A GPU implementation of massively parallel direction splitting for the incompressible Navier–Stokes equations

CMPT 898 Final Report

Adam L. Preuss

Numerical Simulation Laboratory

Department of Computer Science

University of Saskatchewan

`adam.preuss@usask.ca`

April 29, 2013

Abstract

Guermond and Minev proposed a directional splitting algorithm to solve the incompressible Stokes equations. Their algorithm applies the alternating direction implicit method to the viscosity term. The pressure update uses a direction splitting method in order to enforce the incompressibility constraint, as opposed to commonly used projection methods that require the solution of a Poisson equation. The significant part of the algorithm is that all of its implicit equations can be transformed into many independent tridiagonal systems; therefore, each step of the overall algorithm can be solved in $\mathcal{O}(n)$ time, where there are n unknowns. The goal of this course project is to develop a massively parallel GPU fluid solver for the incompressible Navier–Stokes equations using OpenCL and the directional splitting algorithm. GPUs are well suited to massively parallel numerical problems because they have hundreds of small cores specialized for numerical computation. Therefore, GPUs are applicable to solving many tridiagonal systems simultaneously. The non-linear term of the momentum equation is solved using an explicit, semi-Lagrangian method. Numerical tests verify solutions to the Navier–Stokes equation and show that simulations run in linear time with respect to the number of discretized points in the domain. An OpenGL renderer was developed to visualize simulations in two dimensions.

1 Introduction

The Navier–Stokes equations describe the velocity field inside a fluid, typically due to advection, viscosity, body forces and pressure effects. The equations are applicable to two and three dimensional problems and are used in a wide variety of scientific and engineering disciplines. Compressibility of fluids is often negligible, thus an incompressibility constraint is often included as part of the equations. To date, most incompressible Navier–Stokes solution algorithms use projection techniques based on Chorin's method to solve the pressure update [2]. An overview of their many variants is provided in [6]. These algorithms require the solution of a Poisson equation at each time step.

Guermond and Minev proposed a new algorithm to solve the incompressible Stokes equations, using direction splitting techniques [4]. The equations can be split into viscosity, body force and pressure terms (advection is part of the Navier–Stokes equations). The proposed algorithm uses the alternating direction implicit (ADI) method to solve the viscosity term and uses forward Euler to apply body forces [8]. The pressure update is solved using direction splitting which a relaxes the incompressibility condition such that its application is transformed into many one dimensional systems of equations. These equations can be formulated as tridiagonal systems; therefore, each step of the overall algorithm can be solved in $\mathcal{O}(n)$, where there are n unknowns. The direction splitting approach better lends itself to parallel algorithms because the one dimensional systems are independent, in contrast to projection methods that require the solution of a single higher-dimensional Poisson equation. A massively parallel implementation of the direction splitting algorithm is presented in [5], using MPI. It showed excellent preliminary performance, solving 2×10^9 unknowns on a distributed-memory cluster of 1024 processors.

The objective of this course project is to implement a GPU solver for the incompressible Navier-Stokes equations based on the algorithm proposed by Guermond and Minev. GPUs are comprised of hundreds of small cores specialized for numerical computation. They are therefore well suited to the solution of many independent one dimensional differential equation systems. The ADI method is applied to the viscosity term and a semi-Lagrangian method is applied to the advection term of the Navier–Stokes equations [1]. Body forces are solved using forward Euler. Numerical experiments show that the elapsed runtime of simulations to scale linearly, as theoretically predicted. Additionally, an OpenGL renderer was written to visualize the simulations for both presentation and visual verification.

Section 2 presents an overview of fluid simulation mathematical theory, including the traditional projection method of solving the Navier–Stokes equations. Section 3 describes the implementation of the new method, comparing and contrasting it to the projection method. Section 4 discusses simulation results and the OpenGL renderer. Section 5 presents a progress overview and theory for future work that has not yet been implemented.

2 Theoretical Background

This section introduces the incompressible Navier–Stokes equations and the traditional approach to solving them. Understanding the traditional approach clarifies what makes the new algorithm so much better suited to a massively parallel solver.

Mathematical notation will remain consistent through this report and is defined as follows. All scalers are italicized lower-case letters, for example, the pressure p. Vectors are bold and lower-case letters. Matrices are bold uppercase letters. The eigenvalue equation $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ is an example that demonstrates all three cases. The first component of a vector will be its matching lower case letter; all other components will follow in lower case letters (for example, the two dimensional velocity vector $\mathbf{u} = (u, v)^T$.) Superscripts refer to temporal discretization and subscripts refer to spatial discretization points.

2.1 Navier–Stokes equations

The incompressible Navier–Stokes equations govern velocity inside a fluid. They are a system of coupled nonlinear partial differential equations that are commonly written as:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) - \mu\nabla^2\mathbf{u} + \nabla p = \mathbf{f} \qquad \text{in } \Omega\times[0,T] \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \qquad \text{in } \Omega\times[0,T] \tag{2}$$

$$\mathbf{u}|_{\partial\Omega} = 0, \quad \mathbf{u}|_{t=0} = \mathbf{u}_0 \qquad \text{in } \Omega$$

where t is time, $\mathbf{u} = \mathbf{u}(t)$ is the velocity, μ is the viscosity, $p = p(t)$ is the pressure, $\mathbf{f} = \mathbf{f}(t, \mathbf{x})$ is a smooth body force, T is the final time, and Ω is the domain. Velocity, pressure and body forces are all functions of position; viscosity and density are constants. The nonlinear term $\mathbf{u}\cdot\nabla\mathbf{u}$ is known as the advection term. For simplicity, Dirichlet boundary conditions are assumed where the normal velocity component is zero and the tangential component is prescribed.

Equation 1 is known as the *momentum equation*, which is most commonly solved using a method-of-lines discretization method, starting with a specified initial condition and time. The momentum equation is comprised of four different problems that can be separated into advection, viscosity, body forces, and pressure update.

All fluids exhibit changes in density in response to compression (explaining, for instance, the ability to hear underwater due to the propagation of sound waves), however, the compressibility of many simulated liquids is negligible. Typically, simulated fluids are treated as incompressible by adding an *incompressibility constraint*, shown in equation 2. The constraint forces velocity to be divergence-free so that the fluid will remain at a constant density throughout the entire domain.

2.2 Operator splitting

Equation 1 is a linear combination of terms that can easily be split into multiple pieces, allowing each one to be integrated separately at every timestep. Using first-order forward Euler, a solution timestep might look something like:

$$\begin{aligned}
\mathbf{u}_1^{n+1} &= \mathbf{u}^n - \Delta t \mathbf{u} \nabla \mathbf{u} && \text{(Advection)} \\
\mathbf{u}_2^{n+1} &= \mathbf{u}_1^{n+1} + \Delta t \mu \nabla^2 \mathbf{u} && \text{(Viscosity)} \\
\mathbf{u}_3^{n+1} &= \mathbf{u}_2^{n+1} + \Delta t \mathbf{f} && \text{(Body force)} \\
\mathbf{u}^{n+1} &= \mathbf{u}_3^{n+1} + \Delta t \nabla p && \text{(Pressure)}
\end{aligned}$$

where Δt is the stepsize, and $\mathbf{u}_1^{n+1}$, $\mathbf{u}_2^{n+1}$, and $\mathbf{u}_3^{n+1}$ are intermediate steps. In general, any higher-order numerical integration method can be used for each split component, but it is not often of benefit because the above approach is a first order split. Section 3.4 discusses a second order approach to solving the Navier–Stokes equations with direction splitting.

2.3 Grids

The domain is discretized on a uniform Eulerian grid storing velocity and pressure. The grid is staggered to simplify finite volume calculations for spatial derivatives and to avoid skipping values when computing finite differences, as discussed in section 2.4. It is aptly named a marker-and-cell (MAC) grid [7]. All values are stored at cell centres with the exception of fluid velocities, which are stored at cell borders. Each velocity component $\mathbf{u} = (u, v)^T$ is staggered separately. Figure 1 shows two and three dimensional MAC grids, storing pressure and velocity values.

