PARALLEL COMPUTATION OF SOME INCOMPRESSIBLE VISCOUS FLOW PROBLEM BY BLOCK DECOMPOSITION

A Project Report Submitted

in Partial Fulfilment of the Requirements

for the Degree of

BACHELOR OF TECHNOLOGY

in

Mathematics and Computing

by

Sizil Krishna Goojar

(Roll No. 10012332)

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

April 2014

CERTIFICATE

This is to certify that the work contained in this project report entitled "**Parallel Computation of some Incompressible viscous flow problem by block decomposition**" submitted by **Sizil Krishna Goojar** (**Roll No.: 10012332**) to Indian Institute of Technology Guwahati towards partial requirement of **Bachelor of Technology** in Mathematics and Computing has been carried out by him under my supervision and that it has not been submitted elsewhere for the award of any degree.

Guwahati - 781 039
April 2014

(Dr. Jiten Chandra Kalita)
Project Supervisor

ABSTRACT

The main aim of this project is to obtain the flow pattern of some incompressible viscous fluid by using optimised iterators and algorithms. This project's aim was to further enhance the speed of computation by solving them parallely using advance parallel API's like CUDA. The aim of this phase is to solve larger grid domain with greater computational economy to get finer results which was not possible using serial codes before.

ACKNOWLEDGEMENT

I feel a great privilege in expressing my deepest and most sincere gratitude to my supervisor Prof. Jiten C Kalita (Dept. of Mathematics) for the most valuable guidance and influential mentorship provided to us during the course of this project. Due to his technical advices, exemplary guidance, persistent monitoring and constant encouragement throughout the course of our project, I was able to complete the project through various stages. I take this opportunity to express our profound gratitude and deep regards to all other people who helped us in every possible way to get my project comfortably completed.

Contents

List of Figures

Nomenclature

ρ	Density of fluid
ω	Vorticity function
ψ	Stream function
t	Time
g	Body force per unit mass
Re	Reynolds number
V	Velocity vector
u	Horizontal component of velocity vector
v	Vertical component of velocity vector
$CUDA$	Compute Unified Device Architecture
API	Application programming interface
$OpenMP$	Open Multi-Processing

Chapter 1

Introduction

Computational fluid dynamics is a branch of fluid mechanics that concerns with the development of various approximating schemes and numerical algorithm for solving and analysing fluid flows. The question arises why do we need CFD? The Navier-Stoke's equation can completely describe fluid flow under any conditions. For incompressible viscous flows the Navier-Stoke's equation is given as follows.

$$\rho\left(\frac{\partial V}{\partial t} + V.\bigtriangledown V\right) = -\bigtriangledown p + \mu\bigtriangledown^2 V + \rho g \tag{1.1}$$

The term on left hand side is represents the material derivative of the velocity while terms on the right hand side correspond to pressure gradient, viscous force and body force respectively. This equation is analogous to the Newton's second law of motion.

The Navier-Stoke's equation is a nonlinear partial differential equation of second order for which no known analytical solutions exists. Hence engineers rely on CFD for predicting flow patterns, pressure and velocity variations

etc.

One of the method used for approximation is the finite difference method which we have used throughout.

1.1 Finite difference Method

In mathematics, finite-difference methods are numerical methods for approximating the solutions to differential equations using finite difference equations to approximate derivatives. Here we aproximate derivatives of various orders by Taylor series expansion. The following derivation illustrates the method. The taylor series expansion of any function about point x_0 is -

$$\begin{aligned}
f(x_0+h) &= f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f^{(2)}(x_0)}{2!}h^2 + ... + \frac{f^{(n)}(x_0)}{n!}h^n + R_n(x) \\
f(x_0+h) &= f(x_0) + f'(x_0)h + R_1(x) \\
f(a+h) &= f(a) + f'(a)h + R_1(x) \\
\frac{f(a+h)}{h} &= \frac{f(a)}{h} + f'(a) + \frac{R_1(x)}{h} \\
f'(a) &\approx \frac{f(a+h) - f(a)}{h}
\end{aligned}$$

The above equation is and example of forward difference. The other two types of difference schemes possible are backward difference and central difference. The backward difference and central difference schemes for first order derivatives respectively are,

$$\begin{aligned}
f'(a) &\approx \frac{f(a) - f(a-h)}{h} \\
f'(a) &\approx \frac{f(a+h) - f(a-h)}{2h}
\end{aligned}$$

Using similar procedure as discussed above the generalised finite difference scheme of a second order derivative for a function of two variable is,

$$f_{xy}(x,y) \approx \frac{f(x+h,y+k) - f(x+h,y-k) - f(x-h,y+k) + f(x-h,y-k)}{4hk} \tag{1.2}$$

The above form of the discritized equation has been used throughout.

1.2 CUDA

The CUDA (Compute Unified Device Architecture) programming model is a heterogeneous model in which both the CPU and GPU are used. In CUDA, the host refers to the CPU and its memory, while the device refers to the GPU and its memory. Code run on the host can manage memory on both the host and device, and also launches kernels which are functions executed on the device. These kernels are executed by many GPU threads in parallel. And since GPU is more efficient and powerful in handling large numbers and calculation, it makes CUDA programming model suitable for solving CFD problems.

Given the heterogeneous nature of the CUDA programming model, a typical sequence of operations for a CUDA C program is:

1. Copy data from main mem to GPU mem
2. CPU instructs the process to GPU
3. GPU execute parallel in each core
4. Copy the result from GPU mem to main mem

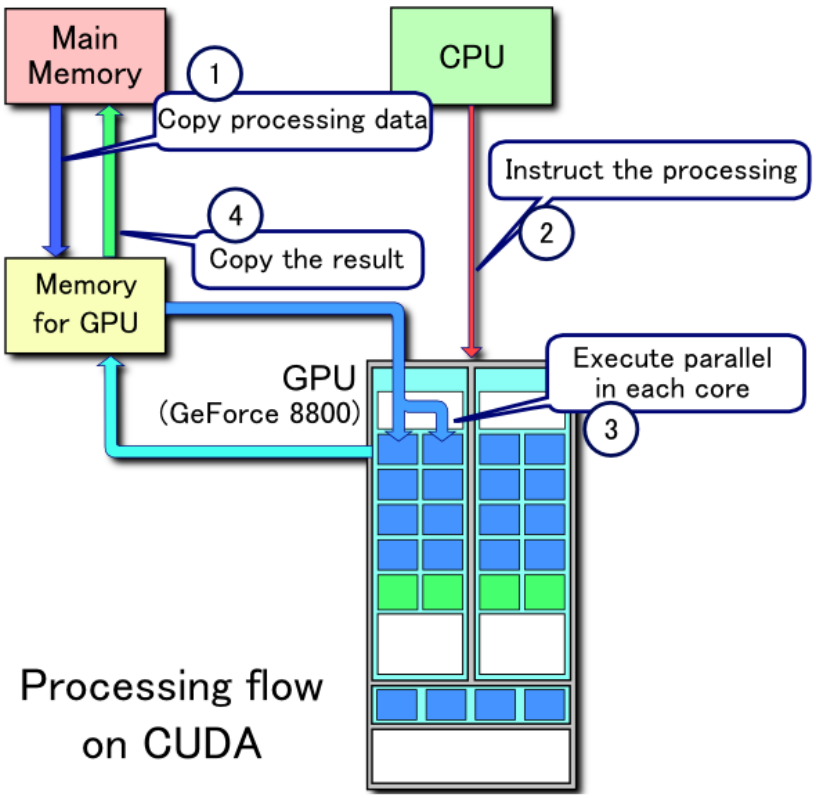


Figure 1.1: The CUDA processing flow

Chapter 2

Theoretical aspects

In this chapter we discuss the theory behind the formulation of steady state and unsteady state flow and also something about the various equation solving methods.

2.1 Steady flow

Steady flow means that no flow parameters (velocity, pressure etc) depend on time. The time dependant terms in Navier-Stoke's and continuity equations drop off. The governing equations are,
Continuity equation for 2-D incompressible flow,

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{2.1}$$

And the momemtum equation which is given at 1.1.

