Numerical Solver For Incompressible Fluid Navier-Stokes Equation

High performance scientific computing in aerospace

Final report - Module A

Enrico Tirri, 10742316 - Group A

February 27, 2025

Contents

1 Introduction **1**

2 Mathematical Formulation **1**
2.1 Governing Equations . . . 1
2.2 Boundary and Initial Conditions . . . 1
2.2.1 Initial Condition . . . 1
2.2.2 Boundary Conditions . . . 1
2.3 Discretization Approach . . . 2

3 Numerical Methodology **2**
3.1 Spatial Discretization: Full Staggered Grid . . . 2
3.2 Momentum Equation Space Discretization . . . 2
3.2.1 Convective Term Discretization . . . 3
3.2.2 Diffusion Term Discretization . . . 3
3.2.3 Pressure Gradient Discretization . . . 3
3.3 Momentum Equation Time Integration . . . 3
3.4 Pressure Correction and Incompressibility Enforcement . . . 4
3.4.1 The Poisson Solver . . . 4
3.5 Stability Condition . . . 5
3.6 The Final Algorithm . . . 6

4 Implementation Details **6**
4.1 The Staggered Grid . . . 6
4.2 The Boundary Conditions . . . 6
4.3 Domain Decomposition . . . 7
4.4 Parallelization . . . 7
4.5 Optimizations . . . 8

5 Testing **8**
5.1 Verification . . . 8
5.2 Testcase Validation . . . 8
5.3 Benchmark . . . 9

6 Results and Discussion **9**

1 Introduction

The Navier-Stokes equations govern the motion of fluid flows and play a fundamental role in numerous scientific and engineering applications, including aerodynamics, weather modeling, and biomedical flows. In the incompressible case, these equations impose a divergence-free condition on the velocity field, creating a strong coupling between pressure and velocity. This coupling presents significant numerical challenges, requiring efficient and stable solvers to achieve accurate solutions.

Developing robust numerical methods for solving the incompressible Navier-Stokes equations has been an area of active research. Traditional approaches include finite difference, finite volume, and finite element methods, each with distinct advantages depending on the problem at hand. One of the primary difficulties in solving these equations numerically is enforcing the incompressibility constraint while ensuring stability and efficiency. Various strategies, such as projection methods, pressure correction schemes, and fully coupled formulations, have been proposed to address this challenge. However, computational cost, accuracy, and scalability remain key concerns, particularly for high Reynolds number flows and complex geometries.

This report presents a numerical solver specifically designed for the incompressible Navier-Stokes equations, focusing on accuracy, stability, and computational efficiency. We describe the mathematical formulation, discretization strategy, and implementation details of our approach. The solver is validated using benchmark problems, and its performance is assessed against target references.

2 Mathematical Formulation

The motion of an incompressible fluid is governed by the Navier-Stokes equations, which describe the conservation of momentum and mass. These equations can be written in their dimensionless form as:

2.1 Governing Equations

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{Re}\nabla^2 \mathbf{u} + \nabla p = \mathbf{f}(t), \quad (1)$$

$$\nabla \cdot \mathbf{u} = 0, \quad (2)$$

where:

- $\mathbf{u} = (u, v, w)$ is the velocity field,
- p is the pressure field,
- Re is the Reynolds number, defined as:

$$Re = \frac{UL}{\nu} \quad (3)$$

  where U is a characteristic velocity, L is a characteristic length, and ν is the kinematic viscosity,

- $\mathbf{f}$ represents external forces, such as gravity or body forces.

The first equation represents momentum conservation, where the terms correspond to unsteady acceleration, convective acceleration, viscous diffusion, and pressure gradient forces. The second equation enforces the incompressibility condition, ensuring that the velocity field remains divergence-free.

2.2 Boundary and Initial Conditions

To solve these equations numerically, appropriate boundary and initial conditions must be specified:

2.2.1 Initial Condition

The velocity field at $t = 0$ is given as:

$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x}) \quad (4)$$

where $\mathbf{u}_0(\mathbf{x})$ is a predefined initial velocity distribution.

2.2.2 Boundary Conditions

Different types of boundary conditions may be applied, depending on the problem setup:

- **Dirichlet boundary condition (no-slip condition):** The velocity is specified on solid boundaries:

$$\mathbf{u} = \mathbf{u}_b \quad \text{on} \quad \partial\Omega_D \quad (5)$$

- **Periodic boundary condition:** The velocity field is periodic across boundaries, commonly used in simulations of homogeneous turbulence or channel flows.

2.3 Discretization Approach

To numerically solve the incompressible Navier-Stokes equations, the spatial and temporal derivatives must be discretized. Common methods include:

- **Finite Difference Method (FDM)** - Uses discrete grid points to approximate derivatives.
- **Finite Volume Method (FVM)** - Ensures local conservation of mass and momentum by integrating over control volumes.
- **Finite Element Method (FEM)** - Uses basis functions to approximate velocity and pressure fields over elements.

In this work, the chosen discretization scheme is finite difference with staggered grids, reasons are several:

- Simplicity and efficiency
- Stable and accurate numerical solution while maintaining the incompressibility constraint.
- Based on stencil methods we can easily achieve N-order approximations
- Several known techniques for time discretization, in particular we will use a linear multi-step method known as **Runge-Kutta scheme**

3 Numerical Methodology

To solve the incompressible Navier-Stokes equations numerically, we employ a full staggered grid for spatial discretization combined with Runge-Kutta scheme for time integration.

