and the ending indices given (e,e2,e3).

Starting with the slice along the i-axis, the slices I wanted to send to the neighbouring processes lie between (s,s2,s3) and (s,e2,e3), as well as between (e,s2,s3) and (e,e2,e3) respectively. These form slices at the "top" and "bottom" of the cuboid. These slices consist of (e2-s2+1) contiguous vectors of length (e3-s3+1) along the k-axis, with the space between the start of each vector being z integers apart along the contiguous memory. Due to the regular spacing between these vector I was able to create those slices using an MPI vector of (e2-s2+1) blocks, with block size (e3-s3+1) and a stride of z.

The slices along the j-axis were obtained similiarly. The slices that I wanted to send the neighbouring processes this time lied between (s,s2,s3) and (e,s2,e3) as well as between (s,e2,s3) and (e,e2,e3). These formed slices at the "left" and "right" sides of the cuboid. This time the slices consisted of (e-s+1) contiguous vectors of length (e3-s3+1) each along the k-axis. The spacing between these vectors was y*z integers. Once again, using the regular spacing between these vectors I was able to create the slice using an MPI vector of (e-s+1) blocks, with block size(e3-s3+1) and a stride of y*z.

Obtaining the ghost slice along the k-axis proved to be dissimilar to the other two axes. Along this axis the slices that I wanted to send were between

(s,s2,s3) and (e,e2,s3), as well as (s,s2,e3) and (e,e2,e3). These slices are at the "front" and "back" of the cuboid. These slices now consited of singular integers which were spaced z integers apart along the k-axis. These integers form a slice of row length (e2-s2+1) and column length (e-s+1). The issue with this kind of arrangement of array elements is that there is now irregular spacing between each element of this cuboid slice, which means that it is not possible to form an MPI vector to use for exchanges between neighbours.

To circumvent the above issue I decided to form MPI subarrays containing the above k-axis slices. MPI subarray is a datatype that lets you specify a subarray of a specified size along each of the array's axes, along with starting points. In this case I had to specify four different subarrays - two for the ghost slices that were to be sent, and two for the ghost slices that were to be recieved, as these were all in four separate places in the array. Doing so allowed me to send and receive the k-axis slices during ghost slice exchanges.

To divide the array across the processors I once again used MPI_Dims_create to divive the number of processes across all three dimensions as equally as possible. This gave me the dimensions of the number of processes along each axis. I calculated each processes' position along the three axes by deriving the following formulae:

$$\text{i-axis position } = \text{rank}(mod)(\text{number of processes along i-axis})$$

$$\text{j-axis position} = \left(\frac{\text{rank - (rank(mod)(number of processes along j-axis))}}{\text{number of processes along j-axis}}\right)(mod)(\text{number of processes along j-axis})$$

$$\text{k-axis position} = \frac{\text{rank - (rank(mod)(number of processes along j-axis x number of processes along i-axis))}}{\text{number of processes along j-axis x number of processes along i-axis}}$$

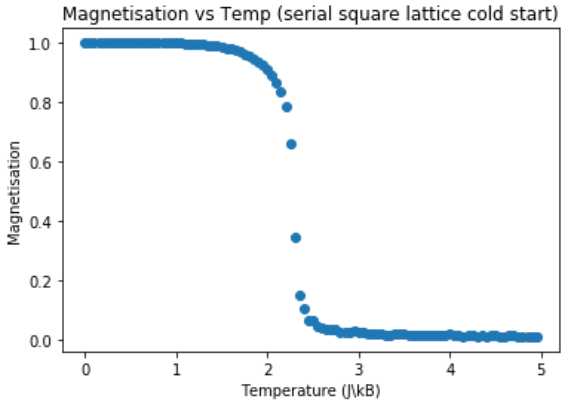
3.13 Potts Model

Changing the code into the Potts model required quite a few major changes. First, I changed the way the grids are initiated so that for a cold start in a q-state model all the spins are assigned the spin q, and for a hot start all the spins are assigned any spin between 1 and s. I also had to change the way energy is calculated across the grid.

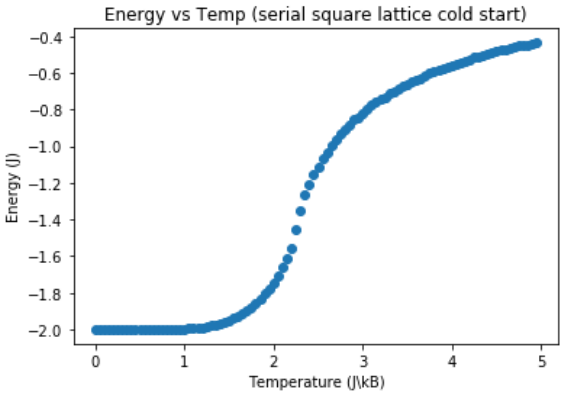
Parallelising the Potts model was the same as as parallelising the Ising model.