The Stream function vorticity formation of the problem is,

$$\bigtriangledown^2 \psi = -\omega \tag{2.2}$$

Non dimensional Navier-Stoke's equation for 2-D incompressible flow in ψ-ω form is,

$$\frac{\partial \psi}{\partial y}.\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}.\frac{\partial \omega}{\partial y} = \frac{1}{Re}\left[\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2}\right] \tag{2.3}$$

The discretized form of the previous equations are The boundary conditions vary accordingly to the problem.

$$\begin{aligned} \psi_{i,j} &= \frac{\psi_{i+1,j} + \psi_{i-1,j} + \beta^2(\psi_{i,j+1} + \psi_{i,j-1}) + (\Delta x)^2 \omega_{i,j}}{2(1+\beta^2)} \\ \beta &= \frac{\Delta x}{\Delta y} \end{aligned} \tag{2.4}$$

where, Δx is grid size in X direction and Δy is grid size in Y direction.

$$\omega_{i,j} = \frac{\begin{aligned}\omega_{i+1,j} + \omega_{i-1,j} + \beta^2(\omega_{i,j+1} + \omega_{i,j-1}) - \frac{1}{4}\beta Re(\omega_{i+1,j} - \omega_{i-1,j})(\psi_{i,j+1} - \psi_{i,j-1}) + \\ \frac{1}{4}\beta Re(\omega_{i,j+1} - \omega_{i,j-1})(\psi_{i+1,j} - \psi_{i-1,j})\end{aligned}}{2(1+\beta^2)} \tag{2.5}$$

To solve both the above equations we have used the Gauss-Seidel method.

2.2 Unsteady State

Unsteady state means that various quantities like pressure,velocity etc depend on time. For the unsteady state formulation we have used the Alternating Direction Implicit(ADI) method.

Alternating Direction Implicit (ADI) method is a finite difference method for solving parabolic, hyperbolic and elliptic partial differential equations. It is most notably used to solve the diffusion equation in two or more dimensions. It is an example of an operator splitting method. The governing equations for unsteady state analysis of flow are,
Continuity Equation is the same as that for steady state in the case of 2-D incompressible flow.

The Navier-Stoke's equation in ψ-ω form is

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \omega}{\partial x}\frac{\partial \omega}{\partial y} = \frac{1}{Re}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right) \tag{2.6}$$

The discretized form of previous equations are
The continuity equation,

$$\begin{aligned} \beta &= \frac{\Delta x}{\Delta y} \\ \psi_{i,j} &= \frac{\psi_{i+1,j} + \psi_{i-1,j} + \beta^2(\psi_{i,j+1} + \psi_{i,j-1}) + (\Delta x)^2 \omega_{i,j}}{2(1+\beta^2)} \end{aligned} \tag{2.7}$$

Where all the terms are at the same time step.

The discretized Navier-Stoke's Equation is-
$XSweep$:

$$\begin{aligned} &\frac{\omega_{i,j}^{n+\frac{1}{2}} - \omega_{i,j}^{n}}{\frac{\Delta t}{2}} + v_{i,j}^{n}\left[\frac{\omega_{i,j+1}^{n} - \omega_{i,j-1}^{n}}{2h}\right] + v_{i,j}^{n+\frac{1}{2}}\left[\frac{\omega_{i+1,j}^{n+\frac{1}{2}} - \omega_{i-1,j}^{n+\frac{1}{2}}}{2h}\right] = \\ &\frac{1}{Re}\left[\left[\frac{\omega_{i+1,j}^{n+\frac{1}{2}} + \omega_{i-1,j}^{n+\frac{1}{2}} - 2\omega_{i,j}^{n+\frac{1}{2}}}{h^2}\right] + \left[\frac{\omega_{i,j+1}^{n} + \omega_{i,j-1}^{n} - 2\omega_{i,j}^{n}}{h^2}\right]\right] \end{aligned} \tag{2.8}$$

$YSweep$:

$$\frac{\omega_{i,j}^{n+1} - \omega_{i,j}^{n+\frac{1}{2}}}{\frac{\Delta t}{2}} + v_{i,j}^{n+1}\left[\frac{\omega_{i,j+1}^{n+1} - \omega_{i,j-1}^{n+1}}{2h}\right] + v_{i,j}^{n+\frac{1}{2}}\left[\frac{\omega_{i+1,j}^{n+\frac{1}{2}} - \omega_{i-1,j}^{n+\frac{1}{2}}}{2h}\right] = \frac{1}{Re}\left[\left[\frac{\omega_{i+1,j}^{n+\frac{1}{2}} + \omega_{i-1,j}^{n+\frac{1}{2}} - 2\omega_{i,j}^{n+\frac{1}{2}}}{h^2}\right] + \left[\frac{\omega_{i,j+1}^{n+1} + \omega_{i,j-1}^{n+1} - 2\omega_{i,j}^{n+1}}{h^2}\right]\right] \quad (2.9)$$

Remark 2.2.1. h represents Δx in X sweep and Δy in y sweep. X Sweep and Y Sweep are explained in section 2.4

To solve the continuity equation Gauss-Seidel method was used while to solve the Navier-Stoke's equation Tri-Diagonal matrix algorithm.

For solving $Ax = B$ the following algorithms have been used.

- Gauss-Seidel method
- Tri Diagonal Matrix Algorithm

2.3 Gauss-Seidel method

The Gauss Seidel method is an iterative technique for solving a square system of n linear equations with unknown x,

$$Ax = b$$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad (2.10)$$

where $A \epsilon R^{n,n}$, $b \epsilon R^{n}$ and $x \epsilon R^{n}$ Then the decomposition of A into its lower triangular component and its strictly upper triangular component is given by,

$$A = L_* + U$$

$$L_* = \begin{bmatrix} a_{1,1} & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix}, U = \begin{bmatrix} 0 & a_{1,2} & \dots & a_{1,n} \\ 0 & 0 & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \tag{2.11}$$

The system of linear equations may be rewritten as,

$$L_* x = b - Ux$$

The Gauss Seidel method now solves the left hand side of this expression for x, using previous value for x on the right hand side. Analytically, this may be written as,

$$x^{k+1} = L_*^{-1}(b - Ux)$$

However, by taking advantage of the triangular form , the elements of x(k+1) can be computed sequentially using forward substitution,

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>1} a_{ij} x_j^{(k)} \right), i, j = 1, 2, \dots, n \tag{2.12}$$

The procedure is generally continued until the changes made by an iteration are below some tolerance, such as a sufficiently small residual.

2.4 Tri Diagonal matrix Algorithm

In numerical linear algebra, the tridiagonal matrix algorithm, also known as the Thomas algorithm (named after Llewellyn Thomas), is a simplified form of Gaussian elimination that can be used to solve tridiagonal systems of equations. A tridiagonal system for n unknowns may be written as,

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$$

where $a_1 = 0$ and $c_n = 0$

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & 0 \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix} \tag{2.13}$$

The forward sweep consists of modifying the coefficients as follows, denoting the new modified coefficients with primes,

$$c'_i = \begin{cases} \frac{c_i}{b_i} & : \quad i = 1 \\ \frac{c_i}{b_i - c'_{i-1} a_i} & : \quad i = 2, 3, 4, ..., n-1 \end{cases} \tag{2.14}$$

$$d'_i = \begin{cases} \frac{d_i}{b_i} & : \quad i = 1 \\ \frac{d_i - d'_{i-1} a_i}{b_i - c'_{i-1} a_i} & : \quad i = 2, 3, 4, ..., n \end{cases} \tag{2.15}$$

The solution is then obtained by back substitution,

$$x_n = d'_n \tag{2.16}$$

$$x_i = d'_i - c'_i x_{i+1}, \quad i = n-1, n-2, ..., 1 \tag{2.17}$$

Chapter 3

Results

This Chapter contains following problem statements and and their solutions.

- Steady State Lid-Driven Cavity
- Unsteady State Lid-Driven Cavity
- Staggered Lid-Driven Cavity
- Cross Lid-Driven Cavity
- Backward step flow

3.1 Steady State Lid-Driven Cavity

The problem is to find the flow in a square domain where the top lid is moving with velocity u. The governing equations are (2.2) (2.3), whereas the discretized equations are (2.4) (2.5).
The purpose of this problem is to get acquainted with the implementation of the Steady State Navier-Stoke's equation. The Lid Driven cavity problem is the standard problem for testing various numerical schemes.

