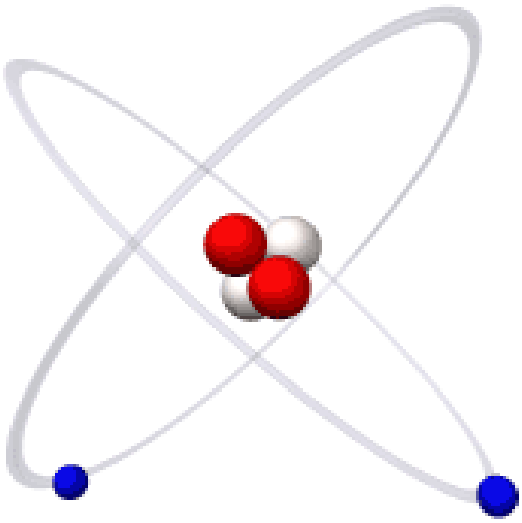

A Monte-Carlo Ising Simulation of Helium Mixing

High-Performance Scientific Computing CSCI 5576

Florian Rappl / Christoph Preis

May 2011

Contents

1 Introduction

Our goal is to write a simulation of ^{3}He and ^{4}He mixing. These mixtures are important for cooling systems below liquid helium temperatures. Our simulation is based on a hybrid Monte Carlo and Ising model. The Ising model was originally introduced to describe magnetism using a lattice of up and down spins.
By extending this model with another degree of freedom we can use it to obtain a phase diagram of ^{3}He, ^{4}He mixtures. Monte Carlo methods are based on the random generation of lattice configurations and an accept / reject decision based on for instance the conservation of energy.
In the second chapter we will discuss the physical properties of our problem and also talk about the statistical observables we will measure in our code.
In the third chapter we explain the main loop of our code and discuss some of the used MPI techniques in greater detail. Our code supports three modes of operation: A single core mode without any MPI calls, a normal MPI mode and another mode that makes use of the shared memory on the PSC SGI Blacklight machine. In the latter mode we split up the lattice between the 16 cores on a blade and furthermore run different lattices on different blades.
In the fourth chapter we present physical results and benchmarks of our application on the TeraGrid machines Frost, Blacklight and Trestles. For big enough lattices we observed excellent speedups and sometimes even saw super linear behaviour. The specifications of those machines have been provided in the HPSC class and can also be found on the TeraGrid website[9].

2 Theoretical background and methods

2.1 Monte Carlo methods

Monte Carlo (MC) methods are very popular for simulations that are too expensive to compute deterministically, i.e. may QCD simulations. A Monte Carlo algorithm generally consists of these steps (Fig. 2.1) [8]:

1. Make a random change to the system.
2. Evolve the system after certain rules (i.e. following the Hamiltonian or partition function).
3. Check for plausibility (for example conservation of energy) and accept or reject the change.
4. Repeat the algorithm several times.

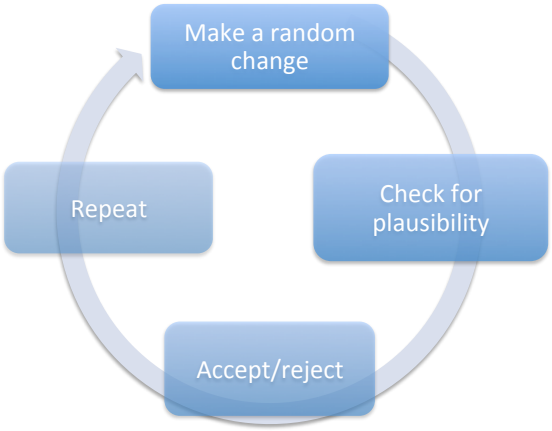

Figure 2.1: A rough outline of an iteration in a Monte Carlo simulation

A very simple example for a Monte Carlo algorithm as shown in fig. 2.1 would be the following one. As setup we take:

- $2N$ coins,
- a coin that shows head has value $v_i = +1$, else $v_i = -1$ and

- we only allow configurations that are $\sum(v_i) = 0$.

The algorithm would now be:

- Flip a random number of coins, then
- compute $a = \sum(v_i)$.
- If $a = 0$ we accept the new configuration, else we reject the change.
- Finally we repeat the algorithm as long as necessary.

In practical applications there is more work to be done than outlined by this simple algorithm. Some algorithms based on Monte Carlo are presented by Tom DeGrand[2]. Our project will illustrate this by building the Ising simulation of Helium using Monte Carlo step by step.

2.2 The Ising model

The Ising model is a very well studied lattice model used to describe physical systems such as magnetization and phase transitions. The Ising model is usually applied to a two or three dimensional lattice with N sites that have discrete states of freedom. Since the Ising model has first been applied to magnetism these are usually called *spins*. In our case every spin can take the values ± 1 and 0.
The Hamiltonian of the Ising model is

$$\mathcal{H} = -\sum_{\langle i,j \rangle} J s_i s_j - H \sum_i s_i, \tag{2.1}$$

where $\langle i, j \rangle$ denotes all pairs of nearest neighbors, J the coupling between these and H the external field. The magnetization is present in form of $\sum_i s_i$. If the coupling J is positive neighboring parallel spins (both -1 or $+1$) are favoured, since this leads to an overall smaller energy. If on the other hand the coupling constant is negative the system tends to form neighboring anti-parallel spins in order to decrease its energy. For finite temperatures this parallel or anti parallel ordering is however always disturbed by random spin-flips. The two dimensional Ising model was the first model of this kind to show a phase transition. It has first been solved analytically by Onsager[6].
A heath bath Monte Carlo algorithm for the Ising model can for instance be implemented like this[8]:

- Randomly pick a site i with coordinates (x, y, z),
- compute the sum m_i over all neighboring spins at the site i. Example: In a 2D lattice if all nearest neighbor spins are up $m_i = 4$. If three are up and one is down $m_i = 2$.
- Calculate the energies $E_\pm = \pm J m_i \pm H$ for the spin at the site i to be $+1$ or -1.

- Set the spin at the site i to ± 1 with probability P of

$$P_{\pm} = \frac{\exp(-\beta E_{\pm})}{\exp(-\beta E_{+}) + \exp(-\beta E_{-})}. \tag{2.2}$$

- Repeat.

More details about an Ising model simulation can be found at the summer school project website of the University of Basel[10].

