Applications of Fast Fluid Dynamics

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Balavarun Pedapudi
ID No. 2013B5AB0462P

Under the supervision of:

Dr. Aakash C. Rai

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS
May 2018

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS

Abstract

Bachelor of Engineering (Hons.) Manufacturing Engineering

Applications of Fast Fluid Dynamics

by Balavarun Pedapudi

There is a need for real time simulations, especially in situations where time is of essence, for example simulating the spread of a biological contaminant requires results in less than real time so as to prevent the spread of the contaminant. This thesis aims to implement and verify Fast Fluid Dynamics, a fast and stable numerical method for incompressible fluid flows. The implementation should be faster than traditional CFD methods to verify the claim that FFD is faster. The implementation at its current capacity can evolve a lid driven cavity, a benchmark CFD problem with reasonable accuracy and is faster than a commercial CFD solver.

Acknowledgements

I would like to thank my thesis advisor Dr. Aakash C. Rai for proposing this unique problem statement for the thesis and for motivating and guiding me throughout the span of this endeavour.

I would also like to thank my family for their support and love since day one.

Contents

List of Figures

Chapter 1

Fast Fluid Dynamics

1.1 Introduction

Traditional fluid solvers evolve system by numerically solving the Navier-Stokes equations. These methods albeit thorough are very expensive computationally and are not suited to evolve for cases where the resources are not extensive or if the results of the simulation are required in a much lesser time than a full CFD solver. For example, while evacuating a building during a fire, it would be very helpful to predict the spread of smoke, or other harmful gases. Time is a very important factor in situations like these and there is a need for a quick solution rather than scientific precision in the simulation.

Fast Fluid Dynamics was developed to cater to this need, the seminal paper for this topic was published by Jos Stam [5]. Fast Fluid Dynamics has been used to model real time or faster than real time simulations [8] of airflow in buildings and the speed has been reported to be 50 times faster than CFD processes, however it depends strongly on the grid resolution, time step and flow conditions.

1.2 Fast Fluid Dynamics - Theory

Fast Fluid Dynamics is used to simulate incompressible fluid flows and is unconditionally stable which gives it an advantage over CFD solvers as the Courant factor is not a limiting factor for the time step to be taken.

The Continuity equation for incompressible fluids is given by

$$\nabla . u = 0 \tag{1.1}$$

Where u is the velocity vector field.

The Navier-Stokes equation is represented as

$$\frac{\partial u}{\partial t} = -(u.\nabla)u - \frac{1}{\rho}\nabla p + \nu\nabla^2 u + f \tag{1.2}$$

where ν is the kinematic viscosity of the fluid, ρ is the density and f is an external force.

The Helmholtz-Hodge decomposition of vector fields is used to proceed. This result states that any vector field w can be split as

$$w = u + \nabla q \tag{1.3}$$

Here the vector field w is split into u, the divergence free component of w and ∇q, the gradient of a scalar field q.

Since u is divergence free,

$$\nabla.u = 0 \tag{1.4}$$

A projection operator P is now introduced, which when acted upon a vector field w gives the divergence free part.

$$P(w) = u \tag{1.5}$$

Taking the divergence of L. H. S. and the R. H. S. of the equation 1.3

$$\nabla.w = \nabla^2 q \tag{1.6}$$

This is a Poisson equation, a solution to this is used to obtain u

$$\frac{\partial u}{\partial t} = \mathbf{P}\big(-(\mathbf{u}.\nabla)\mathbf{u} + \nu\nabla^{\mathbf{2}}\mathbf{u} + \mathbf{f}\big) \tag{1.7}$$

Using the fact that the velocity field u is gradient free, and that pressure is a scalar field when acted upon by the projection operator would give zero.

The equation 1.7 is the fundamental equation upon which a stable fluid solver is to be built. This is solved using an initial state $u_0 = u(x, 0)$ and advancing it by a time step Δt in four steps which will be discussed in detail.

The f term in the equation 1.7 is added iteratively and will be called as the force step. The $(u.\nabla)u$ step resembles an advection equation which is solved using the method of characteristics,

This will be referred to as the advection step. The term $\nu\nabla^2 u$ in the R. H. S. of the equation is solved in the diffusion step. The final step is the projection step when the projection operator is applied.

1.3 Steps for Fast Fluid Dynamics

The u at the start of the timestep is taken as $w_0(x) = u(x,t)$ The force step is easy to evolve, if it is assumed that the force doesn't vary much during the timestep, it is calculated as

$$w_1(x) = w_0(x) + \Delta t F(x,t) \tag{1.8}$$

The next step is the advection step, A disturbance in the domain of the fluid propagates according to the expression $-(u.\nabla)u$. The parameters of the fluid remain conserved along streamlines and this fact is used to back trace the streamline by time Δt and these parameters are used at the next timestep.

The point x is traced back through the velocity field w_1 by time Δt. This defines a path $p(x, -\Delta t)$, the parameters are then obtained by bilinear interpolation of the nearest grid points.

$$w_2(x) = w_1(p(x, -\Delta t)) \tag{1.9}$$

This method is unconditionally stable which makes it possible to take larger Δt without considering the grid size into account.