The Boundary conditions used in our problem are,

$$\psi = 0, \quad \forall i, j$$

$$LeftWall \;:\; \omega_{i,j} = -\frac{2\psi_{2,j}}{(\Delta x)^2}, \quad for 2 \leq j \leq j_{max-1} \tag{3.1}$$

$$RightWall \;:\; \omega_{imax,j} = -\frac{2\psi_{imax-1,j}}{(\Delta x)^2}, \quad for 2 \leq j \leq j_{max-1} \tag{3.2}$$

$$BottomWall \;:\; \omega_{i,1} = -\frac{2\psi_{i,2}}{(\Delta y)^2}, \quad for 2 \leq i \leq i_{max-1} \tag{3.3}$$

$$TopWall \;:\; \omega_{i,jmax} = -\frac{2\psi_{i,jmax-1} + 2\Delta y U}{(\Delta y)^2}, for 2 \leq i \leq i_{max-1} \tag{3.4}$$

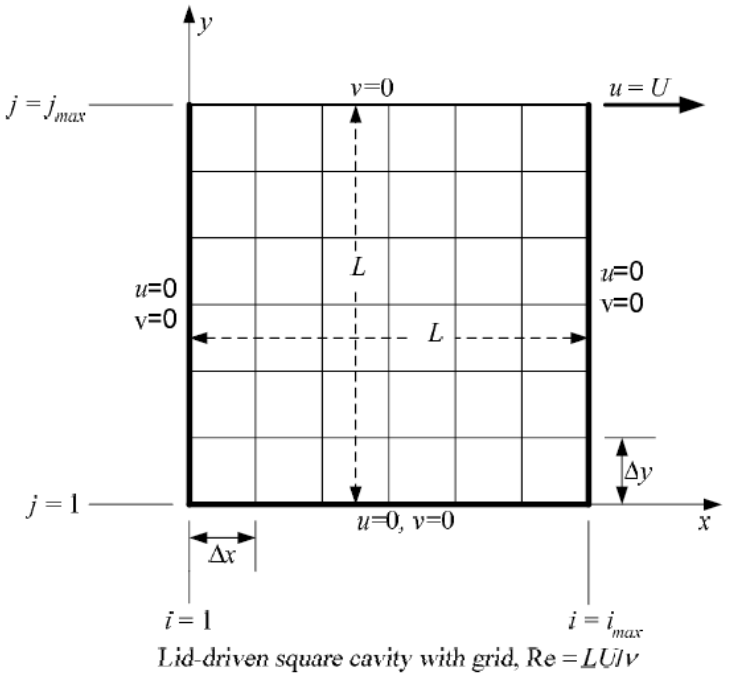


Figure 3.1: The problem domain of Lid-Driven cavity

We tackled this problem solving a $i_{max}xj_{max}$ size matrix though iterative solver. In our case we used Gauss-Seidel method to converge to our solution by minimizing error. The error calculation criteria used is as follows,

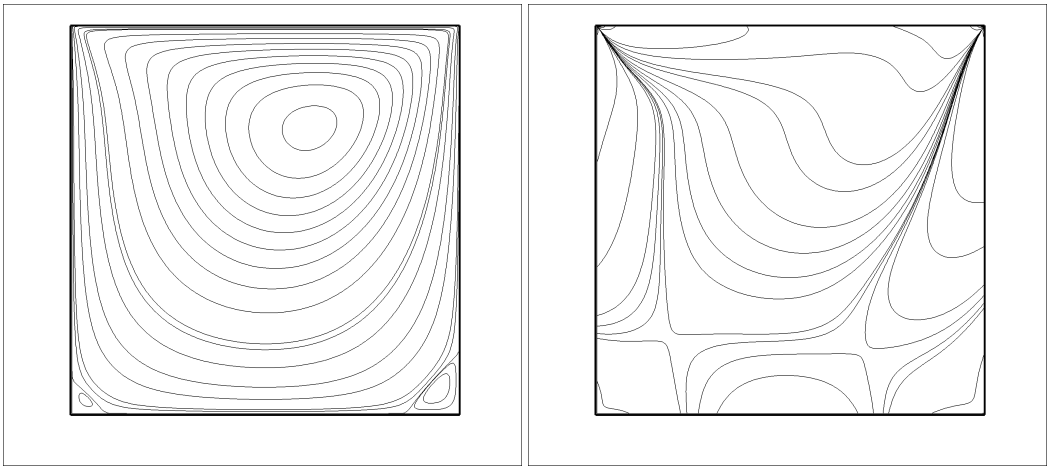
$$\frac{\Sigma_{i,j}\left|\omega_{i,j}^{k+1}-\omega_{i,j}^{k}\right|}{\Sigma_{i,j}\left|\omega_{i,j}^{k+1}\right|}<\epsilon,\ (\epsilon\approx 10^{-5} or other acc petable) \tag{3.5}$$

For higher Reynolds number we had few difficulty in achieving converged solutions. Thus we had to used successive over relaxation to converge our solution and step up to higher Reynolds from lower Reynolds number instead of jumping directly to higher Reynolds number. At higher Reynolds number we can clearly see the secondary and tertiary vortices being formed at the corners.

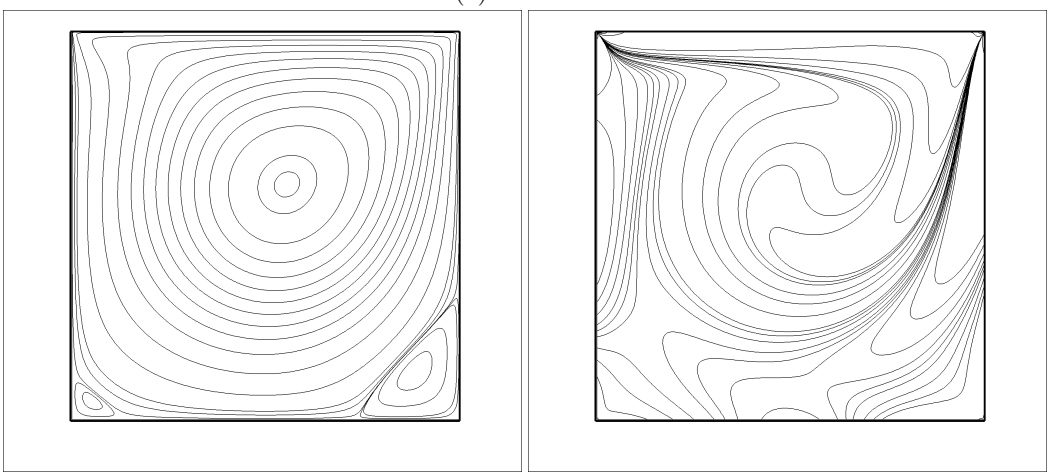
The Pseudo-Code is as follows,

```
do{
        SetBoundaryConditions
        Iterate psi using Gauss-Seidel
        Iterate omega using Gauss-Seidel
        Calculate u and v
        Calculate error
}while error>10^-5
```

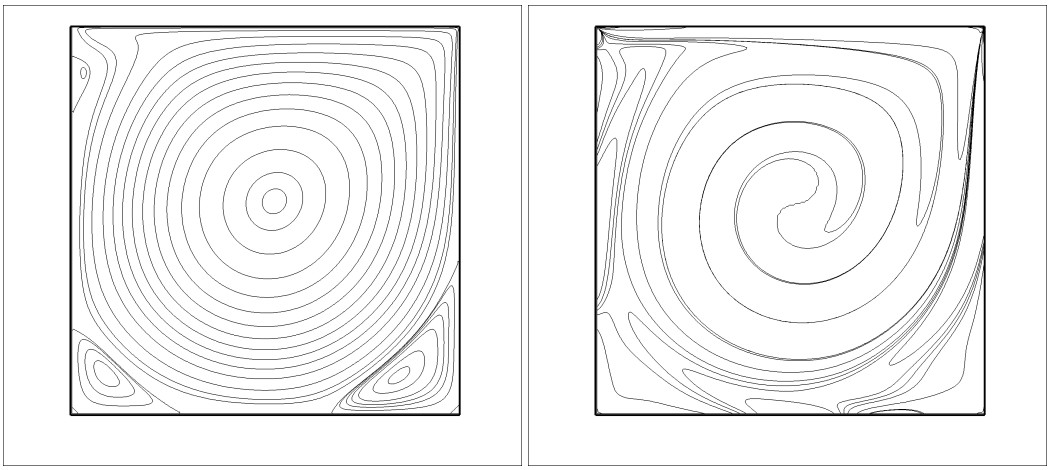
Following generated outputs demonstrate the flow pattern of incompressible viscous fluid at different Reynolds number([6]).