2.3 A short introduction to Helium mixing

^{3}He-^{4}He-Mixtures are very important for cooling systems with temperatures lower than 1 K. The process where these mixtures are used is called dilution refrigeration. The physical background is that by mixing these isotopes of Helium energy is being consumed by the system - energy in form of temperature. In more detail, energy is required to transport the ^{3}He atoms from the ^{3}He-rich phase into the ^{3}He-poor phase. If the atoms can be made to continuously cross this boundary, they effectively cool the mixture. Because the ^{3}He-poor phase cannot have less than a few ^{3}He percent at equilibrium, even at absolute zero, dilution refrigeration can be effective at very low temperatures. The volume in which this takes place is called mixing chamber.
Even though this kind of system has been studied very well experimentally, the theoretical framework is still missing or incomplete. An approach in form of an Ising Model has been introduced by Blume et al. (1971)[5]. They propose a modified Ising Hamiltonian

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} s_i s_j - H \sum_{\langle i,j \rangle} s_i^2 s_j^2 + \Delta \sum_i s_i^2 - N(zH_{33} + \mu_3). \tag{2.3}$$

The spin can take the values $0, \pm 1$ in this model, assigning $0 \rightarrow ^3$He and $\pm 1 \rightarrow ^4$He. Thus as in the standard Ising model the average of the total spin is given by (where N denotes the total number of sites)

$$M = \frac{1}{N} \sum_{i=1}^{N} \langle s_i \rangle. \tag{2.4}$$

Since we have now two different types of *atoms* in the system we need to distinguish between them. We can obtain the number of ^{3}He, N_3, and ^{4}He, N_4, atoms by calculating

$$N_3 = \sum_{i=1}^{N} (1 - s_i^2), \tag{2.5}$$

$$N_4 = \sum_{i=1}^{N} s_i^2. \tag{2.6}$$

The Hamiltonian described in eq. 2.3 is not very useful to evaluate. Since the paper of Blume, Emery and Griffiths (1971)[5] only deals with $J \gg H$ we use $H \approx 0$. Our unit set will be $J = 1$,

and $\Delta = \mu_3 - \mu_4 \equiv -\mu$ in units of J. The later comes from our approximation $\mu_3 = 0$. Therefore our Hamiltonian looks like

$$\mathcal{H} = -\sum_{\langle i,j \rangle} s_i s_j - \mu \sum_{i=1}^{N} s_i^2. \tag{2.7}$$

Most statistics we are interested in are analogous to the standard Ising model. Next to the ones already mentioned (the internal energy of the system can be calculated by using the Hamiltonian from eq. 2.7, the *magnetization* of the system can be evaluated using eq. 2.4 and the particle number observables are already written down explicitly) we can use the specific heat C which is given as the variance of the energy, i.e.

$$C = \beta^2 \left(\langle E^2 \rangle - \langle E \rangle^2 \right) = \frac{\partial E}{\partial T}. \tag{2.8}$$

The usage of this observable will be very important for finding the critical line (β_c, μ_c) as well as detecting the tricritical point in that line, β_{tc}, μ_{tc}. We can also use C to measure the critical exponent ν. Another very important statistic will be the magnetic susceptibility, which can be evaluated by using

$$\chi = \beta \left(\langle M^2 \rangle - \langle M \rangle^2 \right) = \frac{\partial M}{\partial \beta}. \tag{2.9}$$

This statistic has a very deep meaning for physical research, because by evaluating plots and data of χ one will be able find the critical exponent γ.

2.4 Calculating statistics

Updating our system is just one (but very important) part of the problem. In order to get physical values (so called observables) out of it we need to follow the rules of statistical mechanics and calculate the observables after the iterations. In the end we will obtain variables that will be

- normalized to the value of one site or even
- normalized to a probability, i.e. from zero to one.

Since the project is not focused on the physics we will just state what kind of statistics we gather and how we calculate them in general. The usage or correctness of the gathered statistics will not be discussed in detail. In the next chapter we will describe the parallelization of calculating the statistics.

2.4.1 Probability for Helium 3 and Helium 4

We have states ± 1 for Helium-4 and 0 for Helium-3. Therefore we can say that all N sites can have either one of them and that

$$N_4 \equiv \sum_i s_i^2 \tag{2.10}$$

will get us the number of He-4 sites when we sum over all sites i, where s_i is the value ± 1 or 0 of the site. By performing

$$N_3 \equiv N - N_4 \tag{2.11}$$

we obtain the number of He-3 sites. To get the probability we just have to divide through N,

$$n_3 = \frac{N_3}{N}, \qquad n_4 = \frac{N_4}{N}, \tag{2.12}$$

which gives us the chance that any site is in either state. In our code we will actually compute

$$N_4 = \left(\frac{1}{R} \sum_r \sum_{x,y,z} s_{x,y,z}^2 \right), \tag{2.13}$$

where we have the number of repetitions (lattice configurations) R, summed over the configurations r and over all the sites on the x, y and z coordinates.

2.4.2 Energy of the system: The Hamiltonian

The Hamiltonian is the most important statistic of the system, since it is used to determine if we accept or reject a change of a state in the lattice. We therefore calculate

$$H = \left(\frac{1}{R} \sum_r \sum_{x,y,z} \left[\Delta_{x,y,z} + \mu s_{x,y,z}^2 \right] \right), \tag{2.14}$$

where we use the same notation as in eq. 2.13. We define the Δ over

$$\Delta_{x,y,z} \equiv s_{x,y,z} \times \left(s_{x-1,y,z} + s_{x+1,y,z} + s_{x,y-1,z} + s_{x,y+1,z} + s_{x,y,z-1} + s_{x,y,z+1} \right). \tag{2.15}$$

The expression in brackets will be used to determine the sum over the nearest neighbors and will therefore be outsourced as a function to avoid code duplicates and support extension and maintenance of the code.

2.4.3 The specific heat

The specific heat is variable which tells physicists a lot about the properties of the system. Splitting the computation up in different parts will be hard for this one. It is calculated over

$$C = \beta^2 \left(\left[\sum_r \frac{1}{R} \left\{ \sum_{x,y,z} \Delta_{x,y,z} + \mu s_{x,y,z}^2 \right\}^2 \right] - \left[\sum_r \frac{1}{R} \left\{ \sum_{x,y,z} \Delta_{x,y,z} + \mu s_{x,y,z}^2 \right\} \right]^2 \right), \tag{2.16}$$

where we use the same notation as in eq. 2.13.

2.4.4 The susceptibility

The (magnetic) susceptibility is also an interesting observable. It has the same statistical character as the specific heat and is calculated in a similar fashion. Instead of a part of the Hamiltonian we use the part for the magnetization which was donated as the Helium-4 particles. We have

$$\chi = \beta^2 \left(\left[\sum_r \frac{1}{R} \left\{ \sum_{x,y,z} s_{x,y,z}^2 \right\}^2 \right] - \left[\sum_i \frac{1}{R} \left\{ \sum_{x,y,z} s_{x,y,z}^2 \right\} \right]^2 \right). \tag{2.17}$$

Those are the statistics that are calculated and saved in the statistics evaluation file. We see that we do not need much communication between the nodes and blades. The details of our approach are explained in the next chapter.

