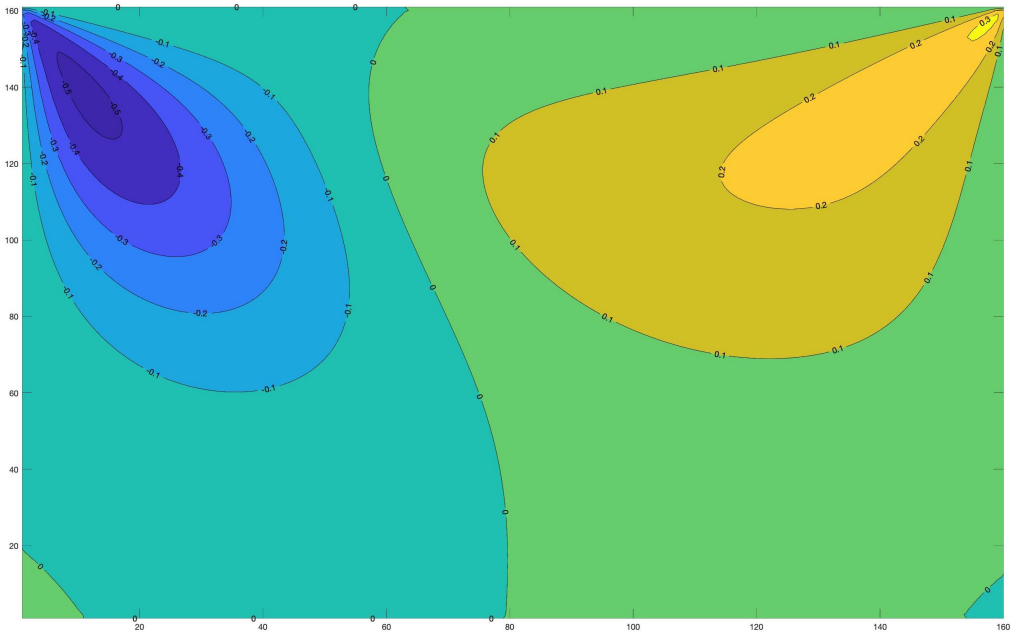
Imperial College
London

2D Navier-Stokes Equations of a Lid-Driven Cavity using the Vorticity Stream-Function Formulation



Academic Responsible:	Dr. Chris Cantwell
Department:	Aeronautics
Course:	H401 MEng Aeronautical Engineering
Module:	High Performance Computing
Academic year:	2019/2020

Student:	Eu Wen Aron Teh
CID:	01327944
Personal tutor:	Dr. Yongyun Hwang
Date:	25/03/2020

Department of Aeronautics
South Kensington Campus
Imperial College London
London SW7 2AZ
U.K.

1 Computational Results

Graphs for the steady-state horizontal velocity along $x = 0.5$ and the vertical velocity along $y = 0.5$ are shown in Figures 1 and 2 below, plotted for a 161 by 161 grid with domain size $L_x = L_y = 1$.

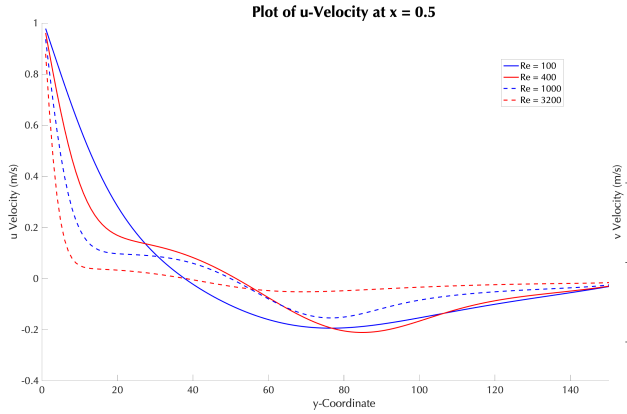


Figure 1

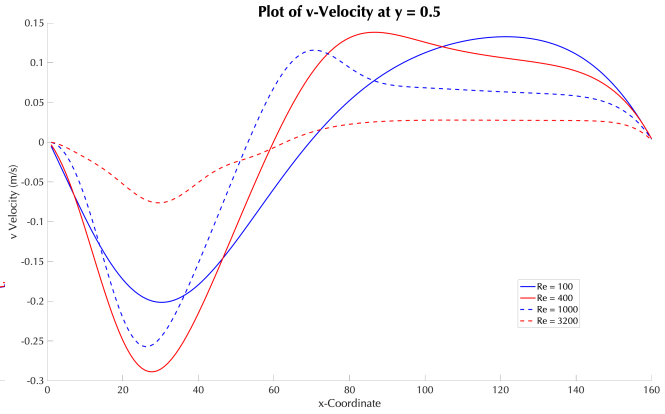


Figure 2

The steady-state vorticity and stream-function contour plots for the same grid and domain as above are shown for $Re = 100$ in Figures 3 and 4 below.