4 Results

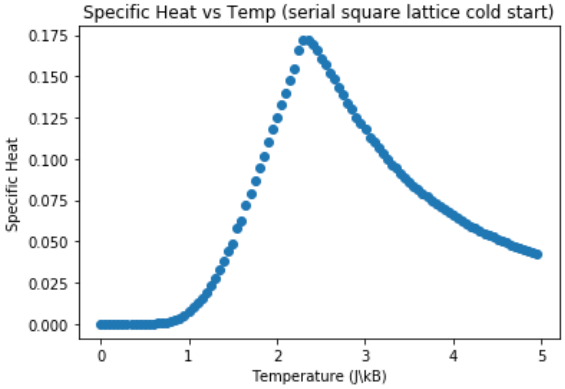
4.1 2D Square Grid Sequential Ising Model



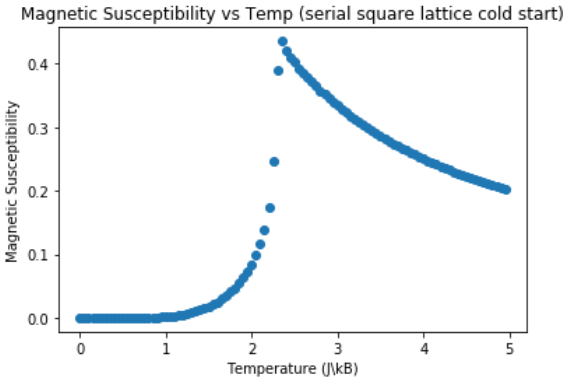
(a) fig 1



(b) fig 2



(c) fig 3



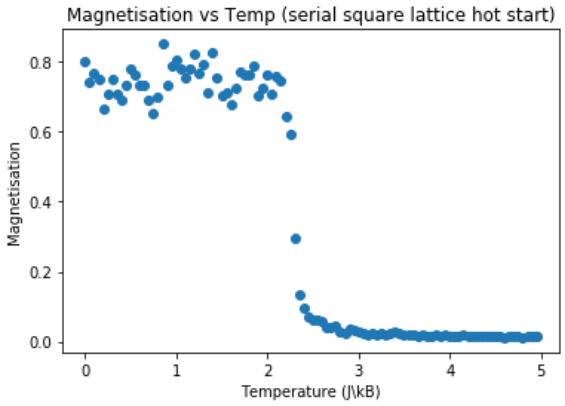
(d) fig 4

Figure 8: Average of 50 simulations on a 100x100 grid, running 1000 metropolis sweeps on each temperature with 100 equally-spaced temperature samples between 0 and 5.

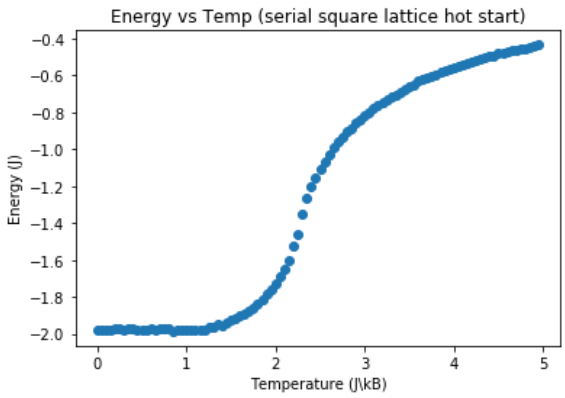
I first ran my simulations on a cold start. As predicted by the analytic solution, the two dimensional square lattice Ising model expieriences a phasse transition at $T_c \approx 2.27 \frac{J}{K_b}$. The observables all follow expected behaviour, with magnetisation sharply dropping to zero at around the critical Temperature, while the energy sharply rises from its ground state towards zero. Specific heat and magnetic susceptibility also show an expected spike at T_c, which is analogous to the spike towards infinity in an infinitely sized system.

My next test involved running the same simulation on a hot start. The

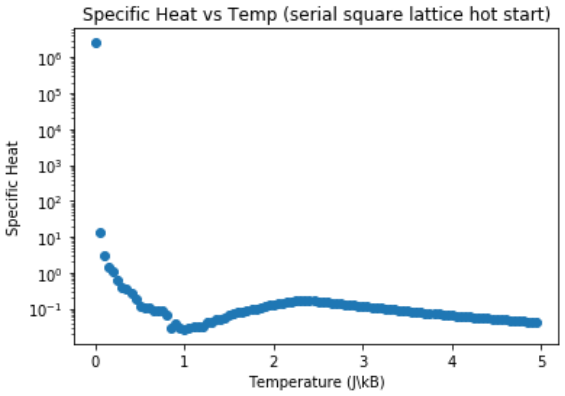
results were now much different than the analytic results. Although there is a sharp decrease in magnetisation at around the same T_c as for the cold start, there appear to be relatively high flactuations magnetisation before this temperature, while the energy graph remains almost indistinguishable. The specific heat and magnetic suceptibility now show much different behaviour, with both having extremely large spikes near zero. The specific heat drops extremely fast before showing behaviour consistent with the cold start at $T \approx 1$, and once again spiking around T_c. Magnetic susceptiblity exhibits also a very sharp decrease, however in this case the behaviour only becomes consistent with the cold start at around T_c.