3 Coding and MPI

In this chapter we will discuss the actual implementation of the physical model. Since the focus of the class was on parallelization we will briefly discuss the general operation of the program and then talk about the used MPI functionality in greater detail. Our information on implementing MPI was taken from the information given in the HPSC class and written in Pacheco's book[7]. For producing our makefiles we took the tutorial from Liz Jessup's book[3].

3.1 General operation

In this section the run of our application will be discussed. Our code will run with or without MPI depending on the compiler flag MPICOMM. If MPICOMM is set MPI will be initialized directly after the start of the program. In the initialize routine every process is assigned a global index and a lattice index. The lattice index reflects the position of the process in its lattice. Every process with lattice index = 0 will act as a lattice master and compute the statistics of its lattice. The process with global index = 0 will compute the statistics of its lattice and also collect global statistic form all lattices.
After MPI has been initialized the global master will run a command line parser. Here the user can enter the configuration for the next run, like lattice size, β iterations or number of repetitions. Alternatively these preferences can be read in from a configuration file. The configuration is broadcasted among all processes and the program enters the main loop. For every value of β and μ a number of warm up runs is performed to bring the lattice into equilibrium. The warm up use the same update routine as the actual iterations, but without recording any statistics.
Before the actual iterations are done an array to store the statistical data in is generated on each process. After every iteration the processes store the observed data of their sub lattice in this array. After the set number of iterations are performed all arrays are gathered and summed by the lattice masters. If the code runs more than one lattice the global master collects this data from all lattices and averages over it. Now new values for β and μ are set and the loop starts again with warm up iterations for this configuration. fig. 3.1 gives a graphical overview of our program.

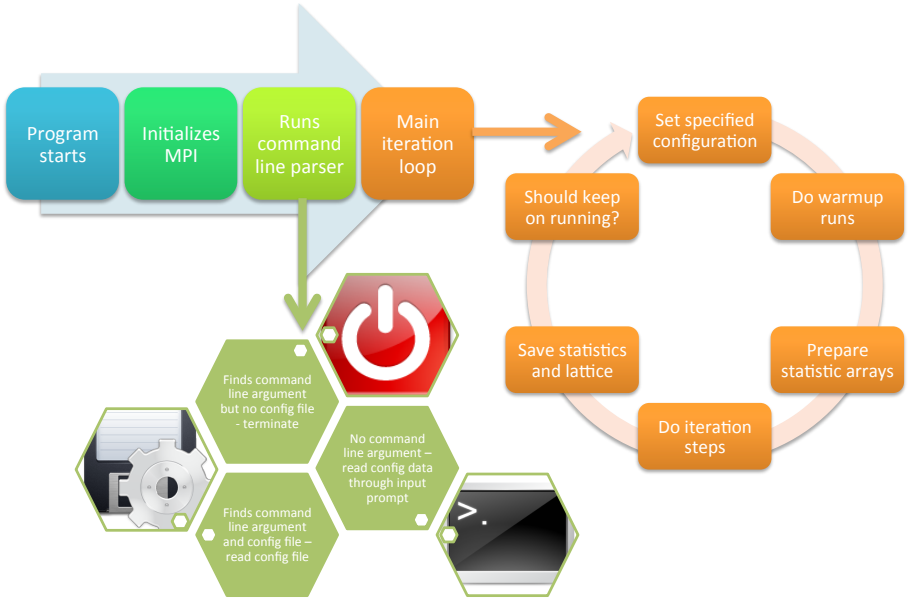

Figure 3.1: Flow chart of our programm if MPI is enabled. The orange circle represents the main loop.

3.2 MPI: Wrapping of MPI functions

Our code makes frequent use of MPI calls, for instance to broadcast to configuration data before the computation starts, to move columns and rows betweens different processes during the computation and to gather results to calculate statistics in the end. On the other side we want to be able to run our code without any MPI calls if we do not have access to a parallel computer and for debugging. In order to be able to do this we wrapped all our MPI calls in communication functions. The syntax for these is COMM_[*Name of corresponding MPI call*]. The call functions use flags to determine if MPI should be enabled for the current build. If the code runs on a single core without MPI communication is often obsolete and the COMM_ calls are simply empty.
One of the MPI functions we wrapped is the simple MPI_Send.

```
1 int COMM_Send(void* message, int count, int dest, int tag)
2 {
3 #ifdef MPICOMM // execute MPI code
4   return MPI_Send(message, count, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
5 #else //no communication needed
6   return 0;
7 #endif /*MPICOMM*/
8 }
```

3.3 MPI: Derived data types

In MPI broadcast and reduce calls we use derived data types to pack a number of different properties into one variable and then send them out in one message. This is for example done before the computation starts, when the root process collects the configuration data for the calculations. This configuration data contains i.e. the lattice size in x, y and z direction, the number of iteration steps of μ and β, the number of warm up iterations. These parameters are either read in from a file or can be entered by the user in the command line. Since all processes need to know these we are distribution them using a wrapped MPI broadcast of a custom structure *input data*.

```
1  struct inputData
2  {
3    int nx, ny, nz; /* lattice dimensions */
4    ...
5  };
6
7  struct inputData data;
8
9  void distributeInputData()
10 {
11 #ifdef MPICOMM // Broadcast input data. Process 0 sends, all other processes receive the data.
12   MPI_Bcast(&data, sizeof(struct inputData), MPI_CHAR, 0, MPI_COMM_WORLD);
13 #else // nothing to distribute
14 #endif /*MPICOMM*/
15 }
```

3.4 MPI: Custom communicators

In order to maximize parallelism in our program we split up our problem in two ways. We have a number of processors compute on one lattice and also different blades of processors compute on different lattices. In this setup we mainly need two types of communication: One among processes working on one lattice i.e. for exchanging ghost rows or computing statistics on the lattice and another one for inter blade communication to average over all lattices.
For intra blade communication we split up MPI_COMM_WORLD into i new communicators MPI_COMM_BLADE. Each of these contains all processors working on one lattice using MPI_COMM_Split and the lattice index as a split key. For inter blade communication we made one more custom communicator MPI_COMM_MASTERS that contains the root process of every lattice. Split key for this communicator is the *nodeIndex* variable, which contains the MPI rank of a process in its lattice.

```
#ifdef MPICOMM
    MPI_Comm MPI_COMM_BLADE;
    MPI_Comm MPI_COMM_MASTERS;
#endif /*MPICOMM*/

// Blade communicator: All processes on one blade in a communicator
MPI_Comm_split(MPI_COMM_WORLD, latticeIndex, globalIndex, &MPI_COMM_BLADE);

// Master communicator: All blade masters in a communicator.
MPI_Comm_split(MPI_COMM_WORLD, nodeIndex, globalIndex, &MPI_COMM_MASTERS);
```