(a) Re=100



(b) Re=400



(c) Re=2000

Figure 3.2: Stream Lines and Vorticity plots at grid size 257 and Reynolds number 100, 400 and 2000 respectively for Lid-Driven cavity problem.

3.2 Unsteady State Lid-Driven Cavity

The problem is to find the evolution of flow of a incompressible viscous fluid in a square domain where the top lid is moving with velocity U. The governing equations are (**??**) (2.6), whereas the discritized equations are (2.7), (2.8) and (2.9).

The purpose of this problem is to get familiar with the unsteady state formulation of the Lid-Driven cavity problem via Alternate Direction Implicit (ADI) method and implementation of Tri Diagonal matrix algorithm. This problem requires getting the flow pattern of incompressible viscous fluid after certain time interval till the flow reaches final Steady state.

The Boundary conditions for this problem are,

$$\psi = 0, \forall i, j$$

$$LeftWall \quad : \quad \omega_{i,j} = -\frac{2\psi_{2,j}}{(\Delta x)^2}, \quad for 2 \leq j \leq j_{max-1} \tag{3.6}$$

$$RightWall \quad : \quad \omega_{imax,j} = -\frac{2\psi_{imax-1,j}}{(\Delta x)^2}, \quad for 2 \leq j \leq j_{max-1} \tag{3.7}$$

$$BottomWall \quad : \quad \omega_{i,1} = -\frac{2\psi_{i,2}}{(\Delta y)^2}, \quad for 2 \leq i \leq i_{max-1} \tag{3.8}$$

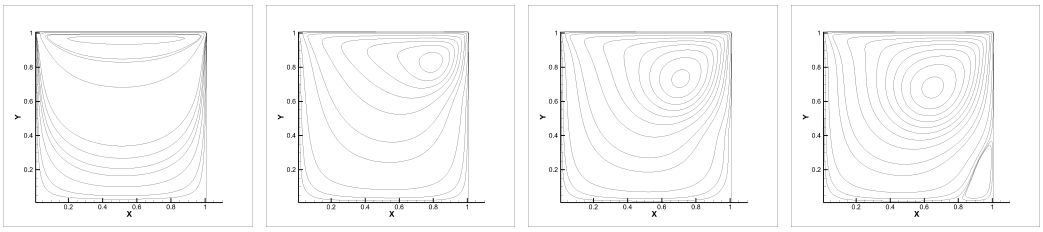
$$TopWall \quad : \quad \omega_{i,jmax} = -\frac{2\psi_{i,jmax-1} + 2\Delta y U}{(\Delta y)^2}, for 2 \leq i \leq i_{max-1} \tag{3.9}$$

We introduced Tri-Diagonal Matrix Algorithm (TDMA) to Solve ω instead of conversing it by using Gauss-Seidel. Since TDMA is an exact solver whereas Gauss-Seidel is an iterative method, this upgrade gave us better performance.

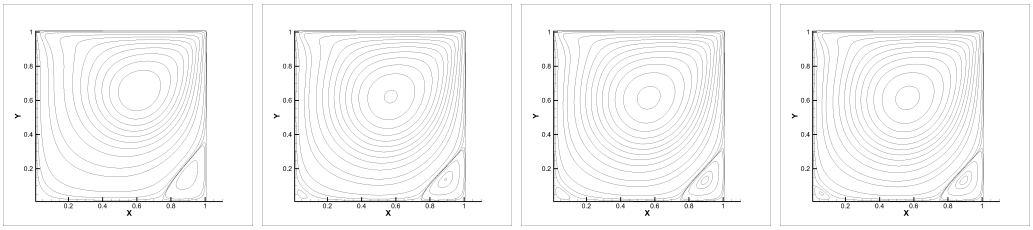
The Pseudo-Code used for this problem is as follows,

```
do{
        SetBoundaryConditions
        Iterate psi using Gauss-Seidel
        Solve omega using TDMA
        Calculate u and v
        Calculate error
        Write File after certain time step (Iteration)
}while error>10^-5
```

The following results are obtained for Reynolds number 400 at *gridsize* 129 at *timestep* 0.001. Let T represents total time taken to reach steady state then following are the results at different time steps.



(a) t=1 (b) t=T/16 (c) t=2T/16 (d) t=3T/16



(e) t=T/4 (f) t=2T/4 (g) t=3T/4 (h) t=T

Figure 3.3: The evolution of stream function in unsteady state Lid-Driven cavity for Reynolds number 400.

3.3 Staggered Cavity

In this problem a square domain with two square cavities on the lower right and upper left of the domain was given with top and bottom lids moving anti-parallel with same velocity ([7])).

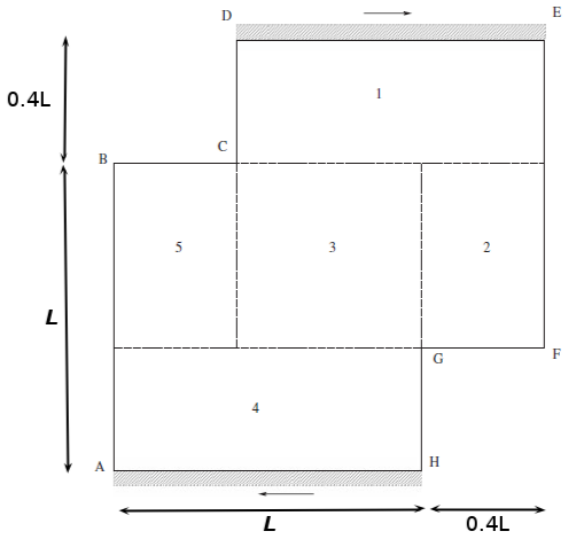


Figure 3.4: The problem domain for staggered cavity.

This problem is difficult to solve because of the blocked domains at the upper left and lower right corner. To overcome this problem we had three options.

- Force zero entries in the omega and psi at block domains after every iteration.
- Divide the domain into two smaller sub-domains and solve these domains individually.
- Divide the domain into five smaller sub-domains and solve these domains individually.

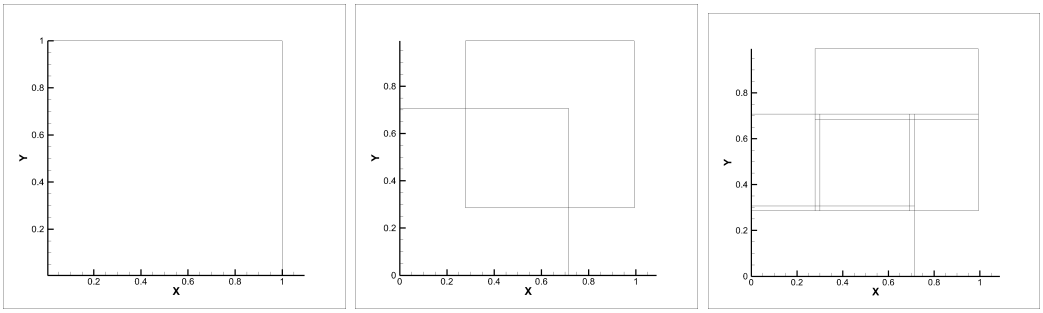
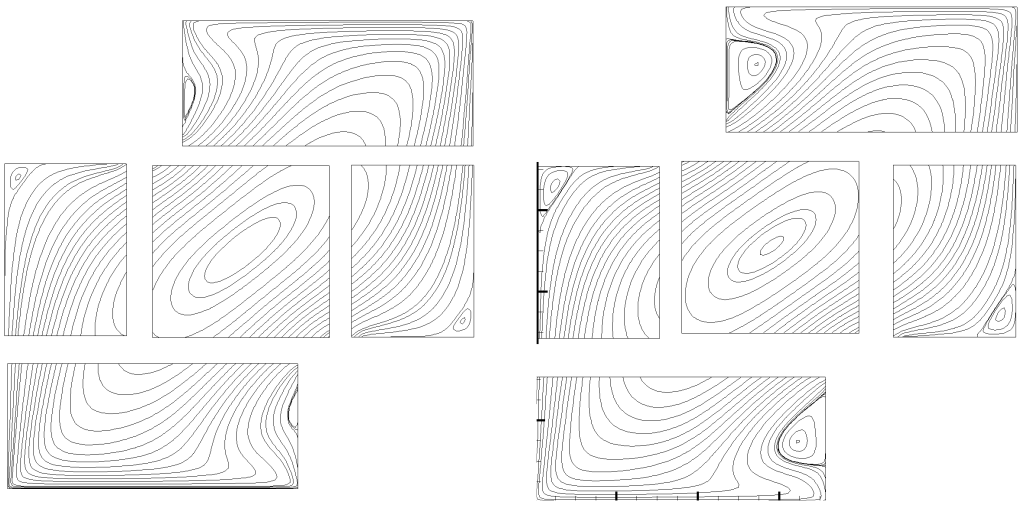