The next step is to solve the diffusion equation given by the ∇^2 term in the R. H. S. of 1.7

$$\frac{\partial w_2}{\partial t} = \nu\nabla^2 w_2 \tag{1.10}$$

This is a well known problem and is solved in this approach using an implicit method and reducing the system of linear equations to a matrix form.

$$(\mathbf{I} - \nu\mathbf{\Delta t}\nabla^{\mathbf{2}})\mathbf{w_3}(\mathbf{x}) = \mathbf{w_2}(\mathbf{x}) \tag{1.11}$$

Discretizing the ∇^2 operator and using the implicit method yields a system of sparse linear system.

The fourth step involves applying the projection operator, which makes the resultant vector field divergence free. This involves resolving the Poisson problem in 1.6

$$\nabla^2 q = \nabla . w_3 \tag{1.12}$$

Calculating q by solving this Poisson equation, the gradient of q is to be calculated and subtracted from w_3 to get the resultant divergence free velocity vector field

$$w_4 = w_3 - \nabla q \tag{1.13}$$

The Poisson equation is solved by discretizing it into a system of sparse linear system, and representing it in matrix form.

These are the steps that are to be done in the particular order, iteratively, to obtain the evolution of the velocity vector field through time t in steps of Δt

Chapter 2

Lid Driven Cavity - FFD implementation

2.1 Lid Driven Cavity

The Lid Driven cavity is a well researched benchmark problem for incompressible fluid flow. In this thesis, a solver which implements Fast Fluid Dynamics for a two dimensional Lid Driven cavity is developed [1]. The code is written using MATLAB/Octave and an open source code [3] has been referred to.

In the code developed during the thesis, all units are non dimensional. The Cavity is taken to be a square of side 1 unit with the lid at the top moving with a speed of 1 unit.

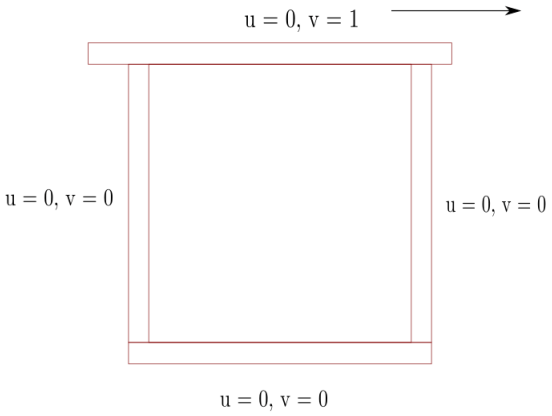


FIGURE 2.1: A 2D Lid Driven cavity

Since this implementation is for a two dimensional Lid Driven cavity, only the x and y components of the velocity are considered.

The x component of the velocity vector field is denoted by u and the y component by v. This is different from the previous notation used in section 1.2

The boundary conditions of the lid as seen in fig 2.1 specify that both x and y components of the velocity at the side and bottom lids are 0. The x component of the velocity at the top of the lid is 1 unit while the y component of velocity is zero. No-slip conditions are assumed at the boundary interface.

FFD is implemented in the order of steps discussed in 1.3. Since the external force in this case is zero, the first force addition term can be skipped.

2.2 Advection step

The advection step from 1.9 needs to be solved first. This is done using the method of characteristics which states that the velocity of the fluid at a particular time t is same as the velocity it had a time step dt ago at a distance dx in the streamline.

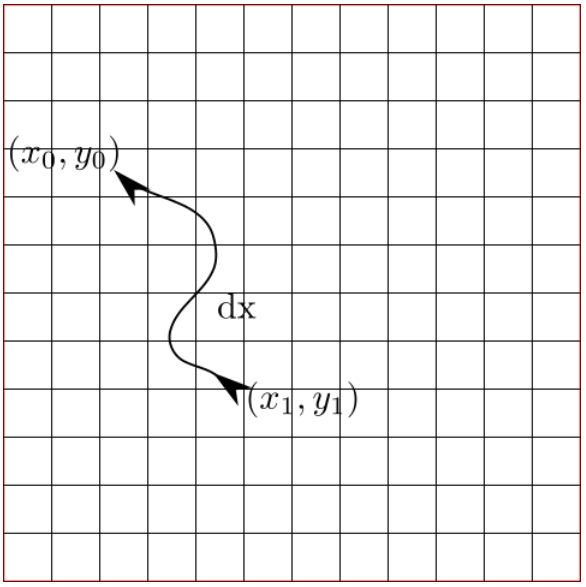


FIGURE 2.2: Back tracing particles through streamlines

Velocity at the time step is determined by back tracing the particle through its streamline and using bi-linear interpolation of the nearby cells to obtain the average velocity.

As seen in figure 2.2, to perform the advection step, the particle needs to be traced back through its streamline over a time dt and the velocity at the time is obtained by bi-linear interpolation of the nearby grid points. Assuming that at time t, the u and v velocities at a point (x_0, y_0) are required, tracing the path back to its previous location at say (x_1, y_1), the bi-linear interpolation is obtained as a function of (x_1, y_1) and the velocities of the neighbouring grid points.