3.5 MPI: Broadcast and Reduce

As already mentioned we use MPI broadcasts to distribute data among all processes. To gather data from different processes we use wrapped MPI_Reduce calls. The example below shows part of the computation of a lattice statistic. We use our custom MPI_COMM_BLADE communicator to send data from all processes of one lattice to the root process of the lattice. This lattice master sums over the collected vectors and returns the values to the statistics routine.

```
void vectorsReduceSum(double* vec, double* sum, double* sumsq)
{
#ifdef MPICOMM
  double out; /* reduce vectors from all sites */
  MPI_Reduce(vec, &out, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_BLADE);
  if (nodeIndex == 0) /* sum on root node of each blade */
  {
    *sum = out;
    *sumsq = out * out;
  }
#else /* execute single processor code */
  *sum = *vec;
  *sumsq = (*vec) * (*vec);
#endif /*MPICOMM*/
}
```

3.6 Communication in detail

In order to obtain the best performance possible it was necessary to reduce communication cost. An usual approach to Monte Carlo would be to pick a site randomly. While this strategy does not make any difference in serial mode, we would suffer from the huge communication

cost in parallel. The communication cost comes directly from the required amount of ghostsite-exchanges, which have to occur after each iteration step. In order to avoid this we needed a different strategy.

3.6.1 Red/Black checkerboard

Our strategy is a so called checkerboard approach. We give every site a parity, which represents red and black or in other words even and odd. The parity is given by the coordinates of the site, i.e. the parity is $(x + y + z)$ mod 2. This is illustrated in fig. 3.2.

A red (even) site will always just need the information of the black (odd) neighbors and

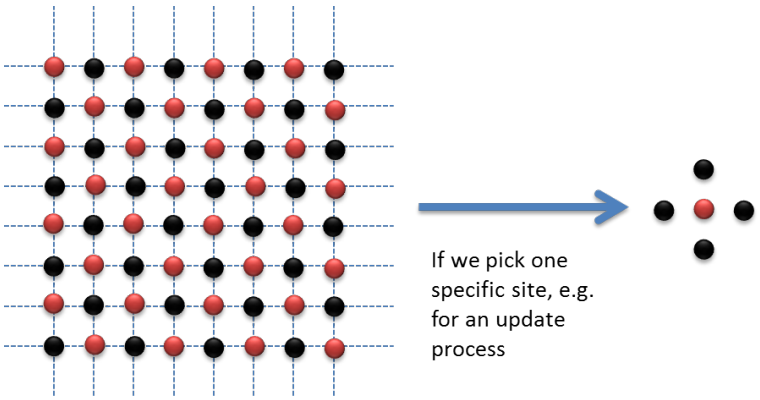

Figure 3.2: The red/black checkerboard illustrated in 2D. Every site is surrounded by sites with the opposite parity.

as a black site will just need the information of its red neighbors. We can therefore save communication by just iterating through all the red sites without exchanging boundary site information after each iteration. Once we did all red sites we will exchange boundary information and iterate over all the black sites. This is continued until we the configured amount of iterations. Therefore the communication cost is reduced by a factor of (with the small volume v, the total volume V and the number of processors p)

$$\frac{n_x n_y n_z}{2} = \frac{v}{2} = \frac{V}{2p}. \tag{3.1}$$

The bigger the problem the more communication we save using this algorithm. The downside is that when p reaches $V/2$ we have actually no more gain.

3.6.2 Sending and receiving ghostsites

For sending and receiving ghostsites we needed to write a proper algorithm. Our main purpose was to avoid a deadlock, which occurs when all try to send or receive at the same time. Therefore every send and receive step had to be split up in two parts. Since we have six of those steps (we have a left, right, top, bottom, up and down ghostarea) we have to do 12 steps in total.

First all the even processors in the current direction will send, while the odd processors receive, then the other way round. For determining if a processor is even or odd in the current direction we used that $q_x = 1$, $q_y = p_x$ and $q_z = p_x p_y$. Therefore we just calculate

$$\frac{p}{q_i} + p \bmod q_i, \tag{3.2}$$

where p is the current processor ID to determine the parity in the current direction i. Doing that we will find the parities as shown in the example in fig. 3.3.

This algorithm is now able to prevent a possible deadlock and also defines clearly who is sending

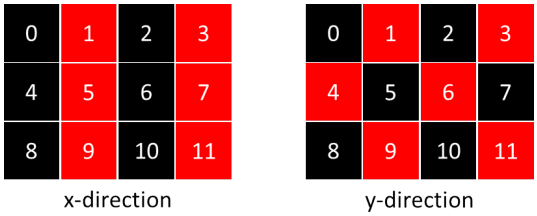

Figure 3.3: The parity of the processors in x-direction on the left and y-direction on the right. The black color denotes even, while red denotes odd.

and receiving to who. We just have to follow $p \pm q_i$ to send and receive in the i-th direction.

Therefore the receiving processor always determines the processor it receives from in the current

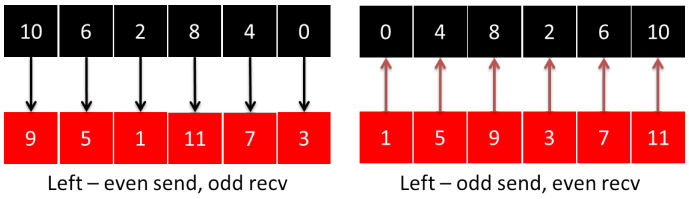

Figure 3.4: A send/recv process of the left ghostsites with the example processor grid from above.

direction as it would determine the processor to send to in the opposite direction. fig. 3.4 shows this process with the processor grid from fig. 3.3. In the case where only one processor is in the corresponding direction we just save the ghostsites we would send from the left side as if they would have been received from a processor that is located on the right side of the current processor. Therefore we can ignore the ghostsite property when accessing neighbor information - we always behave as if we want to access rows that are stored in order to obtain periodic boundary conditions.

4 Results

While building this project we had two main goals in mind. The first was to verify the statistics that Blume et. al[5] had obtained using Mean-Field-Theory. This goal includes showing that this system has a so called *tricritical* point as well as obtaining a phase diagram as known from Thermodynamics. The phase diagram is shown in fig. 4.1. The tricritical point has been a matter of huge interest in recent physics research. I. e. Lee [4] wrote a paper about a connection between super symmetry and the tricritical point in any Ising model. However this is no new topic, since Badke[1] did already some research on this - about 20 years earlier.

