UNIVERSITY
OF WARSAW

Introduction to Computational Fluid Dynamics (CFD) simulations using OpenFOAM

Student Team Project

Summer Semester 2023

Dawid Woś
Joydeep Sarkar
Mark Passia
Mateusz Kapusta

Supervised by
Rishabh P. Sharma

Contents

1 Introduction

OpenFoam is a Computational Fluid Dynamics (CFD) framework designed for the simulations of fluids/continuum mechanics, developed initially at Imperial Collage London [1]. Since the beginning, this software has been popularized as a open and easy to extend framework suitable for different variety of problems, involving diffusion, multiphase fluids and Magnetohydrodynamics (MHD). It is written in C++ language and utilizing modern features like object oriented programing (OOP). In this project a basic introduction to the CFD using OpenFoam is given. The structure of the report is the following. In the first part, we give a brief description about the physics behind the equations used in OpenFOAM. The 2nd part talks about the governing equation, i.e the transport equation and its properties. In the 3rd part, we showcase the special simulation case that is the Couette flow and finally we end with the report with discussions about the simulations results we obtained in this project report.

2 Advection-diffusion equation

2.1 Physics

We start with the physics behind a advection-diffusion equation and can be handled within OpenFOAM, an open-source computational fluid dynamics (CFD) software [2]. At the end of this chapter reader should be able to reproduce this simple example from the beginning.

$$D_T \nabla^2 T + \frac{\partial T}{\partial t} = v \cdot \nabla T, \tag{1}$$

where:

- T is the scalar field (e.g., temperature, concentration)
- D_T is the diffusion coefficient
- v is the velocity field

Diffusion

The term $D_T \nabla^2 T$ represents the diffusion process, where the scalar field T spreads out over space. This term is crucial in modeling how substances like heat or chemical concentrations diffuse through a medium.

Advection

The term $\mathbf{v} \cdot \nabla T$ represents the advection of T due to the fluid flow. If T represents temperature, this term captures how the fluid's motion transports heat within the system.

Time Evolution

The time derivative term $\frac{\partial T}{\partial t}$ describes how the scalar field T changes over time due to both diffusion and advection.

2.2 Solver

We also need to modify the equations in the main solver body.

```
phi = fvc::flux(U);
fvScalarMatrix TEqn
(
    fvm::ddt(T)
    +fvm::div(phi,T)
    -fvm::laplacian(DT, T)
    ==
    fvOptions(T)
);
```

Two of those options combined are enough to create a new solver for the OpenFoam. We also need to add additional term in the `system/fvSchemes` file as we need a scheme for a divergence. While diffusion equation is parabolic in nature, after including transport element it becomes hyperbolic in nature. In order to solve the problem, one defines **fluxes**, which are denoted as ϕ. In the discretization scheme we are once again defining cells, with centers and surfaces. While **u** is a vector field defined in the **center** of the cells, ϕ is defined as the $\phi_i = u_i S_i$, where u_i is a interpolated value of i-th component of the velocity, S_i is a surface of the cell. Basic idea involves using the values defined at surface instead of those defined at centers to evaluate the derivative $\frac{\partial}{\partial x}$, which makes the scheme more robust. While **u** has 3 components, ϕ has 8 defined at each of centers of the cell's surfaces. In OpenFoam there is clear distinction between the objects:

```
volVectorField U
(
IOobject
(
    "U",
    runTime.timeName(),
    mesh,
    IOobject::MUST_READ,
    IOobject::AUTO_WRITE
),
mesh
);
surfaceScalarField phi
(
IOobject
(
    "phi",
    runTime.timeName(),
    mesh,
    IOobject::NO_READ,
```

```
        IOobject::NO_WRITE
    ),
    fvc::flux(U)
    );
```

Velocity field is defined as `volVectorField` while ϕ is `surfaceScalarField`. In the declaration of ϕ we demand not to write down the field and do not read it. Whole field is defined using `flux` method which tells OpenFoam to automatically tie the definition with **u**.

OpenFOAM includes solvers like 'scalarTransportFoam' to solve such equations, taking into account the effects of both diffusion and advection. For this we have different kinds of solvers. Some of which we can see below.

Predefined Solvers: OpenFOAM provides a wide range of predefined solvers tailored for different types of CFD problems, such as incompressible flows, compressible flows, multiphase flows, heat transfer, chemical reactions, and more. For example:

- simpleFoam is used for steady-state incompressible flow.
- pisoFoam is used for transient incompressible flow.
- icoFoam is used for laminar incompressible flow.