First, computing u in terms of the velocities of the neighbouring points, the u at the top and bottom boundaries are interpolated using x_1 and these are again interpolated along the y direction.

$$u_{lower} = (x_{right} - x_1)u_{bottom\ left} + (x_1 - x_{left})u_{bottom\ right} \tag{2.1}$$

$$u_{upper} = (x_{right} - x_1)u_{top\ left} + (x_1 - x_{left})u_{top\ right} \tag{2.2}$$

$$u_{new} = (y_{top} - y_1)u_{lower} + (y_1 - y_{bottom})u_{upper} \tag{2.3}$$

Similarly for v

$$v_{lower} = (x_{right} - x_1)v_{bottom\ left} + (x_1 - x_{left})v_{bottom\ right} \tag{2.4}$$

$$v_{upper} = (x_{right} - x_1)v_{top\ left} + (x_1 - x_{left})v_{top\ right} \tag{2.5}$$

$$v_{new} = (y_{top} - y_1)v_{lower} + (y_1 - y_{bottom})v_{upper} \tag{2.6}$$

In case the projected path goes through a particular boundary, the velocity at the boundary is taken as the flow doesn't exist outside the bounds of the cavity.

The vectorized implementation of this step in particular is discussed in Appendix A

2.3 Diffusion Step

The next step is solving the diffusion step in 1.10.

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u \tag{2.7}$$

Discretizing the above equation in a 2-dimensional space yields,

$$\frac{u_{i,j}^{t+1} - u_{i,j}^{t}}{\Delta t} = \nu \left(\frac{u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1} - 2u_{i,j}^{t+1}}{\Delta x^2} + \frac{u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1} - 2u_{i,j}^{t+1}}{\Delta y^2} \right) \tag{2.8}$$

This particular case has $\Delta x = \Delta y$. Now $u_{i,j}^{t+1}$ is to be obtained from $u_{i,j}^{t}$. Representing the entire set of equations as the product of two matrices,

$$u_{i,j}^{t+1} - \lambda(4u_{i,j}^{t+1} - u_{i+1,j}^{t+1} - u_{i-1,j}^{t+1} - u_{i,j+1}^{t+1} - u_{i,j-1}^{t+1}) = u_{i,j}^{t} \tag{2.9}$$

Where $\lambda = \frac{\nu \Delta t}{\Delta x^2}$

Boundary conditions should be applied, i.e. x component of the velocity should be zero every other boundary except the lid. The y component of velocity is zero at all boundary zones.

$$u_{0,j}^{t+1} = 0 \tag{2.10}$$
$$u_{i,0}^{t+1} = 0 \tag{2.11}$$
$$u_{N,j}^{t+1} = 1 \tag{2.12}$$
$$u_{i,N}^{t+1} = 0 \tag{2.13}$$

The diffusion equation is to be solved for a square of $N-2$ cells length. The final matrix equation would be

$$\begin{bmatrix} 1-4\lambda & \lambda & 0 & \cdots & \lambda & 0 & \cdots & 0 \\ \lambda & 1-4\lambda & \lambda & \cdots & 0 & \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda & 0 & \cdots & 1-4\lambda \end{bmatrix} \times \begin{bmatrix} u_{2,2}^{t+1} \\ u_{2,3}^{t+1} \\ \vdots \\ u_{N-2,N-2}^{t+1} \end{bmatrix} = \begin{bmatrix} u_{2,2}^{t} + \lambda u_{1,2} \\ u_{2,3}^{t} + \lambda u_{1,3} \\ \vdots \\ u_{N-2,N-2}^{t} + \lambda u_{N-1,N-1}^{t} \end{bmatrix}$$

Which can be represented as

$$A_{(N-2)^2 \times (N-2)^2} \times {u_{i,j}^{t+1}}_{(N-2)^2 \times 1} = b(u^t)_{(N-2)^2 \times 1} \tag{2.14}$$

Instead of taking the inverse of the A matrix and passing it as an argument for every time step, MATLAB has a special function mldivide [4] which is optimized to solve for a linear set of equations when sparse matrices are involved. The A matrix for diffusion is after all a penta-diagonal sparse matrix.

2.4 Projection step

The next step would be to make the resultant u velocity field divergence free to satisfy the condition that the fluid is incompressible which was discussed in 1.1.

For the projection step, the divergence free part of the velocity field obtained after the diffusion step is to be removed. This involves calculation of q, the scalar field as was shown in 1.3. Taking the divergence of both R. H. S. and L. H. S. of 1.3,

$$\nabla . w = \nabla^2 q \tag{2.15}$$

The w in this case would be the u velocity field after the diffusion step. To calculate q, this Poisson equation needs to be solved.

