Parallelising Spin Models on Different Geometries

Filip Sosnowski

September 17, 2021

1 Abstract

Statistical systems such as the Ising and the Potts models are often simulated using Monte Carlo methods. My aim is to use these methods via the Metropolis algorithm and simulate the Ising and Potts models. Starting with the square lattice, I then extend the models to the triangular, hexagonal and cubic geometries. I study the second order phase transitions of these models, and study the first order phase transitions of the Potts model. I perform a brief analysis of stable and metastable states encountered in these models. I study the autocorrelations of the Metropolis algorithm. I also the parallelise my simulations using OpenMPI via ghost cell exchanges, decomposing the different lattices in 1,2 and 3 dimensions. I analyse the scalability of my parallel code.

List of Figures

Contents

2 Introduction

2.1 The Ising Model

The Ising model is a mathematical model used to study the properties of a thermodynamic system. Named after Ernst Ising, it is one of the most used models of statistical mechanics in studying phase transitions, which are transitions of a physical system into a different state of matter.[1] Classically, the Ising model deals with the ferromagnetic properties of a system, showing how a system's magnetic properties change with temperature. Studying these magnetic 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 four main observables of interest in this system are magnetisation, energy, magnetic susceptiblity and specific heat capacity. Given a fixed temperature, these properties tend to flactuate randomly, 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 values of these 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 accompanied 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]

2.2 Measuring Observables

The magnetisation 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 the absolute value of the average magnetisation is often taken. Given a system of N particles, 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 energy 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 of the sum $< i, j >$ means that i and j are closest neighbours. The interaction of each two neighbours is taken to be the product of their spins. 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 to account for double counting of neighbour pairs, as pairs $< i, j >$ and $< j, i >$ are treated as distinct in this method despite being the same pair.

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 observables are[3]:

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

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

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 $T_C = 2.269185J$. 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. This is a second order phase transition, meaning that the average magnetisation vanishes continuously. Furthermore, magnetic susceptibiliy of this system shows divergence at T_C. However, this kind of divergence only happens for an infinitely sized system, and in the case of a finite system the phase transition is represented by a sudden spike instead.

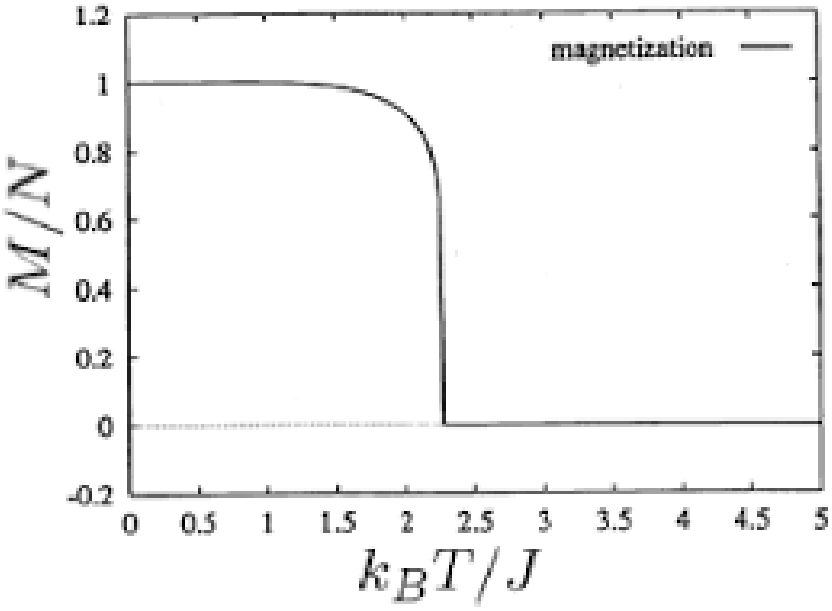


Figure 1: Phase transition for the square grid Ising model shown by sudden change in average magnetisation.[4]

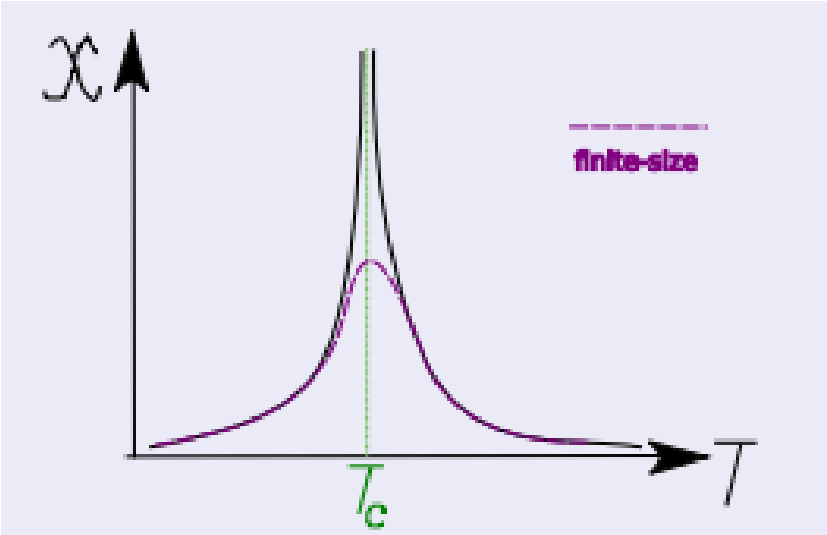


Figure 2: Phase transition in the Ising model shown by divergence of mangetic susceptibility.[4]

The average energy of the system experiences a similiar phase transition to the average magnetisation, rising from -2J towards 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 simplified model that is not reflected by reality, the different angles between closest neighbours can be ignored. My main geometries of interest are triangular, hexagonal and cubic lattices. Over the course of this report 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[5]:

$$H = -J \sum_{,i,j.} (1 - \delta(\sigma_i, \sigma_j))$$

Where the δ stands for the delta Kronecker function. This function compares two variables and returns 1 if they are equal, and 0 if they are not.

The formulae for measuring the other observables remain 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 second order phase transitions at the same temperatures. However, as the number of possible states increases, the system's behaviour changes. For the two dimensional lattices with J>0, the system experiences second order phase transitions for q<5. However for q>4 these transitions become first order, meaning that there is a discontinuity at T_c. In the case of three dimensions and higher, the system experiences first order phase transitions for q>2.[6]

I will be applying the Potts model to all the geometries previously discussed and studying their 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" [7].

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

For a discrete random variable X_t with N measurements labelled from 0 to N-1 the autocorrelation function is given by[8]:

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

In the Metropolis algorithm, the MCMC algorithm I used in this project, the autocorrelation is expected to decrease expotentially from Markov step 0.

2.7 Scalability

Scalability is the property of a system that describes its ability to handle a growing amount of work when additional resources are added. In the case of High Performance Computing this is used to describe how the execution time of a parallel program is affected when additional processors are added.

There are two notions of scalability in HPC - weak scaling and strong scaling.[9] In strong scaling the problem size is fixed, and the scaling is measured as run time against the number of processors used. In an ideal situation the run time falls in inverse proportion to the number of processes. In weak scaling the workload per core is fixed, and "perfect" scaling means that the runtime is unchanged as more cores are used.

Amdahl's law is another important concept in HPC scalability. Every typical program is made up of two components - parts that can be parallelised and parts that must be ran sequentially. Amdahl's law states that for such a program the runtime t_n on n processors is given by:[10]

$$t_n = (\frac{p}{n} + (1 - p))t_1$$

where p is the fraction of the task that can be parallelised and t_1 is the serial runtime.

In my calculations of the scalability of my code I assumed that the non-parallelisable section of my code is negligible. This means that the strong scaling (ie. speedup) is then given by:

$$\frac{t_1}{t_n}$$

I will be mostly focusing on strong scaling.

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. This criterion requires that a state transition from state x to state 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.[11]

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. I used the first method. 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 $\Delta E > 0$, 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 measured in terms of J/K_b.

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 100x100 statically allocated square array and wrote a function to initiate for both hot and cold starts. 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. Another bonus of these arrays is that they are contiguously allocated in memory, which came in useful when parallelising the code.

Next, I had to modify the metropolis algorithm so that it is more suited to computation. I removed the Boltzmann constant from the acceptance step and assimilated it into the temperature T, 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 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 I assumed this would have a significant effect when performing 1000 sweeps for each temperature. However after running some tests this didn't seem to affect the runtime in any significant way, which is probably due to the efficiency of modern compilers.

I ran the simulation 50 times to obtain an average for each observable at each temperature, with the temperature ranging from 0.0001 to 5.0001 in increments of 0.05. I couldn't start with an initial temperature of 0 as it was giving incorrect results, so I decided to start with a temperature close to zero - hence 0.0001.

To measure the average magnetisation per site I decided to make my results be 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 susceptibility I multiplied the variance of magnetisation by 1/T. Next, for specific heat I multiplied the variance of energy by $1/T^2$.

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

In order to parallelise the code I decided to use 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 between each sweep.

As reading arrays along the rows is faster than along the columns 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. These were followed by an MPI_Waitall that ensured these exchanges are made before the next sweep is performed.

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 during hot starts 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 by the ghost exchange.

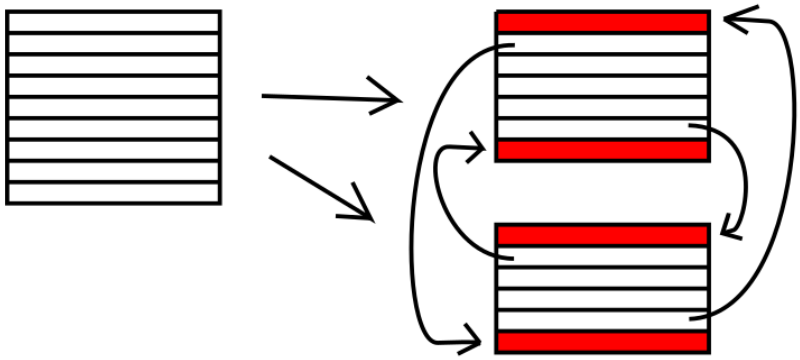


Figure 3: Example of one dimensional decomposition between two processes. An array of 9 rows is split is into 5 rows for process 1 and 4 rows for process 2, with the red columns signifying the ghost columns received from the other process. The arrows show the direction and destination of the row exchanges.

3.4 Alternate One Dimensional Ghost Exchange

A possible bug that can occur during MPI ghost exchanges is that sometimes exchanges can happen before a particular sweep is completed. To test against this I wrote an alternate exchange method that ensures this doesn't happen. The method performs the sweeps and exchanges in a checkboard-style manner. For example, it can first sweep through every odd-numbered processes' grid section, and then odd processes sends halo information to the even-numbered processes. The same steps are then repeated for every even process.

Although this kind of exchange is very safe, it is not efficient. One of the biggest drawbacks of this is now that twice as many Monte Carlo loops have to be performed at each temperature. Combined with the additional logic gates included in this exchange version, this method is expected to experience a very poor speedup, especially compared with the original ghost halo exchange method.