Customization: Users can modify existing solvers or develop new ones to suit specific needs, thanks to the flexibility of OpenFOAM's object-oriented C++ structure. Such as solvers like **marklaplacianFoam**, **joylaplacianFoam**, **davlaplacianFoam**.

joylaplacianFoam

The solver joylaplacianFoam is a modified solver based on the need. At the core, the solver consists of 4 files, createFields.H, joylaplacianFoam.C Make and phi.H. The file joylaplacianFoam.C is the of major significance and it's modification is done based on what we need the solve to do. It looks like the following.

```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"

    #include "createTime.H"
    #include "createMesh.H"

    simpleControl simple(mesh);

    #include "createFields.H"
    #include "phi.H"
    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nCalculating temperature distribution\n" << endl;
```

```
    while (simple.loop(runTime))
    {
        Info<< "Time = " << runTime.userTimeName() << nl << endl;

        fvModels.correct();

        while (simple.correctNonOrthogonal())

        {
            fvScalarMatrix TEqn
            (
                fvm::ddt(T) - fvm::laplacian(DT,T) + fvc::div(phi,T)
             ==
                fvModels.source(T)
            );

            fvConstraints.constrain(TEqn);
            TEqn.solve();
            fvConstraints.constrain(T);
        }

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}
```

Another file(matlaplacianFoam.C) trying to execute the same function in other machine.

```
#include "fvCFD.H"
#include "fvOptions.H"
#include "simpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    argList::addNote
    (
```

```
        "Laplace equation solver for a scalar quantity."
    );

    #include "postProcess.H"

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"
    simpleControl simple(mesh);

    #include "createFields.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nCalculating temperature distribution\n" << endl;

    while (simple.loop())
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {

            //#include "phi.h"
//phi = fvc::flux(U);
            fvScalarMatrix TEqn
            (
                fvm::ddt(T)
               +fvm::div(phi,T)
               -fvm::laplacian(DT, T)
             ==
                fvOptions(T)
            );

            //fvVectorMatrix Ueqn
            //(
            //    fvm::ddt(U)
            //   +fvm::div(U)
            // ==
            //    fvOptions(U)
            ///)
            ///

            TEqn.relax();
```

```
            fvOptions.constrain(TEqn);
            TEqn.solve();
            fvOptions.correct(T);
        }

        #include "write.H"

        runTime.printExecutionTime(Info);
    }

    Info<< "End\n" << endl;

    return 0;
}
```

In both the above code, the following section is used to represent the advection-diffusion equation for a scalar field T:

```
fvm::ddt(T) - fvm::laplacian(DT, T) + fvc::div(phi, T)
```

Explanation of Each Term

Transient Term (`fvm::ddt(T)`):

$$\frac{\partial T}{\partial t} \tag{2}$$

This term represents the time derivative of the scalar field T, indicating how T changes over time.

Diffusion Term (`- fvm::laplacian(DT, T)`):

$$-\nabla \cdot (D_T \nabla T) \tag{3}$$

Here, D_T is the diffusion coefficient, and this term represents the diffusion process, where the scalar field T spreads out over space.

Advection Term (`+ fvc::div(phi, T)`):

$$\nabla \cdot (\phi T) \tag{4}$$

This term represents the advection or convection of T due to fluid flow. ϕ is related to the flow's velocity field, carrying T along with it.

2.3 Numerical Schemes

After determination of the grid and equation, we need to translate differential equation to our grid. There are may different types of challanges associated with the choice of numerical schemes. All properties associated with schemes are defined in `system/fvSchemes` file. A simple example of such directory may look like this:

```
FoamFile
{...
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear orthogonal;
    laplacian(DT,T) Gauss linear corrected;
}

divSchemes
{
    default none;
    div(phi,T) Gauss <keyword>
}


interpolationSchemes
{
    default linear;
}
```

Let's start with `ddTSchemes`. This entry specifies how a time derivative is translated. In case of simple Euler scheme we are approximating

The time derivative $\frac{\partial T}{\partial t}$ is handled using time-stepping schemes, such as implicit or explicit methods, to ensure numerical stability and accuracy.