The second goal was to make some investigations about the parallel performance of this code.

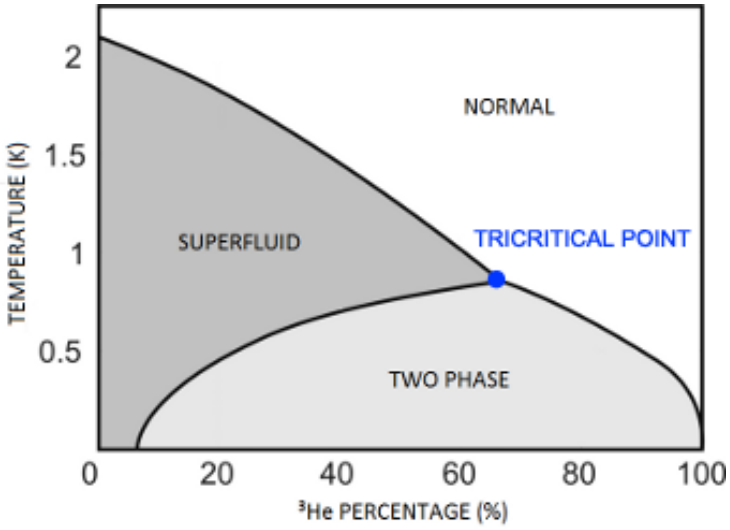

Figure 4.1: Phase diagram of ^{3}He-^{4}He mixing

It is well known that Ising models are in general *embarrassingly parallel* algorithms. That means that we should at least get a linear speed-up for most configurations and on most systems. We already know that PSC SGI Blacklight is a really fast system and that we designed a part of our program especially for this system makes us in a way biased towards it. However we are still interested in the performance difference between **PSC SGI Blacklight** and **SDSC Trestles**. Also NCAR Frost will be evaluated in order to show a comparison.

4.1 Physical

With the statistics described in ch. 2 we are able to determine if we reached our goal. In order to read out the statistic output, which has been saved in a XML file, we wrote a program in C#. This program had a nice interface where we could perform modifications on the data as well as filtering it. In the end we could export the statistics to various file formats. A screenshot of the program is shown in fig. 4.2.

Even though we could export the data as CSV or Excel file we did pick QtiPlot as our favored

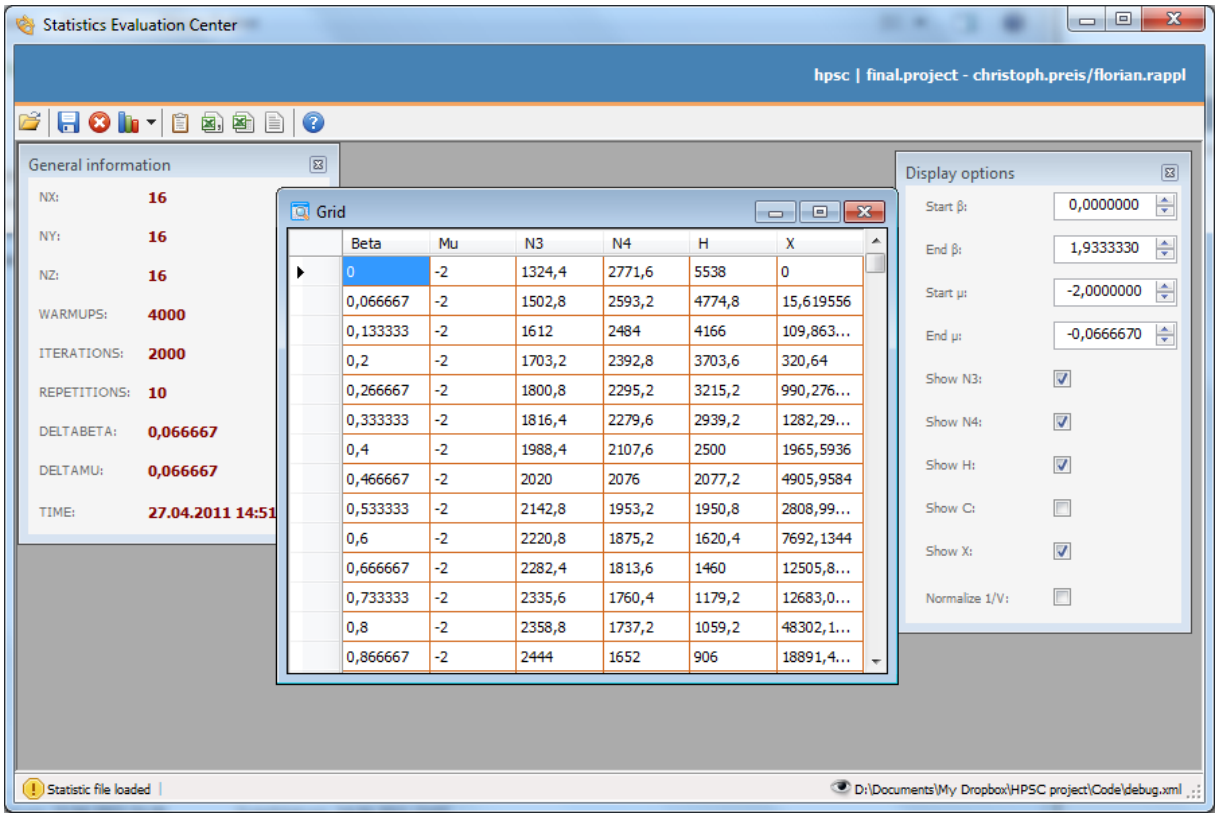

Figure 4.2: A screenshot of the XML file parser program

evaluation program. The reason is that QtiPlot offers us 3D-plots. Those are required in order to work with statistics obtained in a 2D parameter space.

We were mostly interested in the energy, i.e. the Hamiltonian, H as well as the specific heat C and the Helium-3 particle number N_3. While H is only important due to curiosity (and of course from a physical point of view), C and N_3 are the statistics that can be compared to the Blume[5] paper.

The configuration that has been used in order to get all these plots was a 32×32 lattice (2D in order to follow the basic paper of Blume et al.[5]) with 5000 warm-up runs and 10000 iterations each time we measure something. In order to get smooth results we set the repetitions to 20

and increased β and μ in steps of 0.01.

Starting with the Hamiltonian we see the tricritical point as well as the critical line, which separates one phase from another. This change is marked by a huge jump in the energy level in the μ plane as well as in the β plane. The two plots are shown in fig. 4.3.