Figure 3.5: Domains representing the Computation part of various options.

First option was rejected because it requires unnecessary solving of two extra blocks which were not in domain. The second option lead us to solution but was computationally inefficient as it required unnecessary computation of two sub domains with larger intersection. Which lead us to use third option which is both computationally efficient and makes the code easy to parallelize.

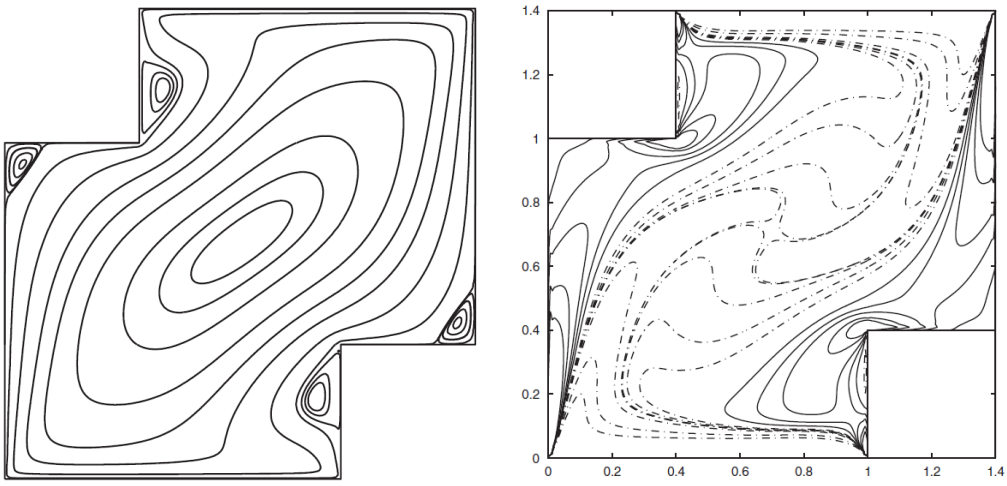


(a) Re 400 (b) Re 800

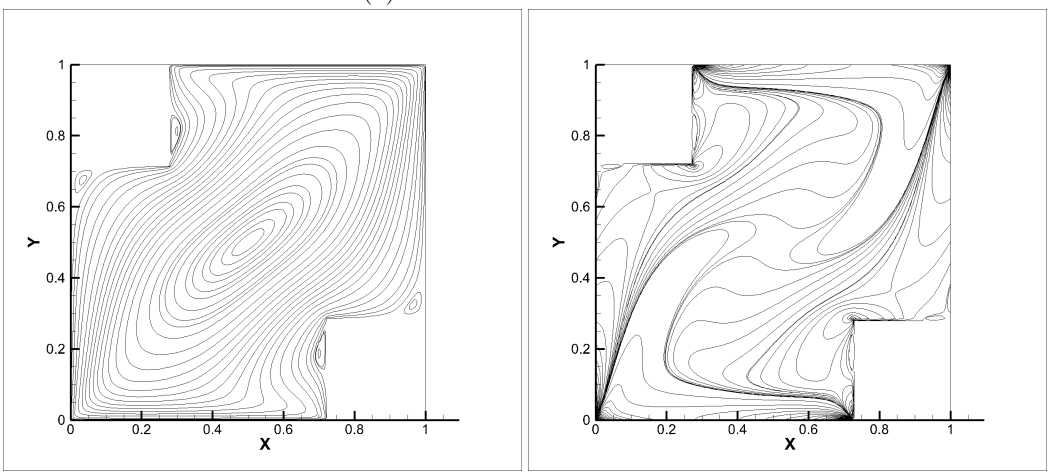
Figure 3.6: Sample demonstration of Stream line computation by dividing the domain into 5 blocks.

Psudo-Code for this problem

```
function Solve(BLOCK){
        Iterate psi using Gauss-Seidel
        Solve omega using TDMA
        Calculate u and v
}
do{
        SetBoundaryConditions
        Solve(BLOCK1)
        Solve(BLOCK2)
        Solve(BLOCK3)
        Solve(BLOCK4)
        Solve(BLOCK5)
        Calculate error
}while error>10^-5
```

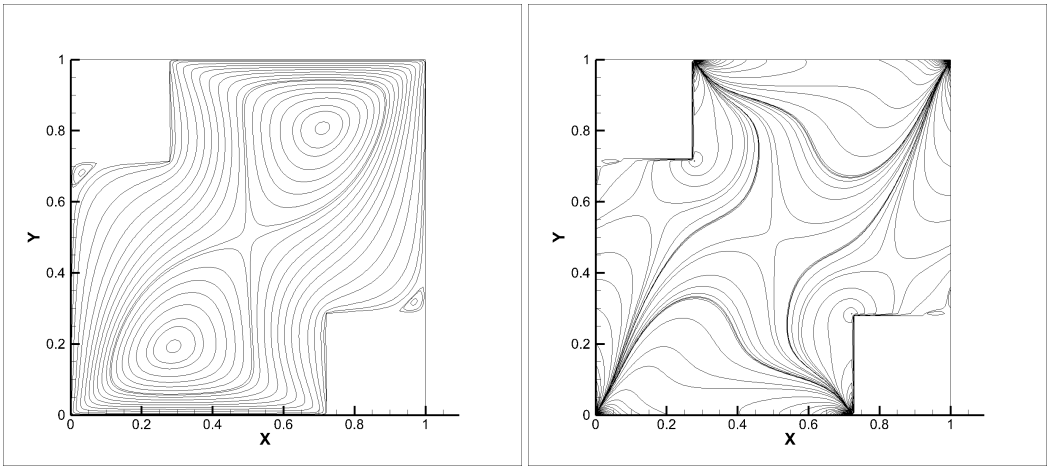


(a) Results from Zhou et al

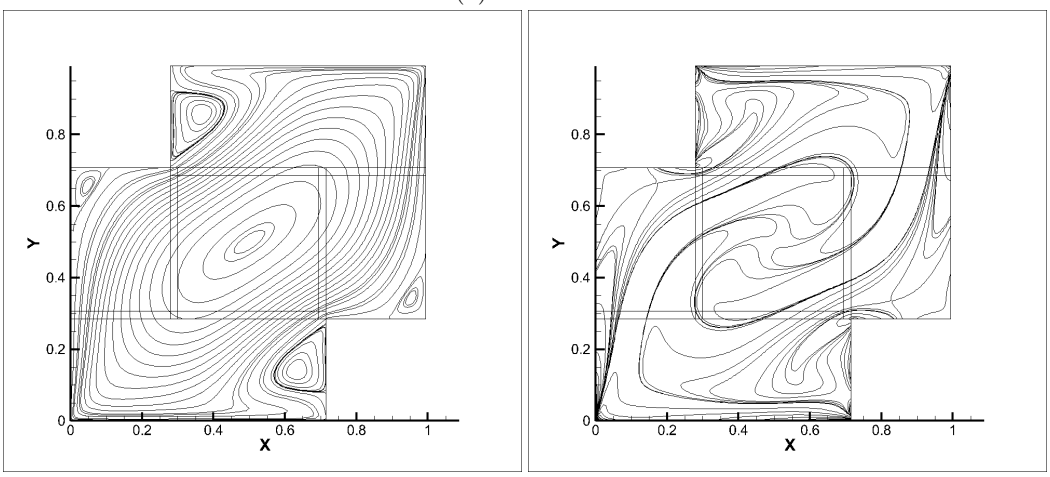


(b) Our Results

Figure 3.7: Streamlines and Vorticity plot in Staggered Cavity for Re 400



(a) Re=100



(b) Re=1000

Figure 3.8: The Streamlines and Vorticity plot in Staggered Cavity for Reynolds number 100 and 1000 respectively.