All spatial derivative are approximated using second-order finite difference schemes and Runge-Kutta method uses the Van der Houwen/Wray coefficients to achieve maximum third-order accuracy in time while maintaining robust stability properties.

3.1 Spatial Discretization: Full Staggered Grid

A full staggered grid (also known as a Marker-And-Cell or MAC grid) is used for the spatial discretization of the velocity and pressure fields. In this grid arrangement:

- The pressure p is stored at the cell centers.
- The velocity components (u, v, w) are stored at the cell faces.

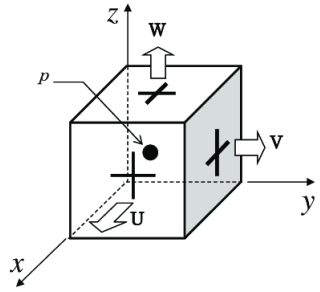

Figure 1: Space cell with full staggered components

This configuration helps to avoid pressure-velocity decoupling and naturally enforces the incompressibility constraint.

NOTE: From here on, all indices of the discretized unknowns (i, j, k) refer to the "Cell" concept, where each Cell contains one unknown per component, as illustrated in the Figure 1. Specifically, the index i corresponds to the x-axis, j to the y-axis, and k to the z-axis.

3.2 Momentum Equation Space Discretization

Second-order spatial discretization is achieved using central difference schemes; because computations occur at staggered grid positions, custom formulas are defined for each velocity component:

3.2.1 Convective Term Discretization

$$[(\mathbf{u}\cdot\nabla)\mathbf{u}]_{i,j,k} \approx \begin{bmatrix} u_{i,j,k}\frac{u_{i+1,j,k}-u_{i-1,j,k}}{2\,\Delta x} + v|_{u_{i,j,k}}\frac{u_{i,j+1,k}-u_{i,j-1,k}}{2\,\Delta y} + w|_{u_{i,j,k}}\frac{u_{i,j,k+1}-u_{i,j,k-1}}{2\,\Delta z} \\ u|_{v_{i,j,k}}\frac{v_{i+1,j,k}-v_{i-1,j,k}}{2\,\Delta x} + v_{i,j,k}\frac{v_{i,j+1,k}-v_{i,j-1,k}}{2\,\Delta y} + w|_{v_{i,j,k}}\frac{v_{i,j,k+1}-v_{i,j,k-1}}{2\,\Delta z} \\ u|_{w_{i,j,k}}\frac{w_{i+1,j,k}-w_{i-1,j,k}}{2\,\Delta x} + v|_{w_{i,j,k}}\frac{w_{i,j+1,k}-w_{i,j-1,k}}{2\,\Delta y} + w_{i,j,k}\frac{w_{i,j,k+1}-w_{i,j,k-1}}{2\,\Delta z} \end{bmatrix} \tag{6}$$

where $a|_{b_{i,j,k}}$ stands for interpolation of component a on component b in cell (i,j,k).
For brevity, we present only $v|_{u_{i,j,k}}$, others can be trivially derived:

$$v|_{u_{i,j,k}} = \frac{v_{i,j,k} + v_{i+1,j,k} + v_{i,j-1,k} + v_{i+1,j-1,k}}{4} \tag{7}$$

3.2.2 Diffusion Term Discretization

$$\left[\nabla^2\mathbf{u}\right]_{i,j,k} \approx \begin{bmatrix} \frac{u_{i+1,j,k}-2\,u_{i,j,k}+u_{i-1,j,k}}{\Delta x^2} + \frac{u_{i,j+1,k}-2\,u_{i,j,k}+u_{i,j-1,k}}{\Delta y^2} + \frac{u_{i,j,k+1}-2\,u_{i,j,k}+u_{i,j,k-1}}{\Delta z^2} \\ \frac{v_{i+1,j,k}-2\,v_{i,j,k}+v_{i-1,j,k}}{\Delta x^2} + \frac{v_{i,j+1,k}-2\,v_{i,j,k}+v_{i,j-1,k}}{\Delta y^2} + \frac{v_{i,j,k+1}-2\,v_{i,j,k}+v_{i,j,k-1}}{\Delta z^2} \\ \frac{w_{i+1,j,k}-2\,w_{i,j,k}+w_{i-1,j,k}}{\Delta x^2} + \frac{w_{i,j+1,k}-2\,w_{i,j,k}+w_{i,j-1,k}}{\Delta y^2} + \frac{w_{i,j,k+1}-2\,w_{i,j,k}+w_{i,j,k-1}}{\Delta z^2} \end{bmatrix} \tag{8}$$

3.2.3 Pressure Gradient Discretization

$$[\nabla p]_{i,j,k} \approx \begin{bmatrix} \frac{p_{i+1,j,k}-p_{i,j,k}}{\Delta x} \\ \frac{p_{i,j+1,k}-p_{i,j,k}}{\Delta y} \\ \frac{p_{i,j,k+1}-p_{i,j,k}}{\Delta z} \end{bmatrix} \tag{9}$$

3.3 Momentum Equation Time Integration

Time integration of the incompressible Navier-Stokes equations is performed using an explicit Runge-Kutta scheme.