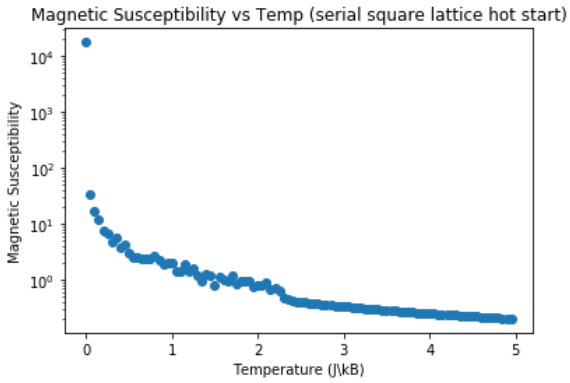
(a) fig 1



(b) fig 2



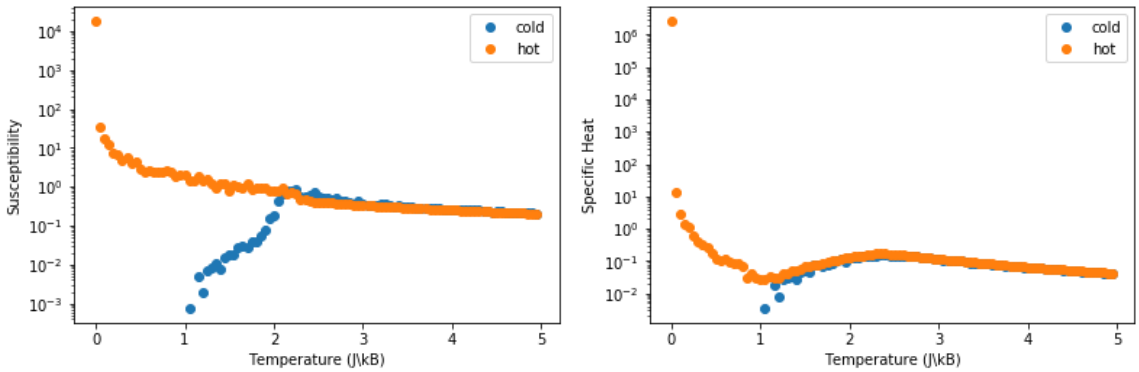
(c) fig 3



(d) fig 4

Figure 9: Average of 50 simulations on a 100x100 grid, running 1000 metropolis sweeps on each temperature with 100 equally-spaced temperature samples between 0 and 5.

The large distinction between the cold start and the hot start is very ap-



(a) fig 1

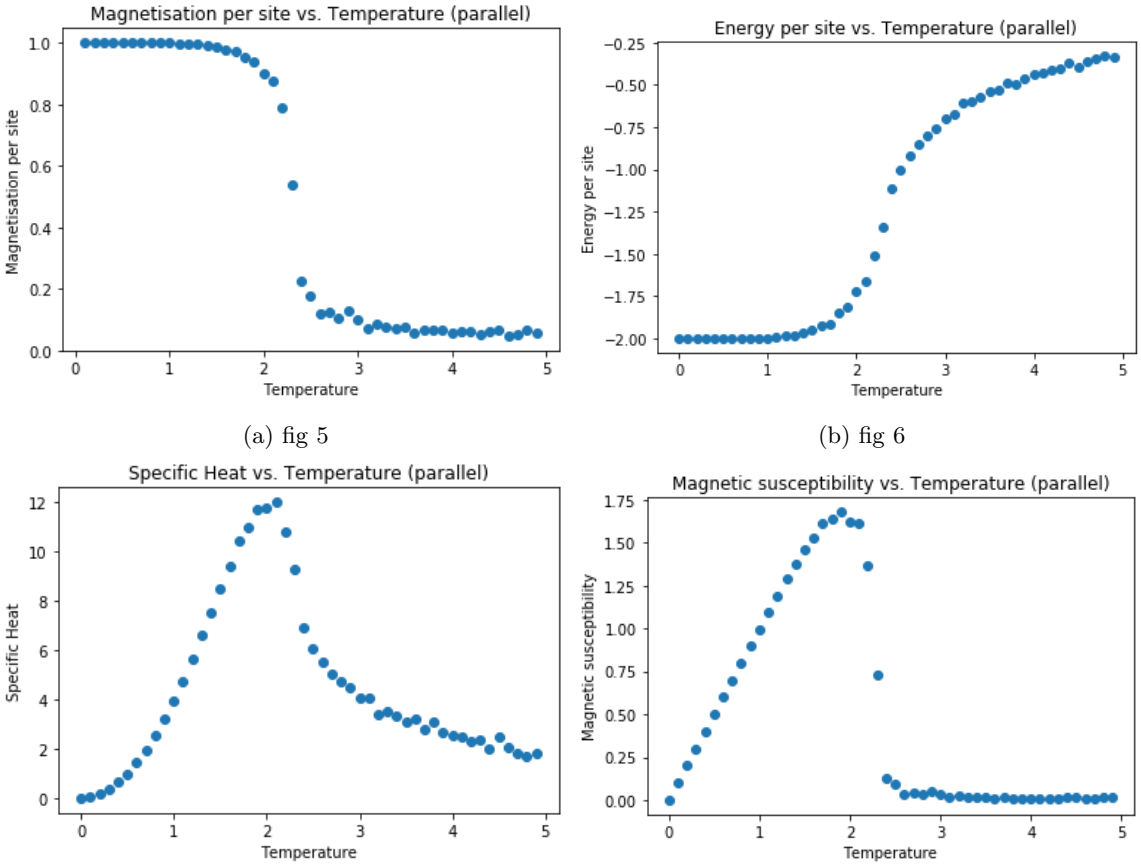
(b) fig 2

Figure 10: Comparison of Magnetic Susceptibility and Specific Heat for both the cold start and the hot start. The two starts show extremely different behaviour near T=0, however they eventually settle into same behaviour.

parent. Whereas I am unsure as to what is causing the differences near T=0 in the magnetic suceptibility and the specific heat, I believe the inconsistency in the average magnetisation before the Curie temperature can be explained via the existence of metastable states. These states would be energetically stable enough so that they would remain unchanged after an extremely large amount of Metropolis sweeps despite not being the system's equilibrium at a given temperature.