Remark 3.3.1. As we can see the two primary vortices are combing into one as we go from Reynolds number 100 to 400. ([3])

3.4 Cross Cavity

The domain of this problem is as follows and top and bottom lids are moving anti-parallel to each other. In this problem we have used the same approach to solve that we had used to solve staggered cavity problem that is by diving the domain into five sub-domains.

Boundary conditions used to solve this domain is as follows,

$$
\begin{aligned}
LeftWall: \quad & \omega_{0,j} = \tfrac{v_{0+1,j}-v_{0,j}}{\Delta x} && \omega_{gridi-k,j} = \frac{v_{gridi-k+1,j} - v_{gridi-k,j}}{\Delta x} \\
RightWall: \quad & \omega_{k-1,j} = \tfrac{v_{k-1,j}-v_{k-2,j}}{\Delta x} && \omega_{gridi-1,j} = \frac{v_{gridi-1+1,j} - v_{gridi-2,j}}{\Delta x} \\
BottomWall: \quad & \omega_{i,0} = -\tfrac{u_{i,1}-u_{i,0}}{\Delta y} && \omega_{i,l} = -\frac{u_{i,l+1} - u_{i,l}}{\Delta y} \\
TopWall: \quad & \omega_{i,gridj-l-1} = -\tfrac{u_{i,gridj-l-1}-u_{i,gridj-l-2}}{\Delta y} && \omega_{i,gridj-1} = -\frac{u_{i,gridj-1} - u_{i,gridj-2}}{\Delta y}
\end{aligned}
$$

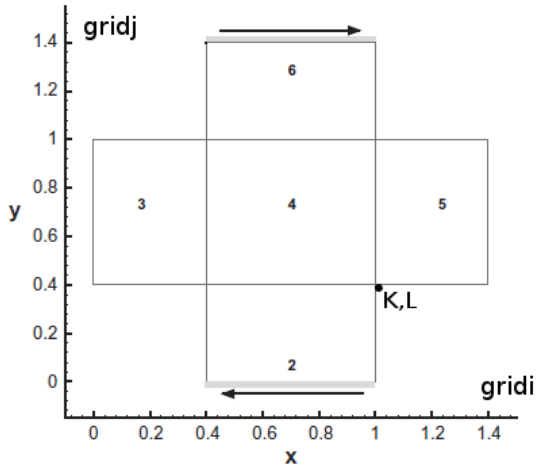
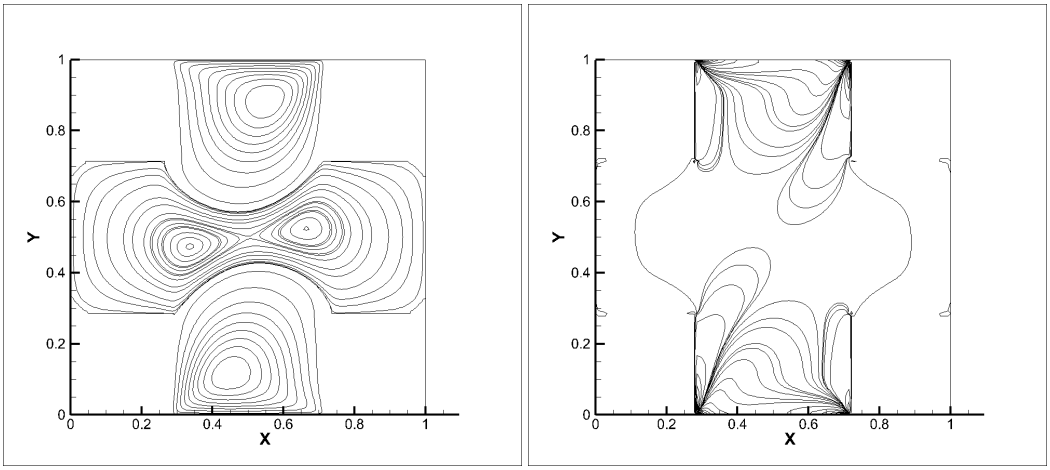
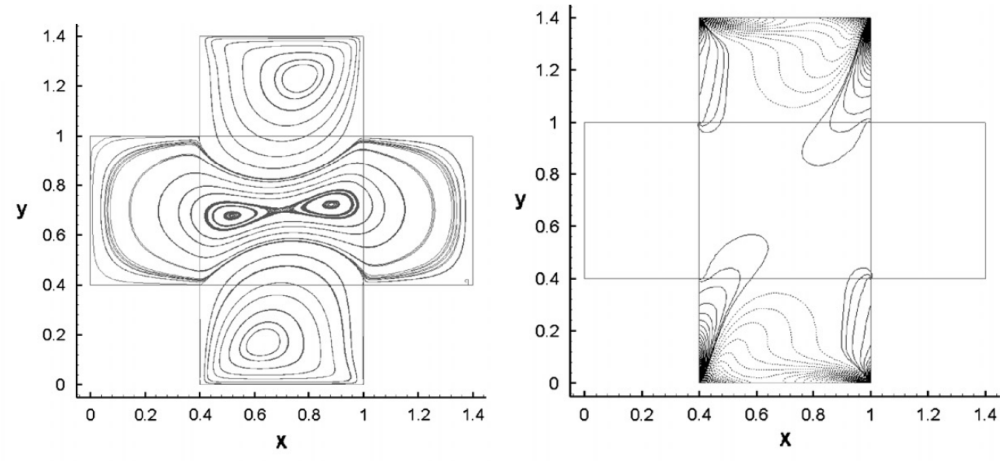


Figure 3.9: Problem domain of Cross Cavity.

Remark 3.4.1. Pseudo-Code for this problem is same as of Staggered cavity.



(a) Our Results



(b) Results from De Vicente et al [2]