At many phases of the simulation, it is necessary to sample velocity and pressure at locations that are not exactly lined up with their stored grid points. In such cases, velocity and pressure can be calculated using bilinear interpolation, described in section 2.4. Two dimensional finite volume

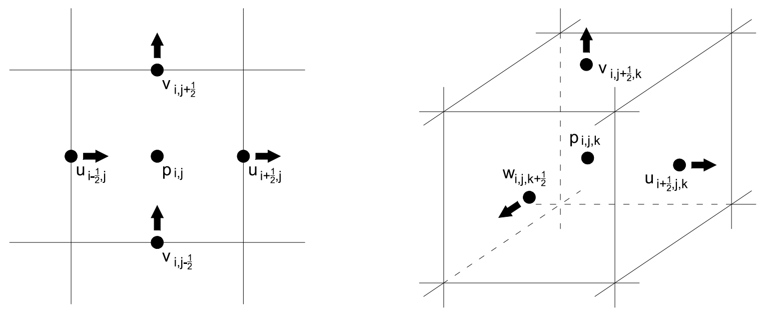

Figure 1: MAC grids for two and three dimentions. Notice how the pressure points are located at the cell centres and the velocity points are located on the cell borders. In this figure, the back three velocity points are not shown on the three-dimensional cell, to avoid confusion.

calculations, required in later sections, are defined as follows:

$$
\begin{aligned}
\nabla p_{i+\frac{1}{2},j} &= \frac{p_{i+1,j} - p_{i,j}}{\Delta x} \qquad & \nabla p_{i,j+\frac{1}{2}} &= \frac{p_{i,j+1} - p_{i,j}}{\Delta y} \\
\nabla u_{i,j} &= \frac{u_{i+\frac{1}{2},j} + u_{i-\frac{1}{2},j}}{\Delta x} \qquad & \nabla v_{i,j} &= \frac{v_{i,j+\frac{1}{2}} + v_{i,j-\frac{1}{2}}}{\Delta y} \\
\nabla^2 p_{i,j} &= \frac{p_{i-1,j} - 2p_{i,j} + p_{i+1,j}}{\Delta x} + \frac{p_{i,j-1} - 2p_{i,j} + p_{i,j+1}}{\Delta y}
\end{aligned}
$$

The three dimensional finite volume calculations are written as:

$$
\begin{aligned}
\nabla p_{i+\frac{1}{2},j,k} &= \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta x} \qquad & \nabla p_{i,j+\frac{1}{2},k} &= \frac{p_{i+1,j,k} - p_{i,j,k}}{\Delta y} \qquad & \nabla p_{i,j,k+\frac{1}{2}} &= \frac{p_{i,j,k+1} - p_{i,j,k}}{\Delta z} \\
\nabla u_{i,j,k} &= \frac{u_{i+\frac{1}{2},j,k} + u_{i-\frac{1}{2},j,k}}{\Delta x} \qquad & \nabla v_{i,j,k} &= \frac{v_{i+\frac{1}{2},j,k} + v_{i,j-\frac{1}{2},k}}{\Delta y} \qquad & \nabla w_{i,j,k} &= \frac{w_{i,j,k+\frac{1}{2}} + w_{i,j,k-\frac{1}{2}}}{\Delta z} \\
\nabla^2 p_{i,j} &= \frac{p_{i-1,j,k} - 2p_{i,j,k} + p_{i+1,j,k}}{\Delta x} + \frac{p_{i,j-1,k} - 2p_{i,j,k} + p_{i,j+1,k}}{\Delta y} + \frac{p_{i,j,k-1} - 2p_{i,j,k} + p_{i,j,k+1}}{\Delta z}
\end{aligned}
$$

2.4 Advection

Advection is the transport of a quantity in a velocity vector field. The advection equation for an arbitrary quantity q is written as follows:

$$\frac{\partial q}{\partial t} = (\mathbf{u} \cdot \nabla)q$$

The quantity q is advected in the divergence-free velocity field. This equation can be applied to the nonlinear term in the Navier–Stokes equation. When solving the advection equation, note that quantities should only be advected in a divergence-free velocity field. Such an approach applies when advecting velocity as well. If one is integrating velocity with a higher order method, one must take care not to update the velocity field until all stages have finished, otherwise, it will not be divergence-free for all stages in the method.

Naively, the advection step can be calculated using central differences. In two dimensions, this is written as:

$$\frac{\mathbf{q}^{n+1} - \mathbf{q}^{n}}{\Delta t} = u\frac{q^n_{i+1,j} - q^n_{i-1,j}}{\Delta x} + v\frac{q^n_{i,j+1} - q^n_{i,j-1}}{\Delta y}$$

This method can produce poor results with turbulent flow. The method does not take into account the value of the point at which the derivative is being calculated, $(q^n_{i,j})$. A superior approach is to use a method that is first order but does not experience the same inaccuracies due to skipping values as centred differences.

Advection is calculated using a semi-Lagrangian approach that is guaranteed to be stable [10]. For every point on the grid, $\mathbf{x}_p$, trace back the trajectory of an imaginary particle in the velocity field. The quantity at this traced position becomes the new value at $\mathbf{x}_p$. It is unlikely that the traced position actually falls on a stored grid point, thus it is calculated using bi- or trilinear interpolation. Bilinear interpolation for quantity q_p at point $\mathbf{x}_p$ where $x_p \in [x_i, x_{i+1}]$ and $y_p \in [y_j, y_j + 1]$ is calculated as follows:

$$\alpha = \frac{x_p - x_i}{\Delta x} \qquad \beta = \frac{y_p - y_j}{\Delta y}$$
$$q_p = (1-\beta)\left[(1-\alpha)q_{i,j} + \alpha q_{i+1,j}\right] + \beta\left[(1-\alpha)q_{i,j+1} + \alpha q_{i+1,j+1}\right]$$

Trilinear interpolation is calculated as:

$$\alpha = \frac{x_p - x_i}{\Delta x} \qquad \beta = \frac{y_p - y_j}{\Delta y} \qquad \gamma = \frac{z_p - z_k}{\Delta z}$$
$$q_p = (1-\gamma)\left\{(1-\beta)\left[(1-\alpha)q_{i,j,k+1} + \alpha q_{i+1,j,k+1}\right] + \beta\left[(1-\alpha)q_{i,j+1,k+1} + \alpha q_{i+1,j+1,k+1}\right]\right\}$$
$$+ \gamma\left\{(1-\beta)\left[(1-\alpha)q_{i,j,k} + \alpha q_{i+1,j,k}\right] + \beta\left[(1-\alpha)q_{i,j+1,k} + \alpha q_{i+1,j+1,k}\right]\right\}$$

A graphical representation of bilinear interpolation is shown in figure 2 where q is stored at cell centres. The complete advection algorithm in two or three dimensions is as follows for each coordinate:

- Let $\mathbf{x}_p$ be the point at i, j or i, j, k.
- Let $\mathbf{x}'_p = \mathbf{x}_p - \Delta t\mathbf{u}_{i,j}$ be the traced back point. In practise, a higher-order integration method

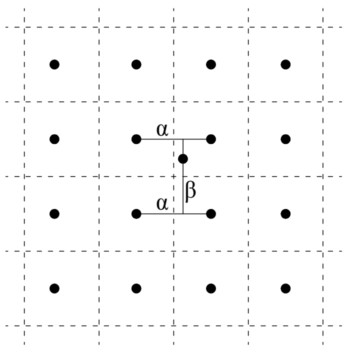

Figure 2: Example of bilinear interpolation.

is often used. The solver in this paper uses a second-order Runge–Kutta method as follows:

$$\mathbf{x}'_p = \mathbf{x}_p - \Delta t \mathbf{u}(\mathbf{x}_p - \frac{\Delta t}{2}\mathbf{u}(\mathbf{x}_p))$$

where $\mathbf{u}(\mathbf{x})$ is calculated using bi- or trilinear interpolation as explained above.

