# SCIFM0004 Accelerated Computing Mini Project
## *The Lebwohl-Lasher Model of Liquid Crystals*

Dr. Simon Hanna
Version 2

October 18, 2023

## Introduction

In this project, you are asked to take a Python program for simulating the 2D Lebwohl-Lasher liquid crystal model, and accelerate it using the various approaches described in the course (Workshops AC1 and AC2). More details are given below. The following background describes what is modelled in the provided Python program. You should look at the code while reading through this.

# Background

Liquid crystals are materials that combine the orientational order of a crystal with the positional order of a liquid. They can be thought of as liquids in which the molecules have a tendency to align with each other. Typical liquid crystal molecules have a long rigid (rod-like) core to promote alignment, but flexible tails to prevent crystallisation. For commercial applications, liquid crystal molecules often have a permanent dipole oriented along the long axis of the molecule, which enables them to couple with external electric fields. In liquid crystal displays, electric fields are used to reorient the liquid crystal molecules between optical polarisers, to induce changes in the optical response of the device. The first liquid crystal displays were introduced in digital watches in the 1970s and there has been a continuing development ever since, for which reason an understanding of the phase behaviour of liquid crystal materials is very important.

There are many liquid crystal phases, with differing degrees of positional and orientational order. We will consider only the simplest, the nematic phase, in which positional order is short-range or liquid-like, while the orientational order is induced by the combination of steric (shape-related) and energetic (van der Waals, electrostatic, dipolar) interactions. The typical phase behaviour of such a system will be:

$$\text{Crystal} \xrightarrow{\ T_{C-N}\ } \text{Nematic} \xrightarrow{\ T_{N-I}\ } \text{Isotropic}$$

For commercial applications, it is important that the range between the crystal to nematic transition, $T_{C-N}$, and the nematic to isotropic transition, $T_{N-I}$, comfortably spans the normal ambient temperature range. The upper temperature limit, $T_{N-I}$, is that at which all orientational ordering is lost and the device would become useless. So it is easy to understand why the an understanding of phase behaviour can be important, both technologically and scientifically.

The Lebwohl-Lasher model[1] is used to simulate the behaviour of nematic liquid crystals, by dividing space into cells on a lattice, and assigning an average orientation, $\theta_j$, to each cell. The local orientation of a liquid crystal system is usually referred to as the liquid crystal *director*. Figure 1 shows the initial configuration and a later configuration of such a model. Adjacent cells are then able to interact through an energy term written as:

$$u_j = -\frac{\epsilon}{2} \sum_{i=1}^{n} \left( 2 \cos^2(\theta_j - \theta_i) - 1 \right) \tag{1}$$

where the sum is taken over the $n$ nearest neighbours of the lattice site and $\epsilon$ is the maximum interaction energy between pairs of adjacent liquid crystal directors.

Lebwohl and Lasher performed Metropolis Monte Carlo simulations on this model, consisting of making random changes to the orientations of the directors that were either accepted

[1]Lebwohl, P.A. and Lasher, G., "Nematic-Liquid-Crystal Order—A Monte Carlo Calculation", *Phys. Rev. A*, **6**, 426–429 (1972).
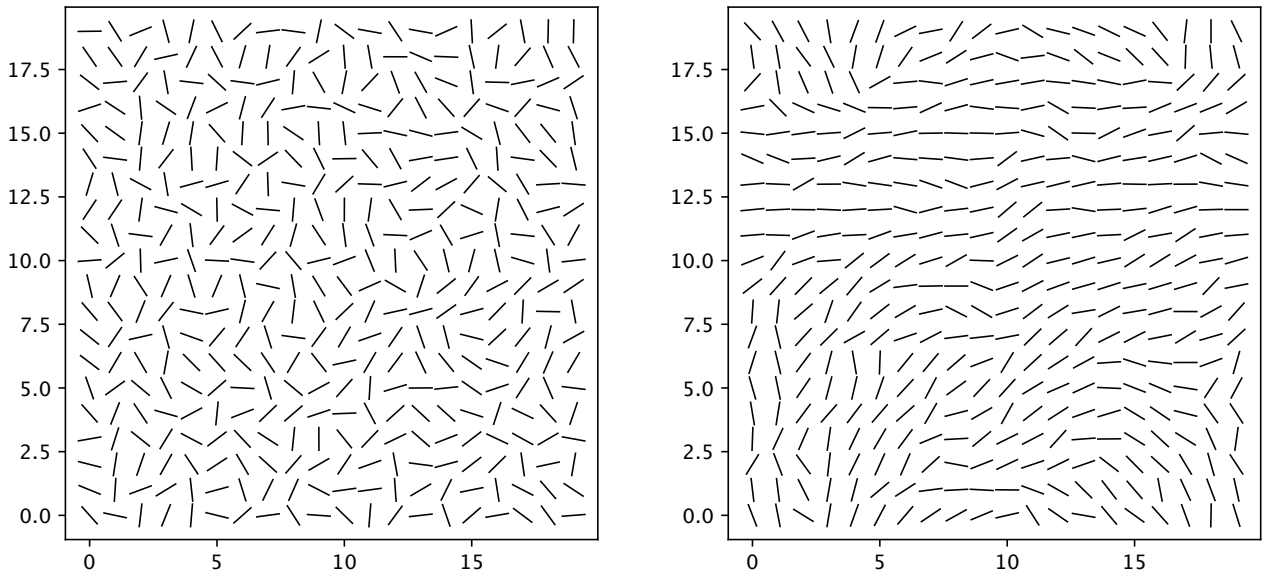
Figure 1: Two frames from a Monte Carlo simulation of the 2D Lebwohl-Lasher model with lattice side length of $50$ and reduced temperature $T^* = 0.5$: left hand image shows initial random configuration; right hand image shows configuration after 50 Monte Carlo steps (MSC).