In a semi-discrete form, the momentum equation is written as

$$\begin{aligned} \frac{\partial\mathbf{u}}{\partial t} &= -(\mathbf{u}\cdot\nabla)\mathbf{u} + \frac{1}{Re}\nabla^2\mathbf{u} - \nabla p + \mathbf{f}(t) \\ &= \mathbf{R}(\mathbf{u}, p, t), \end{aligned} \tag{10}$$

Let $\mathbf{u}^n$ denote the velocity field at time t^n and Δt the time step. The Runge-Kutta update is given by

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \sum_{i=1}^{s} b_i\,\mathbf{k}_i, \tag{11}$$

where s is the number of stages, and the intermediate stage contributions $\mathbf{k}_i$ are computed as

$$\mathbf{k}_i = \mathbf{R}\left(\mathbf{u}^{(i)}, p^{(i)}, t + c_i\,\Delta t\right), \tag{12}$$

with $\mathbf{u}^{(i)}$ and $p^{(i)}$ being the intermediate velocity and pressure values at stage i. The coefficients b_i (as well as the weights used to compute the intermediate stages) are selected to achieve the desired order of accuracy and stability. Maximum third order accuracy can be achieved by using the Van der Houwen/Wray coefficients (Figure 2).

After the Runge-Kutta update, the intermediate velocity field is corrected to enforce the divergence-free condition. This is typically accomplished using a projection method that involves solving a Poisson equation for a pressure correction term, thereby ensuring that the updated velocity field satisfies the incompressibility constraint.

c_i	b_1	b_2	b_3
0	0	0	0
8/15	8/15	0	0
2/3	1/4	5/12	0
1	1/4	0	3/4

Figure 2: Van der Houwen/Wray coefficients for Runge-Kutta scheme

3.4 Pressure Correction and Incompressibility Enforcement

To enforce the divergence-free condition, a Chorin-Temam projection method is applied after the Runge-Kutta update:

1. **Compute an intermediate velocity:**

$$\mathbf{u}^* = \mathbf{u}^{n+1} \quad \text{(prior to pressure correction)}. \tag{13}$$

2. **Solve the pressure Poisson equation:**

$$\nabla^2 \phi = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*. \tag{14}$$

3. **Correct the velocity field:**

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t \, \nabla \phi. \tag{15}$$

4. **Update the pressure:**

$$p^{n+1} = p^n + \phi. \tag{16}$$

Note: Explicit boundary conditions for the pressure field are not always available (they are provided only for periodic boundaries), which can complicate the solution of the Poisson equation. However, by imposing homogeneous Neumann boundary conditions on the pressure correction ϕ, we can effectively address this limitation while maintaining the scheme's overall second-order accuracy.

3.4.1 The Poisson Solver

In this section, we consider the one-dimensional (1D) Poisson problem as a representative case. The governing equation is

$$\nabla^2 p = b, \tag{17}$$

which, when discretized using a second-order finite difference scheme, yields a linear system of the form

$$A\mathbf{x} = \mathbf{b}. \tag{18}$$

Assuming that the matrix A is diagonalize-able, the solution can be obtained efficiently via its eigen-decomposition. Let us denote:

1. H, the matrix whose columns are the right eigenvectors of A:

$$H = \begin{bmatrix} \boldsymbol{\eta}_1 & \boldsymbol{\eta}_2 & \dots & \boldsymbol{\eta}_N \end{bmatrix}$$

$$\text{with} \qquad A\,\boldsymbol{\eta}_j = \lambda_j \, \boldsymbol{\eta}_j, \tag{19}$$

2. Ψ, the matrix whose columns are the left eigenvectors of A:

$$\Psi = \begin{bmatrix} \boldsymbol{\psi}_1 & \boldsymbol{\psi}_2 & \dots & \boldsymbol{\psi}_N \end{bmatrix},$$

$$\text{with} \qquad \boldsymbol{\psi}_j^\dagger A = \lambda_j \, \boldsymbol{\psi}_j^\dagger, \tag{20}$$

3. Λ, the diagonal matrix of eigenvalues:

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_N \end{bmatrix}. \tag{21}$$

Under the assumptions that A is diagonalize-able (which implies $A^\dagger = A$) and that the eigenvectors satisfy the ortho-normality conditions

$$\Psi^\dagger H = H \Psi^\dagger = I, \tag{22}$$

the eigen-decomposition of A is given by

$$AH = H\Lambda. \tag{23}$$

Multiplying the linear system $A\mathbf{x} = \mathbf{b}$ by $\Psi^\dagger$ yields

$$\begin{aligned} \Psi^\dagger A H \Psi^\dagger \mathbf{x} &= \Psi^\dagger \mathbf{b}, \\ \Lambda \, \tilde{\mathbf{x}} &= \tilde{\mathbf{b}}, \\ \tilde{x}_i &= \lambda_i^{-1} \, \tilde{b}_i, \end{aligned} \tag{24}$$

where we have defined $\tilde{\mathbf{x}} = \Psi^\dagger \mathbf{x}$ and $\tilde{\mathbf{b}} = \Psi^\dagger \mathbf{b}$. The computational complexity associated with this approach is as follows:

- The transformation from $\mathbf{b}$ to $\tilde{\mathbf{b}}$ requires $O(N^2)$ operations.