- Set $q_{i,j}$ or $q_{i,j,k}$ to be the interpolated value of q at $\mathbf{x}'_p$.

2.5 Body forces

Body forces are conceptually the simplest part of the fluid simulation because they are applied to the entire liquid without any dependence on the liquid velocity or pressure. Generally, body forces are constant, for example, gravity. However, body forces can be functions of time and position. They cannot be functions of velocity because the body force update would become a nonlinear ODE. The body force update can be written as:

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \mathbf{f}(t^n)$$

Forward Euler is an acceptable method for this update. If $\mathbf{f}$ is constant, then higher order methods will be of no benefit. A higher order Runge–Kutta method can be used if $\mathbf{f}$ is not a constant function.

2.6 Viscosity

Fluid viscosity is a measurement of a fluid's resistivity to deformation. The equation for the velocity term is written as:

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

A first order approach to the viscosity term can be written:

$$\begin{aligned}\mathbf{u}^{n+1}_{i,j,k} = \mathbf{u}^{n}_{i,j,k} &+ \Delta t \left(\frac{\mathbf{u}_{i-1,j,k} - 2\mathbf{u}_{i,j,k} + \mathbf{u}_{i+1,j,k}}{\Delta x} \right) \\ &+ \Delta t \left(\frac{\mathbf{u}_{i,j-1,k} - 2\mathbf{u}_{i,j,k} + \mathbf{u}_{i,j+1,k}}{\Delta y} \right) \\ &+ \Delta t \left(\frac{\mathbf{u}_{i,j,k-1} - 2\mathbf{u}_{i,j,k} + \mathbf{u}_{i,j,k+1}}{\Delta z} \right)\end{aligned}$$

in three dimensions. The two dimensional equivalent is the same, neglecting the last term.

2.7 Projection method

The pressure update is probably the most complicated step. After the first three velocity updates (advection, viscosity and body forces), the velocity field might no longer be a divergence-free field. The projection method is what most algorithms use to enforce incompressibility, an idea first introduced by Chorin[2]. Conceptually, it *projects* the velocity field back onto a divergence-free space. Plugging the pressure update into the incompressibility condition $\nabla \cdot \mathbf{u}^{n+1} = 0$ yields a new equation that can be solved for the pressure:

$$\nabla^2 p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u} \tag{3}$$

Solving this equation can be broken down into steps. The following analysis pertains to two dimensions, but is easily extended to three. First, one must compute the right-hand-side by calculating divergence on the MAC grid for each grid point. This is done by using a finite volume method, easy to calculate on the staggered MAC grid. Velocity values are prescribed at the edge of cells so the divergence may be computed as:

$$\begin{aligned}\nabla \cdot \mathbf{u}_{i,j} &= \frac{\partial u_{i,j}}{\partial x} + \frac{\partial v_{i,j}}{\partial y} \\ &= \frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{\Delta x} + \frac{v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}}}{\Delta y}\end{aligned}$$

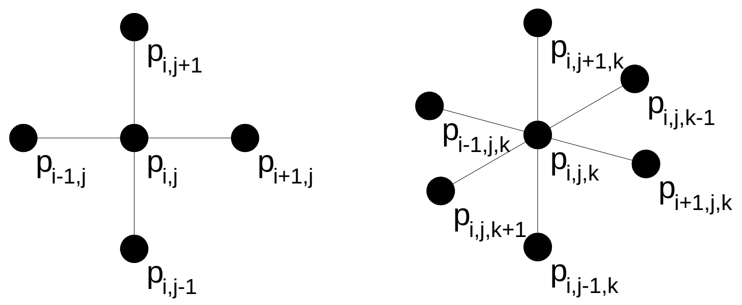

Figure 3: Stencils of the Laplacian operator in two and three dimensions.

The finite difference stencil of the Laplacian is shown in figure 3, giving rise to a banded system. The system is sparse, a tridiagonal system with a single band on either side. The system will be symmetric, thus three vectors must be stored: one for the diagonal, one for the subdiagonal, and one for the band.

$$\nabla^2 p_{i,j} = \frac{p_{i-1,j} - 2p_{i,j} + p_{i+1,j}}{\Delta x^2} + \frac{p_{i,j-1} - 2p_{i,j} + p_{i,j+1}}{\Delta y^2}$$

2.8 Boundary conditions

Implementing boundary conditions is probably the most difficult part of fluid simulation. The normal component of velocity to the solid boundary must be zero to prevent inflow and outflow. The tangential components of the domain are specified as part of simulation parameters.

In two dimensions, boundaries are prescribed for tangential velocities on the left, right, bottom and top. In three dimensions, there are two tangential directions for each face, thus twelve boundary numbers must be prescribed.

Due to grid staggering, the tangential velocities on the boundary do not lie directly on grid points. Instead, ghost points are stored outside the boundary and are calculated via averages. For example, the bottom boundary ghost points of a two dimensional domain are calculated as follows:

$$\frac{u_{i,-\frac{1}{2}} + u_{i,\frac{1}{2}}}{\Delta y} = u_{\text{bottom}}$$

$$\implies u_{i,-\frac{1}{2}} = 2u_{\text{bottom}} - u_{i,-\frac{1}{2}}$$

3 Methods

This section introduces the direction splitting method and its implementation, highlighting how it deviates from the traditional projection approach to solving the incompressible Navier–Stokes equations.

3.1 Velocity update

The algorithm proposed by Guermond and Minev takes an improved approach (in comparison to forward Euler) to the viscosity and body force updates by applying a direction splitting technique proposed by Douglas [8]. It is a variant of Crank-Nicolson time stepping. It accomplishes the same goal as the first three steps described in section 2.2.

The velocity update algorithm is presented here for the sake of completeness. While it is easy to parallelize because it only contains one-dimensional systems of ordinary differential equations, it is unrelated to the projection method alternative that will be described later on. The velocity update algorithm is as follows:

$$\begin{aligned}
&\frac{\boldsymbol{\xi}^{*}-\mathbf{u}^{n}}{\Delta t}-\mu\nabla^{2}\mathbf{u}^{n}+\mathbf{u}^{n}\cdot\nabla\mathbf{u}^{n}+\nabla p^{*}=\mathbf{f}^{n+\frac{1}{2}}\\
&\frac{\boldsymbol{\eta}^{n+1}-\boldsymbol{\xi}^{*}}{\Delta t}-\frac{\mu}{2}\frac{\partial}{\partial x^{2}}\left(\boldsymbol{\eta}^{*}-\mathbf{u}^{n}\right)=0\\
&\frac{\boldsymbol{\zeta}^{n+1}-\boldsymbol{\eta}^{*}}{\Delta t}-\frac{\mu}{2}\frac{\partial}{\partial y^{2}}\left(\boldsymbol{\zeta}^{*}-\mathbf{u}^{n}\right)=0\\
&\frac{\mathbf{u}^{n+1}-\boldsymbol{\zeta}^{*}}{\Delta t}-\frac{\mu}{2}\frac{\partial}{\partial z^{2}}\left(\mathbf{n}^{n+1}-\mathbf{u}^{n}\right)=0
\end{aligned}$$

where $\boldsymbol{\xi}^{n+1}$, $\boldsymbol{\eta}^{n+1}$, and $\boldsymbol{\zeta}^{n+1}$ are intermediate values, and p^{*} is an intermediate pressure value taken from the previous timestep. For the two-dimensional case, omit the last equation, and let $\mathbf{u}^{n+1}=\boldsymbol{\zeta}^{n+1}$.

The nonlinear term is calculated using the semi-Lagrangian method, described in section 2.4. Note that it is also an explicit method. It does not require the solution of a system of equations.