We can already see that something is going on in this system. The Hamiltonian shown in fig.

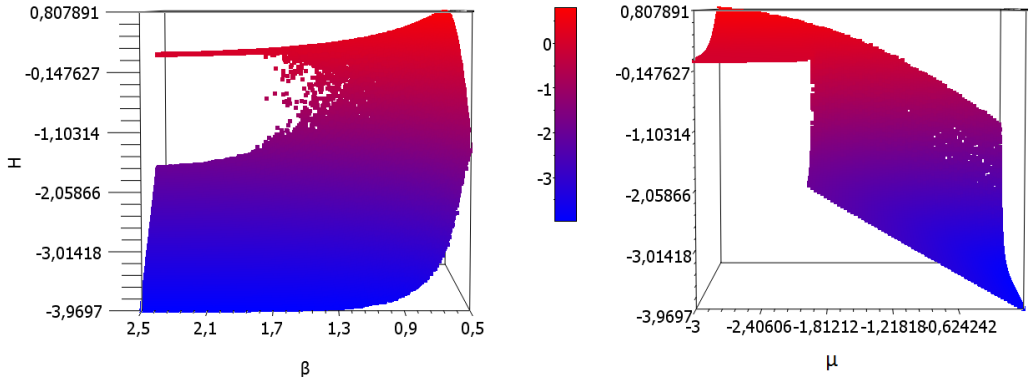

Figure 4.3: The Hamiltonian in the μ plane on the left and β plane on the right

4.3 confirms parts of the paper where our project is based on. We see that we have not only one kind of phase transition but two. One that is going steadily, which is called a second-order phase transition, and one that is in form of a huge jump. The later one is called a phase transition of first-order. Somewhere between those two transitions we have the tricritical point.

We can see evidence for this point in the β plane of the Hamiltonian. Here we see some splitting

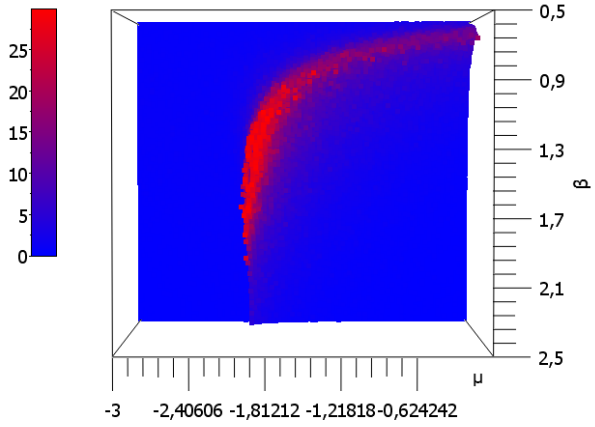

Figure 4.4: The specific heat viewed from the bird's eye view of the parameter space

up with some legacy points between the jump. To determine the tricritical point more exactly we have to use the specific heat C. A 3D plot of the specific heat can be seen in fig. 4.4. In order to find the tricritical point we need to search the maximum of C. In a 3D plot this can

be easily done by coloring and a bird's eye view perspective.
Therefore the color is marking the height of the 3D plot. The more red the surface is the higher the specific heat. On the highest point we do have basically infinity, i.e. that is were the tricritical point is supposed to be. When we compare this point with the splitting up of the Hamiltonian as seen in fig. 4.3 we recognize the same values just more exactly available.
This already confirms a lot of the calculations that Blume et al. did. As a last point we wanted to achieve the phase diagram as obtained by experimental physicists around the world.

The answer that we can give to this riddle is obviously: **Y**es, we can reproduce this phase

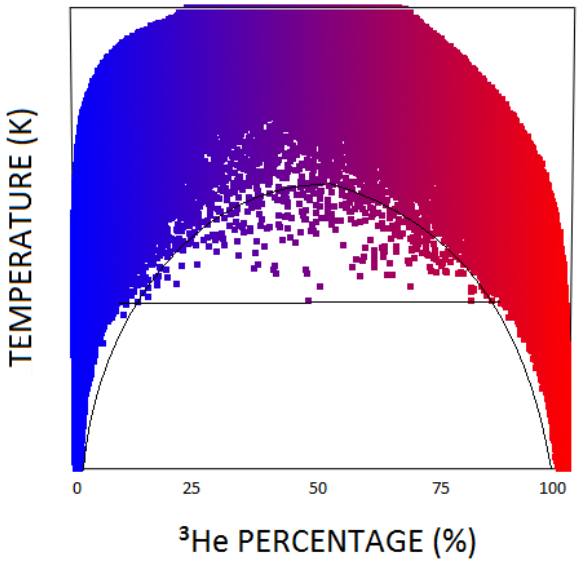

Figure 4.5: Our simulated phase diagram of an Helium-3 with Helium-4 mixing process.

diagram and a lot more exact that Blume et al.[5] could do. Of course the comparison is a little bit unfair taking into consideration that the method of Blume et al. was Mean-Field-Theory (an approximation), while we were doing an Ising simulation (the reality). If we now compare our diagram in fig. 4.5 with the experimental one shown in fig. 4.1 we can see a lot of features of our simulation:

- The two phase state is split up in a meta-stable and an non-stable region.
- Due to the coloring we see a smooth transition from super-fluid to the normal state of the mixing combination.
- We can actually determine how likely a meta-stable phase state is.

So we did have some success obtaining physical results from our project.

4.2 Computational

Parallelizing an Ising code is a thankful job for every high-performance program coder. It is one of those problems that are easy parallelizable and always benefit a lot from running on multiple processors. The communication cost is not very high since we were able to perform some tweaks and luckily have a local action - compared to a non-local action in lattice Quantum Chromodynamics, where lots of communication is required.
We evaluated the gained speedup we could achieve compare to the single core implementation on Frost, Trestles and Blacklight. Since our jobs typically run on less than 512 cores Frost's debug queue allowed very quick benchmarking. On Blacklight and Trestles the queues were longer, but these machines also produced more interesting output, since they have enough memory to run big 3D lattices in a reasonable amount of time. On Blacklight we also made use of the shared memory on blades, which should allow a very very fast ghost row exchange.