- Solving the diagonal system $\Lambda\,\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ requires $O(N)$ operations.
- The inverse transformation from $\tilde{\mathbf{x}}$ back to $\mathbf{x}$ also requires $O(N^2)$ operations.

Thus, for a 1D domain the overall complexity is $O(N \cdot N^2)$, while for a three-dimensional (3D) domain it becomes $O(N^3 \cdot N^2)$.

A significant reduction in computational complexity is achievable by leveraging the Fast Fourier Transform (FFT) for the transformations $\mathbf{b} \to \tilde{\mathbf{b}}$ and $\tilde{\mathbf{x}} \to \mathbf{x}$, thereby reducing the complexity of each transformation to $O(N \log N)$ and the overall complexity to $O(N^3 \cdot N \log N)$. FFT-based methods are applicable when the eigenvectors of A can be expressed in terms of sine and cosine functions.

For instance, consider the function

$$e^{i2\pi \frac{nm}{N}}, \quad 0 \le n, m \le N, \tag{25}$$

which represents a 2D function. We demonstrate that this function is an eigenvector of the discrete Laplacian operator A:

$$\begin{aligned} A\, e^{i\frac{2\pi nm}{N}} &= e^{i\frac{2\pi n(m-1)}{N}} - 2e^{i\frac{2\pi nm}{N}} + e^{i\frac{2\pi n(m+1)}{N}} \\ &= -2e^{i\frac{2\pi nm}{N}} + e^{i\frac{2\pi nm}{N}} \left(e^{i\frac{2\pi n}{N}} + e^{-i\frac{2\pi n}{N}}\right) \\ &= e^{i\frac{2\pi nm}{N}} \left(2\cos\left(\frac{2\pi n}{N}\right) - 2\right). \end{aligned} \tag{26}$$

Since

$$e^{i\frac{2\pi nm}{N}} = \cos\left(\frac{2\pi nm}{N}\right) + i\sin\left(\frac{2\pi nm}{N}\right), \tag{27}$$

the eigenvectors can indeed be represented as a combination of sine and cosine functions, ensuring the applicability of FFT-based solvers.

In our implementation, which utilizes a full staggered grid, the forward transformation (i.e., the mapping $\mathbf{b} \to \tilde{\mathbf{b}}$) is performed using:

- The Discrete Fourier Transform (DFT) for periodic boundary conditions,
- The Discrete Cosine Transform (DCT-II) for Neumann boundary conditions.

The corresponding inverse transformations (i.e., $\tilde{\mathbf{x}} \to \mathbf{x}$) are executed using:

- The Inverse DFT (scaled by $\frac{1}{2N}$) for periodic boundary conditions,
- The Inverse DCT (DCT-III, scaled by $\frac{1}{2N}$) for Neumann boundary conditions.

The eigenvalues for the discretized Laplacian operator are given by:

- For periodic boundary conditions:

$$\lambda_k = -\left(\frac{2\sin\left(\frac{k\pi}{N}\right)}{\Delta x}\right)^2, \tag{28}$$

- For Neumann boundary conditions:

$$\lambda_k = -\left(\frac{2\sin\left(\frac{k\pi}{2N}\right)}{\Delta x}\right)^2. \tag{29}$$

This spectral method, facilitated by the FFT, significantly reduces the computational cost of solving the Poisson equation on a 3D domain.

3.5 Stability Condition

While the numerical discretization method we employ provides several advantages, it is not without limitations. In particular, the pressure correction technique necessitates adherence to the following stability condition:

$$\Delta t_{\max} < c\,\frac{\Delta x^2}{\nu} \propto \Delta x^2\, Re, \tag{30}$$

where:

- $\Delta t_{\max}$ is the maximum allowable time step,
- Δx denotes the spatial discretization step (in an n-dimensional problem, this is taken as the smallest spacing among all axes),
- ν is the kinematic viscosity,
- c is a constant that is specific to the problem,
- Re is the Reynolds number.

3.6 The Final Algorithm

Here, we present the complete algorithm, which incorporates the Runge-Kutta scheme with pressure correction and utilizes the coefficients detailed in Figure 2. The stage contributions, denoted by $\mathbf{R}$, are derived as specified in Equations (10) and (12). Intermediate steps have been streamlined by substituting previously computed results; thus, only the final outcome is reported.