3.2 Pressure update

The pressure update uses a direction splitting technique as well. Consider a generalized version of the pressure-solve given by equation 3:

$$\nabla \mathbf{u} + \Delta t A p = 0 \tag{4}$$

where A is an operator. Note that the projection step from the above method can be recovered by using $A = -\nabla^2$. Guermond and Minev propose an alternative that is shown to have the same convergence properties as the projection method, letting $A = (1 - \frac{\partial}{\partial x^2})(1 - \frac{\partial}{\partial y^2})(1 - \frac{\partial}{\partial z^2})$. The problem has now been transformed into three one-dimensional partial differential equations:

$$\left(1 - \frac{\partial}{\partial x^2}\right)\psi = -\frac{1}{\Delta t}\nabla \mathbf{u} \tag{5}$$

$$\left(1 - \frac{\partial}{\partial y^2}\right)\varphi = \psi \tag{6}$$

$$\left(1 - \frac{\partial}{\partial z^2}\right)p = \varphi \tag{7}$$

where ψ and φ are intermediate values to store the solution for each split direction. For the two-dimensional case, let $A = (1 - \frac{\partial}{\partial x^2})(1 - \frac{\partial}{\partial y^2})$, omit equation 7, and let $p = \varphi$.

There are higher order variants of the method that change it from standard incremental form to rotational incremental form by including additional terms to the pressure update. This is presented in section 3.4.

3.3 Tridiagonal solver

The one dimensional systems that arise from direction splitting are tridiagonal systems. The system is symmetric because finite differences are centred. The sub-diagonal values b are all identical because the same finite difference stencil is applied at all locations on the domain. Almost all of the diagonal elements a are identical for the same reason, with the exception of boundary conditions, which are a special case: a_α and a_β. The system and its solution are written as follows, using the

Thomas algorithm:

$$\begin{bmatrix} a_\alpha & b & & & 0 \\ b & a & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & a & b \\ 0 & & & b & a_\beta \end{bmatrix} \qquad c'_i = \begin{cases} \dfrac{b}{a_\alpha} & i = 0 \\ \dfrac{b}{a - c'_{i-1}b} & i = 1, 2, \ldots, n-2 \end{cases}$$

$$d'_i = \begin{cases} \dfrac{d_i}{a_\alpha} & i = 0 \\ \dfrac{d_i - d'_{i-1}b}{a - c'_{i-1}b} & i = 1, 2, \ldots, n-2 \\ \dfrac{d_i - d'_{i-1}b}{a_\beta - c'_{i-1}b} & i = n-1 \end{cases} \qquad x_i = \begin{cases} d'_i & i = n-1 \\ d'_i - c'_i x_{i+1} & i = n-2, \ldots, 0 \end{cases}$$

3.4 Higher order scheme

The proposed method can be extended to a second order scheme, known as a rotational form. The equations are written as follows:

$$\begin{aligned}
p^{*,n+\frac{1}{2}} &= p^{n-\frac{1}{2}} + \psi^{n-\frac{1}{2}} \\
\rho\left(\frac{\boldsymbol{\xi}^{n+1} - \mathbf{u}^n}{\Delta t} + \mathbf{u}^n \cdot \nabla \mathbf{u}^n\right) &= \mu \nabla^2 \mathbf{u}^n - \nabla p^{*,n-\frac{1}{2}} + \mathbf{f}(t^{n+\frac{1}{2}}) \\
\rho\left(\frac{\boldsymbol{\eta}^{n+1} - \boldsymbol{\xi}^{n+1}}{\Delta t}\right) &= \frac{\mu}{2}\frac{\partial^2}{\partial x^2}\left(\boldsymbol{\eta}^{n+1} - \mathbf{u}^n\right) \\
\rho\left(\frac{\boldsymbol{\zeta}^{n+1} - \boldsymbol{\eta}^{n+1}}{\Delta t}\right) &= \frac{\mu}{2}\frac{\partial^2}{\partial y^2}\left(\boldsymbol{\eta}^{n+1} - \mathbf{u}^n\right) \\
\rho\left(\frac{\mathbf{u}^{n+1} - \boldsymbol{\zeta}^{n+1}}{\Delta t}\right) &= \frac{\mu}{2}\frac{\partial^2}{\partial z^2}\left(\boldsymbol{\zeta}^{n+1} - \mathbf{u}^n\right) \\
\left(1 - \frac{\partial^2}{\partial x^2}\right)\left(1 - \frac{\partial^2}{\partial y^2}\right)\left(1 - \frac{\partial^2}{\partial z^2}\right)\psi^{n+\frac{1}{2}} &= -\frac{1}{\Delta t}\nabla \mathbf{u}^{n+1} \\
p^{n+\frac{1}{2}} &= p^{n-\frac{1}{2}} + \psi^{n+\frac{1}{2}} - \chi\mu\nabla\left(\frac{1}{2}(\mathbf{u}^{n+1} + \mathbf{u}^n)\right)
\end{aligned}$$

In this approach, χ is a parameter ranging from 0 to 1. Note that when $\chi = 0$, the previous method is recovered. All examples in this report choose $\chi = 0.5$.

3.5 Algorithm overview in two dimensions

This section shows the complete derivation of the implementation for the method.

1. Store old values for u and v in $\tilde{u}$ and $\tilde{v}$. Update the boundary conditions for them as well.

$$\begin{aligned}
\tilde{u}_{i,j} &= u_{i,j} & \tilde{v}_{i,j} &= v_{i,j} \\
\tilde{u}_{-1,j} &= u_{n_x,j} = 0 & \tilde{v}_{i,-1} &= v_{i,n_y} = 0 \\
\tilde{u}_{i,-1} &= 2u_S - u_{i,0} & \tilde{v}_{-1,j} &= 2v_W - v_{0,j} \\
\tilde{u}_{i,n_y} &= 2u_N - u_{i,n_y-1} & \tilde{v}_{n_x,j} &= 2v_E - v_{n_x-1,j}
\end{aligned}$$

2. Calculate temporary velocity values $u^*_{i,j}$ and $v^*_{i,j}$ via the momentum equation, using a pressure prediction.

$$\begin{aligned}
&\rho\left(\frac{\mathbf{u}^*_{i,j} - \tilde{\mathbf{u}}_{i,j}}{\Delta t}\right) = \mu\nabla^2\tilde{\mathbf{u}}_{i,j} - \nabla\left(p_{i,j} + \psi_{i,j}\right) \\
\implies &\mathbf{u}^*_{i,j} = \tilde{\mathbf{u}}_{i,j} + \frac{\Delta t\mu}{\rho}\nabla^2\tilde{\mathbf{u}}_{i,j} - \frac{\Delta t}{\rho}\nabla\left(p_{i,j} + \psi_{i,j}\right) \\
\implies &\left\{\begin{array}{rl}
u^*_{i,j} = & \tilde{u}_{i,j} + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{u}_{i-1,j} - 2\tilde{u}_{i,j} + \tilde{u}_{i+1,j}}{\Delta x^2} + \dfrac{\tilde{u}_{i,j-1} - 2\tilde{u}_{i,j} + \tilde{u}_{i,j+1}}{\Delta y^2}\right] \\
& -\dfrac{\Delta t}{\rho}\left[\dfrac{(p_{i+1,j} + \psi_{i+1,j}) - (p_{i,j} + \psi_{i,j})}{\Delta x}\right] \\
v^*_{i,j} = & \tilde{v}_{i,j} + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{v}_{i-1,j} - 2\tilde{v}_{i,j} + \tilde{v}_{i+1,j}}{\Delta x^2} + \dfrac{\tilde{v}_{i,j-1} - 2\tilde{v}_{i,j} + \tilde{v}_{i,j+1}}{\Delta y^2}\right] \\
& -\dfrac{\Delta t}{\rho}\left[\dfrac{(p_{i,j+1} + \psi_{i,j+1}) - (p_{i,j} + \psi_{i,j})}{\Delta y}\right]
\end{array}\right.
\end{aligned}$$

3. Solve the x-direction of the ADI split. Note that the boundary conditions of $u^{**}_{i,j}$ and $v^{**}_{i,j}$ are equivalent to those listed in step 1. Constant terms arising from the boundary conditions may need to be moved to the right-hand side of the equations.