To compare these states with the equilibrium states at $T < T_c$, I reran my hot start simulations and I saved the system configurations for states with magnetisations $M << 1$. I also saved some equilibrium states for $T \approx T_c$ and $T >> T_c$ for comparison.

4.2 2D Square Grid Parallel Ising Model



(a) fig 5

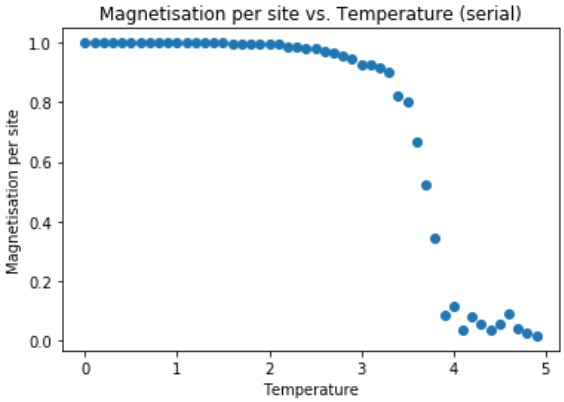
(b) fig 6

(c) fig 7

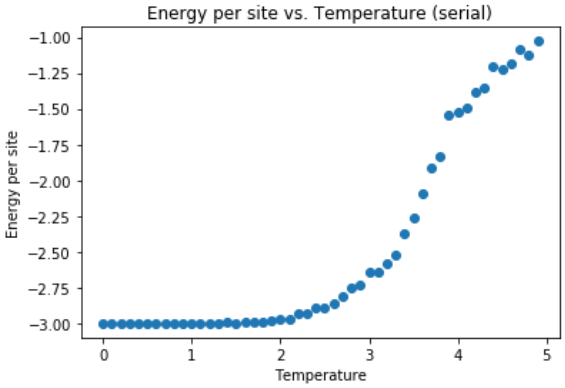
(d) fig 8

Figure 11: Add your own figures before compiling

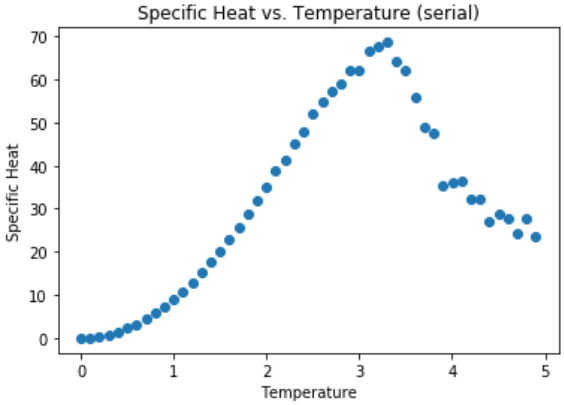
4.3 2D Triangular Grid Sequential Ising Model