Since the velocity field is already known, calculation of $\nabla.u$ is fairly straightforward, keeping in mind the boundary conditions for the simulation. Discretizing the ∇^2 operator for the 2 dimensional case, the projection step now becomes

$$\nabla.w = \frac{q_{i+1,j} + q_{i-1,j} + q_{i,j+1} + q_{i,j-1} - 4q_{i,j}}{\Delta x^2} \tag{2.16}$$

Since $\Delta x = \Delta y$ for the particular grid. Now the system of equations can be represented in a matrix equation which will solve the Poisson equation and give the required q

$$\begin{bmatrix} -\frac{4}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \cdots & \frac{1}{\Delta x^2} & 0 & \cdots & 0 \\ \frac{1}{\Delta x^2} & -\frac{4}{\Delta x^2} & \frac{1}{\Delta x^2} & \cdots & 0 & \frac{1}{\Delta x^2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{\Delta x^2} & 0 & \cdots & -\frac{4}{\Delta x^2} \end{bmatrix} \times \begin{bmatrix} q_{1,1} \\ q_{1,2} \\ \vdots \\ q_{N,N} \end{bmatrix} = \begin{bmatrix} \nabla.u^t_{1,1} \\ \nabla.u_{1,2} \\ \vdots \\ \nabla.u^t_{N,N} \end{bmatrix}$$

This can be solved to obtain q, the scalar field

$$A_{project_{N^2 \times N^2}} \times q_{N^2 \times 1} = \nabla.u_{N^2 \times 1} \tag{2.17}$$

$A_{project}$ also being a sparse penta-diagonal matrix, mldivide is used to solve this set of linear equations rather than passing the inverse of it every time step.

The next part of the projection step is to calculate the gradient of q which is to be subtracted from the velocity field obtained from the diffusion step to make the resultant velocity field divergence free.

$$u_{t+\Delta t} = u - \nabla.q \tag{2.18}$$

This step makes sure that the fluid flow is incompressible and thus gives a stable flow pattern after the time step Δt.

These are the FFD steps applied to a lid driven cavity case. The code developed during this thesis is completely vectorized for faster speeds, most notable difference is observed in the advection step where instead of traversing through the grid one point at a time, which has to go through a large number of points for a high value of N, the vectorized implementation would perform the advection step at a single go, significantly reducing the compute time.

The initial implementation was done using Arrayfire, a general purpose GPU library, since the GPU processes would be fast, especially to solve the Poisson equation. However to compare the speed of the implementation with ANSYS Fluent, a commercial CFD solver available, the code was rewritten using MATLAB to compare the speed of the implementation with Fluent on the same system.

Chapter 3

Results

The steps discussed in the previous chapter which described how to implement Fast Fluid Dynamics in Lid Driven cavity have been applied and the results will be discussed. In this, comparison between the code developed during this thesis, the available Open Source implementation and a commercial CFD solver ANSYS Fluent are compared. In all the cases, a Lid Driven cavity with a grid size of 200×200 is simulated with the lid moving at a speed of 1 unit and the fluid is initially still i.e. both u and v are taken to be zero at all points at the zeroth time step.

3.1 Accuracy

The steady state solution obtained using the Fast Fluid Dynamics implementation is compared to the accepted values of velocity obtained from the Ghia [6] paper.

Shown below are the plots 3.1 3.2 obtained for accuracy of the u velocity profile across the mid point of the cavity for different Reynolds numbers. The results are compared for grid sizes of 64×64, 128×128 and 256×256 to check if the velocity profile at the steady state converges as the number of grid points are increased.

As can be observed, the results converge more and more to the expected u profile as the number of grid points increases. This is a positive test of the convergence of the solution and indicates that grid independence is present.

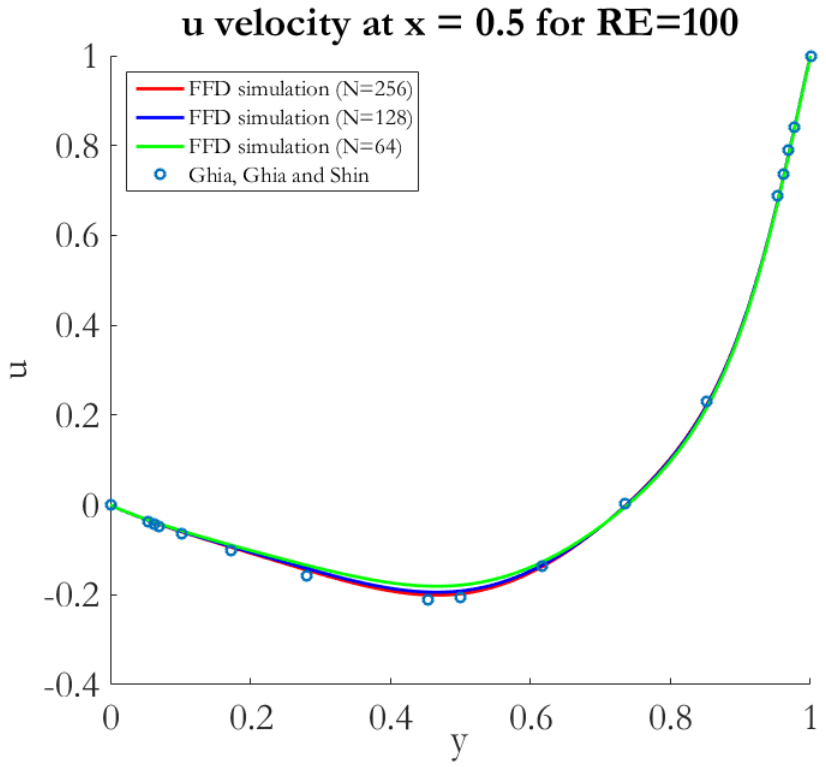


FIGURE 3.1: The u velocity profile across the mid-point of the cavity for a Reynolds number of 100

3.2 Speed

The speed of the code is compared to ANSYS Fluent, a grid of 200×200 is evolved for a flow time of 10 seconds and the compute time of ANSYS, the open source FFD implementation and the code developed during the thesis are compared.