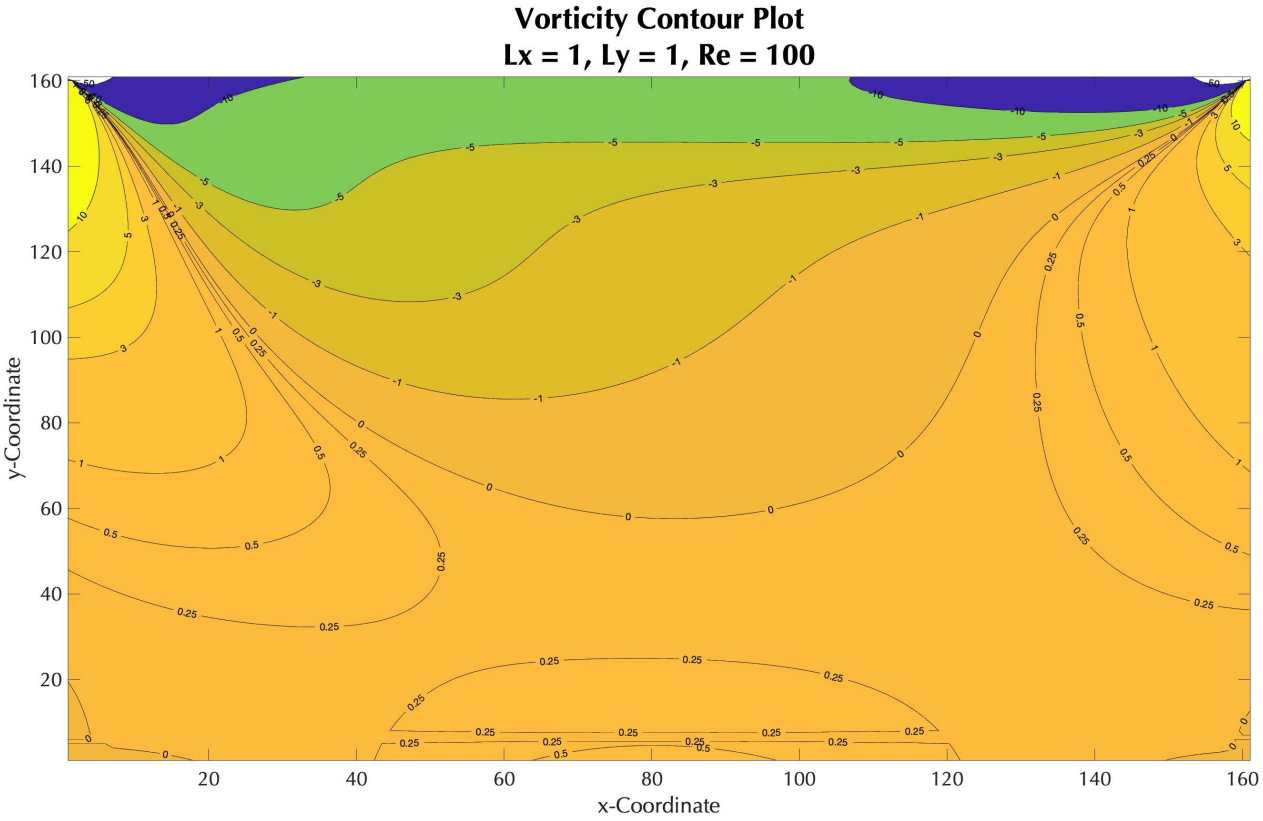


Figure 3

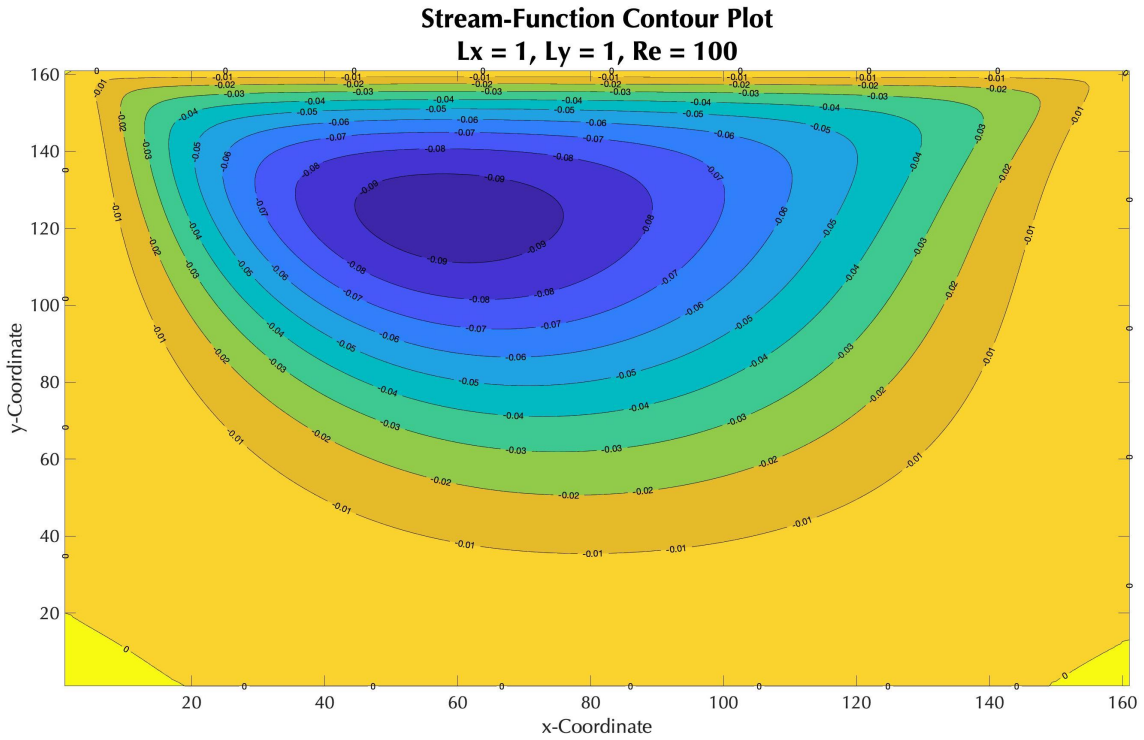


Figure 4

Table 1 shows the coordinates (x,y) of the stream-function minimum at the above Reynolds numbers and domain. Coordinate $(0, 0)$ is defined as the bottom-leftmost grid-point.

Reynolds Number	100	400	1000	3200
Minimum ψ Coordinates	(59, 122)	(56, 109)	(51, 110)	(40, 117)

Table 1

Lastly, the steady-state stream-function contour plots for domain sizes $(L_x, L_y) = (1, 2)$ and $(L_x, L_y) = (2, 1)$ and $Re = 100$ are shown in Figures 5 and 6 below.