$$
\begin{aligned}
\mathbf{R}_I &= -\left(\mathbf{u}^n \cdot \nabla\right)\mathbf{u}^n + \frac{1}{Re}\nabla^2\mathbf{u}^n \\
\mathbf{Y}_2^* &= \mathbf{u}^n + \frac{64\,\Delta t}{120}\mathbf{R}_I - \frac{64\,\Delta t}{120}\nabla p^n \\
\nabla^2\left(\phi_2 - p^n\right) &= \frac{120}{64\,\Delta t}\nabla\cdot\mathbf{Y}_2^* \\
\mathbf{Y}_2 &= \mathbf{Y}_2^* - \frac{64\,\Delta t}{120}\nabla\left(\phi_2 - p^n\right)
\end{aligned}
\tag{31}
$$

$$
\begin{aligned}
\mathbf{R}_{II} &= -\left(\mathbf{Y}_2 \cdot \nabla\right)\mathbf{Y}_2 + \frac{1}{Re}\nabla^2\mathbf{Y}_2 \\
\mathbf{Y}_3^* &= \mathbf{Y}_2 + \frac{50\,\Delta t}{120}\mathbf{R}_{II} - \frac{34\,\Delta t}{120}\mathbf{R}_I - \frac{16\,\Delta t}{120}\nabla\phi_2 \\
\nabla^2\left(\phi_3 - \phi_2\right) &= \frac{120}{16\,\Delta t}\nabla\cdot\mathbf{Y}_3^* \\
\mathbf{Y}_3 &= \mathbf{Y}_3^* - \frac{16\,\Delta t}{120}\nabla\left(\phi_3 - \phi_2\right)
\end{aligned}
\tag{32}
$$

$$
\begin{aligned}
\mathbf{R}_{III} &= -\left(\mathbf{Y}_3 \cdot \nabla\right)\mathbf{Y}_3 + \frac{1}{Re}\nabla^2\mathbf{Y}_3 \\
\mathbf{u}^* &= \mathbf{Y}_3 + \frac{90\,\Delta t}{120}\mathbf{R}_{III} - \frac{50\,\Delta t}{120}\mathbf{R}_{II} - \frac{40\,\Delta t}{120}\nabla\phi_3 \\
\nabla^2\left(p^{n+1} - \phi_3\right) &= \frac{120}{40\,\Delta t}\nabla\cdot\mathbf{u}^* \\
\mathbf{u}^{n+1} &= \mathbf{u}^* - \frac{40\,\Delta t}{120}\nabla\left(p^{n+1} - \phi_3\right)
\end{aligned}
\tag{33}
$$

4 Implementation Details

Our implementation prioritizes three main objectives: performance, understandability, and portability. We selected C++, a modern, efficient, and widely used programming language, but to minimize performance overhead and avoid the complexity of excessive abstraction layers, we adopted a coding style that remains as close to C as possible.

For FFT computations, we opted to utilize the open-source FFTW library [2]. Implemented in C, FFTW is easily integrated and delivers exceptional performance.

4.1 The Staggered Grid

The grid is stored in memory on a per-component basis as a three-dimensional array with row-major indexing. This layout enhances cache locality, facilitates rapid data access, and enables efficient memory transfers (as discussed in the parallelization section).

Additionally, complex and verbose index computations are encapsulated within global macros, thereby providing a simplified triple-index notation for data access.

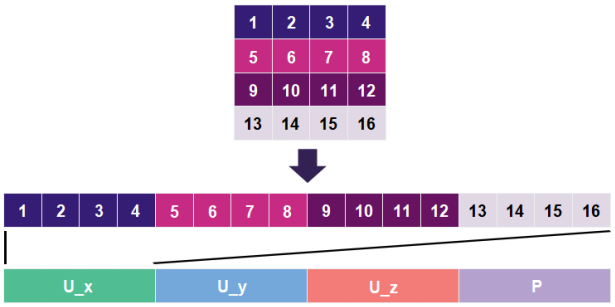

Figure 3: Staggered grid memory organization (2D array is represented)

4.2 The Boundary Conditions

Boundary conditions present an additional challenge, as they must be imposed without incurring excessive computational overhead or significantly increasing memory usage. To address this, our approach employs a layer of "ghost points" that surround the computational domain. This strategy offers several advantages:

1. It decouples the application of boundary conditions from the momentum equations, thereby eliminating the need for conditional branching within the nested computational loops.

2. It permits the implementation of boundary conditions in dedicated functions, which enhances code clarity and maintainability.

3. It facilitates the encapsulation of interprocessor communication for domain decomposition, as discussed in the parallelization section.

4. For large grids, the additional memory requirement, which scales as $O(N^2)$, becomes

negligible compared to the overall domain size, which scales as $O(N^3)$.

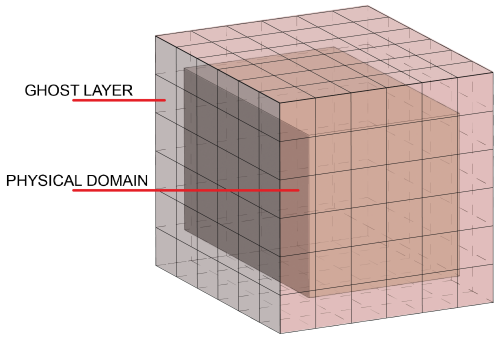

Figure 4: Staggered grid with ghost layer

Due to the staggered grid configuration, a node for every component is not always present at the physical boundary. To address this, correction mechanisms are employed:

1. For Dirichlet boundary conditions, a reverse interpolation is used:

$$U_{\text{ghost}} = 2D - U_{\text{inner}}, \tag{34}$$

where D is the prescribed boundary value.

2. For periodic boundary conditions, the computational domain is extended to include the boundary nodes. This is feasible because the ghost layer is completely populated with the inner boundary values from the adjacent periodic domains.