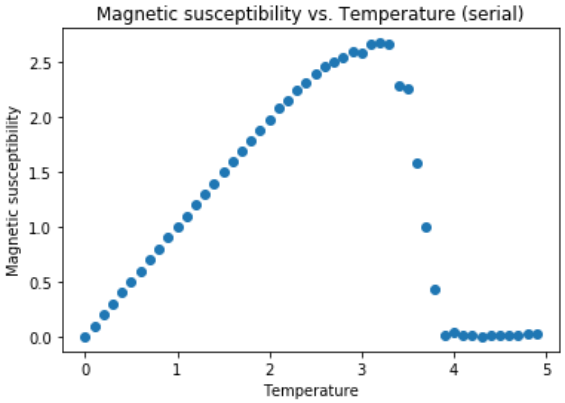
(a) fig 9



(b) fig 10



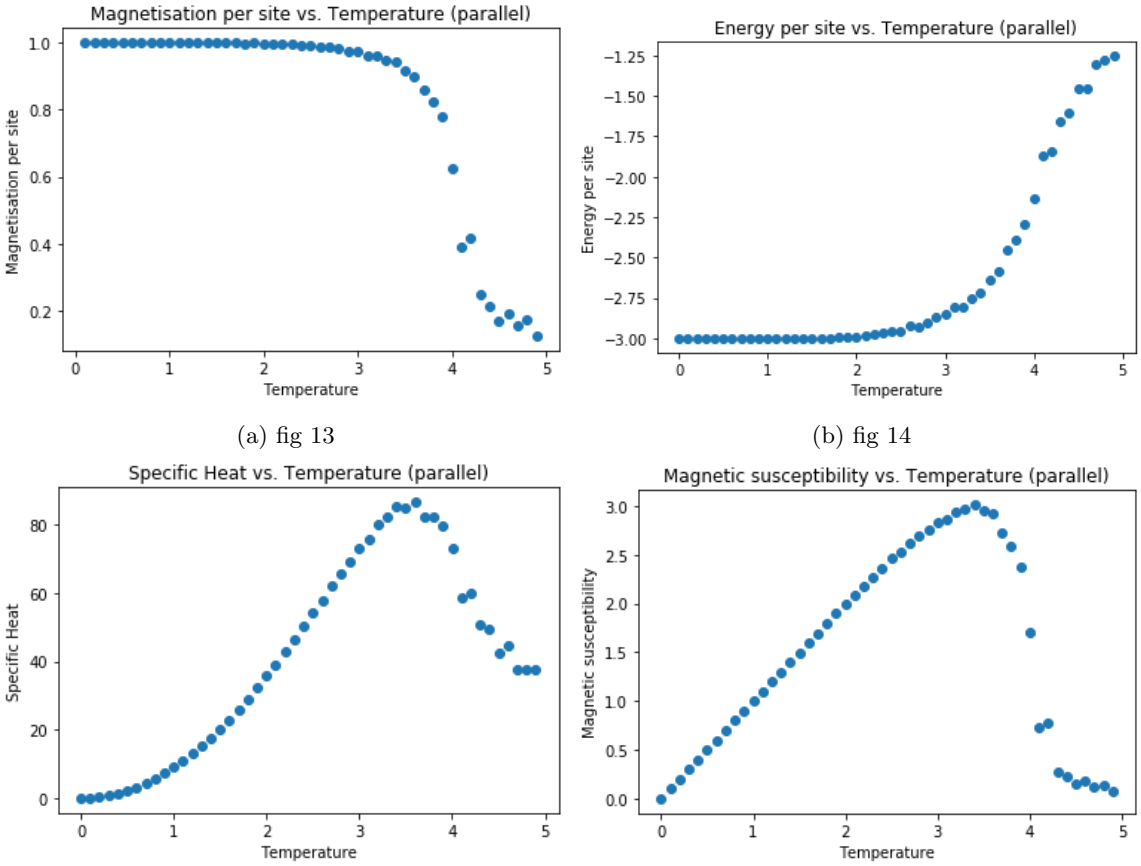
(c) fig 11



(d) fig 12

Figure 12: Add your own figures before compiling

4.4 2D Triangular Grid Parallel Ising Model



(a) fig 13

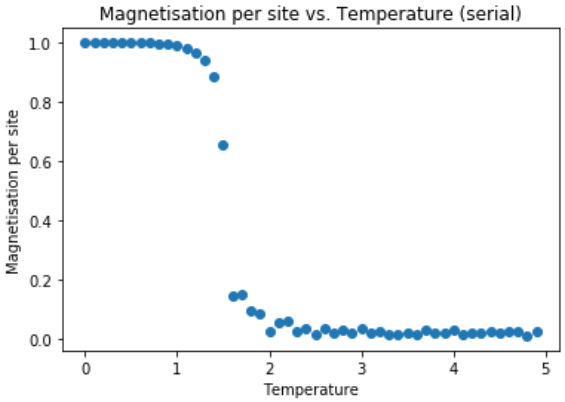
(b) fig 14

(c) fig 15

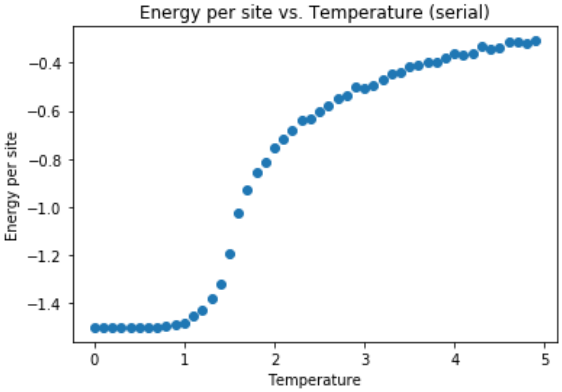
(d) fig 16

Figure 13: Add your own figures before compiling

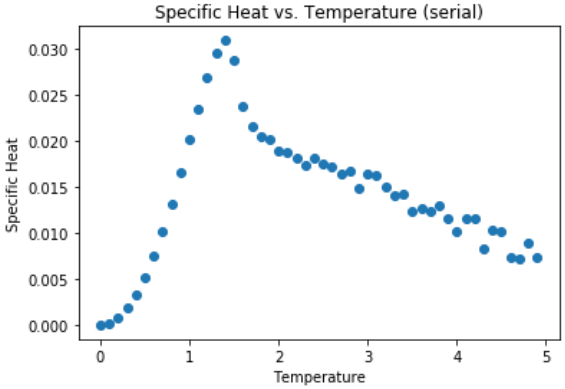
4.5 2D Hexagonal Grid Sequential Ising Model



