Liquid Crystals Report

Ben Morgan 25th April

# Introduction

This plot gives some insight into the vortex that was generated. The top left pane shows the density of the vortex with respect to the x and y coordinates. The bottom left shows the vorticity. The top right pane shows the velocity profile across the vortex. The bottom right plot shows how the error in density and velocity changes over time.

# Numba

I decided that first it would be interesting to try the most naive numba optimisation approach and place the numba @nb.jit handler infront of all\_energy() one energy() and get\_order(). This resulted in a nice speedup from the outset.

I ran some array jobs on blue crystal which ran the unmodified code and the numba optimised code with various grid sizes.

A graph with a line and a number of numbers

Description automatically generated

The numba code is significantly faster at all grid sizes. I calculated how many times faster the numba version was for all datapoints and found that the numba version was on average 10.06x faster than the unmodified version. I ploted these datapoints in the below graph.

A graph with blue dots and numbers

Description automatically generated

The speedup does not seem to depend too much on grid size and the datapoints and the highest and lowest speedups only vary by 5.3%.

This speedup occurs in spite of the fact that the get\_order() function reverted back to object mode (as seen in the below picture). This means that the speedup will have occurred mostly as a result of optimising the all\_energy and one\_energy functions.

A screenshot of a computer program

Description automatically generated

# MPI

I next decided to attempt to use mpi to speed up the program. I thought MC\_step would be a good candidate for mpi parallelisation as it computed values over a large grid that could be split up into smaller pieces. It was also obvious that this could be done without much communication between threads which I believed would make its implementation simple.

I wanted to test the dependency of speed on the number of cores over which the model was ran. I chose a large grid size of 500x500 and scheduled jobs over varying numbers of cores. This graph has been plotted below.

At first as more cores are added the wall time significantly decreases but after around 8 cores were used the benefit of using more cores significantly diminished. As a result, in future MPI runs I decided to use 8 cores as I believed this gave the best balance between speedup and computational cost. Below I have plotted a graph of the wall time multiplied by the number of cores for each run. It shows that the computational expense increases in a roughly linear fasion as more cores are introduced

A graph showing a line of growth

Description automatically generated with medium confidence

At this point the MPI implementation was slower than the numba implementation, so I decided to combine the two.

# MPI + NUMBA