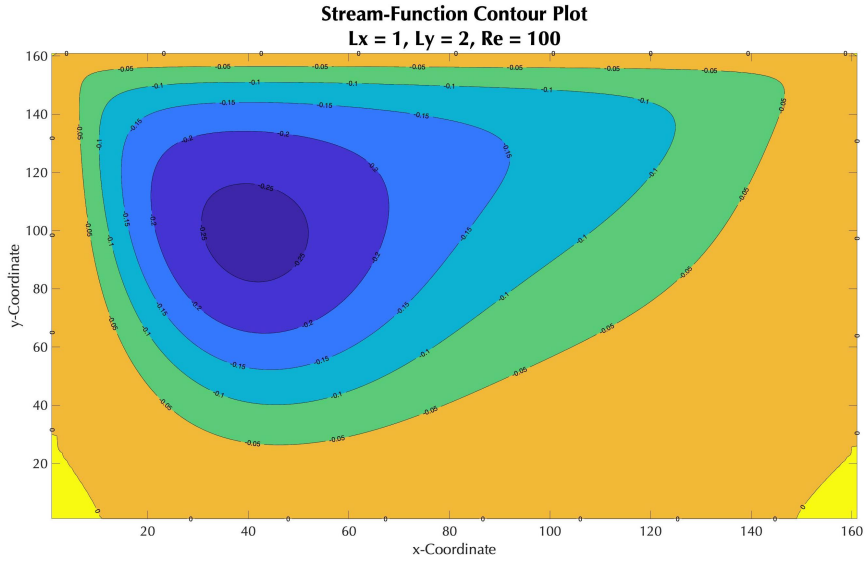


Figure 5

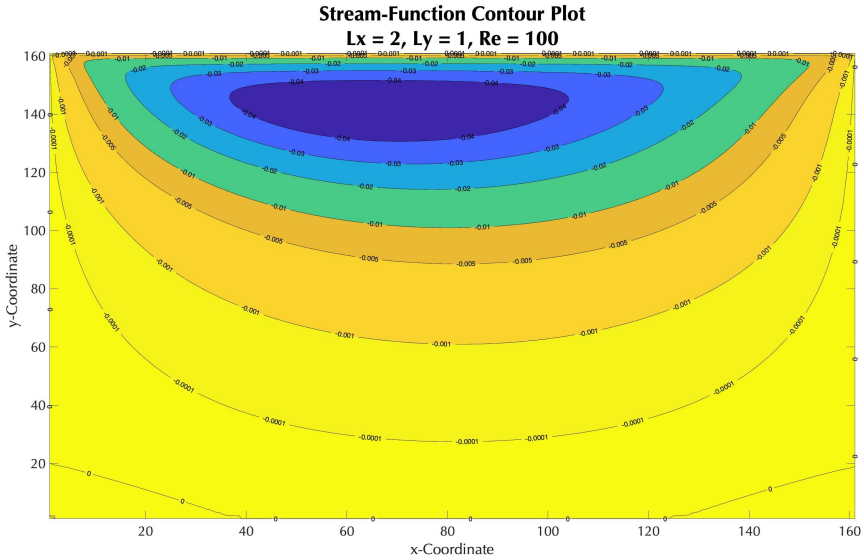


Figure 6

2 Parallelisation and Optimisation Discussion

Solving the Poisson problem occupies the majority of the program run-time, and increasingly dominates the run-time for larger and larger grids, therefore the program efficiency improves most from a parallelisation of the Poisson solver. MPI is initialised at the start of my program in LidDrivenCavitySolver.cpp, and runs in serial until the Poisson problem is reached. To avoid creating and destroying instances of the Poisson solver with each time increment, the instance is initialised outside the time-step loop and its destructor is only run at the end of the program to free memory and finalise MPI.

The Poisson problem requires solving a large linear system to compute the updated stream-function given current vorticity values, with the dimension of the matrix and vector scaling with $(N_x - 2) \cdot (N_y - 2)$. After considering several alternatives, including the conjugate gradient and Jacobi iterative methods, I decided to implement a parallel solver using the parallel SCALAPACK routine PDGBSV. This routine performs LU decomposition of the square A matrix using Gaussian elimination before substituting twice to solve for the unknown vector, which should significantly outperform a manual implementation of an iterative solver. Additionally, the A matrix for the Poisson Problem is banded with an upper and lower bandwidth of $Nx - 2$, allowing for a more efficient banded storage format. The bandwidth to matrix size ratio decreases as N_x and N_y increases, therefore the divide and conquer parallel implementation of PDGBSV becomes more effective the larger the problem size becomes. A one-dimensional BLACS grid is initialised, and each processor reads an equal section of banded A matrix and corresponding right-hand vector values. The processors are then able to solve their linear system subsets independently of one another using the aforementioned approach. The final stream-function vector is then gathered by all processes and is returned to the LidDrivenCavity solver for the next time-step. The effectiveness of this divide and conquer approach decreases when the block size (number of columns each processor solves for) is too small, therefore a check $NB \geq (BWL + BWU) + 1$ is performed both by my program and the routine to ensure the condition is satisfied. This also places a limitation on the number of processors required to exeute this program for a given grid size.

Unfortunately, I was not able to execute a parallel solve with multiple processors due to an issue with my MPI_Allgather() implementation, however the code still ran fully in serial and I was able to produce the full set of results.