$$\begin{aligned}
&\rho\left(\frac{\mathbf{u}^{**}_{i,j} - \mathbf{u}^*_{i,j}}{\Delta t}\right) = \frac{\mu}{2}\frac{\partial^2}{\partial x^2}\left(\mathbf{u}^{**}_{i,j} - \tilde{\mathbf{u}}_{i,j}\right) \\
\implies &\left(1 - \frac{\mu\Delta t}{2\rho}\frac{\partial^2}{\partial x^2}\right)\mathbf{u}^{**}_{i,j} = \mathbf{u}^*_{i,j} - \frac{\mu\Delta t}{2\rho}\frac{\partial^2}{\partial x^2}\tilde{\mathbf{u}}_{i,j} \\
\implies &\left\{\begin{array}{l}
u^{**}_{i,j} - \dfrac{\mu\Delta t}{2\rho\Delta x^2}\left(u^{**}_{i-1,j} - 2u^{**}_{i,j} + u^{**}_{i+1,j}\right) = u^*_{i,j} - \dfrac{\mu\Delta t}{2\rho\Delta x^2}\left(\tilde{u}_{i-1,j} - 2\tilde{u}_{i,j} + \tilde{u}_{i+1,j}\right) \\
v^{**}_{i,j} - \dfrac{\mu\Delta t}{2\rho\Delta x^2}\left(v^{**}_{i-1,j} - 2v^{**}_{i,j} + v^{**}_{i+1,j}\right) = v^*_{i,j} - \dfrac{\mu\Delta t}{2\rho\Delta x^2}\left(\tilde{v}_{i-1,j} - 2\tilde{v}_{i,j} + \tilde{v}_{i+1,j}\right)
\end{array}\right.
\end{aligned}$$

4. Solve the y-direction of the ADI split. The boundary conditions are still the same as listed

in step 1.

$$
\begin{aligned}
&\rho\left(\frac{\mathbf{u}_{i,j}-\mathbf{u}^{**}_{i,j}}{\Delta t}\right)=\frac{\mu}{2}\frac{\partial^2}{\partial y^2}\left(\mathbf{u}_{i,j}-\tilde{\mathbf{u}}_{i,j}\right)\\
\Longrightarrow &\left(1-\frac{\mu\Delta t}{2\rho}\frac{\partial^2}{\partial y^2}\right)\mathbf{u}_{i,j}=\mathbf{u}^{**}_{i,j}-\frac{\mu\Delta t}{2\rho}\frac{\partial^2}{\partial y^2}\tilde{\mathbf{u}}_{i,j}\\
\Longrightarrow &\begin{cases} u_{i,j}-\dfrac{\mu\Delta t}{2\rho\Delta y^2}\left(u_{i,j-1}-2u_{i,j}+u_{i,j+1}\right)=u^{**}_{i,j}-\dfrac{\mu\Delta t}{2\rho\Delta y^2}\left(\tilde{u}_{i,j-1}-2\tilde{u}_{i,j}+\tilde{u}_{i,j+1}\right)\\ v_{i,j}-\dfrac{\mu\Delta t}{2\rho\Delta y^2}\left(v_{i,j-1}-2v_{i,j}+v_{i,j+1}\right)=v^{**}_{i,j}-\dfrac{\mu\Delta t}{2\rho\Delta y^2}\left(\tilde{v}_{i,j-1}-2\tilde{v}_{i,j}+\tilde{v}_{i,j+1}\right)\end{cases}
\end{aligned}
$$

5. Update the boundary condition for $\mathbf{u}_{i,j}$ as in step 1. Then, solve the x-direction of the pressure constraint.

$$
\begin{aligned}
&\left(1-\frac{\partial^2}{\partial x^2}\right)\varphi_{i,j}=-\frac{1}{\Delta t}\nabla\mathbf{u}_{i,j}\\
\Longrightarrow\ &\varphi_{i,j}-\frac{\varphi_{i-1,j}-2\varphi_{i,j}+\varphi_{i+1,j}}{\Delta x^2}=-\frac{1}{\Delta t}\left(\frac{u_{i,j}-u_{i-1,j}}{\Delta x}+\frac{v_{i,j}-v_{i,j-1}}{\Delta y}\right)
\end{aligned}
$$

6. Solve the y-direction of the pressure constraint.

$$
\begin{aligned}
&\left(1-\frac{\partial^2}{\partial y^2}\right)\psi_{i,j}=\phi_{i,j}\\
\Longrightarrow\ &\psi_{i,j}-\frac{\psi_{i-1,j}-2\psi_{i,j}+\psi_{i+1,j}}{\Delta y^2}=\varphi_{i,j}
\end{aligned}
$$

7. Update the pressure based on the constraint.

$$
\begin{aligned}
p_{i,j}&=\tilde{p}_{i,j}+\psi_{i,j}-\chi\mu\nabla\left(\frac{1}{2}(\mathbf{u}_{i,j}+\tilde{\mathbf{u}}_{i,j})\right)\\
&=\tilde{p}_{i,j}+\psi_{i,j}-\frac{\chi\mu}{2}\left(\frac{(u_{i,j}+\tilde{u}_{i,j})-(u_{i-1,j}+\tilde{u}_{i-1,j})}{\Delta x}+\frac{(v_{i,j}+\tilde{v}_{i,j})-(v_{i,j-1}+\tilde{v}_{i,j-1})}{\Delta y}\right)
\end{aligned}
$$

3.6 Algorithm overview in three dimensions

This section shows the implementation steps for a three dimensional domain. Some of the intermediate steps are left out because they have been described in the previous section.

1. Store old values for u, v and w in $\tilde{u}$, $\tilde{v}$ and $\tilde{w}$. Update the boundary conditions for them as well. The subscript letters L, R, F, B, U and D stand for left, right, front, back, up and down

of the domain, respectively.

$$
\begin{array}{lll}
\tilde{u}_{i,j,k} = u_{i,j,k} & \tilde{v}_{i,j,k} = v_{i,j,k} & \tilde{w}_{i,j,k} = w_{i,j,k} \\
\tilde{u}_{-1,j,k} = u_{n_x,j,k} = 0 & \tilde{v}_{i,-1,k} = v_{i,n_y,k} = 0 & \tilde{w}_{i,j,-1} = w_{i,j,n_z} = 0 \\
\tilde{u}_{i,-1,k} = 2u_U - u_{i,0,k} & \tilde{v}_{-1,j,k} = 2v_L - v_{0,j,k} & \tilde{w}_{-1,j,k} = 2w_L - w_{0,j,k} \\
\tilde{u}_{i,n_y,k} = 2u_D - u_{i,n_y-1,k} & \tilde{v}_{n_x,j,k} = 2v_R - v_{n_x-1,j,k} & \tilde{w}_{n_x,j,k} = 2w_R - w_{n_x-1,j,k} \\
\tilde{u}_{i,j,-1} = 2u_F - u_{i,j,0} & \tilde{v}_{i,j,-1} = 2v_F - v_{i,j,0} & \tilde{w}_{i,-1,k} = 2w_U - w_{i,0,k} \\
\tilde{u}_{i,j,n_z} = 2u_B - u_{i,j,n_z-1} & \tilde{v}_{i,j,n_z} = 2v_B - v_{i,j,n_z-1} & \tilde{w}_{i,n_y,k} = 2w_D - w_{i,n_x-1,k}
\end{array}
$$

2. Calculate temporary velocity values $u^*_{i,j,k}$, $v^*_{i,j,k}$ and $w^*_{i,j,k}$ via the momentum equation, using a pressure prediction.