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:

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

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

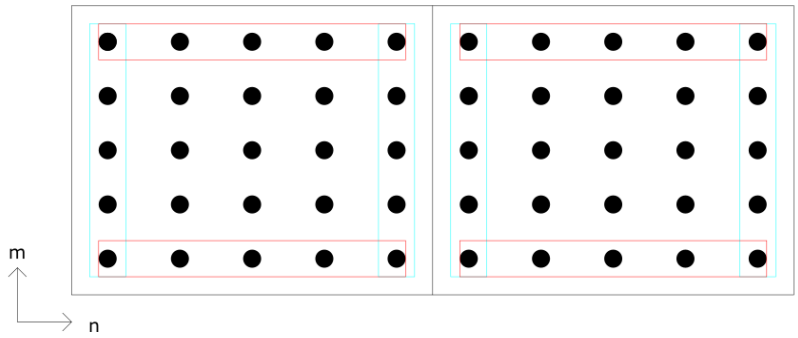


Figure 4: Two dimensional decomposition between two neighbouring processes. The rows in the red box show signify the rows to the neighbouring processes to the top and bottom of the process, while the columns in the blue boxes show the columns shared with neighbouring processes to the left and right of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

3.6 Triangular Ising Model

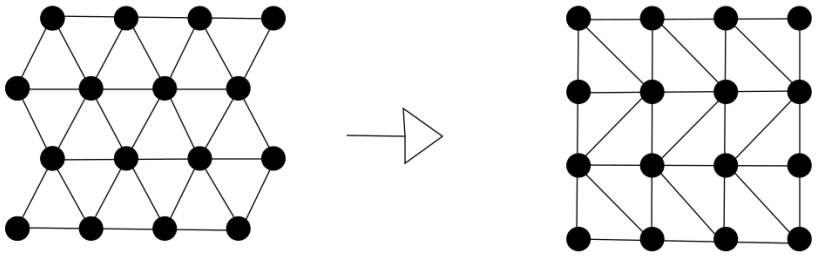


Figure 5: Mapping of a triangular lattice to a square lattice.

To find the nearest neighbours in the triangular Ising model I drew the triangular lattice first then mapped it to a normal square grid by hand. 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. Given a grid g, 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 and specific heat of the system. I was then able to run the program as normal.

3.7 Hexagonal Ising Model

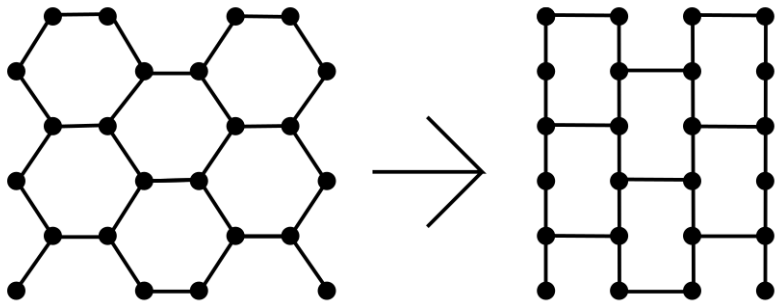


Figure 6: Mapping of a hexagonal lattice to a square lattice.

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 tests were all done on a 100x100 lattice this is not a problem.

Having once again adjusted 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.

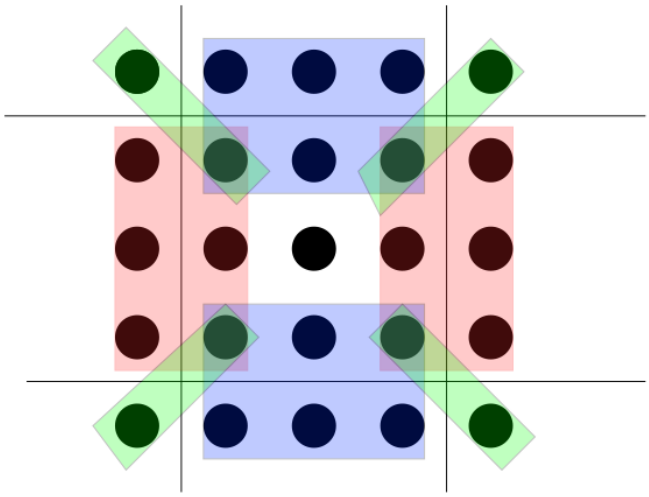


Figure 7: Cell exchanges for a sublattice in the triangular model. The red areas show exchanges of columns, the blue areas show row exchanges, while the green areas show diagonal exchanges.

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 20x20x20 grid.

3.10 Parallelising the Cubic Ising Model

In C, a statically allocated three dimensional array g[i][j][k] with dimensions x, y and z along each respective axis 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 in one dimension 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 2 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 regular spacing between the elements I was able to once again create an 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:

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

$$\text{j-axis position} =$$

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

$$\text{k-axis position} =$$

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

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 q. I also had to change the way energy is calculated across the grid, which required writing a function that calculated the Hamiltonian for the Potts model.

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

3.14 MPI-IO

A potentially useful feature that I decided to include in my code is the ability to save the current configuration of the system at a given time. In order to be able to do this efficiently in parallel I used MPI-IO. MPI-IO provides a low-level interface for carrying out parallel I/O, ie. allows multiple MPI processes to read from or write to a single file. However all MPI-IO is done as binary reads and writes, so it must be provided with additional information so that these are performed correctly.

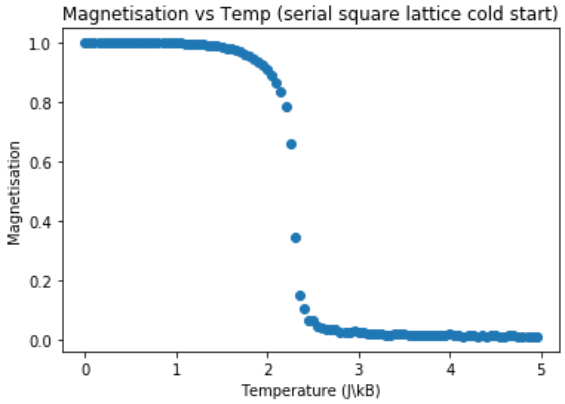
Like in C, MPI-IO uses a file pointer to know where in the file we are currently looking. However in MPI-IO this file pointer isn't shared and each process has its own unique pointer to tell it where is is looking. For my purposes each task would have to access non-contiguous file chunks. This is most efficiently done by setting up a file view, which restricts what parts of the data file each process can see, and therefore tells it what parts of the data can be skipped.

I found that the best way to set up these file views was to create a subarray for each process to tell it what part of the grid belongs to it. This in turn allowed for each process to write into and read the relevant parts of the file.

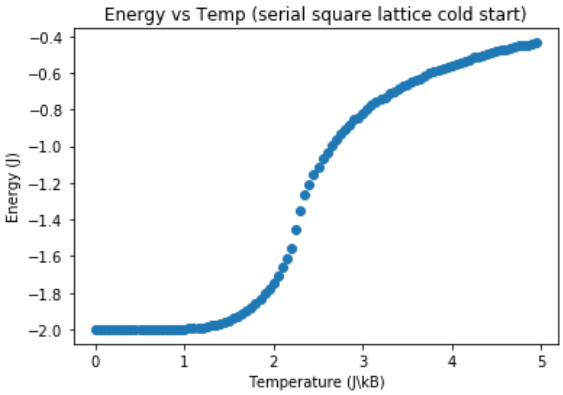
Although in the end I didn't find this function to be useful, it provides additional potential for my code to be tested on any possible grids of interest in the future. It is well suited for vary large grids due to the efficient use of MPI-IO.

4 Results

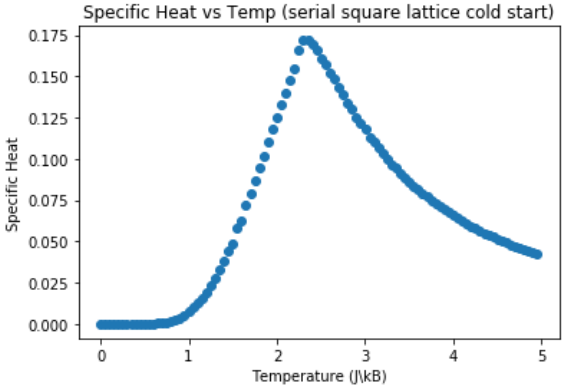
4.1 2D Square Grid Sequential Ising Model



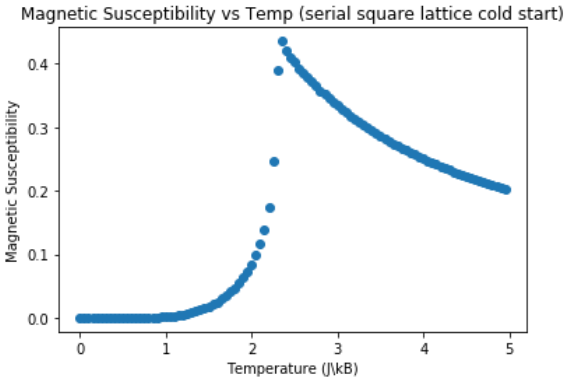
(a) fig 1



(b) fig 2



(c) fig 3



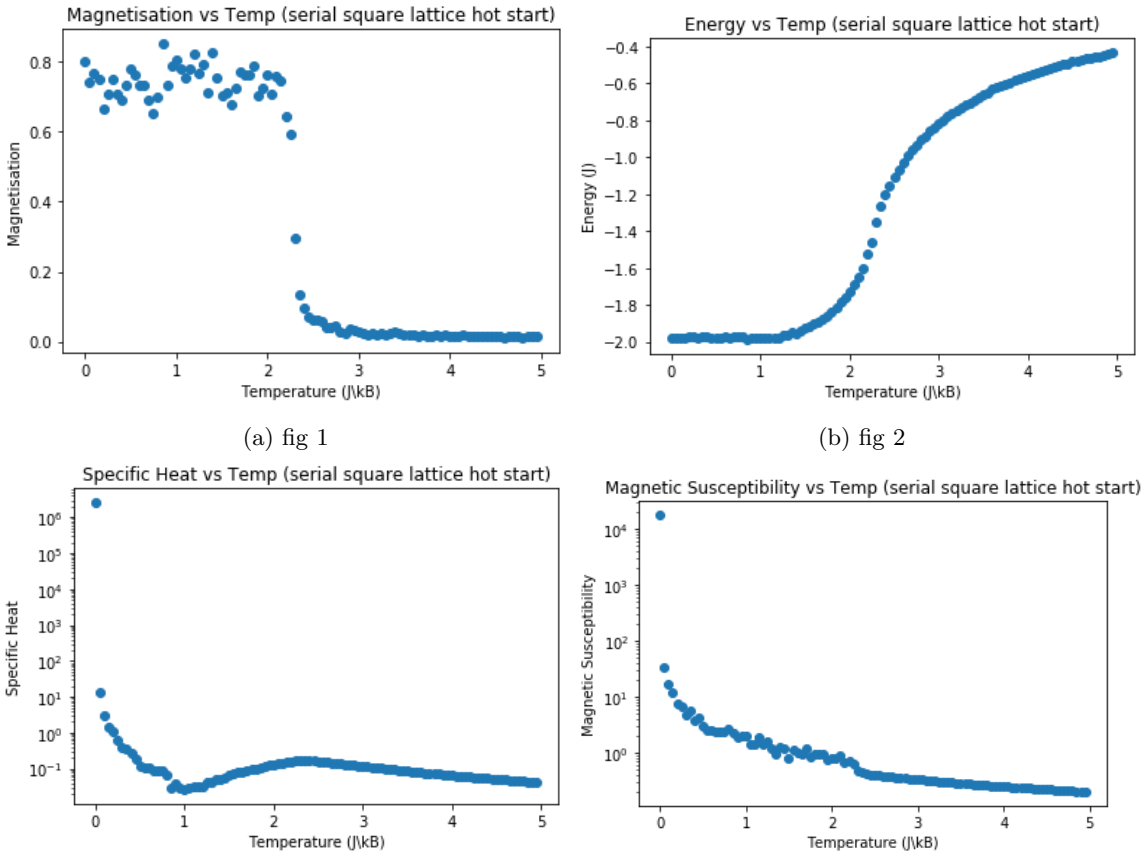
(d) fig 4

Figure 8: Average of 50 cold start simulations on a 100x100 grid for the square lattice Ising model, 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 phase 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 in 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 $T = 0$. 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 from zero, however in this case the behaviour only becomes consistent with the cold start at around T_c.