$$\frac{\partial X}{\partial t} \approx \frac{X_{i+1} - X_i}{\Delta t}, \tag{5}$$

where Δt is a timestep of our simulation. There are few other schemes that might be used such as a Crank-Nicolson time scheme. In similar manner other entries control how other part of equation are handled. Let's take a closer look at `gradSchemes`. A Gauss keyword

states, that a Gauss theorem is used to compute

$$\boldsymbol{\nabla} U = \frac{1}{V} \sum_f \mathbf{S_f} U_f, \tag{6}$$

where V stands for the volume of the cell, $\mathbf{S_f}$ stands for the normal vector to the face f and U_f is a *approximate* value of field U at face f. This is where the second keyword comes in, as `linear` specifies how U_f is beeing calculated. In this case we are using linear approximation and assuming mean value of U field in two neighbouring cells separated by face f. There are different schemes that can be used here, some can be field-dependent as a `upwind` scheme or can be adjusted with additional parameters like `LimitedLinear`.

2.4 Simulation setup

2.4.1 Defining the Domain

In OpenFOAM, the ‘blockMeshDict‘ file is a critical component that defines the computational domain. This file is part of the ‘blockMesh‘ utility, which generates a structured hexahedral mesh for the domain in which equations like the diffusion-advection equation are solved.

The Role of blockMeshDict

The ‘blockMeshDict‘ file describes the geometry and the mesh structure of the computational domain. It is located in the ‘constant/polyMesh‘ directory of your OpenFOAM case. This file contains all the necessary information to create a mesh, including the domain size, the number of cells, and the boundary patches.

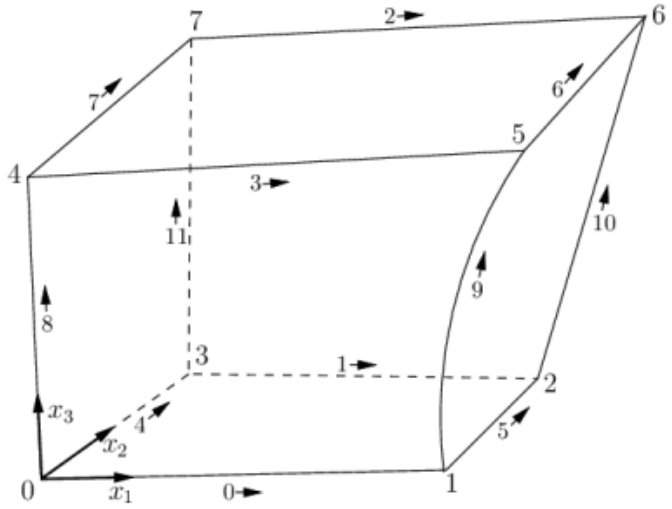

Figure 1: Block of a domain [3]

Key Components

Vertices

Vertices are the corner points of the blocks that define the geometry of the domain. The coordinates of these vertices are specified in the ‘vertices‘ section of the ‘blockMeshDict‘ file.

```
vertices
(
    (0 0 0)   // Vertex 0
    (1 0 0)   // Vertex 1
    (1 1 0)   // Vertex 2
    (0 1 0)   // Vertex 3
    (0 0 1)   // Vertex 4
    (1 0 1)   // Vertex 5
    (1 1 1)   // Vertex 6
    (0 1 1)   // Vertex 7
);
```

Blocks

Blocks are the hexahedral volumes that make up the computational domain. This can be seen as in figure 1 [3]. The ‘blocks‘ section of the ‘blockMeshDict‘ file defines these volumes by connecting the vertices.

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (10 10 10) simpleGrading (1 1 1)
);
```

In the example above, the block connects vertices 0 to 7 to form a cube, and it is subdivided into $10 \times 10 \times 10$ cells.

Edges

In some cases, edges between vertices may be curved. The ‘edges‘ section allows you to specify these curved edges.

Boundary Patches

Boundary patches define the boundaries of the domain where specific conditions (e.g., inlet, outlet, wall) are applied. These patches are crucial for solving equations like the diffusion-advection equation because they define how the scalar field T interacts with the domain boundaries.