4.3 Domain Decomposition

Domain decomposition enables the distribution of computational workload across multiple processors, thereby offering potential performance gains provided that data transfer overhead is effectively managed. Ideally, the domain should be partitioned in a way that minimizes the ratio of boundary cells to interior cells, reducing communication costs and maximizing the efficiency of single-processor computations. While a "bubble" domain, i.e. a small cubic decomposition, would be optimal in this regard, it is not well-suited to our numerical algorithm because the FFT computation requires rapid processing of complete axis-wise data segments. As a compromise, we adopt the "Pencil Decomposition" strategy, which partitions the domain along only two axes rather than all three. This approach strikes a balance between efficient FFT execution and parallel scalability. The domain decomposition is performed using the open-source library 2Decomp [4], originally developed in Fortran and subsequently ported to C++ through the independent open-source project 2Decomp_C [Github repository].

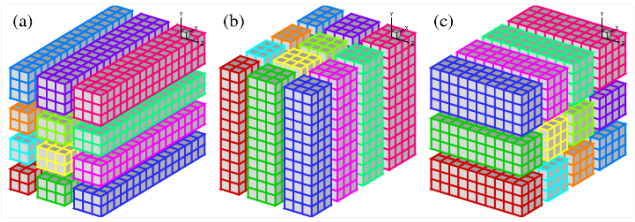

Figure 5: Pencil decomposition

4.4 Parallelization

Due to the computationally intensive nature of the problem, its high synchronization requirements, and the absence of deferrable memory accesses, multithreading was deemed unsuitable for parallelization. Instead, we adopted a multi-core parallelization strategy that leverages the communication layer provided by MPI in the open-source OpenMPI library [3], as well as SIMD instructions when available.

In our implementation, a pencil-wise domain decomposition is employed, whereby each processor is assigned a subdomain. Processors exchange data with their neighbors by sending inner boundary layers and receiving corresponding outer boundary layers, which are stored in ghost layers. This approach allows the interior computations of each subdomain to proceed independently, effectively treating them as single-processor computations, with synchronization occurring only during the enforcement of boundary conditions.

Moreover, the pencil decomposition facilitates the parallelization of FFT computations along individual axes. The C2Decomp library automatically provides a pencil decomposition across all spatial axes and includes a mechanism to transpose domain data, redistributing it among processors according to different pencil orientations. This transposition requires synchronization only during the data exchange phase.

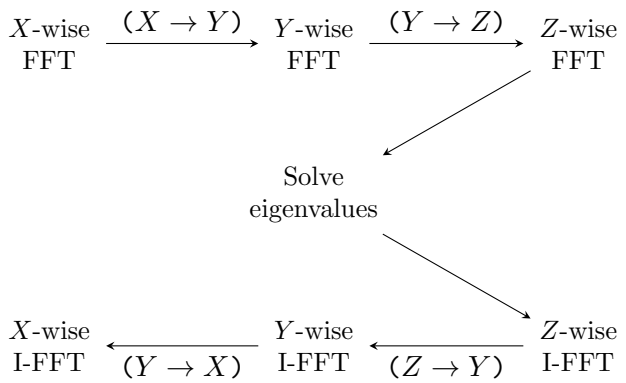

Figure 6: Parallelized Fast Poisson Solver Scheme

4.5 Optimizations

Optimizations have been implemented both in the computational routines and in memory usage. On the computational side, given that the code is predominantly composed of nested loops, significant effort has been devoted to minimizing divergent control flows and applying loop fusion techniques to enhance cache reuse. This has been achieved by eliminating conditional branches within loop bodies, determining loop iteration ranges prior to entering nested loops, employing fixed induction variable increments exclusively, and segregating the computations of the velocity components and pressure to enable the compiler to effectively fuse loops.

Regarding memory usage, an analysis of the data flow revealed that only three data buffers are required:

- **Model buffer:** Serves as the input and output of the solver and includes ghost layers.
- **Temporary buffer:** An intermediate buffer used for computation, which also includes ghost layers.
- **RHS buffer:** Utilized for dual purposes:
  - The velocity component stores the stage contributions from the previous step.
  - The pressure component holds the input and output for the Poisson solver.

Furthermore, no additional buffers are required for MPI communication, thanks to the use of custom MPI datatypes provided by OpenMPI.

5 Testing

5.1 Verification

For verification purposes, we assessed the convergence order of both the velocity and pressure components in space and time. Tests were conducted on a domain $[0,1]^3 \subset \mathbb{R}^3$ with initial and Dirichlet boundary conditions specified from the exact solution:

$$u_{\text{exact}}(x,y,z) = \begin{bmatrix} \sin(2\pi x)\cos(2\pi y)\sin(2\pi z)\,t \\ \cos(2\pi x)\sin(2\pi y)\sin(2\pi z)\,t \\ 2\cos(2\pi x)\cos(2\pi y)\cos(2\pi z)\,t \end{bmatrix},$$
$$p_{\text{exact}}(x,y,z) = \cos(2\pi x)\cos(2\pi y)\sin(2\pi z)\,t. \tag{35}$$

The system's evolution was driven by a manufactured solution derived from the exact domain solution.

The computational error is quantified as the L_2 norm of the difference between the exact and computed solutions:

$$\|E\|_{L_2} = \left(\int_\Omega |u_{\text{exact}} - u_{\text{computed}}|^2 \, d\Omega \right)^{1/2}. \tag{36}$$

In all tests, a Reynolds number of $Re = 1000$ was used to ensure a robust stability regime, and the error was evaluated at $1s$ to mitigate the influence of the initial conditions.