(a) fig 1

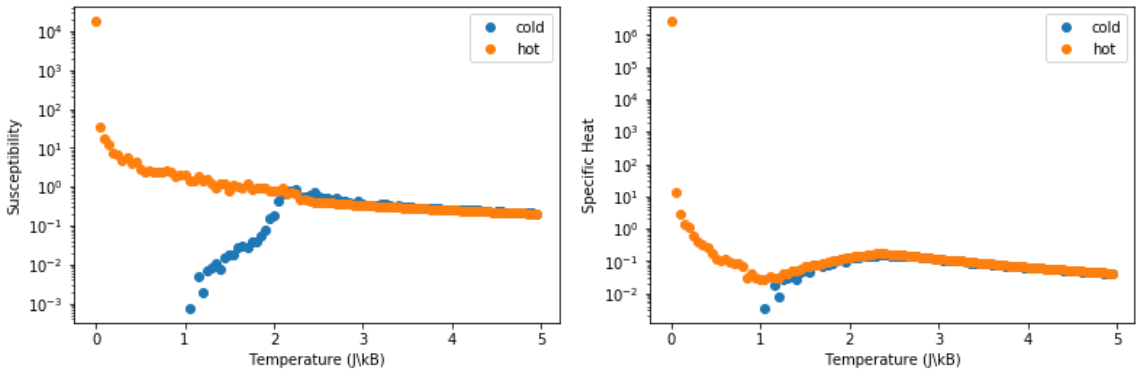
(b) fig 2

(c) fig 3

(d) fig 4

Figure 9: Average of 50 hot start simulations on a 100x100 grid for the square lattice Ising model, 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. Due to the random nature of the how start, it is possible for an initial state to fall into one of these metastable states.

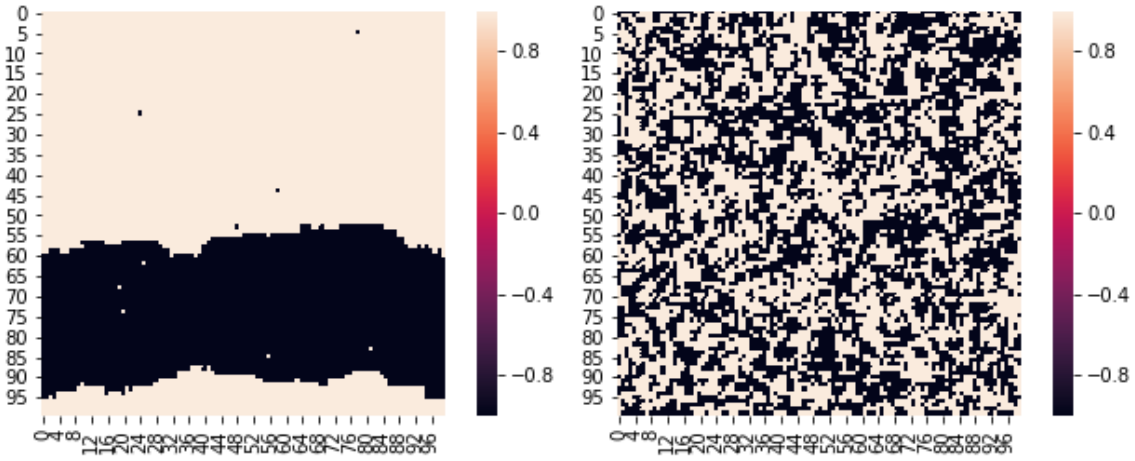
To compare these states with the equilibrium states at $T < T_c$, I reran my hot start simulations and I saved the system configurations for states with magnetisations $M << 1$. I also saved some equilibrium states at $T > T_c$ for comparison. Next, I made heatmaps of the two configurations and compared them.

The metastable states at $T < T_c$ appear to have the same energy as the stable states at these temperatures, however they have much lower magnetisations. Creating heatmaps of these states revealed that they consist of massive bands or clusters of same-spin states, with the system made up of roughly half +1 spins and half of -1 spins. As all the sites in these metastable states seem to be surrounded by niehgbours of the same spin, the average energy of the system tends to be near -2J, ie. the lowest possible energy of the system. However, this is different from a stable state at these temperatures which consist almost entirely of the same spin. A consequence of this is that in the metastable state the magnetisations of the sites effectively cancel out and are close to 0. The system then tends to be stuck in this state for a very long time, as it is close to being in energy equilibrium and a great number of spin flips must be performed to bring this system to its true equilibrium.

The stable states at $T > T_c$ have similiar magnetisation to the metastable

states at $T < T_c$, however have much higher energy. Although these states are also made up of roughly half +1 spins and -1 spins, these are now arranged more randomly, with each site being likely to be surrounded by different neighbours. This causes the energy of the system to be much higher, which is in line with the equilibrium at these temperatures.

Due to the difficult to read nature of the hot start results, I decided to perform the rest of my testing using cold starts only.



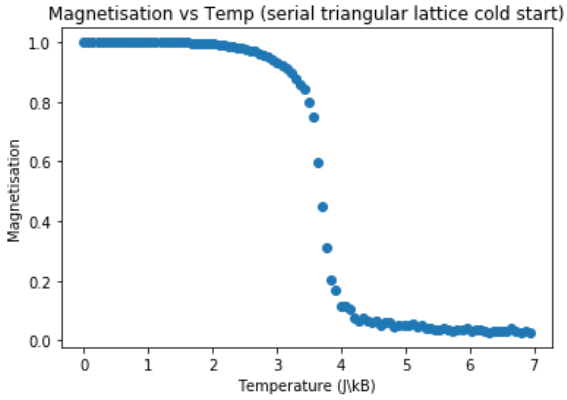
(a) fig 1 (b) fig 2

Figure 11: Left: Metastable state at $T << T_c$. Right: Stable state at $T >> T_c$.

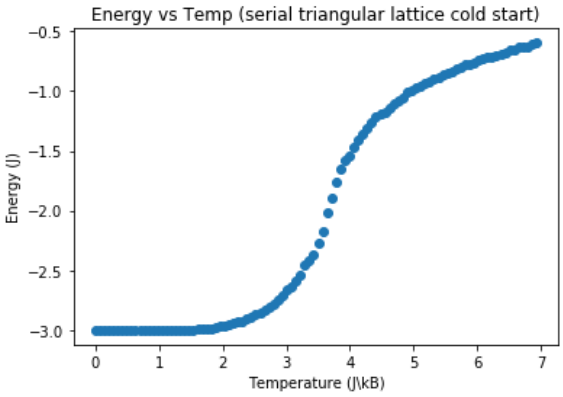
4.2 2D Triangular Grid Sequential Ising Model

Testing my triangular code on a cold start revealed that the system experiences a sharp drop in magnetisation as well as a sharp increase in average energy, implying that the system experiences a phase transition at $T_c \approx 4\frac{J}{K_b}$. Further evidence for this can be seen from the spikes in magnetic suceptibility and specific heat around this temperature.

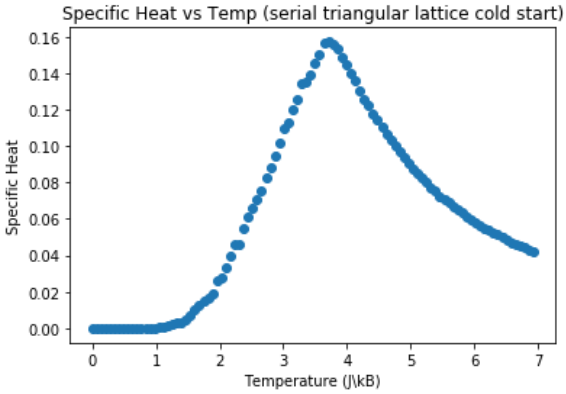
I think that the fact that T_c is higher for this system than it is for the square lattice can be explained by the fact that each site now has a higher number of neighbours. This means that each site now has a bigger range of possible energies, and is thus less likely to be flipped. It also makes sense from a physical standpoint, as the particle is held in place by forces from a higher number of neighbouring particles, so higher energies (and thus higher Temperatures) are needed to flip the spins.



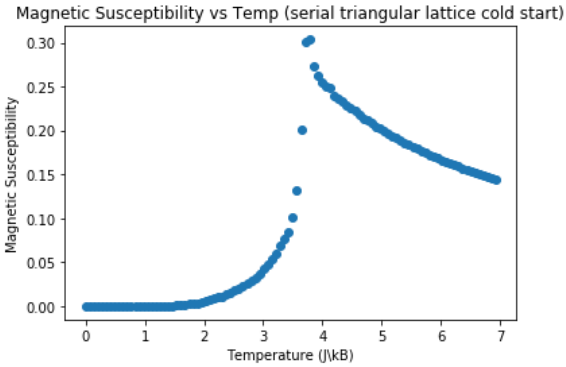
(a) fig 9



(b) fig 10



(c) fig 11



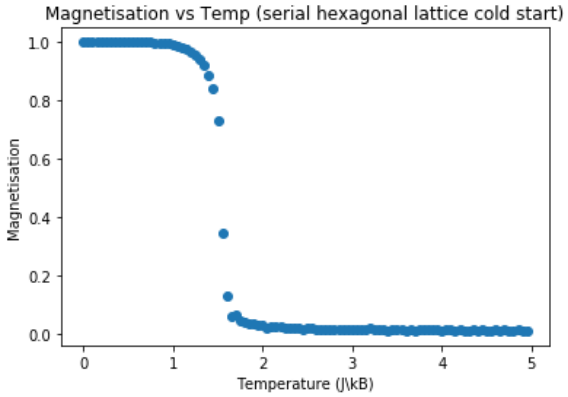
(d) fig 12

Figure 12: Phase transition of the triangular Ising model, measured on a 100x100 grid and averaged over 50 simulations for 100 temperature samples.