Figure 3.10: Stream Lines and Vorticity plots in Cross Cavity

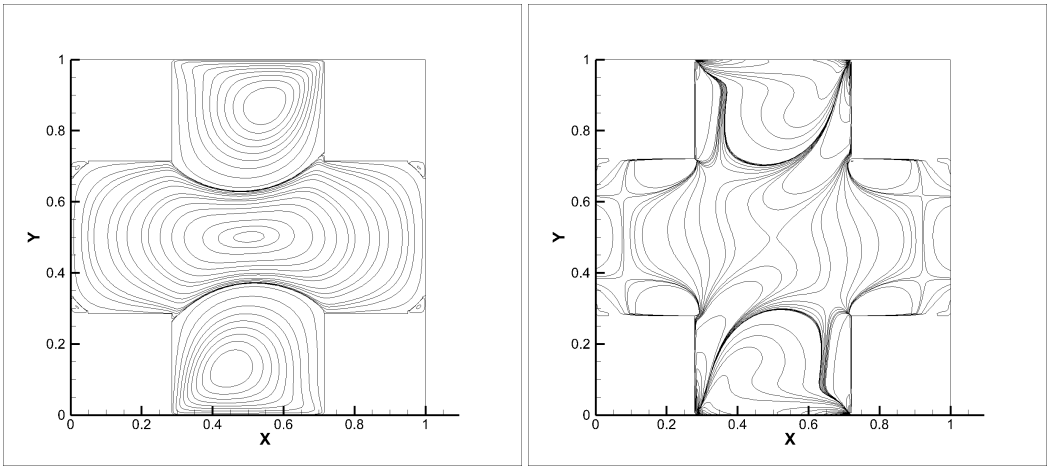


Figure 3.11: Stream Lines and Vorticity plots in Cross Cavity at Re 400

3.5 Backward step flow

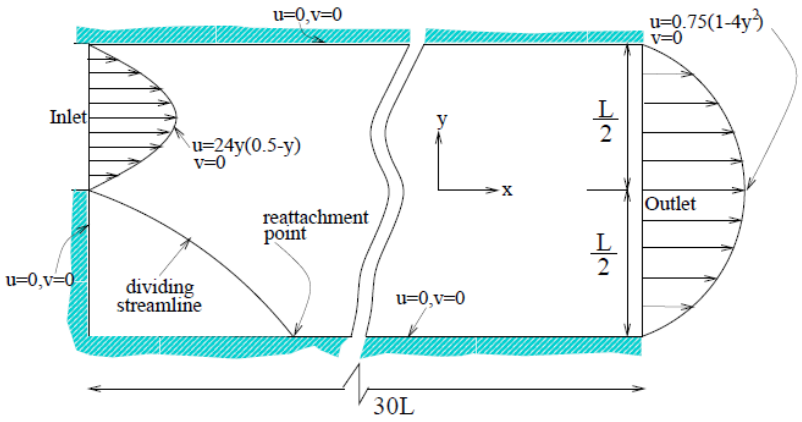


Figure 3.12: Problem domain of the Backward step flow.

In this problem fluid enters the domain with the parabolic profile into a larger cross-section area tube. The outlet conditions correspond to fully developed flow. The importance of this problem is that we are using Inflow and Outflow boundary conditions instead of wall boundary conditions on left and right

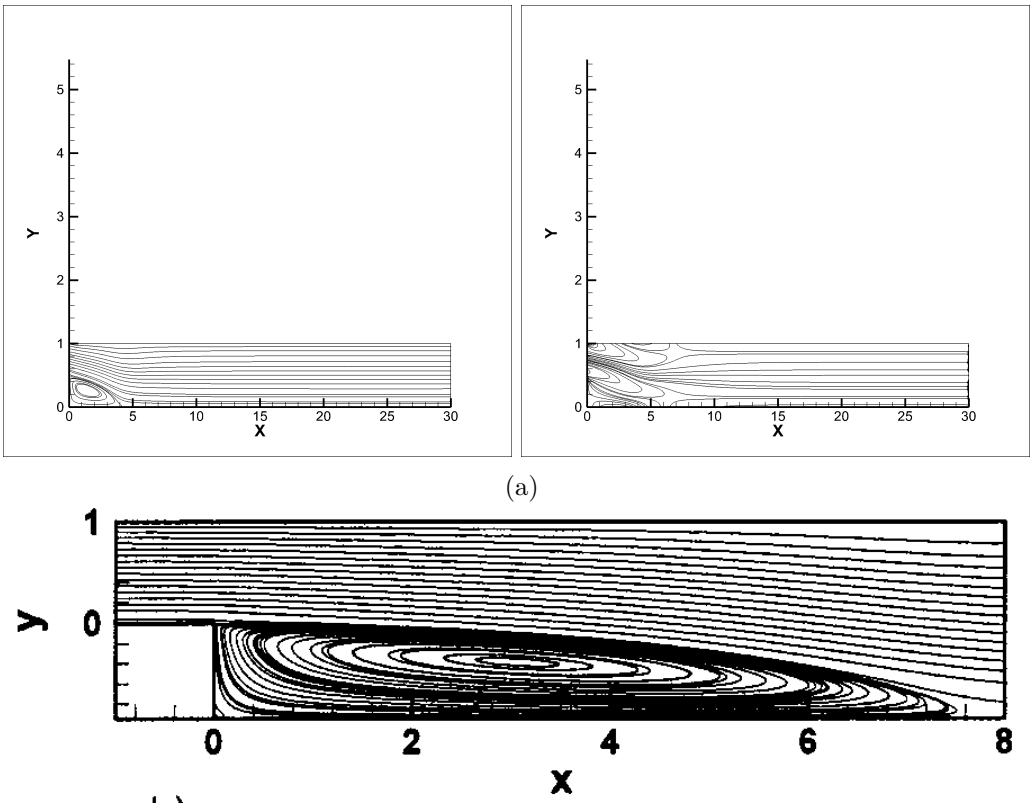
walls respectively. These Boundary conditions are as follows,

$$
\begin{aligned}
TOP &: u = v = \omega = 0; \psi = 0.5 \\
BOTTOM &: u = v = \psi = \omega = 0 \\
INLET &: u = v = \psi = \omega = 0 \qquad at -0.5 \leq y \leq 0 \\
&\quad u = 12y(1-2y), v = 0, \psi = 2y^2(3-4y), \omega = 12(4y-1) at 0 \leq y \leq 0.5 \\
OUTLET &: u = \frac{3}{4}(1-4y^2), v = 0, \psi = \frac{1}{4}(1+3y-4y^3), \omega = 6y
\end{aligned}
$$

The Pseudo-Code used for this problem is as follows,

```
do{
        SetBoundaryConditions
        Iterate psi using Gauss-Seidel
        Solve omega using TDMA
        Calculate u and v
        Calculate error
}while error>10^-5
```

Following is the plots obtained and compared with one of the prior work of G.Biswas et.al.([2])



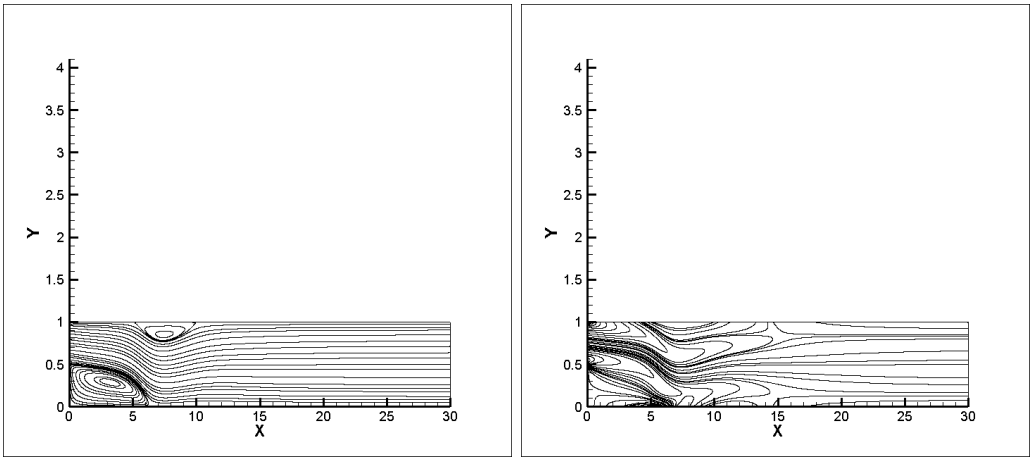
(a)

(b)

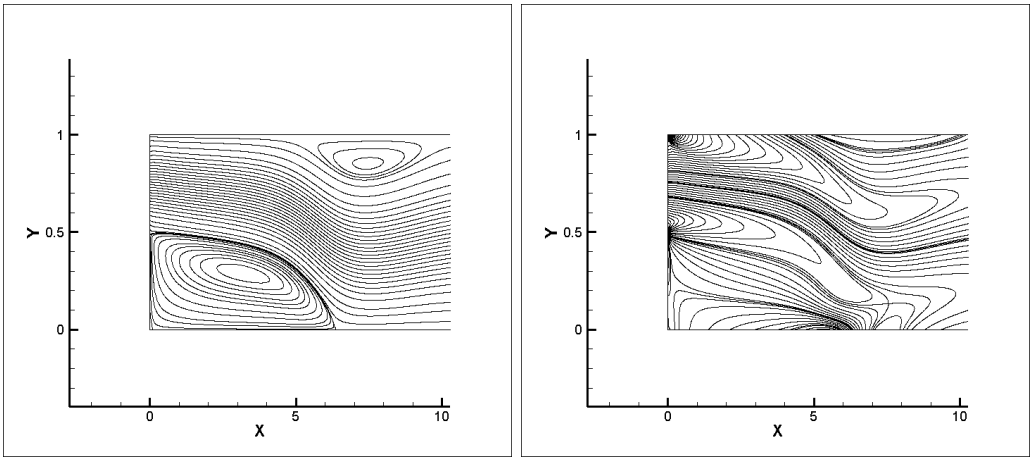
Figure 3.13: Plots at Reynolds number 400. ; (a) Stream Lines and Vorticity plots calculated. (b) Stream Lines plot from G.Biswas et. al.