4.2.1 Frost

On Frost we benchmarked our MPI code on a number of different lattice sizes, which are listed in table 4.1. The computation was performed using n = 2, 4, 8, 16, 32, 64, 128, 256 and 512 cores in co processor mode. We measured the computation time using MPI_Wtime. The jobs ran between 5 minutes and 2 hours, depending on the problem size.
Fig. 4.6 shows the speedup on Frost compared to n = 2 cores. As expected for bigger lattices

Size	Dimensions	Number of nodes	Size in memory
small	64 x 64 x 1	4096	$\approx$ 80 kB
medium	128 x 128 x 1	16384	$\approx$ 320 kB
big	256 x 256 x 1	65536	$\approx$ 1.3 MB
huge	512 x 512 x 1	262144	$\approx$ 5 MB
huge 3D	64 x 64 x 64	262144	$\approx$ 5 MB

Table 4.1: Lattice sizes and node numbers.

the computation speed up scales better with an increasing number of processes. For big lattices we see a weakly super linear behaviour for 16 and 32 processors. For a huge lattice Frost shows a highly super linear behaviour. This effect can be explained by looking at Frost's cache. Frost has 256 kB L2 cache per processor. So a medium lattice can still fit entirely in the cache of two cores, whereas we expect an increasing number of cache misses for big and huge lattices.

We were also interested in how 3D lattices scaled compared to 2D ones. To do this we ran our program using a huge 2D lattice and a 3D lattice with the same number of nodes. Both benchmarks scaled super linear in the beginning as expected from our previous results (see Fig. 4.7). For process counts greater than 64 however the 3D lattice scaled considerably worse

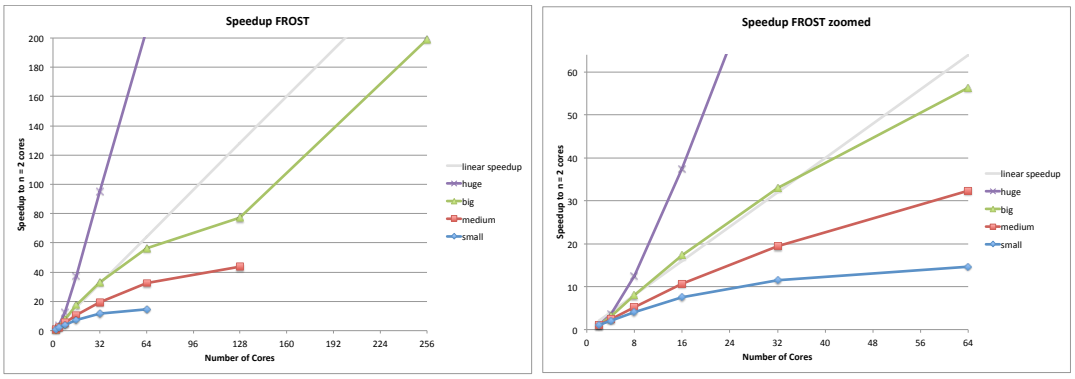

Figure 4.6: Frost benchmarking on different lattice sizes. The right side is zoomed in around zero. Bigger lattices scale better with more processors.

compared to the 2D case. For 128 or more processes we even saw a performance drop. This effect is probably due to the ghost row exchange. Whereas we mostly only need to exchange rows in two directions in the 2D case, in the 3D case we will need to send ghost rows in four or even six directions.

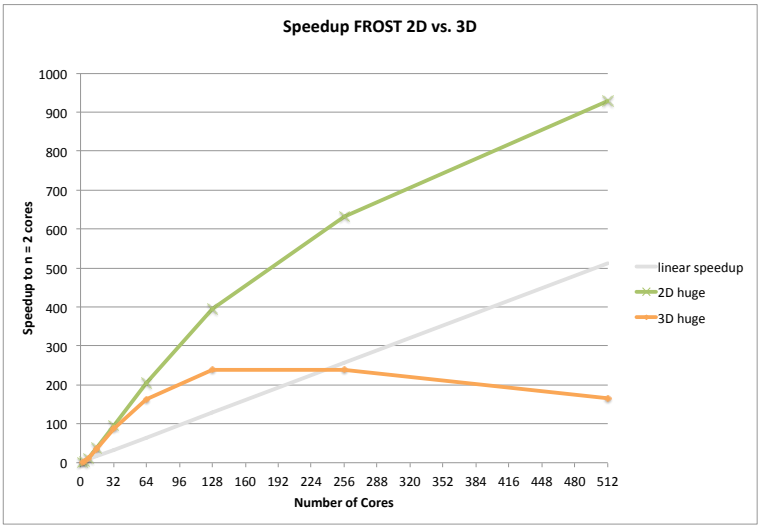

Figure 4.7: 2D versus 3D huge lattices on Frost. The 2D lattice shows a greater speedup, probably due to less communication cost in ghost row exchanges.

4.2.2 Blacklight and Trestles

Blacklight and Trestles are machines that have far more memory than Frost. Those computers also have faster CPUs and more cache-memory. While Blacklight has a gigantic amount of shared memory between the 16 cores of each blade, Trestles has a decent amount of memory

with 2 GB/core, i.e. 64 GB overall.
In order to have a fair test we built the `hpsc_mpi` version of our code on both machines. This means that we did not use the blade feature at Blacklight. We decreased the number of warmpus and iterations according to the size of the sublattices. This means that a run with 128 processors had only 1/4 of the iterations that a run with 32 processors had to go through.
We can see in fig. 4.8 that Blacklight always shows a better performance than Trestles. Both seem to scale down when we reach $N_z = p$. This has already been observed with our Frost benchmarks. In fig. 4.7 we've experienced a dramatic decrease in performance when the number of processes p was bigger than the Z dimension N_z. This is related to a lot more communication due to:

- More processors means less volume per processor. This means the scaling $V/2p$ which has been introduced in the red/black checker board does give us less benefit.
- Splitting up the lattice in two dimensions results in 2 more communication step per exchange ghost sites call. Therefore the communication cost per ghost sites exchange has already been doubled.

We are also able to see that the shared memory of Blacklight is the main reason for the advantage of the machine from the PSC over the SDSC Trestles, since both have nearly the same slopes after the starting point which has been picked at exactly the maximum amount of processors at one Blade.

We also wanted to give some insight about Blacklights performance using the Blade mode.