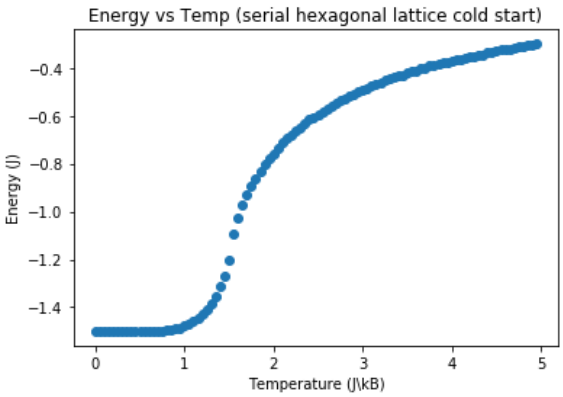
4.3 2D Hexagonal Grid Sequential Ising Model

The cold start on the hexagonal Ising model shows similiar behaviour to the other two models, as it experiences a second order phase transition via a sharp decrease in average magnetisation and average energy, along with a spike in specific heat and suceptibility. The difference in this model is that Curie Temperature now appears to be much smaller at $T_c \approx 1.5\frac{J}{K_b}$.

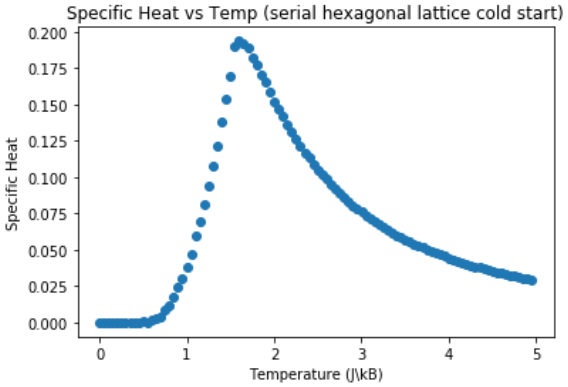
The Curie Temperature, which is lower than the Curie Temperatures for both the square and triangular lattices, was expected due to the fact that each site on this lattice has only three neighbouring spins. This causes the possible energies of each site to be significantly lower than for the other two lattices, which in turn makes it more likely that the spin will be flipped.



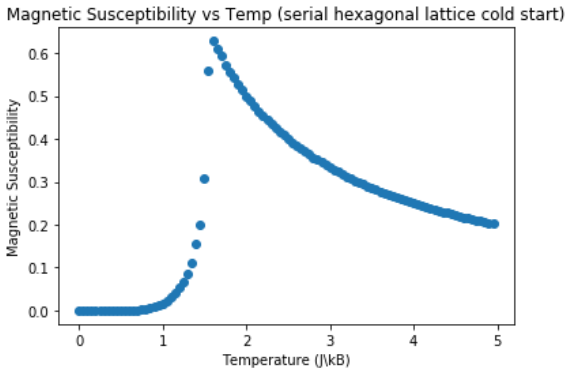
(a) fig 9



(b) fig 10



(c) fig 11



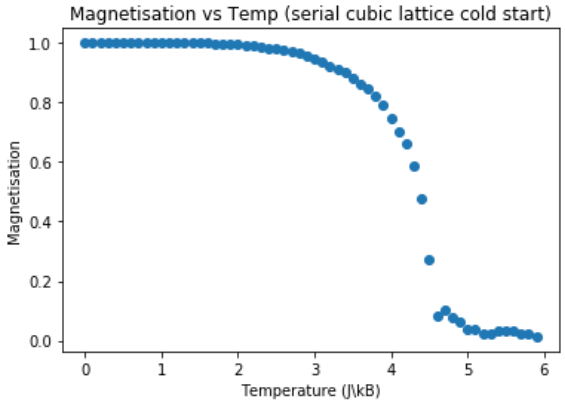
(d) fig 12

Figure 13: Phase transition of the hexagonal Ising model, measured on a 100x100 grid and averaged over 50 simulations for 100 temperature samples.

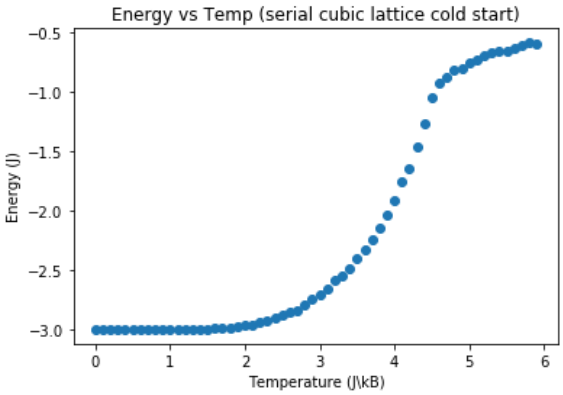
4.4 3D Cubic Grid Sequential Ising Model

Extending the Ising model to three dimensions shows that the behaviour of the model in this geometry is consistent with the results seen at two dimensions. The system still experiences a second order phase transition, with the Curie Temperature at $T_c \approx 4.5 \frac{J}{K_b}$. This is a slightly higher Curie Temperature than the two dimensional triangular lattice, which also has six closest neighbours.

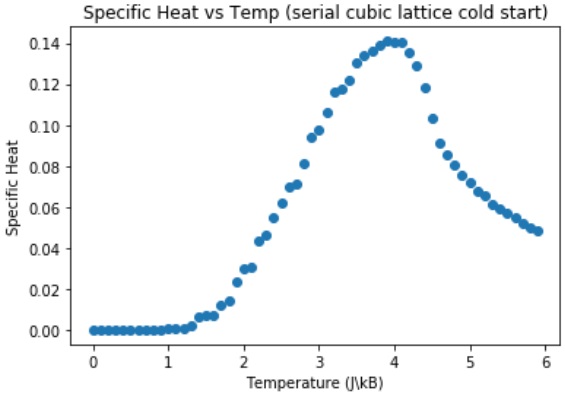
Another difference of the cubic model is that the magnetisation decreases a lot less sharply than it does for the two dimensional models, and appears to flactuate quite a bit past T_c.



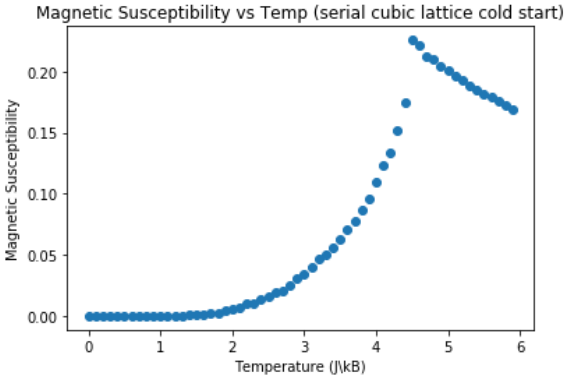
(a) fig 9



(b) fig 10



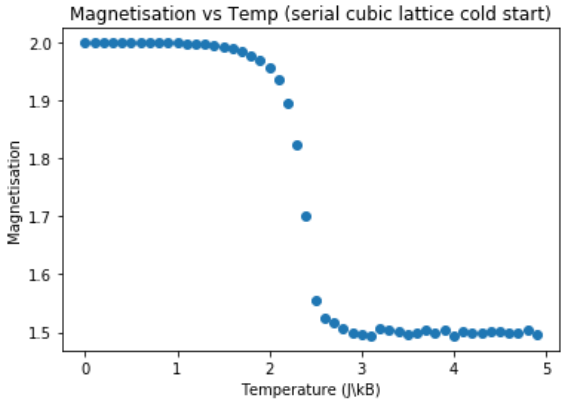
(c) fig 11



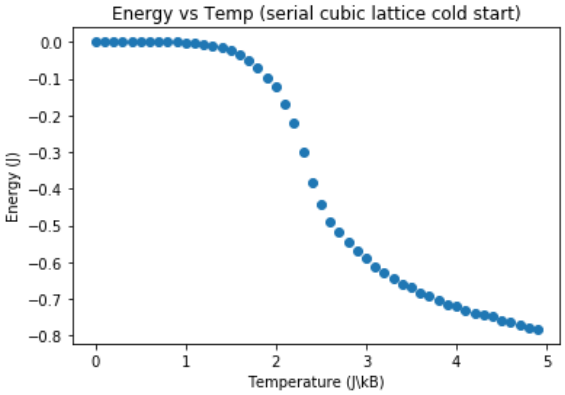
(d) fig 12

Figure 14: Phase transitions of the cubic Ising model on a 20x20x20 cubic lattice, averaged over 50 simulations for 100 temperature samples.

