FYS3150 – Project 4

Kristian Gregorius Hustad (`krihus`)

November 15, 2016

> Nobody actually creates perfect code the first time around, except me.
>
> *Linus Torvalds*

Abstract

In this report, ...

NOTE: All programs and derivations used in them were made in collaboration with Jonas Gahr Sturtzel Lunde (`jonassl`).

GitHub repository at `https://github.com/KGHustad/FYS3150`

1 Introduction

[1]

We aim to study the Ising model in for a square $L \times L$ lattice with periodic boundary conditions.

WRITE MORE

Methods are derived in section 2, implementation considerations and results are given in section 3, and finally conclusions are drawn in section 4.

2 Discussion of methods

We have the following expression for the energy of a given state, i,

$$E_i = -J \sum_{<kl>} s_k s_l \tag{1}$$

and its mean magnetization by

$$M_i = \sum_{k}^{L^2} s_k \tag{2}$$

where s_k, s_l are individual spins.

Further, we have the partition function, which is the sum of the energy for all states. We will denote the set of all possible states by $\mathcal{S}$.

$$Z = \sum_{i \in \mathcal{S}} e^{\beta E_i} \tag{3}$$

In general, a $L \times L$ lattice has 2^{L^2} possible states, i.e. $|\mathcal{S}| = 2^{L^2}$

2.1 The case of $L = 2$

2.1.1 Energy and mean magnetization

We will study the case of $L = 2$ and find analycal expressions, which we will later compare to our numerical results.

2.1.2 Expectation values

It is known that the expectation value for the energy, the energy squared, the absolute magnetization [1], the heat capacity and the magnetic susceptibility are respectively

[1] We will not disciminate between positive and negative magnetization

State	Symmetries	Energy (J)	Mean magnetization
↑ ↑ ↑ ↑	1	-8	4
↑ ↑ ↑ ↓	4	0	2
↑ ↑ ↓ ↓	4	0	0
↑ ↓ ↓ ↑	2	8	0
↓ ↓ ↓ ↑	4	0	-2
↓ ↓ ↓ ↓	1	-8	-4

Table 1: All 16 possible states for $L = 2$

$$\langle E \rangle = -\frac{1}{Z}\frac{\partial}{\partial \beta} Z \tag{4}$$

$$\langle E^2 \rangle = \frac{1}{Z}\frac{\partial^2}{\partial \beta^2} Z \tag{5}$$

$$\langle M \rangle = \frac{1}{Z}\sum_{i \in \mathcal{S}} M_i \tag{6}$$

$$\langle C_V \rangle = \tag{7}$$

$$\langle \chi \rangle = \tag{8}$$

$$\tag{9}$$

3 Implementation and results

3.1 Choice of language

For our implementation, we chose a hybrid Python-C approach – inexpensive operations such as initialization of arrays and extracting the important quantities from the results are done in Python, where we can write short high-level code, while the expensive operations, i.e. the Metropolis algorithm, is carried out in fast C code.

3.2 Parallelization

The Metropolis algorithm cannot easily (and efficiently) be parallelized without side-effects. One could combine several runs with different seeds to get some kind of average, but such an approach is sub-optimal. However, it is trivial to parallelize multiple runs with differing parameters.

For this purpose, we could have extended our C code with OpenMP or used MPI from Python (which has been shown to achieve similar efficiency to MPI from C++ in [2]) but neither OpenMP nor MPI are optimal choices for such a problem. A pool based parallelization model allows for efficient and automated distribution of tasks at run time. The pool size is usually chosen to be equal to the number of logical processors, and then each worker in the pool will fetch and compute tasks until the pool is empty.

Studying running times from our program, we should not be surprised to find that our parallel program induces no measurable overhead [2] and is hence optimal.

4 Conclusion

References

[1] M. Hjorth-Jensen, "Computational physics: Lecture notes fall 2015." [Online]. Available: http://compphysics.github.io/ComputationalPhysics/doc/Lectures/lectures2015.pdf

[2] M. Mortensen and H. P. Langtangen, "High performance python for direct numerical simulations of turbulent flows," *Computer Physics Communications*, vol. 203, pp. 53–65, 2016. [Online]. Available: https://www.duo.uio.no/handle/10852/50300

[2] Strictly speaking, this is only the case for machines that are able to sustain peak clock speed on all cores. Also, hyperthreading is not true parallelism, so in general one cannot expect a speedup greater than the number of cores. For these reasons, a typical laptop will see a higher time usage per job when running in parallel, but that does not mean our program is suboptimal.