After the inlet the flow experiences a larger cross sectional area hence vortices are being formed at the lower left corner. The flow fully develops after travelling a length of about 15 times the cross-sectional diameter.

Increasing the grid size to obtain finer results and secondary vertex was very time taking process in serial code. Thus introducing CUDA to solve problem domain at higher grid size and higher Reynolds number faster.



(a)

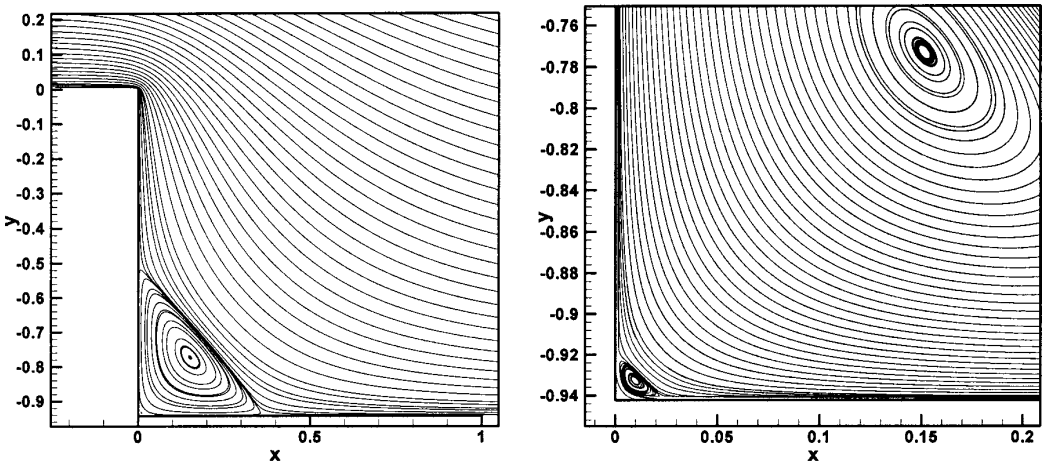


(b)

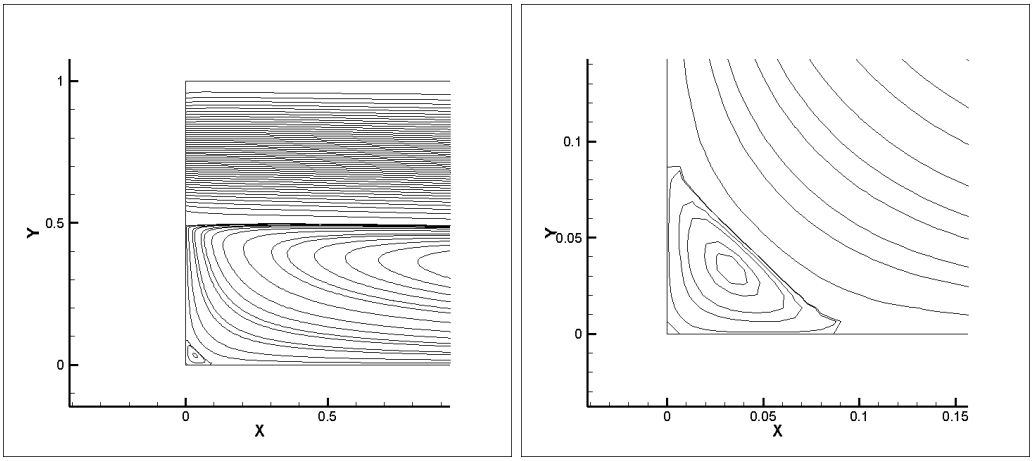
Figure 3.14: Stream Lines and Vorticity plots at Reynolds number 800 on grid size 150x4500. Expansion ratio H/h = 0.15 ; (a) Plots calculated. (b) Zoom to clarify the vortex achieved.

The above results were obtained at Reynolds number 800 using grid size 150x4500. The code was simulated till error 0.001.

Further zooming to the origin to show additional secondary vortex.



(a)



(b)

Figure 3.15: Stream Lines and its further zoom. ; (a) Plots at Re=1 from G. Biswas et.al. (b) Plots calculated at Re=800.

The codes for above simulations were ran on Serial and CUDA platform. Time taken by CUDA and serial code to solve 150x4500 size matrix upto convergence error 0.001,

Size	Serial Code	Cuda Code	Speed up
150x4500	37h 27m	1h 33m	24

3.5.1 Introducing CUDA

The current speed-up was achieved by parallelising the nested loops and various algorithms used in the serial code. This speed up can be further increased by optimizing the algorithms and their CUDA reflections. Since we are using algorithms like Gauss-Seidel, which needs values of current time-step as well as values of last time-step, there is scope of further optimising the current approach towards CUDA reflection of such algorithms.

One of the problem faced while working in global memory and had to change psi entries while other threads might need the old values. This gave two possibilities,

- Store the values of the new iteration in a separate variable. But then i had to swap the variables after each iteration.
- Jump to the Jacobi's method to solve psi iterations.

Thus, currently Jacobi's method was used to parallelize the psi iterations at the cost of few increased number of iterations. Another option faced was using a 2 dimensional array over 1 dimensional. Since CUDA calculate over single dimensional array I had to convert all the two dimensional array into single dimension before sending it to CUDA and vice versa. This of-course means I had to define kernel with respect to single dimensional array.

Conclusion and Summary

In the first phase I have successfully implemented all the above mentioned problems through serial codes. This lays the foundation of working process of obtaining solution through block decomposition. This phase of my work consists of parallel Application programming interface(API) implementation to solve all smaller sub domains parallely instead of subsequently. Since we already have computational results by solving these smaller sub domains sequentially, we've used CUDA to reduce the computational time in this phase.

I've preferred CUDA(Compute Unified Device Architecture) programming model which is created by NVIDIA and implemented by the GPU's(Graphics Processing Units). The GPU, as a specialized processor, addresses the demands of real-time high-resolution 3D graphics compute-intensive tasks. Now GPUs have evolved into highly parallel multi-core systems allowing very efficient manipulation of large blocks of data. These design are more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel.

Bibliography

[1] Garrett Birkhoff, Richard S Varga, and David Young. Alternating direction implicit methods. *Advances in computers*, 3:189–273, 1962.

[2] G Biswas, M Breuer, and F Durst. Backward-facing step flows for various expansion ratios at low and moderate reynolds numbers. *Journal of fluids engineering*, 126(3):362–374, 2004.

[3] Javier De Vicente, Daniel Rodríguez, Vassilis Theofilis, and Eusebio Valero. Stability analysis in spanwise-periodic double-sided lid-driven cavity flows with complex cross-sectional profiles. *Computers & Fluids*, 43(1):143–153, 2011.

[4] Urmila Ghia, Kirti N Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

[5] Jiten C Kalita, DC Dalal, and Anoop K Dass. A class of higher order compact schemes for the unsteady two-dimensional convection–diffusion equation with variable convection coefficients. *International Journal for Numerical Methods in Fluids*, 38(12):1111–1131, 2002.

[6] Natarajan Ramanan and George M Homsy. Linear stability of lid-driven cavity flow. *Physics of Fluids*, 6:2690, 1994.

[7] Predrag M Tekić, Jelena B Rađenović, Nataša Lj Lukić, and Svetlana S Popović. Lattice boltzmann simulation of two-sided lid-driven flow in a staggered cavity. *International Journal of Computational Fluid Dynamics*, 24(9):383–390, 2010.

[8] YC Zhou, BSV Patnaik, DC Wan, and GW Wei. Dsc solution for flow in a staggered double lid driven cavity. *International Journal for Numerical Methods in Engineering*, 57(2):211–234, 2003.