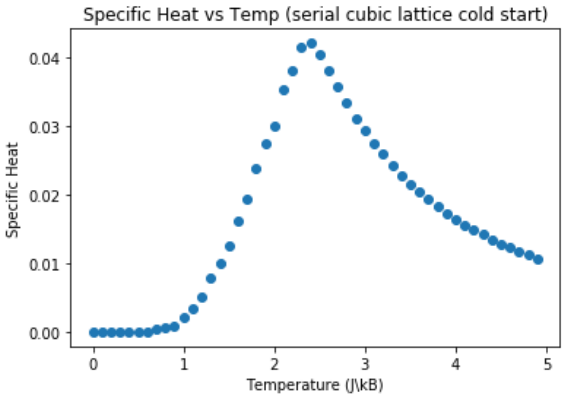
4.5 2D 2-state Potts Model



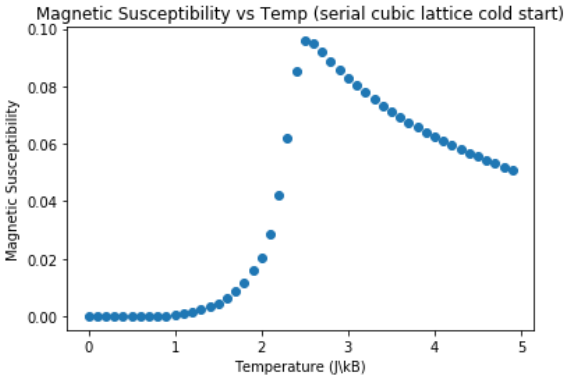
(a) fig 1



(b) fig 2



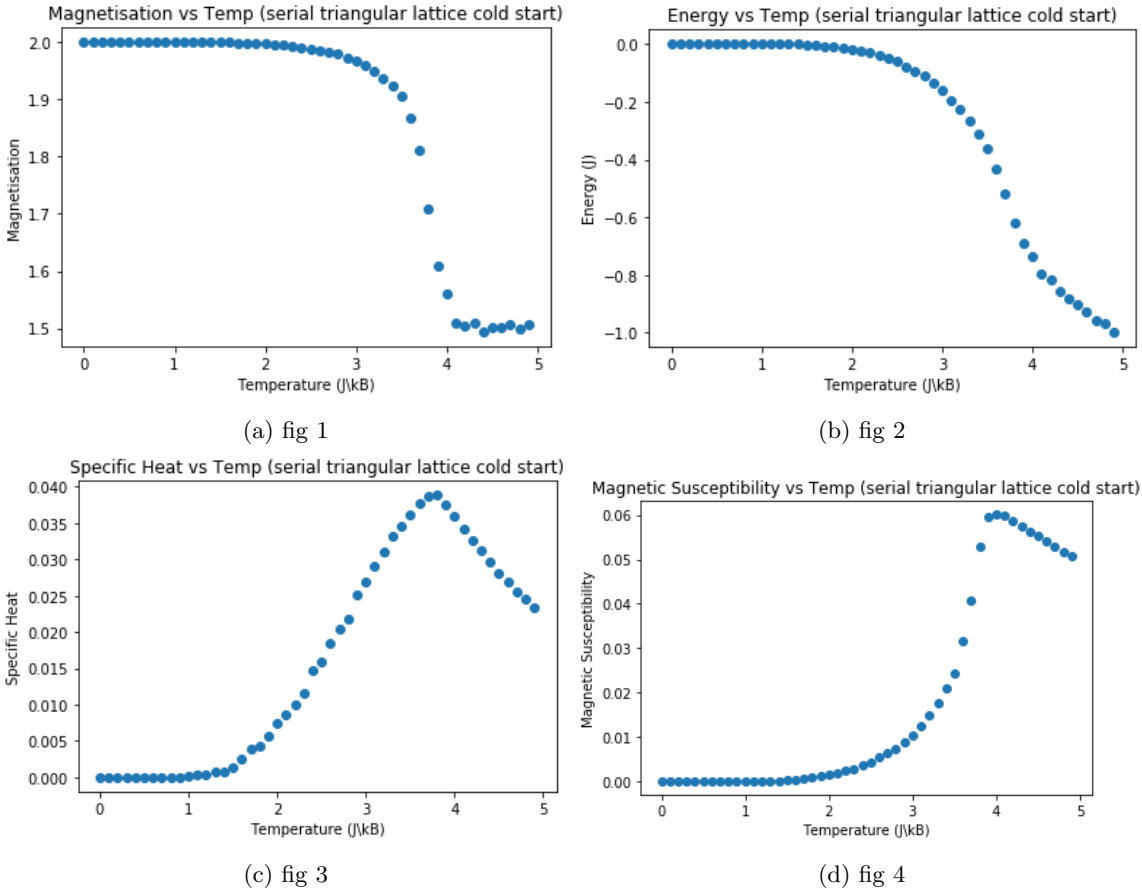
(c) fig 3



(d) fig 4

Figure 15: Phase transitions of the 2D Potts model on a square lattice. The system experiences a phase transitiont at the same temperature as the square lattice Ising model.

Testing the 2-state Potts model on a two dimensional square lattice reveals that the system undergoes a second order phase transition at $T \approx 2.27 \frac{J}{K_b}$, which is the same as the phase transition for the Ising model. This confirms that the 2-state Potts model is equivalent to the Ising model. However there is a difference between the values in magnetisation and energy of the two models, but this is due to the different values of spins and the changed Hamiltonian.



(a) fig 1

(b) fig 2

(c) fig 3

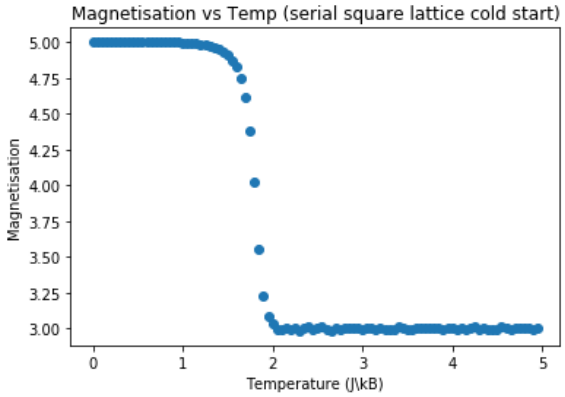
(d) fig 4

Figure 16: Phase transitions of the 2D Potts model on a triangular lattice. The system experiences a phase transition at the same temperature as the triangular lattice Ising model.

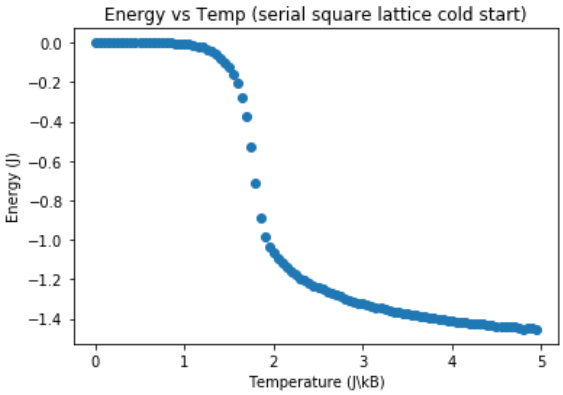
Running my 2-state Potts model on a triangular lattice revealed a similiar result, with the Curie temperature being the same as for the Ising model. This suggests that the 2-state Potts model is equivalent to the Ising model on any geometry.

To see whether the 2-state Potts model is also equivalent to the Ising model in three dimensions, I tested my Potts model code on a three dimensional cubic lattice and compared it with my cubic lattice Ising results. The phase transitions appeared to be at the same temperature, confirming that the 2-state Potts model is also equivalent to the Ising model in three dimensions.

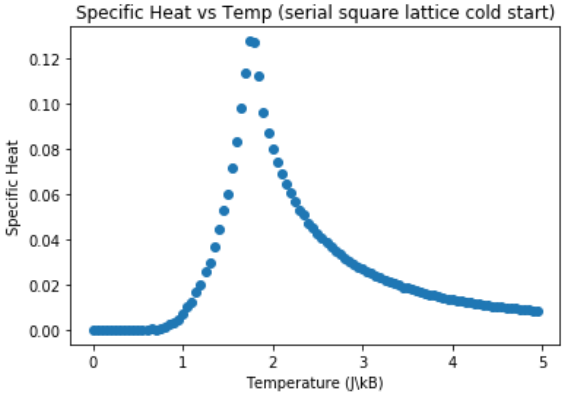
4.6 5-state 2D Potts model



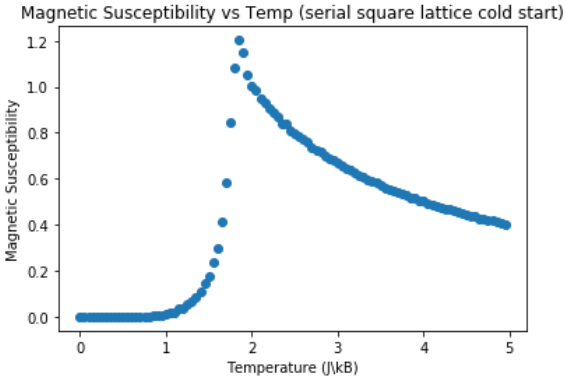
(a) fig 1



(b) fig 2



(c) fig 3



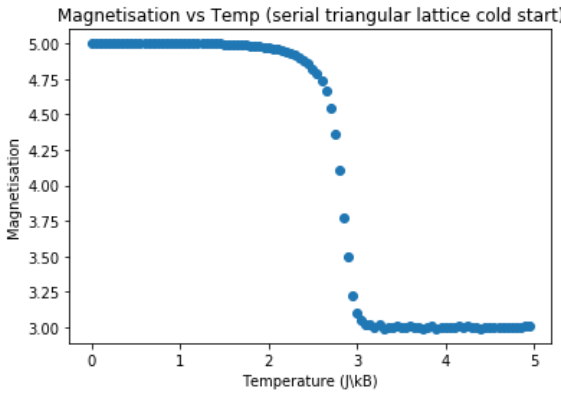
(d) fig 4

Figure 17: Phase transitions of the 2D 5-state Potts model on a square lattice. The system experiences an extremely sharp change in magnetisation, suggesting a first order phase transition.

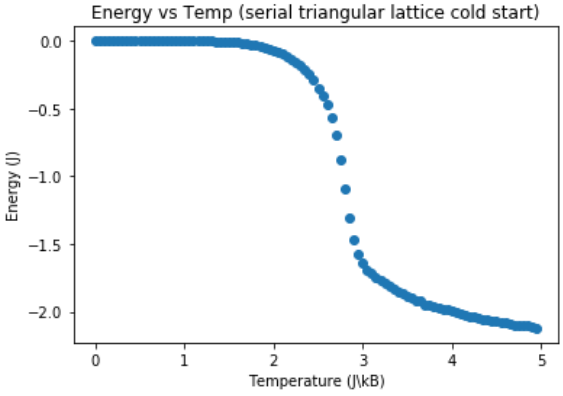
Another test of interest to me was the behaviour of the 5-state Potts model and its first order phase transition. I found the Curie Temperature to be significantly lower than in the 2-state case, at $T \approx 2\frac{J}{K_b}$. During this phase transition the magnetisation experiences a much sharper decrease in magnetisation than in the 2-state case, signifying a first order phase transition. The energy also experiences an extremely sharp decrease, however this appears to be continuous. Compared with the 2-state Potts model, the spikes in specific heat and magnetic susceptibility are now much sharper and bigger.

I repeated the same test on the hexagonal and triangular lattices. The results were similiar, with both the hexagonal and triangular lattices experiencing a first

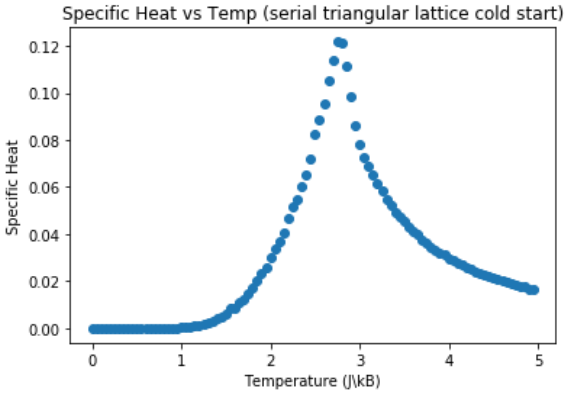
order phase transition, at $T \approx 1.5 \frac{J}{K_b}$ and $T \approx 3 \frac{J}{K_b}$ respectively.



