Imperial College London
Department of Aeronautics
Third Year HPC Coursework Report 2020

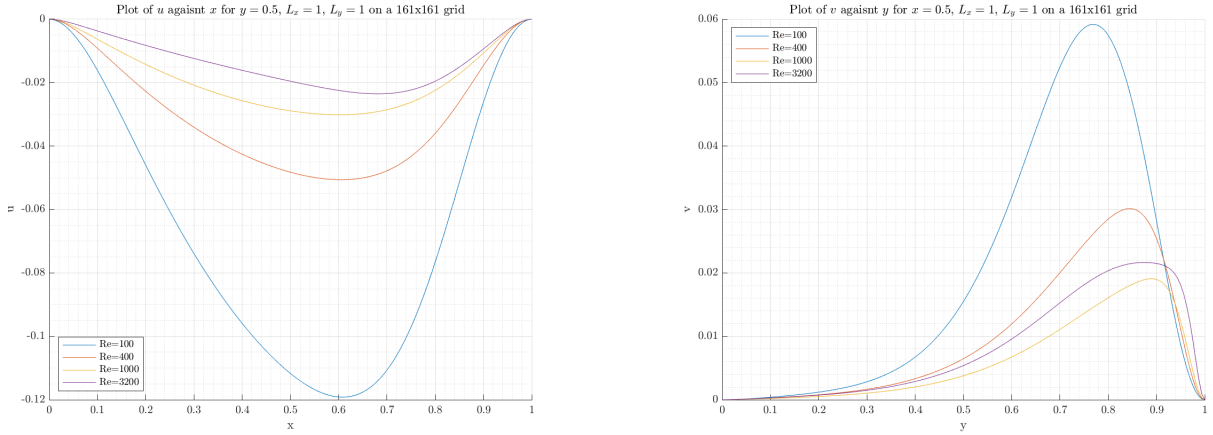
High Performance Computing Coursework

Done by:
Chai Jun, Sean
CID: 01327446

Contents

1 Result of running Lid Driven Cavity Solver

1.1 Velocity Plots

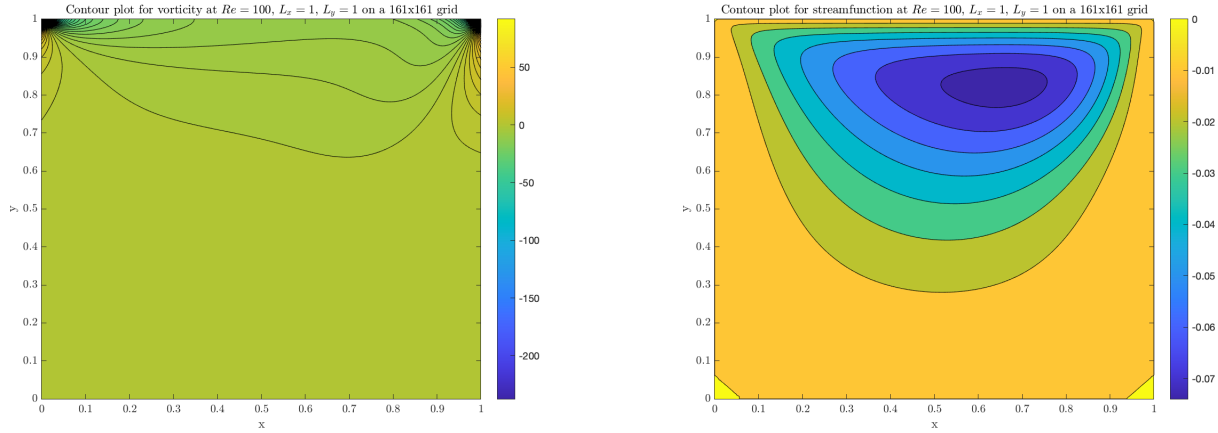


(a) Horizontal velocity u against x along $y = 0.5$

(b) Vertical velocity v against y along $x = 0.5$

Figure 1: Plot of velocities of the steady state solution for Reynolds numbers of 100, 400, 1000 and 3200 using a 161×161 grid and $L_x = L_y = 1$.

1.2 Vorticity and Streamfunction



(a) Vorticity contour plot

(b) Streamfunction contour plot

Figure 2: Contour plots of vorticity and streamfunction at $Re = 100$ on a 161×161 grid and $L_x = L_y = 1$.

1.3 Minimum Streamfunction

Reynolds Number	100	400	1000	3200
x-coordinate				
y-coordinate				

Table 1: Table of x-y coordinates of the streamfunction minimum at the different Reynolds numbers for a 161×161 grid and $L_x = L_y = 1$.