$$
\begin{cases}
u^*_{i,j,k} & = \tilde{u}_{i,j,k} - \dfrac{\Delta t}{\rho}\left[\dfrac{(p_{i+1,j,k}+\psi_{i+1,j,k})-(p_{i,j,k}+\psi_{i,j,k})}{\Delta x}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{u}_{i-1,j,k}-2\tilde{u}_{i,j,k}+\tilde{u}_{i+1,j,k}}{\Delta x^2}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{u}_{i,j-1,k}-2\tilde{u}_{i,j,k}+\tilde{u}_{i,j+1,k}}{\Delta y^2}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{u}_{i,j,k-1}-2\tilde{u}_{i,j,k}+\tilde{u}_{i,j,k+1}}{\Delta z^2}\right] \\
v^*_{i,j,k} & = \tilde{v}_{i,j,k} - \dfrac{\Delta t}{\rho}\left[\dfrac{(p_{i,j+1,k}+\psi_{i,j+1,k})-(p_{i,j,k}+\psi_{i,j,k})}{\Delta x}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{v}_{i-1,j,k}-2\tilde{v}_{i,j,k}+\tilde{v}_{i+1,j,k}}{\Delta x^2}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{v}_{i,j-1,k}-2\tilde{v}_{i,j,k}+\tilde{v}_{i,j+1,k}}{\Delta y^2}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{v}_{i,j,k-1}-2\tilde{v}_{i,j,k}+\tilde{v}_{i,j,k+1}}{\Delta z^2}\right] \\
w^*_{i,j,k} & = \tilde{w}_{i,j,k} - \dfrac{\Delta t}{\rho}\left[\dfrac{(p_{i,j,k+1}+\psi_{i,j,k+1})-(p_{i,j,k}+\psi_{i,j,k})}{\Delta x}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{w}_{i-1,j,k}-2\tilde{w}_{i,j,k}+\tilde{w}_{i+1,j,k}}{\Delta x^2}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{w}_{i,j-1,k}-2\tilde{w}_{i,j,k}+\tilde{w}_{i,j+1,k}}{\Delta y^2}\right] \\
 & + \dfrac{\Delta t\mu}{\rho}\left[\dfrac{\tilde{w}_{i,j,k-1}-2\tilde{w}_{i,j,k}+\tilde{w}_{i,j,k+1}}{\Delta z^2}\right]
\end{cases}
$$

3. Solve the x-direction of the ADI split. Note that the boundary conditions of $u^{**}_{i,j,k}$, $v^{**}_{i,j,k}$ and $w^{**}_{i,j,k}$ are equivalent to those listed in step 1. Constant terms arising from the boundary

conditions may need to be moved to the right-hand side of the equations.

$$\begin{cases} u^{**}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta x^2}\left(u^{**}_{i-1,j,k} - 2u^{**}_{i,j,k} + u^{**}_{i+1,j,k}\right) = u^{*}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta x^2}\left(\tilde{u}_{i-1,j,k} - 2\tilde{u}_{i,j,k} + \tilde{u}_{i+1,j,k}\right) \\ v^{**}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta x^2}\left(v^{**}_{i-1,j,k} - 2v^{**}_{i,j,k} + v^{**}_{i+1,j,k}\right) = v^{*}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta x^2}\left(\tilde{v}_{i-1,j,k} - 2\tilde{v}_{i,j,k} + \tilde{v}_{i+1,j,k}\right) \\ w^{**}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta x^2}\left(w^{**}_{i-1,j,k} - 2w^{**}_{i,j,k} + w^{**}_{i+1,j,k}\right) = w^{*}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta x^2}\left(\tilde{w}_{i-1,j,k} - 2\tilde{w}_{i,j,k} + \tilde{w}_{i+1,j,k}\right) \end{cases}$$

4. Solve the y-direction of the ADI split. The boundary conditions are still the same as listed in step 1.

$$\begin{cases} u^{***}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta y^2}\left(u^{***}_{i,j-1,k} - 2u^{***}_{i,j,k} + u^{***}_{i,j+1,k}\right) = u^{**}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta y^2}\left(\tilde{u}_{i,j-1,k} - 2\tilde{u}_{i,j,k} + \tilde{u}_{i,j+1,k}\right) \\ v^{***}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta y^2}\left(v^{***}_{i,j-1,k} - 2v^{***}_{i,j,k} + v^{***}_{i,j+1,k}\right) = v^{**}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta y^2}\left(\tilde{v}_{i,j-1,k} - 2\tilde{v}_{i,j,k} + \tilde{v}_{i,j+1,k}\right) \\ w^{***}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta y^2}\left(w^{***}_{i,j-1,k} - 2w^{***}_{i,j,k} + w^{***}_{i,j+1,k}\right) = w^{**}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta y^2}\left(\tilde{w}_{i,j-1,k} - 2\tilde{w}_{i,j,k} + \tilde{w}_{i,j+1,k}\right) \end{cases}$$

5. Solve the z-direction of the ADI split. The boundary conditions are still the same as listed in step 1.

$$\begin{cases} u_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta z^2}\left(u_{i,j,k-1} - 2u_{i,j,k} + u_{i,j,k+1}\right) = u^{***}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta z^2}\left(\tilde{u}_{i,j,k-1} - 2\tilde{u}_{i,j,k} + \tilde{u}_{i,j,k+1}\right) \\ v_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta z^2}\left(v_{i,j,k-1} - 2v_{i,j,k} + v_{i,j,k+1}\right) = v^{***}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta z^2}\left(\tilde{v}_{i,j,k-1} - 2\tilde{v}_{i,j,k} + \tilde{v}_{i,j,k+1}\right) \\ w_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta z^2}\left(w_{i,j,k-1} - 2w_{i,j,k} + w_{i,j,k+1}\right) = w^{***}_{i,j,k} - \frac{\mu\Delta t}{2\rho\Delta z^2}\left(\tilde{w}_{i,j,k-1} - 2\tilde{w}_{i,j,k} + \tilde{w}_{i,j,k+1}\right) \end{cases}$$

6. Update the boundary condition for $\mathbf{u}_{i,j,k}$ as in step 1. Then, solve the x-direction of the pressure constraint.

$$\varphi_{i,j,k} - \frac{\varphi_{i-1,j,k} - 2\varphi_{i,j,k} + \varphi_{i+1,j}}{\Delta x^2} = -\frac{1}{\Delta t}\left(\frac{u_{i,j,k} - u_{i-1,j,k}}{\Delta x} + \frac{v_{i,j,k} - v_{i,j-1,k}}{\Delta y} + \frac{w_{i,j,k} - w_{i,j,k-1}}{\Delta z}\right)$$

7. Solve the y-direction of the pressure constraint.

$$\phi_{i,j,k} - \frac{\phi_{i-1,j,k} - 2\phi_{i,j,k} + \phi_{i+1,j,k}}{\Delta y^2} = \varphi_{i,j,k}$$

8. Solve the z-direction of the pressure constraint.

$$\psi_{i,j,k} - \frac{\psi_{i-1,j,k} - 2\psi_{i,j,k} + \psi_{i+1,j,k}}{\Delta z^2} = \phi_{i,j,k}$$

9. Update the pressure based on the constraint.

$$\begin{aligned} p_{i,j,k} &= \tilde{p}_{i,j,k} + \psi_{i,j,k} \\ &- \frac{\chi\mu}{2}\left[\frac{(u_{i,j,k} + \tilde{u}_{i,j,k}) - (u_{i-1,j} + \tilde{u}_{i-1,j})}{\Delta x}\right] \\ &- \frac{\chi\mu}{2}\left[\frac{(v_{i,j,k} + \tilde{v}_{i,j,k}) - (v_{i,j-1} + \tilde{v}_{i,j-1})}{\Delta y}\right] \\ &- \frac{\chi\mu}{2}\left[\frac{(w_{i,j,k} + \tilde{w}_{i,j,k}) - (w_{i,j,w-1} + \tilde{w}_{i,j,w-1})}{\Delta z}\right] \end{aligned}$$

4 Results

This section describes the required OpenCL kernels and results of the numerical experiments. With the exception of the GPU kernels, the project is developed entirely in C++ because it is a fast and versatile language. C++ interfaces directly with memory and low-level libraries, while at the same time offering many complex data structures such as templates and polymorphic classes. The GPU portion is written to use OpenCL, due to its ability to support a wide variety of processing units.