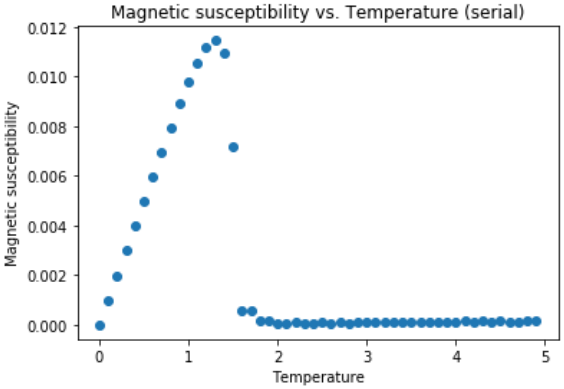
(a) fig 9



(b) fig 10



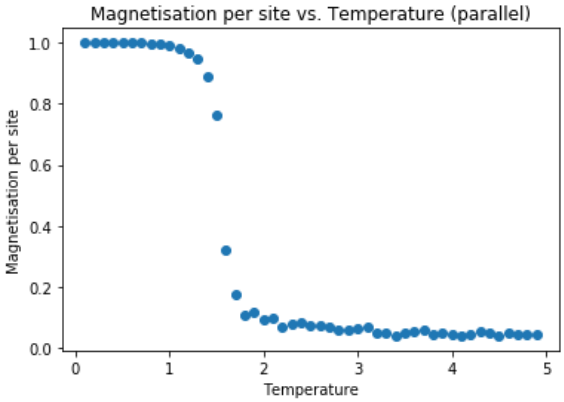
(c) fig 11



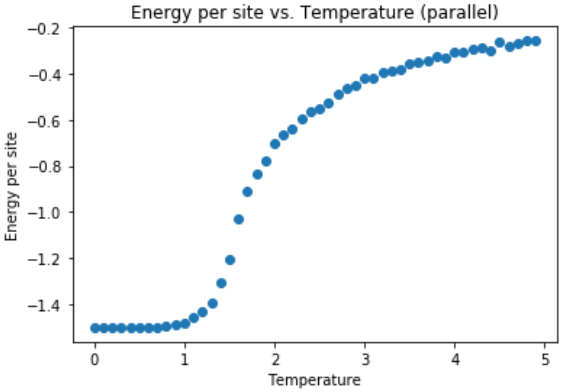
(d) fig 12

Figure 14: Add your own figures before compiling

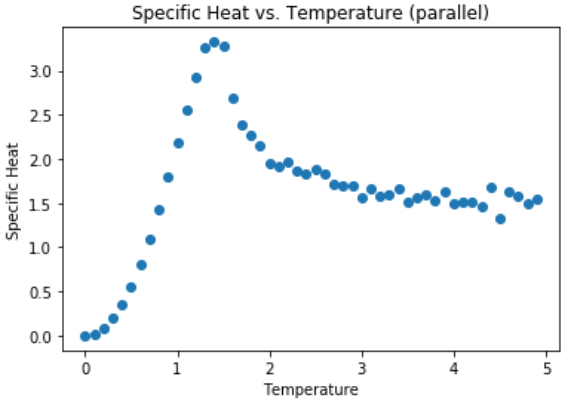
4.6 2D Hexagonal Grid Parallel Ising Model