1.4 Streamfunction Plots for Varying L_x and L_y

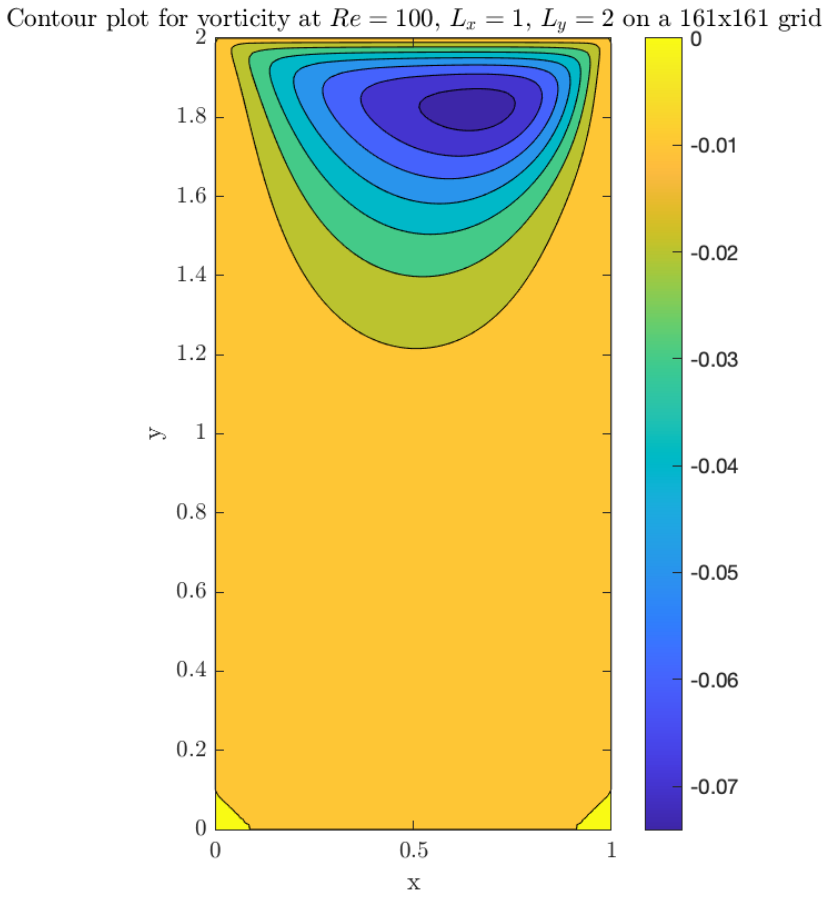


Figure 3: Contour plot of streamfunction at $Re = 100$ on a 161×161 grid and $L_x = 1$, $L_y = 2$.

2 Discussion of Parallel Code

One key decision in implementing the code was to swap around the steps of the algorithm. Since the calculation of the inner vorticity and the boundary vorticity are independent of one another, we can

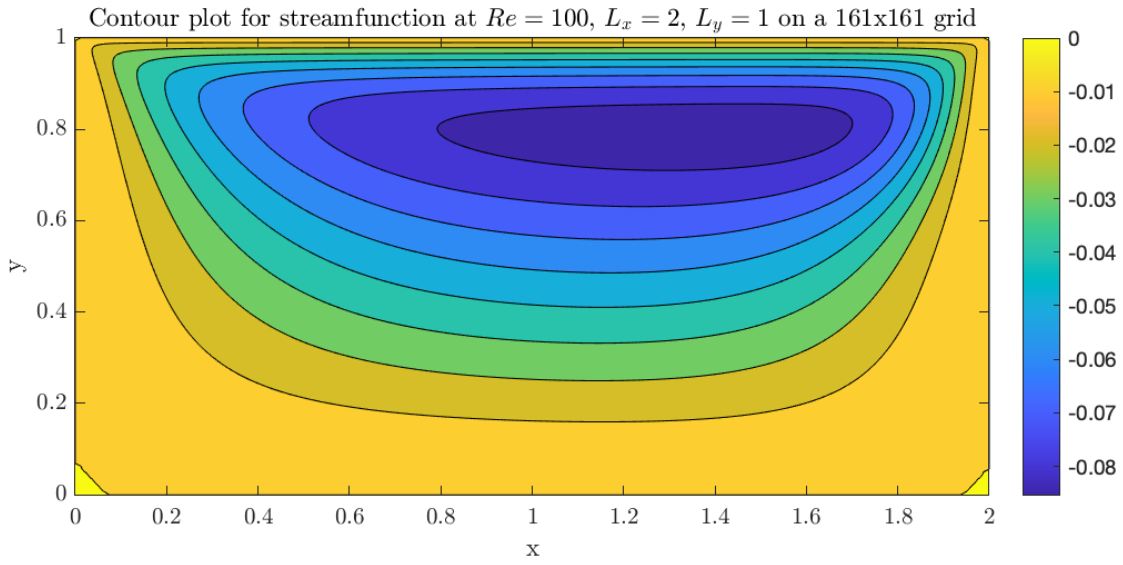


Figure 4: Contour plot of streamfunction at $Re = 100$ on a 161×161 grid and $L_x = 2$, $L_y = 1$.

swap the steps around. By calculating the interior vorticity before the boundary, it allowed for easier parallelising of code as the coded needed to be less careful in

The updating of the boundary conditions was not parallelised as the time taken to send and receive the data take around the same time compared to just a simple "for" loop. As such, the decision was made to not parallelise that portion of the code.

One method of optimising the code was to do the update of vorticity at time t and $t + dt$ in a manner that is consistent with the storage of the data. In our case, that meant updating the values in a column-major format as the matrices were stored as such. This allowed for faster memory access due to the reduced need of traversing through the array to reach the memory address. Additionally, I allocated the same chunk of the array to the same processes for the parallel portion of the code. This allowed each processor to access the same part of the array each time, thereby utilising memory caching which allowed for faster memory access.

Another method of optimising the serial code was to separate it from the parallel one. This reduced the need to