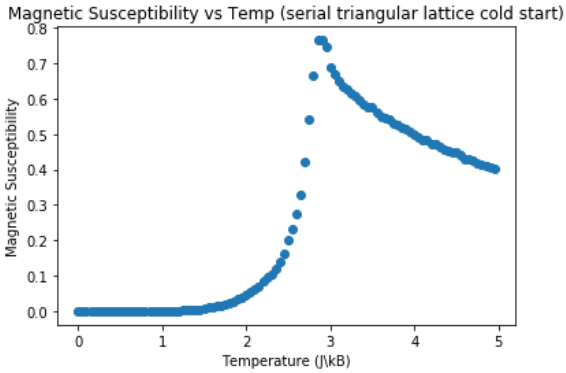
(a) fig 1



(b) fig 2

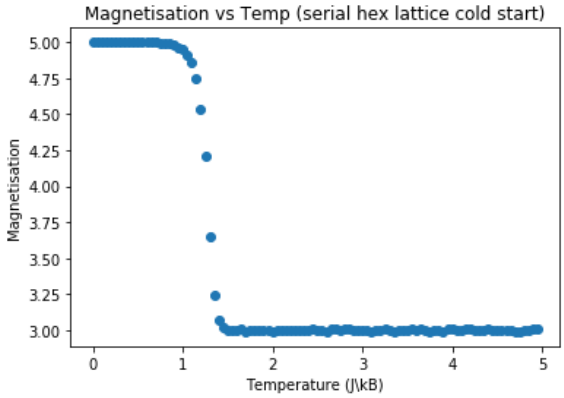


(c) fig 3

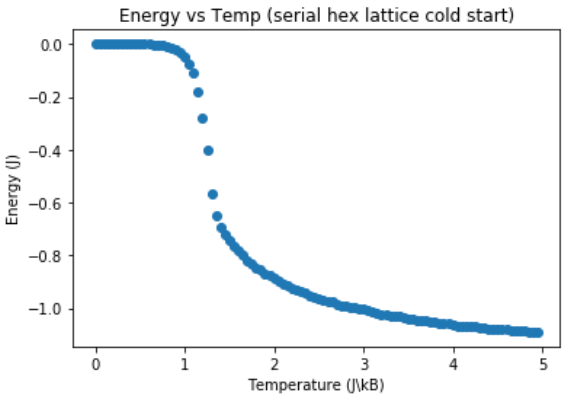


(d) fig 4

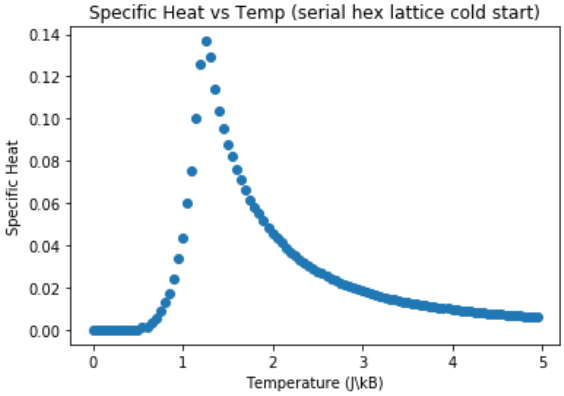
Figure 18: Phase transitions of the 5-state 2D Potts model on a triangular lattice.



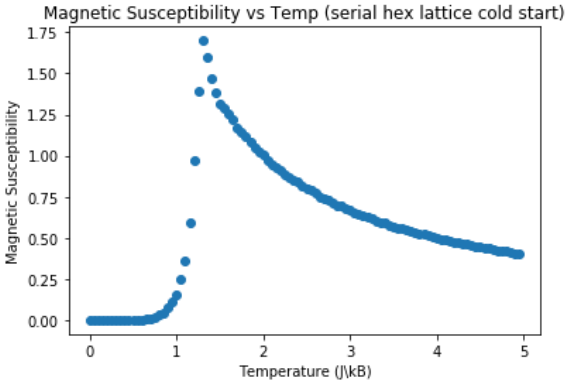
(a) fig 1



(b) fig 2



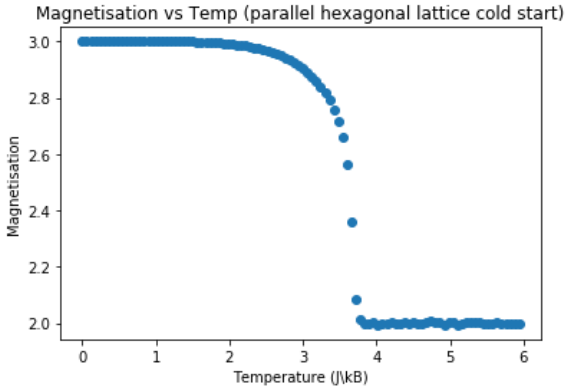
(c) fig 3



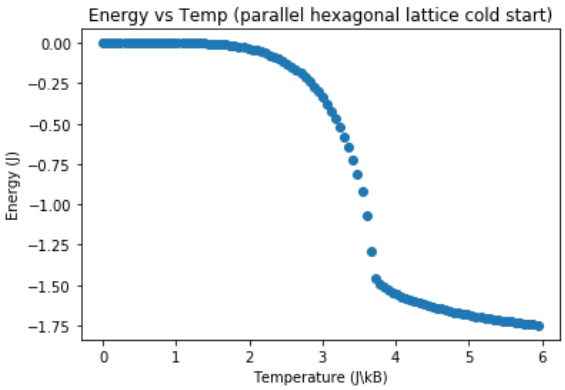
(d) fig 4

Figure 19: Phase transitions of the 5-state 2D Potts model on a hexagonal lattice.

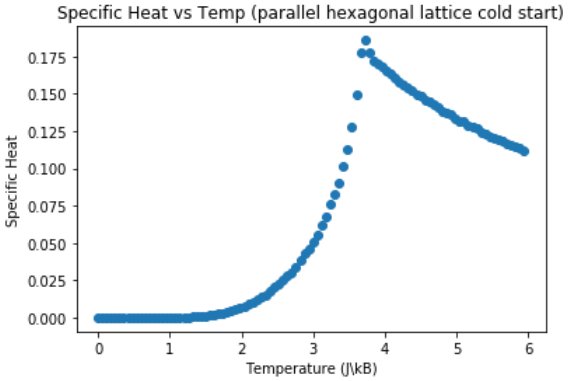
4.7 3-State Cubic potts Model



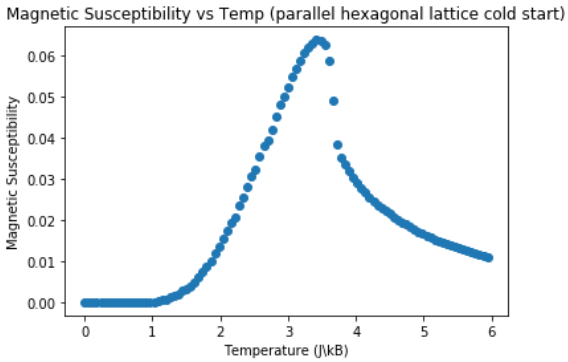
(a) fig 1



(b) fig 2



(c) fig 3



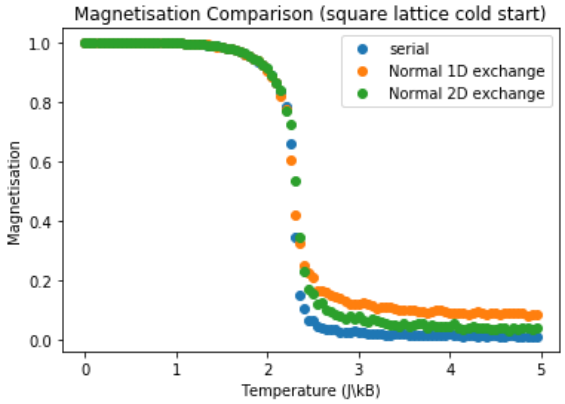
(d) fig 4

Figure 20: Phase transitions of the 3-state 3D Potts model on a cubic lattice. The system appears to exhibit a first order phase transition.

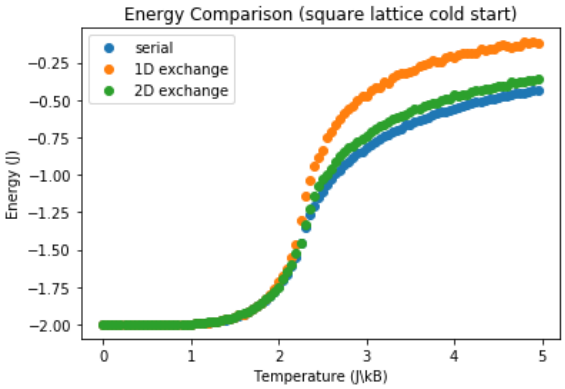
Comparing the 3-state cubic Potts model with the cubic Ising model (which is equivalent to the 2-state Potts model) reveals a much sharper phase transition, confirming that this model experiences a first order phase transition. Furthermore, the system appears to have a lower Curie temperature than the 2-state case, which is in line with my findings for the 5-state 2D Potts model.

4.8 Comparison of Serial and Parallel Results

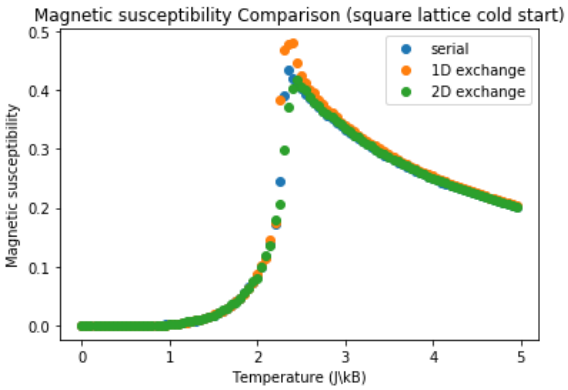
To see the accuracy and effectiveness of my parallel code, I reran some of the simulations above in parallel using 12 processes. Then I graphed my parallel results against my serial results to check their consistency.



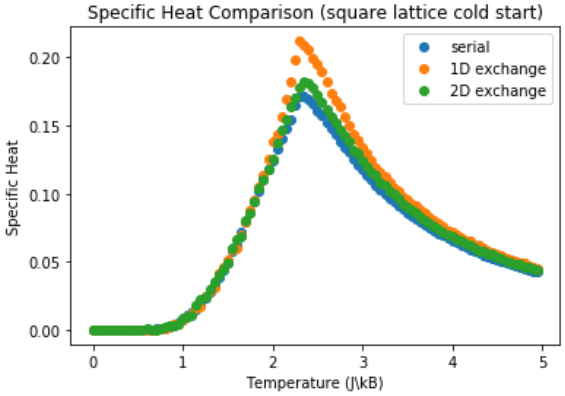
(a) fig 1



(b) fig 2



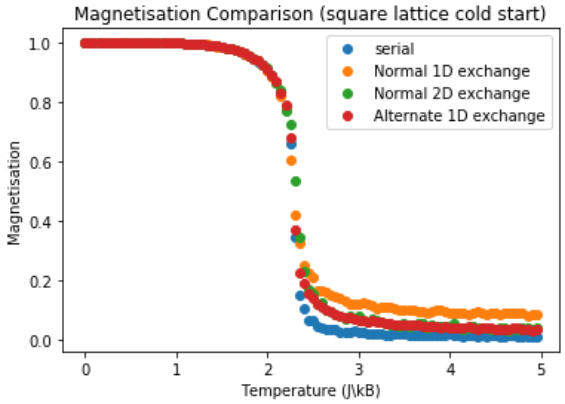
(c) fig 3



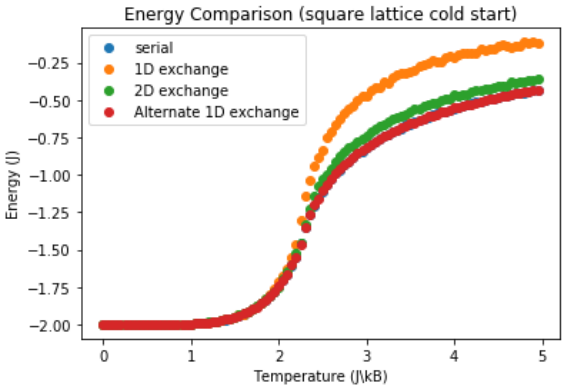
(d) fig 4

Figure 21: Comparison of my serial square lattice Ising model results against the 1D and 2D decomposition results.