or rejected according to a temperature dependent acceptance criterion. In this way they demonstrated the existence of the nematic to isotropic phase transition at a reduced temperature of approximately $T^* = k_B T / \epsilon = 1.1$.

# Metropolis Monte Carlo

Monte Carlo algorithms were first used by researchers at Los Alamos in the 1950's and originally were applied to simulate neutron transport through nuclear piles. The term Monte Carlo is used to describe any algorithm that involves random

numbers to generate different physical configurations. The Metropolis algorithm[2] was originally used to simulate liquids of hard spheres, and introduced a Boltzmann factor as a way of deciding which configurations to keep. The method as published, which we shall be using, was shown to simulate a system at constant temperature in the canonical thermodynamic ensemble. The approach is to generate a sequence of configurations, each by making a change to the previous configuration. This is sometimes referred to as a Markov chain. In the present case, we will make random changes to the angle of one of the directors in the lattice.

The algorithm in full goes as follows:

1. Pick a cell at random;

2. Calculate the energy of the cell, $E_0$;

3. Make a random change to the angle – in the code example attached, I draw a random number form a normal distribution with standard deviation proportional to the temperature of the system;

4. Calculate the energy of the cell again, $E_1$;

5. If $E_1 \leq E_0$ accept the change immediately;

6. If $E_1 > E_0$ generate a random number, $R$, in the range $[0.0, 1.0)$ and the Boltzmann factor for the change. Ac-

[2]N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, *J. Chem. Phys.* **21**, 1087 (1953).

cept the change if:

$$R \le \exp\left(-\frac{E_1 - E_0}{k_B T}\right)$$

Otherwise undo the change.

7. Repeat the process until every cell has been visited on average once i.e. for a lattice of size $N \times N$ this would be $N^2$ times. This is 1 MCS.

8. Repeat the above for a large number of MCS.

# Simulation Process

A simulation proceeds by setting a temperature and then performing Monte Carlo steps while monitoring the energy of the system. The energy should gradually reach an equilibrium after which it will fluctuate about its mean value. Equilibrium properties of the system can be measured and averaged once this equilibrium has been achieved. An example is shown in the left hand graph of Figure 2.

The degree of liquid crystalline order in a system is generally measured using the following order parameter:

$$S = \langle P_2(\cos\theta) \rangle \tag{2}$$

where $\theta$ is the angle between the individual cell director and the overall director, the angle brackets indicating an average
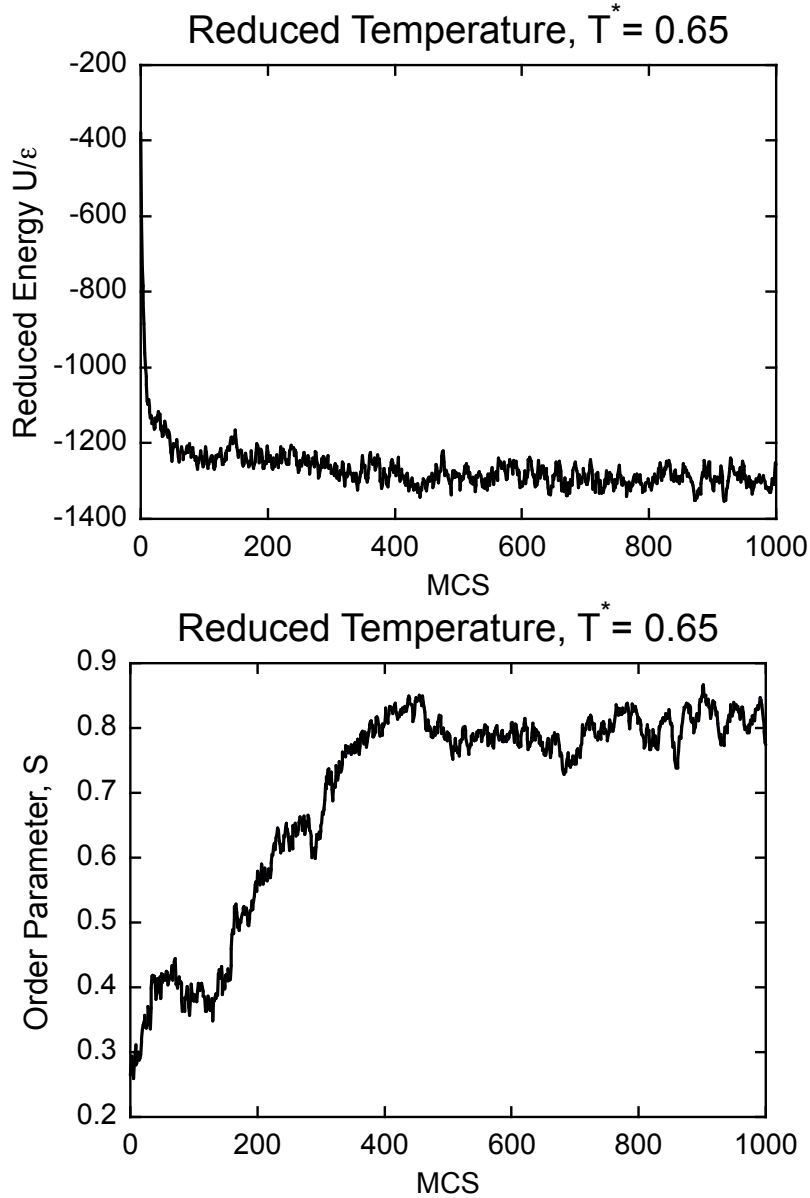
Figure 2: Left hand: Reduced energy versus Monte Carlo step for a Lebwohl-Lasher simulation with $T^* = 0.65$. Right hand: Order parameter versus MCS for the same system.

over all the cells and over multiple configurations drawn from the equilibrium distribution. $P_2$ is the second order Legendre polynomial defined by $P_2(\cos\theta) = \frac{1}{2}(3\cos^2\theta - 1)$. $S = 1$ for a perfectly ordered liquid crystal, while $S \simeq 0.25$ for a random 2D arrangement such as is shown in the left hand frame of Figure 1.

In practice we don't know the system director – the overall direction is not fixed – and so we calculate the order parameter from a liquid crystal order tensor as follows:

$$Q_{\alpha\beta} = \frac{1}{2}\left\langle 3l_{\alpha,i}l_{\beta,i} - \delta_{\alpha\beta} \right\rangle_i, \qquad \alpha, \beta = x, y, z \qquad (3)$$

with the average taken over all cells in a frame. $l_{\alpha,i}$ and $l_{\beta,i}$ are components of the director of cell $i$ expressed as a cartesian unit vector, while $\delta_{\alpha\beta}$ is the Kronecker delta. $S$ is given by the largest Eigenvalue of $Q_{\alpha\beta}$ with the overall system director, which we don't actually need, given by the corresponding Eigenvector. Ideally we should look at $\langle S \rangle$, where we average over several equilibrium configurations. An example of the evolution of the order parameter in a simulation is shown in the right hand side of Figure 2.

**Note:** the random jumps in the director applied during each MCS are drawn from a distribution with width governed by the temperature. This is an attempt to keep the acceptance ratio – the fraction of attempted moves which are successful – to a reasonable value (around 0.5 is ideal). If the acceptance ratio is too small, the system is frozen and takes too long to equilibrate.

# Simulation Results

For a given system size, we are interested in observing the value of liquid crystal order parameter, $S$, as a function of reduced temperature, $T^*$. Taking the above into account, we would run independent simulations at each temperature, being careful to ensure the system has equilibrated i.e. both the energy and order parameter should have reached plateaux. Average quantities should be determined from the equilibrated parts of the simulations. Figure 3 shows the results of such a set of simulations for a $20 \times 20$ 2D lattice. It can be observed that there is a phase transition from orientationally ordered to disordered in the region of $T^* = 0.9$. The difference between this and the published value (see above) is most likely due to the different dimensionality of the two cases (3D versus 2D).

# Aims of the Project

You are provided with a Python program that performs 2D Lebwohl-Lasher simulations as described above. You should familiarise yourself with the operation of the code and make sure you can reproduce the types of result shown in Figures 1–3.