5.2 Testcase Validation

The solver was validated using two established cavity flow test cases.

The first test case is conducted on the domain

$$[0,1] \times [0,1] \times [-1,1] \subset \mathbb{R}^3,$$

with a grid resolution of 128^3, a Reynolds number of $Re = 1000$, and a total of 8000 timesteps with $\Delta t = 10^{-3}$. The initial condition is specified as

$$u(x,y,z) = (0,0,0),$$

and all boundaries are subject to Dirichlet conditions:

$$\begin{cases} u(x,y,z) = (1,0,0) & \text{for } x = 1, \\ u(x,y,z) = (0,0,0) & \text{otherwise.} \end{cases}$$

The second test case is performed on the domain

$$[-0.5,0.5]^3 \subset \mathbb{R}^3,$$

with the same grid resolution (128^3), Reynolds number ($Re = 1000$), and $\Delta t = 10^{-3}$, meanwhile timesteps have been increased to 50000 in order to reach a steady state. The initial condition remains

$$u(x, y, z) = (0, 0, 0),$$

but the boundary conditions differ: periodic boundary conditions are applied along the z-axis, while Dirichlet conditions are imposed on the remaining faces:

$$\begin{cases} u(x, y, z) = (0, 1, 0) & \text{for } x = -0.5, \\ u(x, y, z) = (0, 0, 0) & \text{otherwise.} \end{cases}$$

5.3 Benchmark

A speedup evaluation was carried out using the configuration of Testcase 1 (see Section 5.2). For this evaluation, the simulation was limited to 1000 timesteps on a grid consisting of 64 nodes. The experiments were conducted on a system equipped with an 8-core Ryzen 7 5800H CPU, employing both one-dimensional (1D) and two-dimensional (2D) domain partitioning strategies. For each processor configuration, the speedup results were averaged over five consecutive runs.

6 Results and Discussion

Figure 7 demonstrates that, as anticipated, the method exhibits quadratic convergence with decreasing grid spacing.

Figure 8 demonstrates that, as expected, the velocity components converge quadratically as the timestep size decreases. Notably, an initial convergence rate exceeding quadratic is observed, which can be attributed to the Van der Houwen/Wray coefficients in the Runge-Kutta scheme, designed to achieve third-order temporal accuracy. The subsequent reduction to second-order convergence is likely due to additional error contributions, such as those from the spatial discretization, that are only second-order accurate. In contrast, the pressure does not exhibit quadratic convergence; instead, it appears to converge with an order of approximately 1.5. This behavior is not indicative of a flaw in the solver but is rather a known characteristic of the Chorin-Temam projection method that has been adopted [1].

Testcases results are illustrated in Figure 9, first is illustrated on left, second on the right. In both cases, the numerical solutions obtained are in agreement with known results, thereby confirming the robustness and accuracy of our solver.

Figure 10 demonstrates that the expected linear scalability, which is characteristic of pencil decomposition, has not been fully achieved. In particular, the performance of the two-dimensional (2D) decomposition does not exhibit any significant improvement over the one-dimensional (1D) decomposition (horizontal or vertical). This discrepancy may be partially due to the limited number of processors utilized in the current tests. Additional experiments with a larger number of processors will be conducted to more accurately assess the true scalability and speedup of the solver.