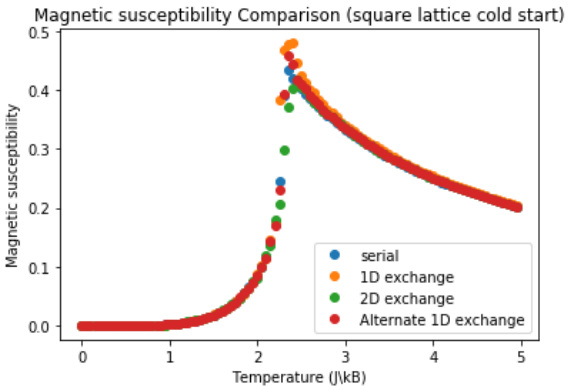
I began my comparison by testing the square lattice. Both the 1D and 2D decomposition models appeared to experience second order phase transitions at the same temperature, however for temperatures $T > T_c$ the values of both magnetisation and energy were significantly higher for the parallel models than for the serial models. This was most apparent in my 1D decomposition results for energy and specific heat, which showed a great difference at high temperatures. For the 2D decomposition these differences were less apparent, however still noticable.



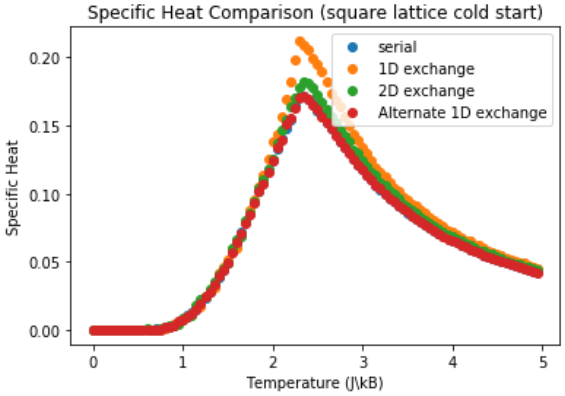
(a) fig 1



(b) fig 2



(c) fig 3

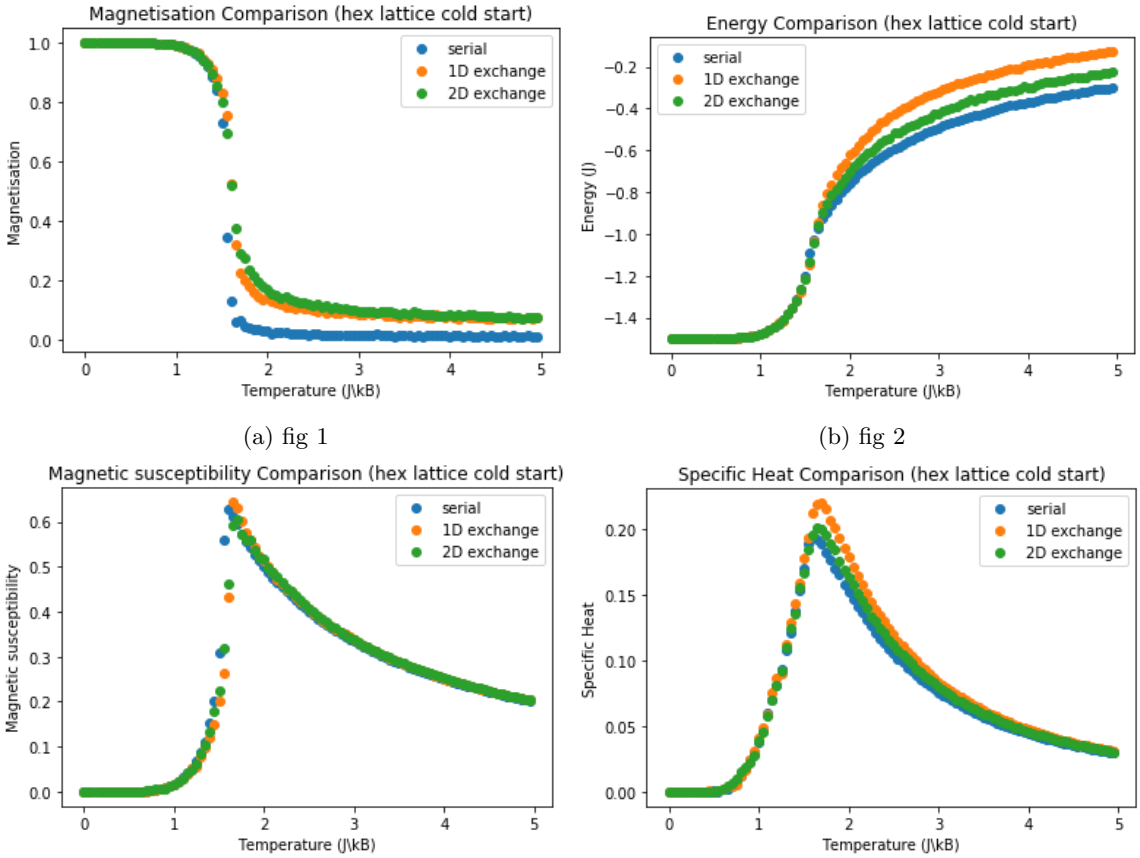


(d) fig 4

Figure 22: Comparison of my serial square lattice Ising model results against the 1D and 2D decomposition results, this time including the alternative 1D exchange results.

In an attempt to find the possible source of these differences, I reran my code using the alternative 1D exchange method. The results were much different now, with the energy and the specific heat graphs being the exact same as for the serial code. However, the average magnetisation and magnetic suceptibility were more in line with the 2D decomposition than with the serial code.

This result hints that part of the issue in the inconsistent rseults may be arrising from a bug in the ghost exchanges, however this doesn't appear to be the full reason as some results are still inconsistent.



(a) fig 1

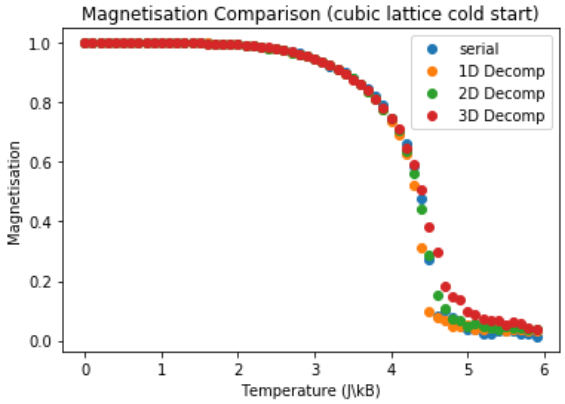
(b) fig 2

(c) fig 3

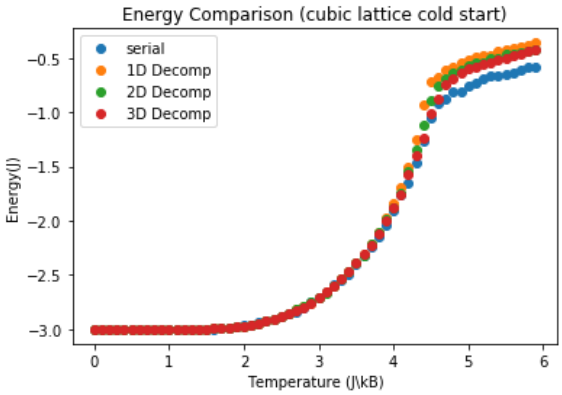
(d) fig 4

Figure 23: Comparison of my serial hexagonal lattice Ising model results against the 1D and 2D decomposition results.

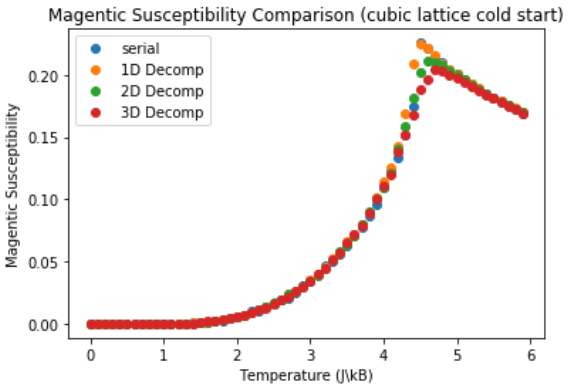
To further test the disrepancies in my result I investigated my results for the hexagonal lattice Ising model. The results were similiar the previous ones, with the parallel models experiencing second order phase transitions at the same temperature as the serial code, but then producing higher values for the observables at $T > T_c$. The only observable that seemed almost indistinguishable from the serial results was the magnetic susceptibility. For the other observables, the 2D decomposition once again produced results more in line with the serial results than the 1D decomposition.



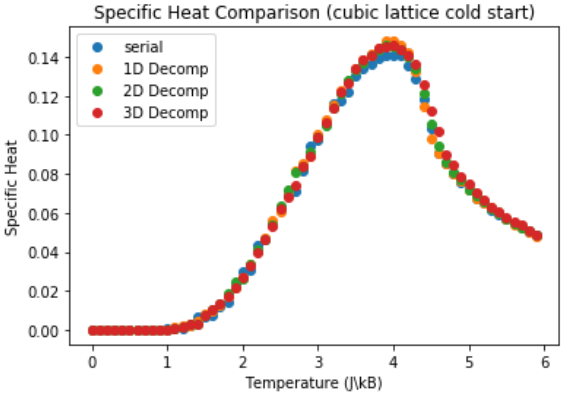
(a) fig 1



(b) fig 2



(c) fig 3



(d) fig 4

Figure 24: Comparison of my cubic square lattice Ising model results against the 1D, 2D and 3D decomposition results.

Finally, I compared the parallel cubic Ising model with the serial Ising model. Surprisingly, the parallel results seemed far more in line with the serial results than any of the parallel results in the 2D case. However some inconsistencies were still apparent at high tempereatures, especially for the average energy which was still significantly higher for the parallel results than for the serial results.

4.9 Scaling of the 2D Ising Model

To measure the parallel scaling of my code for the two dimensional Ising model I timed on my simulations on a different number of cores. First I timed my code on every processor number between 1 and 12. Each time I ran 50 simulations of the 100x100 grid, with 100 temperature samples and 1000 metropolis sweeps

at every temperature.