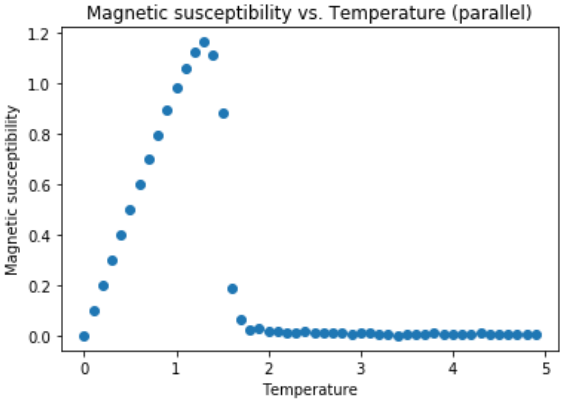
(a) fig 13



(b) fig 14



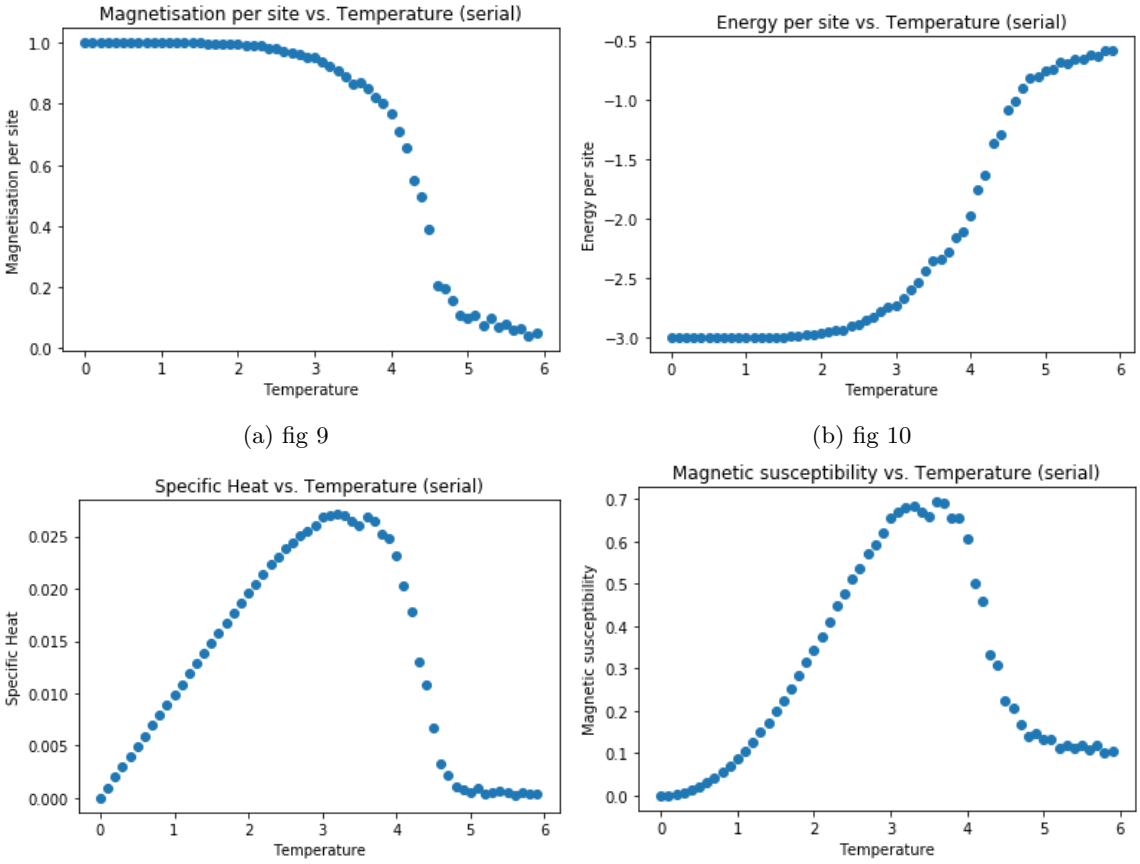
(c) fig 15



(d) fig 16

Figure 15: Add your own figures before compiling

4.7 3D Cubic Grid Sequential Ising Model



(a) fig 9

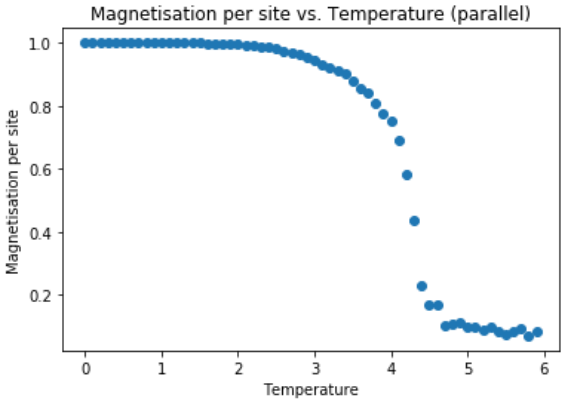
(b) fig 10

(c) fig 11

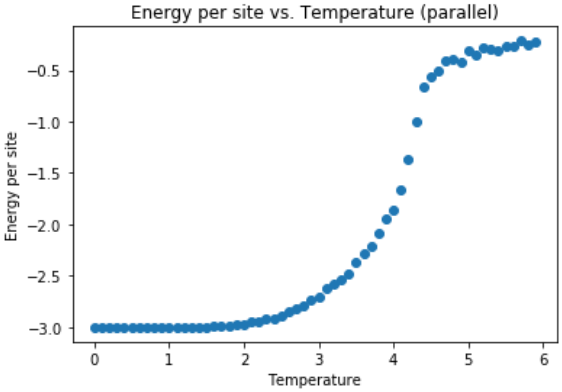
(d) fig 12

Figure 16: Add your own figures before compiling

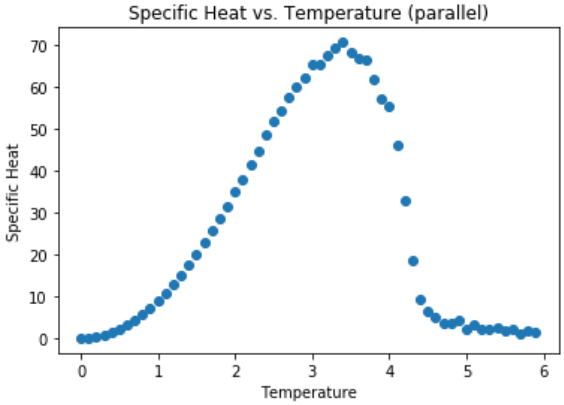
4.8 3D cubic Grid Parallel Ising Model



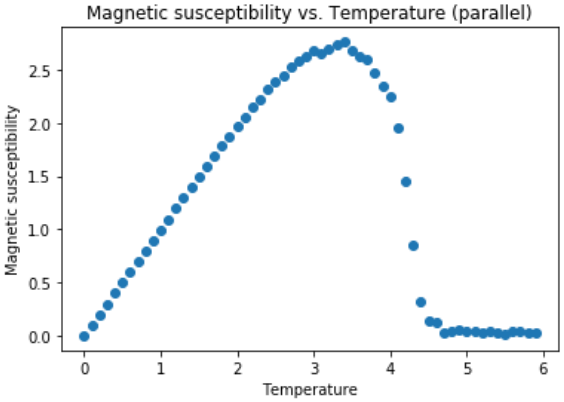
(a) fig 13



(b) fig 14



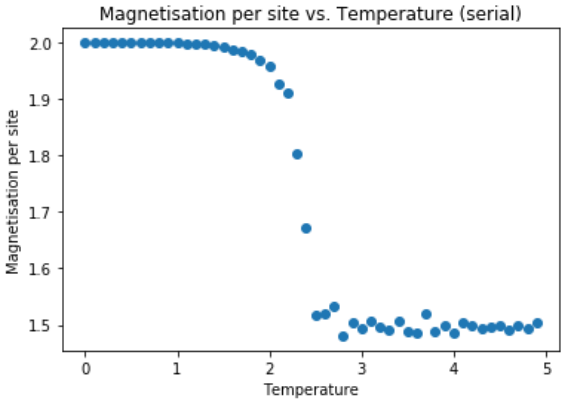
(c) fig 15



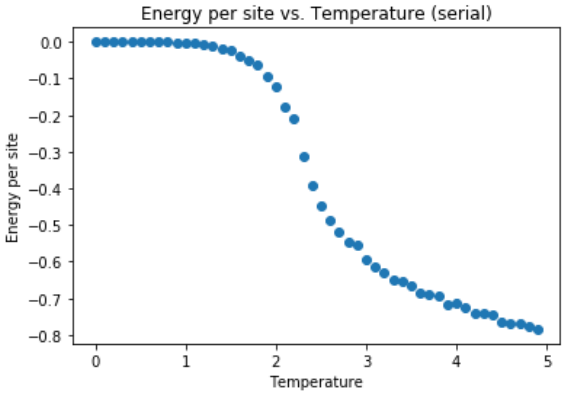
(d) fig 16

Figure 17: Add your own figures before compiling