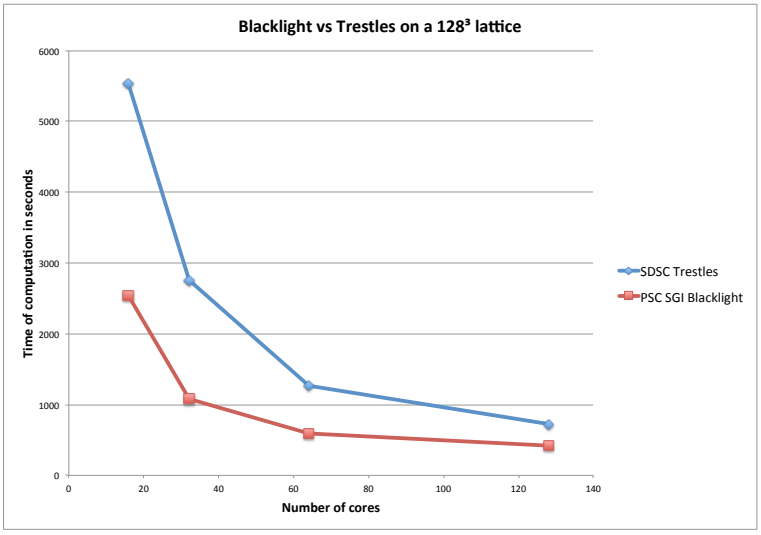

Figure 4.8: The total time for a sample computation with a 128^3 lattice on Blacklight vs Trestles. The number of warmup-runs and iterations have been adjusted to fulfil $128^3/p$ processors.

However this was not possible due to an over usage of Blacklight from other teams. Predicting

those results is fairly easy since a Blade mode would stick at 16 CPUs or 1 Blade and gain a speedup that goes fairly linear with n Blades if the number of Blades is a prime factor of the number of Blades. In our test this would have been the case since the number of repetitions was set to 20, which has primefactors of $2, 2, 5$. So 32 and 64 processors would have scaled linearly, 128 would have scaled worse but still better than 64.

4.3 Conclusion

We could confirm the results that Blume, Emery and Griffiths[5] obtained. Their work is quite impressive, because their calculations were done in an approximation of the system they proposed. That we actually see the same behavior as they predicated combined with a far better accuracy does work in favor of their system and the physics that is behind it.
We also did see that parallelizing an Ising model can give us huge benefit. However we always have to take in mind the lattice size as well as the processor count when we start parallel jobs. A really big processor count makes only sense if the lattice is big enough. Having a lot of processors available can therefore only mean running on gigantic lattices.
In physics we always want to have $N \to \infty$ as close as possible. Therefore setting a bigger lattice size is always an option that should be considered when enough processors are available. The lattice size should be big enough in order to work on splitting one dimension only (e.g. the Z dimension - therefore we only have to send two ghost areas in $X - Y$ dimension - meaning we have only the *up* and *down* send and receives - ergo four rounds in total) and still benefit from the red/black checker board in form of a large enough $V/2p$.

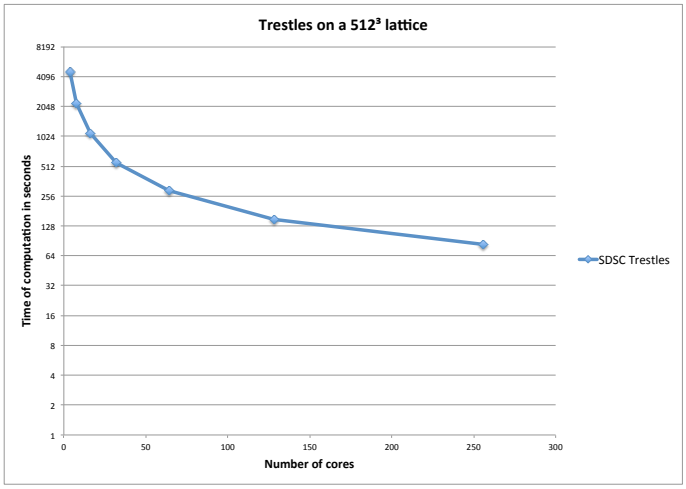

Figure 4.9: Once we reach the lowest limit $\min(N_x, N_y, N_z)$ we stop gaining a computational speedup. The time has been taken for getting 100 values with 5 repetitions.

5 Summary

With our project we hope to show what is important in High-Performance Scientific Computing. On the one side it is important to optimize the code for a specific machine. This is done in order to get also the last bit of performance out of that machine. On the other hand it is also very important to have the code in a general form so that it is easy to port to another machine. Due to our macros and overall code-design we hope to have succeeded in this matter.

Those goals were not the only ones we had in mind when we picked this kind of project. To have an *embarrassingly parallel* algorithm is something everyone would love. But our tests showed that even such algorithms have to run with the right parameters in order to gain speed and performance by using them with a lot of processors. The real physics world is parallel - that's why we have the super-linear speedup in most cases. However the real world does not have communication problems, since communication is most of the time done at the speed of light and without latency.

We showed how we can reduce communication and under what circumstances the cost of communication is overpowering the benefit of a huge processor count. We also showed that our code is actually producing physical outcome. This is another goal we wanted to achieve with our project. Just parallelizing a code is not enough if the parallel algorithm makes the outcome look different from the serial implementation.

Overall we can definitely say that we did not only learn a lot by performing this project, but also used a lot of what we learned in class. We used communicators, parallel file-IO, collective communication, derived datatypes and the exchange of ghost-rows. Our project will be placed online where it can be used for developing all different kind of parallel (or serial) Ising simulations. The code is very flexible and was thought as kind of a demonstration how to use it as a framework for building (parallel) Monte Carlo simulations.

Bibliography

[1] R. Badke. A Monte Carlo Renormalisation Group study of the Two-Dimensional discrete cubic model: A Hint for Superconformal invariance. *J. Phys. A*, **20**:3425, 1987.

[2] T. DeGrand and C. DeTar. *Lattice Methods for Quantum Chromodynamics*. World Scientific Publishing, Singapore, 2006.

[3] E. R. Jessup G. Domik, L. D. Fosdick and C. J. C. Schauble. *Introduction to High-Performance Scientific Computing*. MIT Press, Boston, 1996.

[4] Sung-Sik Lee. Emergence of Supersymmetry at a Critical point of a lattice model. *Phys. Rev. B*, **76(75)**:103, 2007.

[5] V. J. Emery M. Blume and R. B. Griffiths. Ising Model for the λ-Transition and Phase Seperation in ^{3}He-^{4}He Mixtures. *Phys. Rev. A*, **4(3)**:1071, 1971.

[6] L. Onsager. Crystal Statistics. I. A Two-Dimensional Model with a Order-Disorder Transition. *Phys. Rev.*, **65**:117, 1944.

[7] P. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann, Massachusetts, 1996.

[8] James P Sethna. *Entropy, Order Parameters, and Complexity*. Calrendon Press, Oxford, 1st edition, 2009.

[9] TeraGrid Support. Information on TeraGrid computers, 01.05.2011. *https://www.teragrid.org/web/user-support/compute_resources*.

[10] L. Witthauer and M. Dieterle. The Phase Transition of the 2D-Ising Model, 01.05.2011. *http://quantumtheory.physik.unibas.ch/bruder/Semesterprojekte2007/p1/*.