Institute for Theoretical Physics
University of Cologne

Prof. Dr. Simon Trebst
Peter Bröcker, Johannes Helmes

Computational Many-Body Physics

Assignment 2

Summer Term 2015

website: http://www.thp.uni-koeln.de/trebst/Lectures/2015-CompManyBody.shtml
due date: Monday, May 11th, 18:00 - send solutions to helmes [at] thp.uni-koeln.de

5. Parallel programming with MPI

Programming technique

All commands that we want a computer to do are translated to binary operations which are performed by the computer's "brain" — the central processing unit (CPU). If a computer has more than a single CPU-core, commands can be executed in parallel. Nowadays not only high performance computers (HPC) but also desktop machines and laptops do have more than one cpu — typically four. If our computational problem can be divided into independent tasks, we can speed up the cpu time needed by distributing these tasks to different cpu-cores.

A special variant of parallel computing is implemented by so-called *message passing* which means that every process has its own data space in the main memory but it can communicate and exchange data with other processes. In the following we will present a standard called message passing interface (MPI) which defines a bunch of functions that are useful for the communication between processes.

In order to run MPI programmes on your computer you first need to install either MPICH or OpenMPI and second the python library mpi4py (follow these instructions).

Let us straightly dive into python's MPI interface mpi4py by looking at the following example:

```
# Import the MPI library
from mpi4py import MPI

# Your connection to the MPI world
comm = MPI. COMM_WORLD

# Which ID has your process
rank = comm . Get_rank ()

print " Hello World ! I am process #", rank
```

You can run this example by typing

```
mpirun -np 4 python mpi_hello_world.py
```

on the command line. You see that the command `python mpi_hello_world.py` is executed four times. The number of processes is specified by setting the flag `-np`. Moreover, `mpirun` provides the the communication functionality between processes and to this end assigns a unique number (ID) to every process. The above example queries and prints this ID.

In a second example, we illustrate the communication features of MPI. Download and run the script mpi_send_recv.py using different values for `-np`. What happens?

6. Metropolis Algorithm for the Ising Model *10 points*

In this second exercise we will study the thermal phase transition that occurs in the two-dimensional Ising model, the first example of a dynamic system. The Ising model is defined by the Hamiltonian

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

where the spins $\sigma_i = \pm 1$ correspond to “up" and “down" spins and the sum runs over all pairs of nearest neighbor spins. We will consider the ferromagnetic case with $J > 0$.

For the underlying lattice we will consider a square lattice of linear extent L and $N = L \times L$ sites. In order to minimize finite-size effects we will use *periodic* boundary conditions, which results in a lattice with a total of $2N$ bonds.

To identify and characterize the thermal phase transition we will investigate thermal averages for a number of observables, which are generally defined as

$$A(T) = \frac{1}{Z} \sum_i A_i \exp(-\beta E_i)\,,$$

where Z is the partition function of the system, the sum runs over all possible spin configurations i, A_i and E_i are the values which the observable A and energy E have for a given configuration i, and β is the inverse temperature $\beta = 1/(k_B T)$.

We will calculate these thermal averages via Monte Carlo sampling for a range of temperatures $T = 0.1, 0.2, \ldots, 4$ (where we fix units by setting $J = 1$) and system sizes $L = 16, 32, 64$.

1. Implement a **single spin-flip Metropolis algorithm** for this 2D Ising model. We have provided a python skeleton that helps structuring your code.
2. Plot Monte Carlo averages of the **magnetization** $M = \sum_i \sigma_i$ for the full temperature range. Perform measurements only after an initial set of – say – 10,000 thermalization sweeps where one sweep corresponds to N attempted spin flips. After this thermalization phase, perform one measurement for every sweep.
3. Generate a visualization of the dynamics of the Monte Carlo algorithm by drawing an animation of the spin configuration change in sweeps. You could use two differently colored squares for spin “up” and “down”, see Fig. 1. Run this animation for $T = 2.0, 2.26984, 3.0$. A possible implementation for the animation using matplotlib is prepared in the skeleton. Simply uncomment the indicated parts of the code.
4. Plot Monte Carlo averages for the **Binder cumulant** of the magnetization $U = 1 - \frac{\langle M^4 \rangle}{3\langle M^2 \rangle^2}$

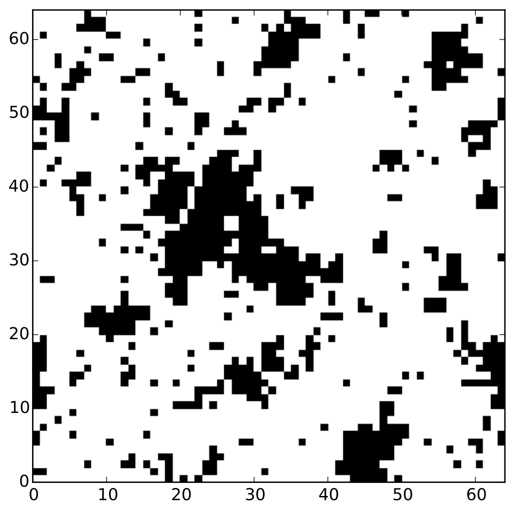

Figure 1: Visualization of a spin configuration of a two-dimensional Ising model.

for the full temperature range.

5. Plot Monte Carlo averages for the **energy** E for the full temperature range.
6. Plot Monte Carlo averages for the **specific heat** C_v, which you can estimate either by (numerically) calculating the derivative dE/dT of the energy curve above, or more directly via $\langle C_v \rangle = \beta^2/N(\langle E^2 \rangle - \langle E \rangle^2)$ by measuring Monte Carlo estimates for E^2.
7. From the two results plotted above can you reproduce the estimate of the thermal phase transition $T_c = 2/\ln(1+\sqrt{2}) \approx 2.269186$ for the square lattice?
8. Implement a **cluster update**, such as the Wolff algorithm explained in the lecture.
9. *Optional exercise:* For a fixed system size $L = 32$ perform $2^{16} = 65536$ measurements of the energy at the thermal transition temperature $T = 2.269186$ for the single spin-flip algorithm and the cluster update algorithm.
   For both sequences perform a **binning analysis** of the sampled energies – for which algorithm does the error converge? Measure the integrated autocorrelation time of the energies in order to determine the dynamical exponent. You can either do it by using the autocorrelation function $C_E(\Delta) = \frac{\langle E_t E_{t+\Delta} \rangle - \langle E_t \rangle^2}{\langle E_t^2 \rangle - \langle E_t \rangle^2}$ (averaging over k) and summing up

$$\tau_E^{(int)} = \sum_{\Delta=1}^{\infty} C_E(\Delta) \tag{2}$$

   or by estimating it from binning mean values

$$\tau_E^{(int)} = \frac{1}{2}\left(\frac{\Delta\overline{E}^{(\infty)}}{\Delta\overline{E}^{(0)}}\right)^2 - \frac{1}{2}. \tag{3}$$

   In fact, the largest possible binning level is $l = 16$ in this example. Try to estimate the dynamical exponent z from $\tau_E^{(int)} \propto L^z$ for the Metropolis and for the Wolff algorithm.