The Lid driven cavity system is solved using the transient state solver in ANSYS Fluent, Since the FFD implementation used in the thesis solves for each time step iteratively and we need to compare similar routines to analyse the speed of both these implementations.

The Fluent simulation being be a CFD implementation, the limit on the time step is imposed by the Courant condition,

$$c = \frac{\Delta t}{\Delta x} \tag{3.1}$$

Where c is the Courant factor which is the determining parameter as to whether a solution will converge or not.

Taking a Courant factor of 0.9, for a square domain of side 1, the Δt would be

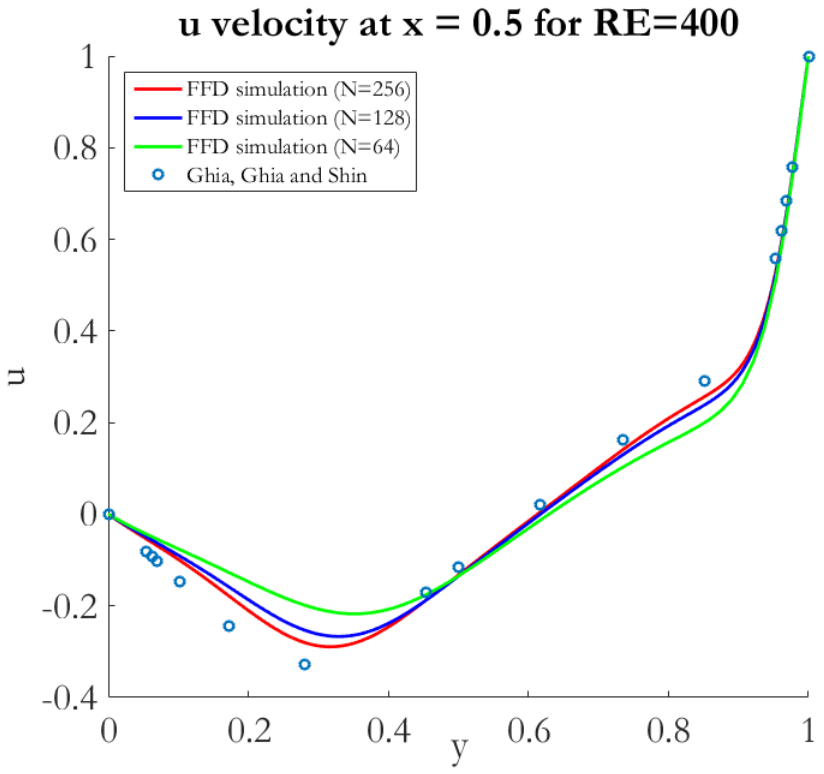


FIGURE 3.2: The u velocity profile across the mid-point of the cavity for a Reynolds number of 400

$$\Delta t = 0.9 \times \frac{1}{200} = 0.0045 \tag{3.2}$$

For a flow time of 10 seconds, the number of iterations are

$$n_{iterations} = \frac{10}{0.0045} = 2222 \tag{3.3}$$

Therefore the CFD implementation would require 2222 iterations.

This is where FFD is advantageous, the time step taken was $dt = 0.02$, which implies that 500 iterations are enough to evolve the system for a flow time of 10 seconds. This cannot be done using traditional CFD processes as the solution would blow up if the courant factor is greater than 1.

Shown below is the mesh that was used to solve for the lid driven cavity in Fluent

As can be seen, the claim that FFD is faster than CFD processes has been verified. However the available open source implementation is not feasible to solve for high resolution grids because the large number of grid points makes the processes which aren't vectorized slow.

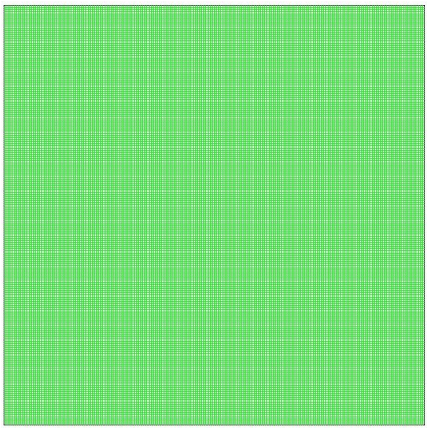


FIGURE 3.3: The 200 x 200 Mesh used to simulate the Lid Driven cavity in ANSYS Fluent

Execution time for Flow time of 10	
Method	CPU Time
ANSYS Fluent	410 seconds
New FFD implementation	137 seconds
Open Source implementation	$>$ 30 minutes

The reason as to why the FFD implementation used in the thesis is faster is explained in Appendix A where the advection step is vectorized. Other processes like calculating the divergence of the scalar field q are also performed using improved methods.

The picture 3.4 shows the fluid flow for the lid driven cavity for a grid size of 200×200 using a time step of $dt = 0.02$.