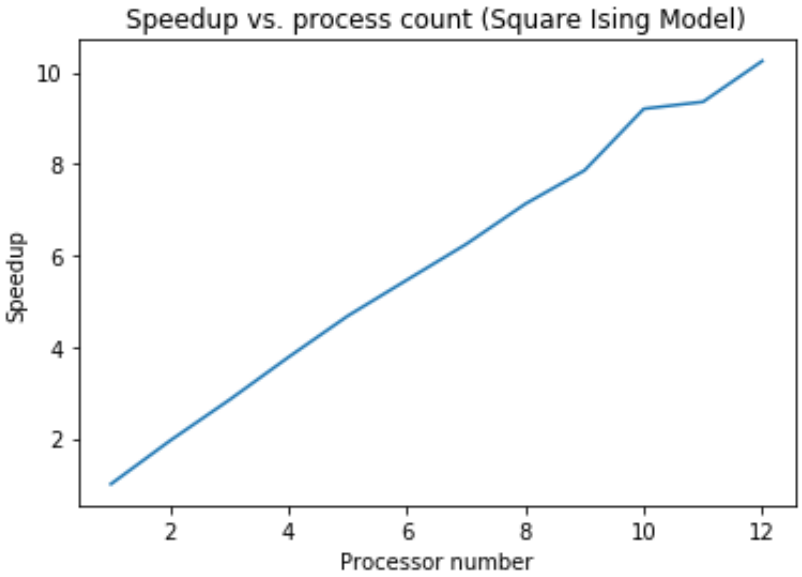


Figure 25: The speedup of the square lattice Ising model.

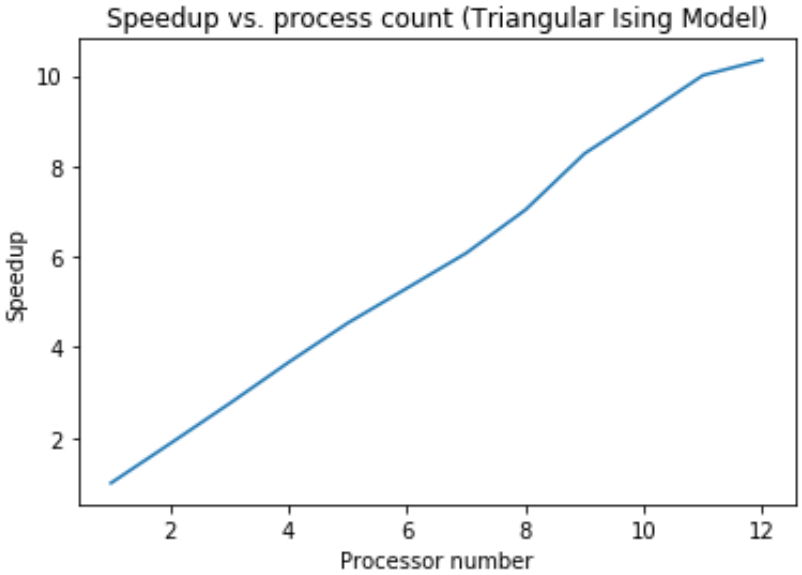


Figure 26: The speedup of the triangular lattice Ising model.

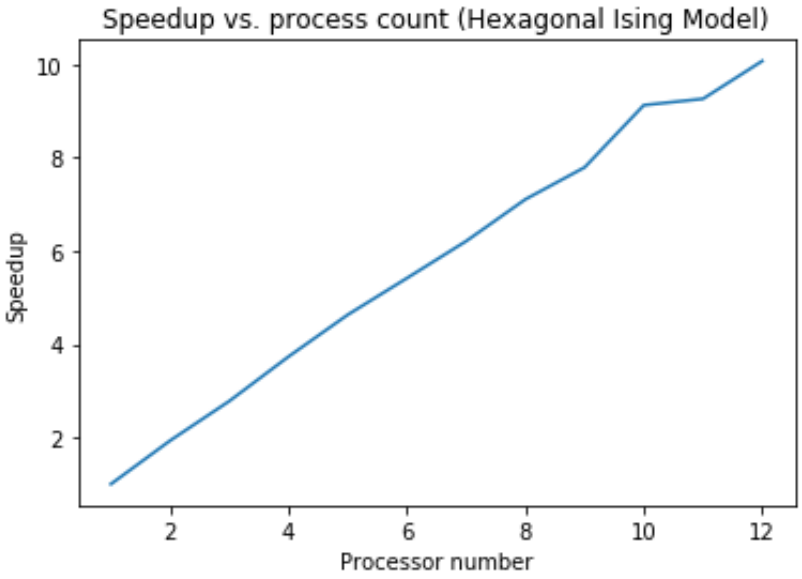


Figure 27: The speedup of the hexagonal lattice Ising mode.

As can be seen from the speedup graphs, the speedups for all three models are almost linear. This is a satisfactory result, as according to Ahmdal's law, speedups obtained by parallel code cannot be faster than a linear speedup. The result also confirms that the purely serial part of my program is close to negligible.

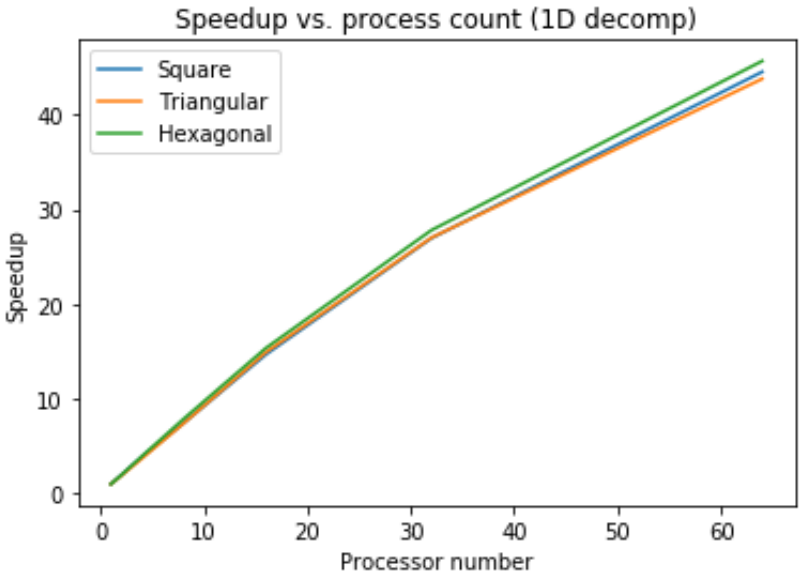


Figure 28: The speedup of the Ising model for all geometries is close to linear even for a large number of cores. The code shows good scalability.

To further test see the capabilities of my code for large-scale applications, I ran my simulations once again on Lonsdale on a varying number of cores between 1 and 64, using a 144x144 array. Despite greatly varying in runtimes (with the triangular model being approximately three times slower than the square lattice ising model), all the models showed nearly identical speedups.

4.10 Autocorrelation

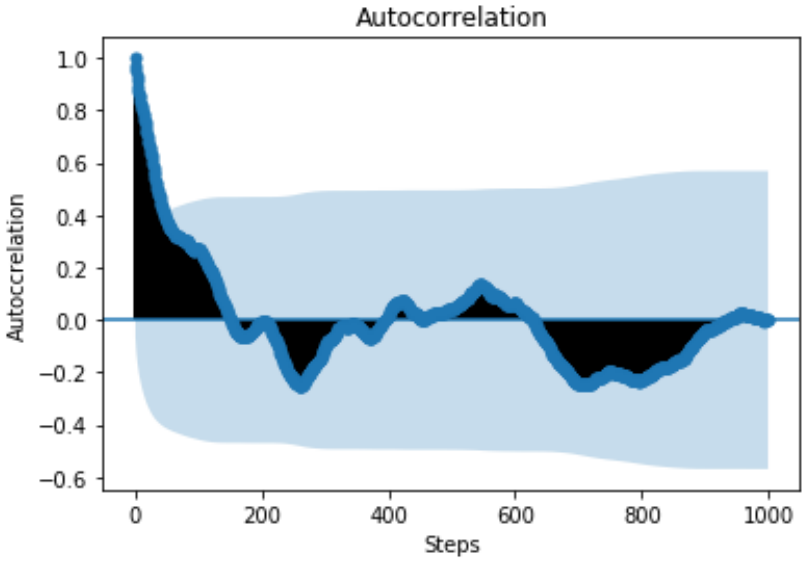


Figure 29: Autocorrelation of the square lattice Ising model

Testing the autocorrelation of the square lattice Ising model revealed an expotential decrease from step 0, which was the expected result. The autocorrelation then flactuated around 0 randomly.

5 Future Work

The Ising and the Potts model are very extensive and it is difficult to study them in great detail over a short amount of time. Given additional time, I would be interested in studying the phase transitions of the models for additional geometries, such as a prism lattice. i would also be interested in finding the behaviour of these models at higher temperatures, to see if any phase transitions occur.

For the Potts model, I would like to explore the phase transitions to higher q-state Potts model, to see how the phase transitions vary as the number of possible states is increased.

I would also like to extend my program to study more generalised models, such as the XY model, which is a continuous version of the Potts model. Another advancement of my code would be to employ algorithms other than the Metropolis algorithm, such as the Swedensen Wang algorithm or the Wolff algorithm. These algorithms have proved to have much better autocorrelations than the Metropolis algorithm.

Finally, if time was to allow for it, I would like to find the source of the disrepancies of the observables at high temperatures in my parallel and serial codes.

Another bug related to the parallel code was that in my 2D decomposition, when exchanging columns, send the full relevant ghost column to neighbour processes lead to the wrong results. However ignoring the last element of the ghost column during exchanges made the results much more in line with my serial code. I can't think of a possible reason behind this and given more time I would love to find out what is causing this.

6 Conclusion

Over the course of this project I

7 Bibliography

[1] https://stanford.edu/~jeffjar/statmech/intro4.html
Standford University Lecture notes on Ising Model

[2] https://www.nhn.ou.edu/~johnson/Education/Juniorlab/Magnetism/2013F-CuriePoint.pdf
University of Oklahoma Dept. of Physics and Astronomy,
Neil McGlohon & Nathan Beck, Tim Corbly & Richard Mihelic

[3] https://www.maths.tcd.ie/~dbennett/js/ising.pdf
"Numerical Solutions to the Ising Model using the Metropolis Algorithm", Danny Bennet

[4]
http://www.thphys.nuim.ie/CompPhysics/2021s1_lectureslides/MP468_2021s1_lectureslides04a.pdf
Maynooth lecture slides on the computational Ising model

[5] https://digital.library.unt.edu/ark:/67531/metadc693193/m2/1/high_res_d/522745.pdf

"Potts-model Grain Growth Simulations: Parallel Algorithms and Applications",
Steven A. Wright, Steven J. Plimpton, Thomas P. Swiler, Richard M. Fye, Michael F. Young,
Elizabeth A. Holm

[6]
https://www.ihes.fr/~duminil/publi/2015%20Currents%20developments%20in%20mathematics.pdf
"Order/disorder phase transitions: the example of the Potts model",
Hugo Duminil-Copin

[7] https://www.investopedia.com/terms/a/autocorrelation.asp
Autocorrelation Definition, Tim Smith

[8] http://www.emanuelschmidt.eu/physik/ising.pdf
"Monte Carlo Simulation of the 2D Ising model", Emanuel Schmidt

[9]
https://www.archer.ac.uk/training/course-material/2016/12/mpi_scaling_manc/Slides/Scaling.pdf
Archer supercomputer training notes

[10] MA5601 Lecture 1, Mike Peardon

[11] https://people.duke.edu/~kh269/teaching/notes/MetropolisExplanation.pdf
"Why does the Metropolis-Hastings procedure satisfy the detailed balance criterion?",
Kris Hauser