Use the profiling tool provided with Python to determine which parts of the code are doing the most work, and focus on accelerating these first of all. The profiler is available in the
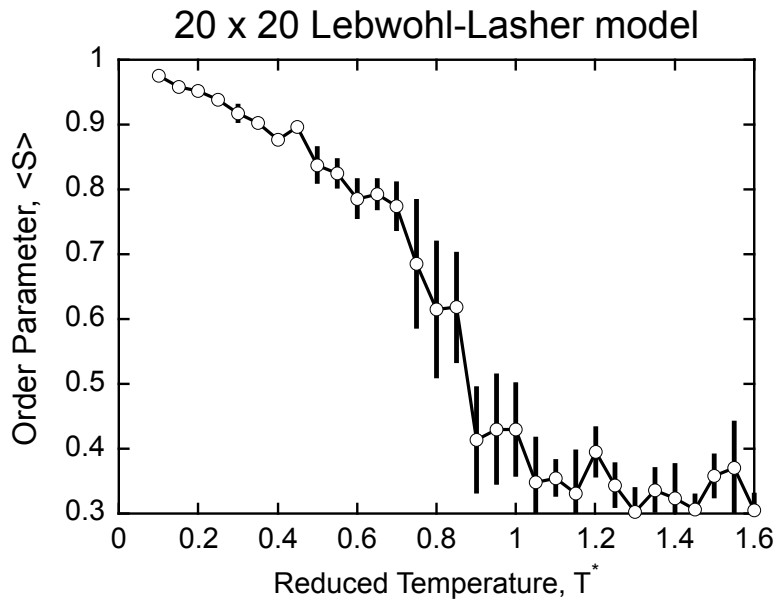
Figure 3: Averaged order parameter $\langle S \rangle$ as a function of reduced temperature $T^*$ for a 2D Lebwohl-Lasher model with size $20 \times 20$ cells. Note the order drops to an isotropic value at around $T^* = 0.9$.

Spyder menu, if you have it, or can be run from the command line as follows:

```
python -m cProfile LebwohlLasher.py 50 50 0.5 0 > prof.txt
```

Note that I am assuming a Linux environment and sending the output to a file. Searching through the output file I find:

```
944296 function calls (936422 primitive calls) in 11.386 seconds

ncalls tottime percall cumtime percall filename:lineno(function)
    51   0.062   0.001   2.065   0.040 LebwohlLasher.py:120(all_energy)
    51   1.531   0.030   1.551   0.030 LebwohlLasher.py:130(get_order)
    50   0.766   0.015   5.815   0.116 LebwohlLasher.py:151(MC_step)
     1   0.001   0.001   9.438   9.438 LebwohlLasher.py:189(main)
     1   0.001   0.001  11.387  11.387 LebwohlLasher.py:23(<module>)
     1   0.000   0.000   0.002   0.002 LebwohlLasher.py:32(initdat)
     2   0.000   0.000   0.000   0.000 LebwohlLasher.py:41(plotdat)
     1   0.000   0.000   0.004   0.004 LebwohlLasher.py:71(savedat)
377500   6.637   0.000   6.637   0.000 LebwohlLasher.py:95(one_energy)
```

Information on how to interpret these numbers is given in the profiler documentation:

`https://docs.python.org/3/library/profile.html`

Targetting those parts of the code that are clearly taking up the most time, use all available techniques to speed up the code. This should include:

- Port the code to mpi4py and run on multiple cores;

- Cythonize individual functions and attempt to apply multi-threading techniques;

- See what opportunities exist for using NumPy vectorization to speed the calculations (can you switch from random to sequential sampling to aid this?);

- Experiment with Numba and see what speed-ups are possible.

**Please note:** it is not expected that all the above will be attempted in the same version of the code. Please try out the different methods and submit your various experiments as separate programs.

**Also note:** This assessment primarily targets the unit ILO on "accelerating high performance scientific programs", but also addresses the ILOs covering version control and testing and validation. Version control is particularly important when trying out lots of different ideas for accelerating code – you may often need to backtrack when the latest attempt fails, or you may see marginal gains from different approaches and need to be able to go back and reassess your findings. Equally, when working with multiple different versions of the code it is important to check that the code is still performing correctly. Therefore I would like you to implement a testing strategy within your code, and I would like you to work in a versioning archive such as GitHub. When assessing your work, I would like to be able to look through your code archive.

An important aspect of this assessment will be the presentation of your findings through a report. The report should be a critical analysis of which computational approaches work best, with comparisons between the methods highlighting speed improvements and commenting on such matters as ease of coding, flexibility of approach, ease of maintenance etc. I would like to see charts detailing speed comparisons and scaling plots where you examine performance of your code versus (i) numbers of threads / tasks, (ii) size of problem. Please make sure you discuss your results in an analytical manner.

# Instructions for submission

Please present your code in the form of a code archive such as a GitHub repository. Please include all programs and support files (including Blue Crystal batch scripts), together with appropriate instructions for building and running the code (e.g. Cython setup scripts). **Please include the address of your archive in your report.** Your report should be uploaded to Blackboard before the **deadline on Wednesday of Week 20, i.e. 13th March 2024** at **12:00 noon**.