3.3 Limitations

Although the Fast Fluid Dynamics implementation developed is much faster than a traditional CFD approach, the accuracy is compromised for the execution time. The solution will be stable regardless of the time step taken but this doesn't ensure that the solution would be accurate. The general shape of the fluid flow can be obtained but this method still lacks in its ability to correctly predict the flow parameters at the steady state down to a very low tolerance.

Another unexplained facet of this method, which is also not explained in this report is how Fast Fluid Dynamics is mathematically consistent with the two governing equations, the Navier-Stokes and the continuity equations. More specifically how it is logical to iterate the governing FFD equation 1.7 in steps.

The paper [5] has not made it explicitly clear as to how a numerically consistent solution can be arrived upon by performing an iteration by taking only a single part of it (For example in the force addition step, the effect of all the other parameters are taken to be zero) and this is

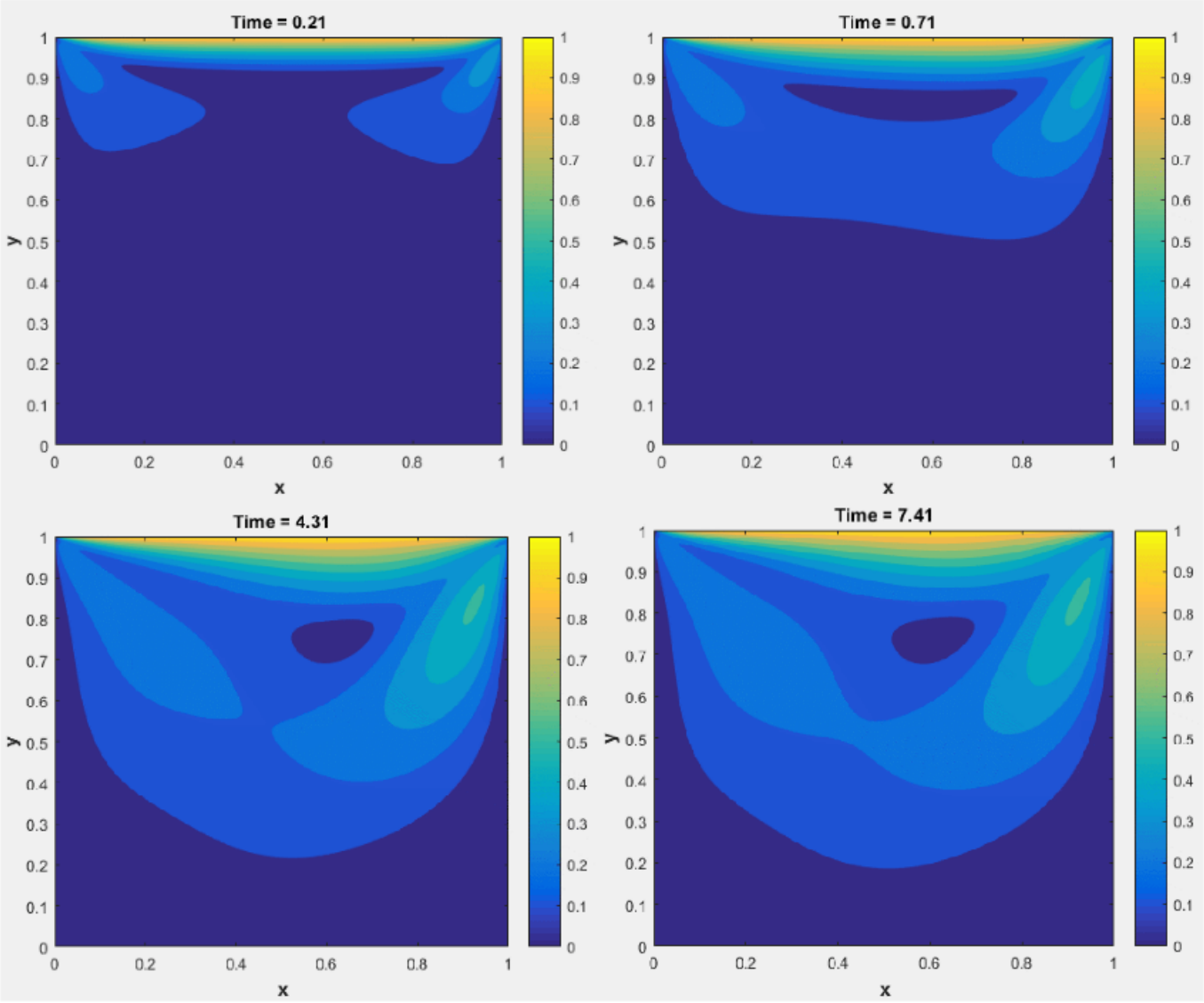


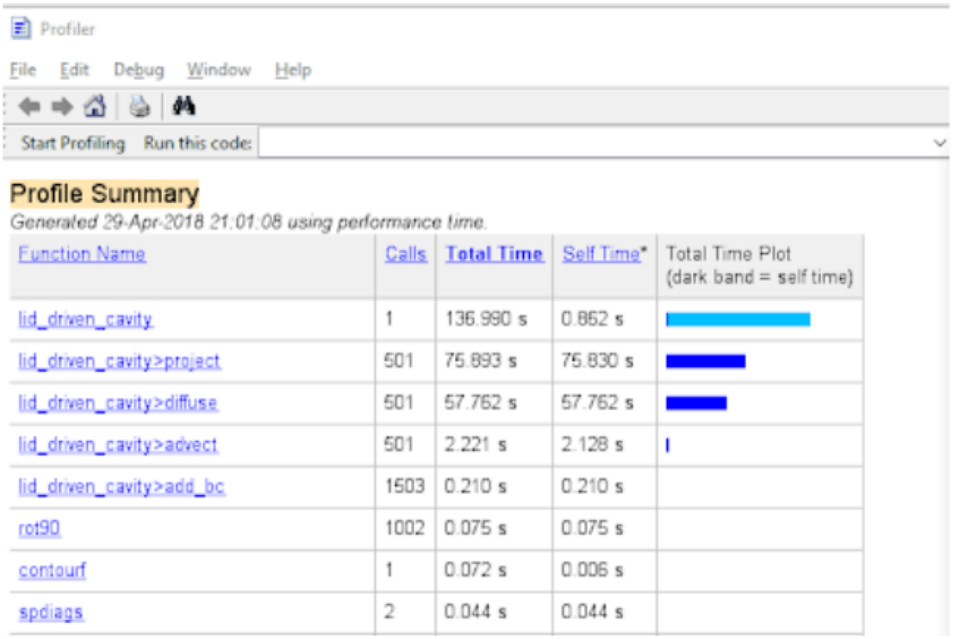
FIGURE 3.4: Fluid flow in a lid driven cavity

done for all the steps. There has been no other document which is readily available, is rigorous, and explains the logic behind the working of this algorithm partly due to the lack in available literature.

3.4 Future Work

There is a promising amount of work which can be done on the basis of this thesis. The most important result is that the code is indeed fast and is a relatively crude implementation. More tests can be performed by altering the boundary and initial conditions. This code can be developed to simulate the flow of smoke in a building, spread of contaminants with some more input.

The order of the steps involved seem to make a difference in the solution obtained. This interesting point has not been mentioned in the literature available, rather the steps are followed without a strong explanation as to why one should do so.


Profiler

File Edit Debug Window Help

Start Profiling Run this code:

Profile Summary

Generated 29-Apr-2018 21:01:08 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
lid_driven_cavity	1	136.990 s	0.862 s	