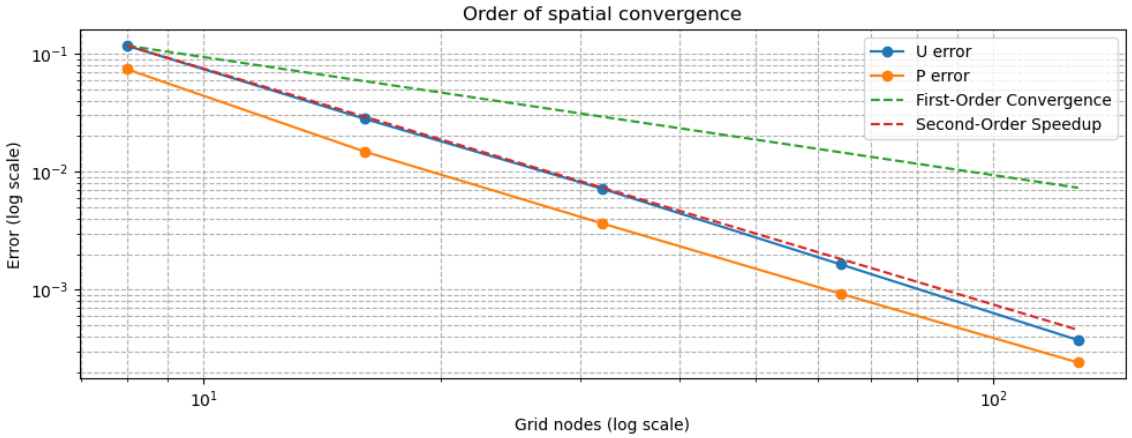

Figure 7: Spatial convergence with $\Delta t = 10^{-4}$

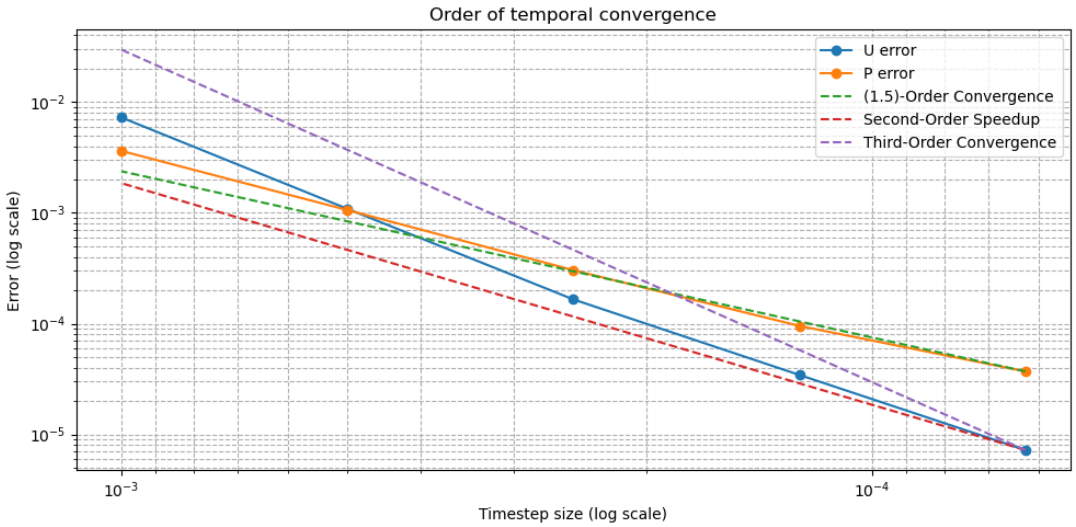

Figure 8: Temporal convergence with grid nodes $n = 32$

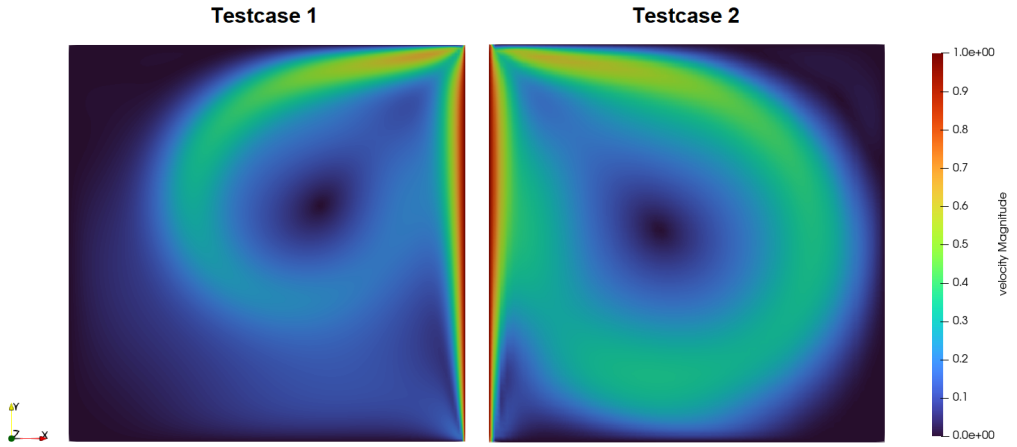

Figure 9: X-Y slice at $z = 0$ of testcases domain result

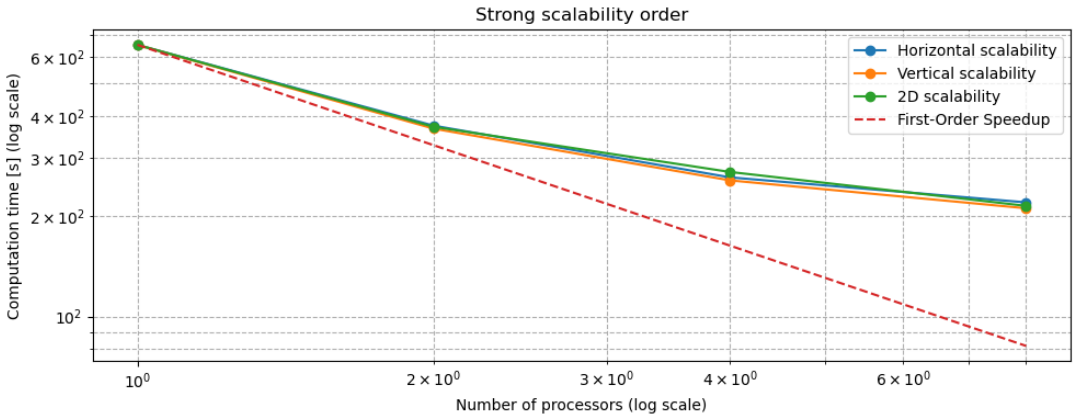

Figure 10: Strong scalability

References

[1] A. J. Chorin. Numerical solution of the navier-stokes equations. *Mathematics of Computation*, 22:745–762, 1968.

[2] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[3] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[4] N. Li and S. Laizet. 2decomp&fft - a highly scalable 2d decomposition library and fft interface. 2010.