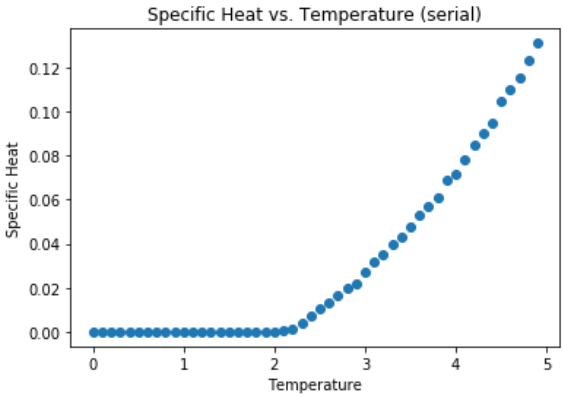
4.9 2D Potts Square Grid Model (2 spins)



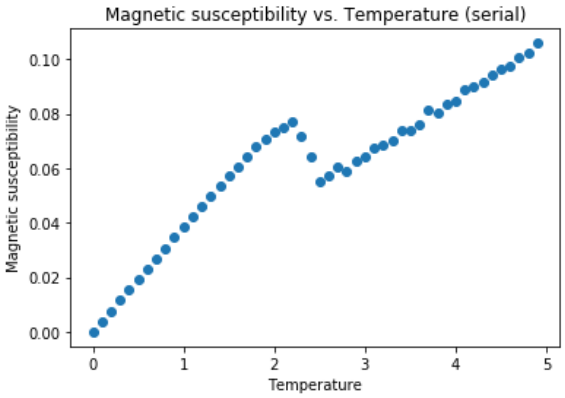
(a) fig 1



(b) fig 2



(c) fig 3

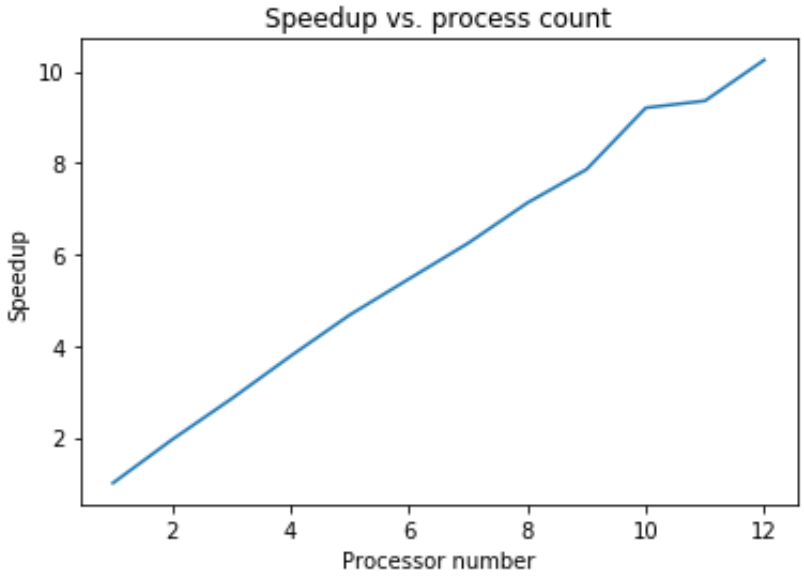


(d) fig 4

Figure 18: Add your own figures before compiling

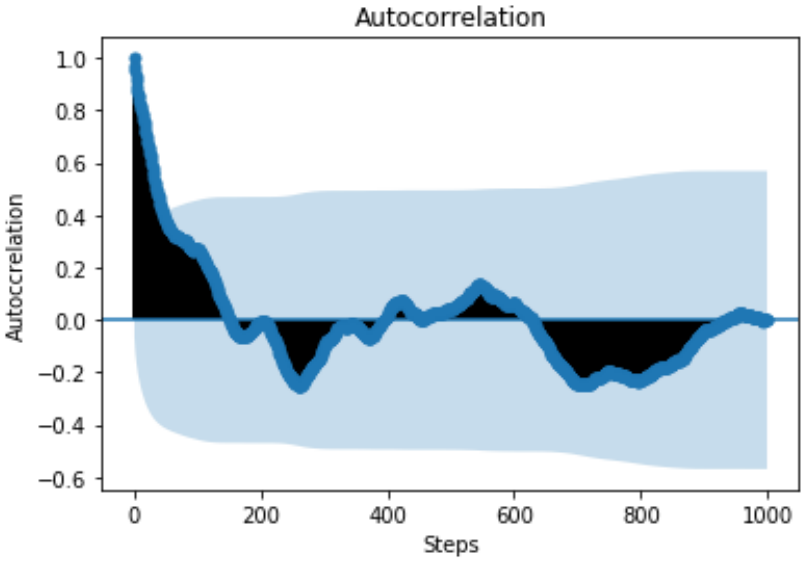
As can be seen from the phase transition at around T=2.27, the 2-spin Potts model is equivalent to the Ising model.

4.10 Speedups Square Ising Model



The speedup graph above shows the near-linear speedup obtained by my code. According to Ahmdal's law, speedups obtained by parallel code cannot be faster than a linear speedup, so this is a good result.

4.11 Autocorrelation



5 Sources