lid_driven_cavity>project	501	75.893 s	75.830 s	
lid_driven_cavity>diffuse	501	57.762 s	57.762 s	
lid_driven_cavity>advect	501	2.221 s	2.128 s	
lid_driven_cavity>add_bc	1503	0.210 s	0.210 s	
rot90	1002	0.075 s	0.075 s	
contourf	1	0.072 s	0.006 s	
spdiags	2	0.044 s	0.044 s	

FIGURE 3.5: Profile of execution time for each function.

As seen from 3.5 the code, although fast has its rate determining steps in the matrix multiplications, specifically during the projection and diffusion step require big matrices (of the order $(N \times N)$) which would be faster if implemented in GPUs as they are faster for array operations [2]. Moreover applying a Finite-Element scheme to this theory would make the matrix operations much easier which would in turn allow for larger domains of simulation.

3.5 Conclusion

The aim of the thesis which was to develop a code which implements Fast Fluid Dynamics and verifies the claim that FFD is indeed faster than CFD methods has been achieved. The code is completely vectorized and the improvements made on the available code is starkly visible.

However a strong mathematical reasoning behind evolving the system in steps neglecting the effects of the other parameters and the reason why the order is chosen (Force step, Advection step, Diffusion step, and Projection step) has not been found. There is scope for building up on the results obtained from the work done during this thesis as is detailed in section 3.4.

Appendix A

Vectorized Advection Algorithm

The benefits of vectorized operations are well known [7] and is very useful, especially when dealing with large arrays as we are while performing the advection step.

Refer to the figure 2.2, the given initial parameters are the location of all the grid points (x_1, y_1) and the velocity (u, v)at all these grid points from the previous time step.

The x_1 and y_1 for all the grid points are stored in a $N \times N$ array, they are labelled as x_1tile and y_1tile. The word tile is chosen to represent all the information regarding the particular parameter.

Similarly two more $N \times N$ arrays labelled $utile$ and $vtile$ store the value of u and v at all the resolution points.

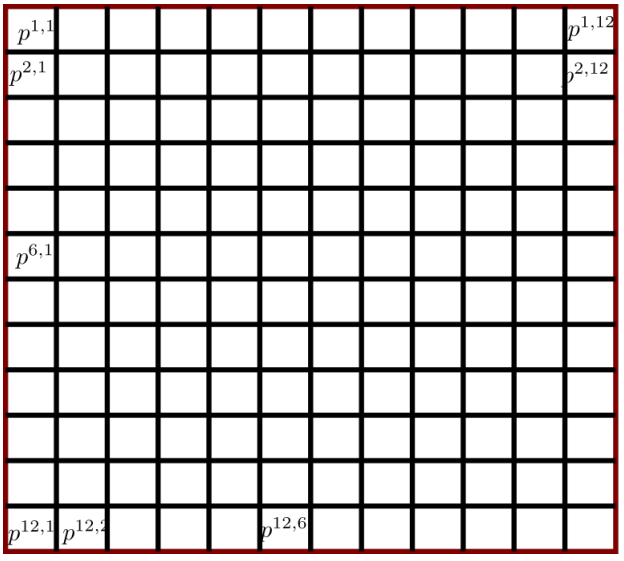


FIGURE A.1: A tile which consists of the value of parameter p at all the resolution points.