4.1 Kernels

The fluid solver is implemented in OpenCL. Once initial conditions have been set, all computations are performed on the GPU; the only time it is necessary to transfer GPU memory back to main memory is when output is stored to disk.

Small concurrent OpenCL programs that run on the GPU are called kernels. A separate kernel is run for each point on a two or three dimensional grid. Modern GPUs have several hundred cores that each run a kernel. The OpenCL model is shown in figure 4. Computation units are divided into work groups that can be synchronized.

A GPU can run several hundred instances of a kernel in parallel, one for each core. In the OpenCL model, it is not possible to have synchronized kernels across an work groups, because the domain can be much larger than the number of GPU cores. Thus, the fluid simulator requires several kernels for each simulation step. They are as follows for a two dimensional simulation. These kernels pertain to the steps listen in section 3.5.

- Apply velocity boundary conditions to the domain as described in step 1.
- Apply step 2 which solves the momentum equation. This kernel also applies the nonlinear term and body forces. Additionally, this kernel sets the right-hand side of the x-direction

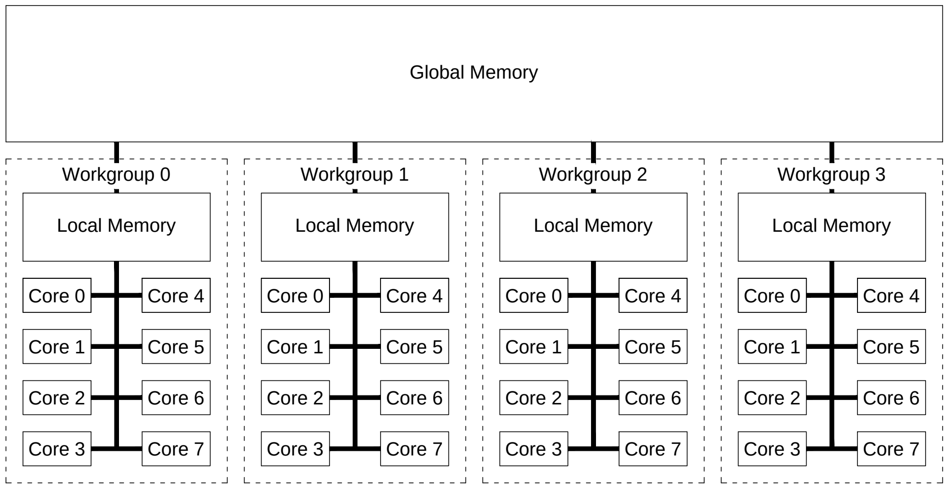

Figure 4: Description of the OpenCL model. Each work group has access to a small amount of a local memory that is much faster to access than global memory. Synchronization is possible between cores in a work groups but not between work groups.

split in step 3 because the right-hand side only depends $\mathbf{u}_{i,j}*$ from the momentum equation.

- Solve tridiagonal systems for u and v in the x-direction. The kernel domain size is now n_y.
- Compute the right-hand side for the y-direction split used in step 4. This must be a separate kernel because the previous kernels solve $\mathbf{u}_{i,j}^{**}$ in the x-direction and those values are required in the y-direction.
- Solve tridiagonal systems for u and v in the y-direction. The kernel domain size is now n_x. Additionally, compute the right-hand side of the x-direction pressure update because it does not depend on $\mathbf{u}_{i,j}^{**}$ at all.
- Solve the tridiagonal system for pressure in the x-direction in step 5.
- Solve the tridiagonal system for pressure in the y-direction in step 6. There is no right-hand side to compute.
- Apply the pressure correction as described in step 7.

Three dimensional simulations require additional kernels to account for the additional dimension in the direction split. The complete set of kernels pertaining to section 3.6 are:

- Apply velocity boundary conditions to the domain as described in step 1.

- Apply step 2 which solves the momentum equation. This kernel also applies the nonlinear term and body forces. Additionally, this kernel sets the right-hand side of the x-direction split in step 3 because the right-hand side only depends $\mathbf{u}_{i,j}*$ from the momentum equation.

- Solve tridiagonal systems for u, v and w in the x-direction. The kernel domain size is now $n_y \times n_z$.

- Compute the right-hand side for the y-direction split used in step 4. This must be a separate kernel because the previous kernels solve $\mathbf{u}^{**}_{i,j,k}$ in the x-direction and those values are required in the y-direction.

- Solve tridiagonal systems for u, v and w in the y-direction. The kernel domain size is now $n_x \times n_z$.

- Compute the right-hand side for the z-direction split used in step 5.

- Solve tridiagonal systems for u, v and w in the z-direction. The kernel domain size is now $n_x \times n_y$. Additionally, compute the right-hand side of the x-direction pressure update because it does not depend on $\mathbf{u}^{***}_{i,j,k}$ at all.

- Solve the tridiagonal system for pressure in the x-direction in step 6.

- Solve the tridiagonal system for pressure in the y-direction in step 7. There is no right-hand side to compute.

- Solve the tridiagonal system for pressure in the z-direction in step 8. There is no right-hand side to compute.

- Apply the pressure correction as described in step 9.

4.2 Scaling

Numerical experiments are conducted on a single core of a distributed memory cluster with a GPU. The GPU specifications are:

- NVIDIA Corporation, Tesla M2075

- 14 cores, 1147MHz, 5375MB

- Max work group size: 1024

- Local mem: 48kB

Number of Unknowns	**Runtime (s)**
4.0×10^3	0.64
1.6×10^4	2.54
6.6×10^4	8.20
2.6×10^5	38.23
1.0×10^6	266.05
4.1×10^6	3096.28

Table 1: Timing results of the simulation. These results were run 3 times each, taking the minimum time per run.

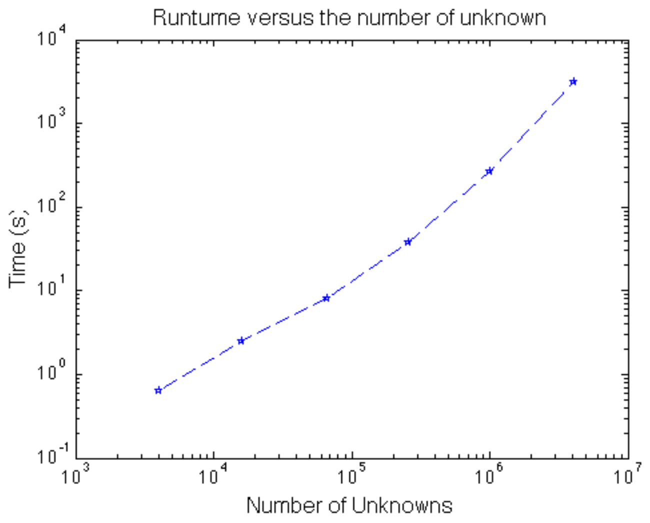

Figure 5: Logarithmic plot of the fluid simulation runtimes from the above table. Notice how the scaling is linear.

Table 1 shows runtimes for a lid-driven cavity as it is scaled up to its maximum allowable problem size (limited by the GPU memory). The simulations are run for 1000 steps. Figure 5 shows linear scaling, as predicted because tridiagonal systems are solved $\mathcal{O}(n)$. The Reynold's number does not affect these results because all systems are solved directly with a constant stepsize.

4.3 Renderer

A renderer was developed in OpenGL to display results of the fluid simulations in two-dimensions (a three dimensional version is part of future work). The renderer animates the simulations. Figure 6 shows the steady-state solution of a lid-driven cavity with top and bottom boundaries prescribed in the same direction. Figure 7 shows the steady-state solution where the top boundary is prescribed and the rest are zero.

The renderer draws traces in the velocity field by advecting particles in the domain. The traces

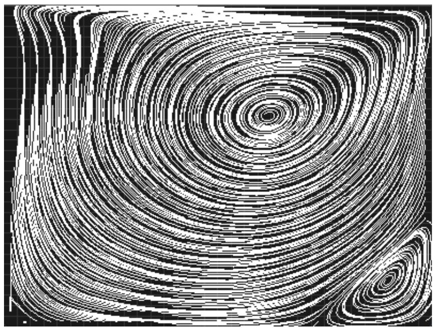

Figure 6: Steady state solution where top and bottom boundaries are prescribed in opposite directions. The left and right boundaries are both set to zero.