As seen in figure A.1, a tile in this context means an array which stores all the values of the parameter in question in a single array. Now, to determine the (x_1, y_1) from the tiles of x_0, y_0, u and v. The x_1 tile is obtained by subtracting u tile multiplied by Δt from the x_0 tile. In case the value of x_1 goes beyond the value at the boundary (For the cavity, the boundaries are at $x = 0$ and $x = 1$), the boundary values are taken as the streamlines cannot come from beyond the domain of the simulation.

```
y0_linspace  = linspace(0, 1, n);
y0_tile_temp = repmat(y0_linspace, n, 1);
y0_tile      = y0_tile_temp(2:end-1, 2:end-1);
x0_tile      = transpose(y0_tile);


zero_tile = zeros(size(u_tile));
one_tile = ones(size(u_tile));

% back tracing with the boundary conditions in place
x1_tile = min(max(x0_tile - u_tile * dt, zero_tile), one_tile);
y1_tile = min(max(y0_tile - v_tile * dt, zero_tile), one_tile);
```

Above is the code segment which highlights the application of this logic using MATLAB

Since x_1 and y_1 for all points are known, the indices of the right, bottom, top, and the left elements are stored in different tiles.

Using these indices, another array operation which takes in the tiles consisting of the indices as well as the initial *u tile* and *v tile*, The velocity of the neighbouring points are obtained.

The velocities of the neighbouring elements are also stored in tiles. $u_{bottom\ left}$, $u_{bottom\ right}$, $u_{top\ left}$, $u_{top\ right}$ and corresponding neighbouring v velocities are also stored. Shown below is the implementation of this logic

```
u_top_left     = reshape(flat_u(i_left_flat  + j_top_flat),...
                                 n_minus_2, n_minus_2);
u_top_right    = reshape(flat_u(i_right_flat + j_top_flat),...
                                 n_minus_2, n_minus_2);
u_bottom_right = reshape(flat_u(i_right_flat + j_bottom_flat),...
                                 n_minus_2, n_minus_2);
u_bottom_left  = reshape(flat_u(i_left_flat  + j_bottom_flat),...
                                 n_minus_2, n_minus_2);
```

The next step would be to perform the bi-linear interpolation to obtain the velocity at a time-step Δt before. This is done by following equations 2.1 through 2.6. The resultant *u new tile* is returned as the velocity after the advection step.

```
u_x_interpolation_lower = ((x_right_tile - x1_tile).* u_bottom_left ...
                         + (x1_tile - x_left_tile).*u_bottom_right)/delta_x;

u_x_interpolation_upper = ((x_right_tile - x1_tile).* u_top_left ...
                         + (x1_tile - x_left_tile).*u_top_right)/delta_x;

u_new_tile = ((y_top_tile - y1_tile).* u_x_interpolation_lower ...
           + (y1_tile - y_bottom_tile).* u_x_interpolation_upper)/delta_x;
```

Thus the function to perform the advection step would not traverse the entire grid point by point but instead does it in a single go.

The benefits of this can be clearly seen in the execution time comparison table where the vectorized FFD implementation is much faster than an unvectorized implementation.

Bibliography

[1] *Implementation of Lid Driven Cavity using Fast Fluid Dynamics*. URL: https://github.com/Balavarun5/fast_fluid_dynamics (visited on 05/03/2018).

[2] J. Sugerman K. Fatahalian and P. Hanrahan. "Understanding the Efficiency of GPU Algorithms for Matrix-Matrix Multiplication". In: *Association of Computing Machinery* (2004). URL: https://graphics.stanford.edu/papers/gpumatrixmult/gpumatrixmult.pdf.

[3] *Lid Driven Cavity - FFD implementation*. URL: https://github.com/lukasbystricky/FastFluidDynamics (visited on 05/03/2018).

[4] *mldivide*. URL: https://in.mathworks.com/help/matlab/ref/mldivide.html (visited on 05/03/2018).

[5] Jos Stam. "Stable Fluids". In: (1999). URL: http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/ns.pdf.

[6] C. T. SHIN U. GHIA K. N. GHIA. "High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method". In: *Journal of Computational Physics* (1982). URL: https://pdfs.semanticscholar.org/211b/45b6a06336a72ca064a6e59b14ebc520211c.pdf.

[7] Wikipedia contributors. *Array programming — Wikipedia, The Free Encyclopedia*. [Online; accessed 4-May-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Array_programming&oldid=838811287.

[8] Wangda Zuo and Q Chen. "Real-time or faster-than-real-time simulation of airflow in buildings". In: 19 (Mar. 2009), pp. 33–44.