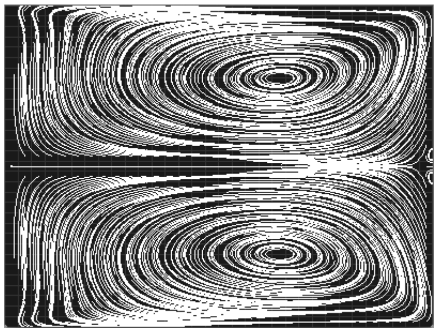

Figure 7: Steady state solution where the top boundary is prescribed and the rest are zero.

are integrated over many timesteps using forward Euler. Figure 8 shows a system with opposing boundaries where top and bottom boundaries are in opposite directions.

4.4 Verification and Validation

The OpenCL kernels and `c++` source together comprise of just under 3000 lines of code. Therefore, some form of incremental testing is necessary to locate bugs. Unit testing is the first step in verification to ensure the fluid solver is functioning correctly. A small test suite was written to evaluate key numerical functions. Some examples of units tests include testing interpolation of the grid, tridiagonal solvers, and finite difference calculations. Knowledge that these components function correctly allows development to run much more efficiently, narrowing the search domain for possible bugs in the code.

The domain is either a two dimensional rectangle or a three dimensional rectangular prism. Depending on boundary conditions, it can therefore exhibit some form of symmetry. For instance, a two dimensional simulation with equal top and bottom boundary conditions will have the top and bottom half of the domain be “mirrors“ of each other and shown in Figure 7. Such symmetry

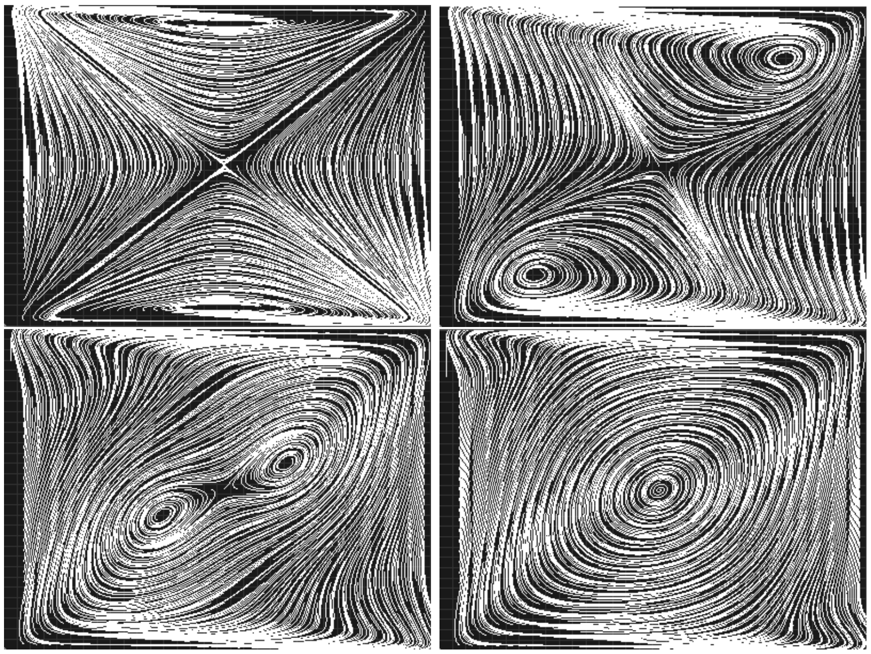

Figure 8: Four frames of a simulation approaching a steady state, where top and bottom boundaries are prescribed in opposite directions.

is crucial to detecting problems that provide reasonable solutions but have hidden “off by one” errors. Additionally, all problems will exhibit rotational symmetry, another method to verify that the fluid simulator is functioning correctly.

The driven cavity is a problem commonly used to evaluate numerical methods for fluid simulation. Numerical experiments are shown to graphically match results described in [3].

5 Progress overview and future work

The direction splitting algorithm for the incompressible Navier–Stokes equations was implemented for GPU using OpenCL. It solves the advection term using a semi-Lagrangian method, the viscosity term using ADI splitting, body forces using forward Euler and the pressure update using a relaxation of the incompressibility constraint. Numerical simulation shows the runtime to scale linearly. Therefore, the primary goal of this project has been accomplished. Additionally, a renderer has been developed using OpenGL to display the velocity field as it changes over time.

The following sections discuss future directions for the fluid solver.

5.1 CFL Condition

The Navier–Stokes equations are used to update velocities inside the liquid. The simulation steps are limited by the Courant-Friedrichs-Lewy (CFL) condition, which guarantees that numerical solutions to the simulation will converge [1]. The condition is written as follows:

$$\Delta t = C \left[\frac{\Delta x}{\max(u)} + \frac{\Delta y}{\max(v)} \right]$$

where C is is the Courrant number, Δx and Δy refer to grid cell size, and u and v are the fluid velocity values. In parallel, the problem reduces to finding the maximum value in a dataset, which can easily be solved by dividing the dataset into a number of sub-domains, having each parallel worker find the maximum value within its domain, and then repeating the processes until the overall maximum has been reached.

5.2 Level sets

It is often interesting to model the movement of a liquid in air, perhaps with solid boundaries at the edge or in the middle of the domain, thus introducing free surface and solid boundary conditions. The interface between liquid and air can be modelled using a concept known as a level set. Each cell in the MAC grid is assigned a value $\phi(x)$: an implicit surface function whose value is equal to the signed distance between cells centres and the nearest point on the surface. Cells with $\phi < 0$ contain liquid; cells with $\phi > 0$ contain air. The interface between the two is taken to be where $\phi = 0$. A more detailed study of level sets is presented in [9]. To represent a dynamic free surface, the entire level set must be advected according to the velocity field at each timestep. That is, we must solve the advection equation $\frac{\partial \phi}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = 0$ to update the free surface. The level set must then be recomputed because the advection integration does not preserve signed distances to surfaces. Level sets can easily be computed in parallel, noting that at any given step, the maximum distance the free surface can move is limited by the CFL condition. Thus, the level set can computed on a series of local domains.

References

[1] Bridson, R. *Fluid simulation for computer graphics.* A K Peters, Wellesley, 2007.

[2] Chorin, A. J. Numerical solution of the navier–stokes equations. *Math Comp 22* (1968), 745–762.

[3] Guermond, J.-L., Migeon, C., Pineau, G., and Quartapelle, L. Start-up flows in a three-dimensional rectangular driven cavity of aspect ratio 1:1:2 at Re = 1000. *J Fluid Mech 450* (January 2002), 169–199.

[4] Guermond, J.-L., and Minev, P. A new class of fractional step techniques for the incompressible Navier-Stokes equations using direction splitting. *Comptes Rendus Mathematique 348*, 9-10 (2010), 581–585.

[5] Guermond, J.-L., and Minev, P. A new class of massively parallel direction splitting for the incompressible Navier-Stokes equations. *Comput Methods Appl Mech Engrg 200*, 23-24 (2011), 2083–2093.

[6] Guermond, J.-L., Minev, P., and Shen, J. An overview of projection methods for incompressible flows. *Comput Methods Appl Mech Engrg 195* (2006), 6011–6045.

[7] Harlow, F. H., and Welch, J. E. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Phys. Fluids 8*, 12 (1965), 2182–2189.

[8] Jim Douglas, J. Alternating direction methods for three space variables. *Numerische Mathematik 4*, 1 (1962), 41–63.

[9] Osher, S. J., and Fedkiw, R. P. *Level Set Methods and Dynamic Implicit Surfaces*, vol. 153 of *Applied mathematical sciences*. Springer-Verlag, Berlin, 2003.

[10] Stam, J. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 121–128.