```
boundary
{
```

```
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (1 2 6 5)
        );
    }
    wall
    {
        type wall;
        faces
        (
            (0 1 5 4)
            (2 3 7 6)
            (3 2 1 0)
            (4 5 6 7)
        );
    }
};
```

2.4.2 Mesh generation

In OpenFOAM, the geometry of the domain is defined using mesh files. For a simple heat diffusion simulation, we create the mesh using the `blockMesh` utility, which reads the mesh description from a file named `blockMeshDict` located in the `system` directory. In which both the geometry and the mesh of the simulated object are defined. Here are the keywords used in the exaple blockMeshDict file[1]: In our project we have used some of the exampular cases of running this command - blockMesh and the resultant visualization is done via paraview. We have visualized cases of pipe, cylinder and sphere. The advanced case of pipe, in terms of couette flow is something which we would see in section 3.2

[1]It is also possible to simulate in OpenFOAM using third-party CAD models. This can be done by converting the CAD models into a suitable mesh format using tools like `snappyHexMesh` or third-party software such as `gmsh`.

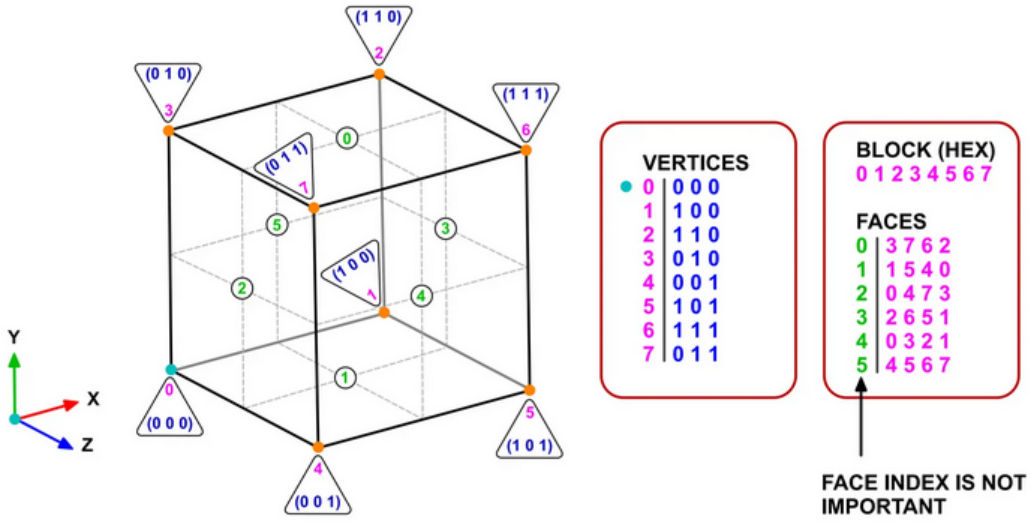

Figure 2: Relating block with that of the Mesh Structure [4]

2.4.3 Pipe

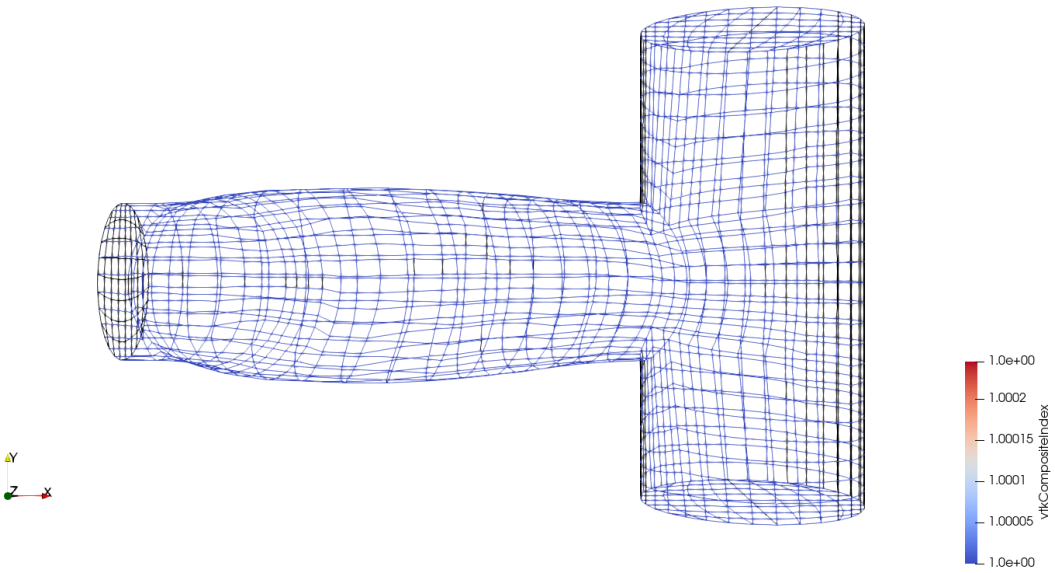

Figure 3:

2.4.4 Sphere

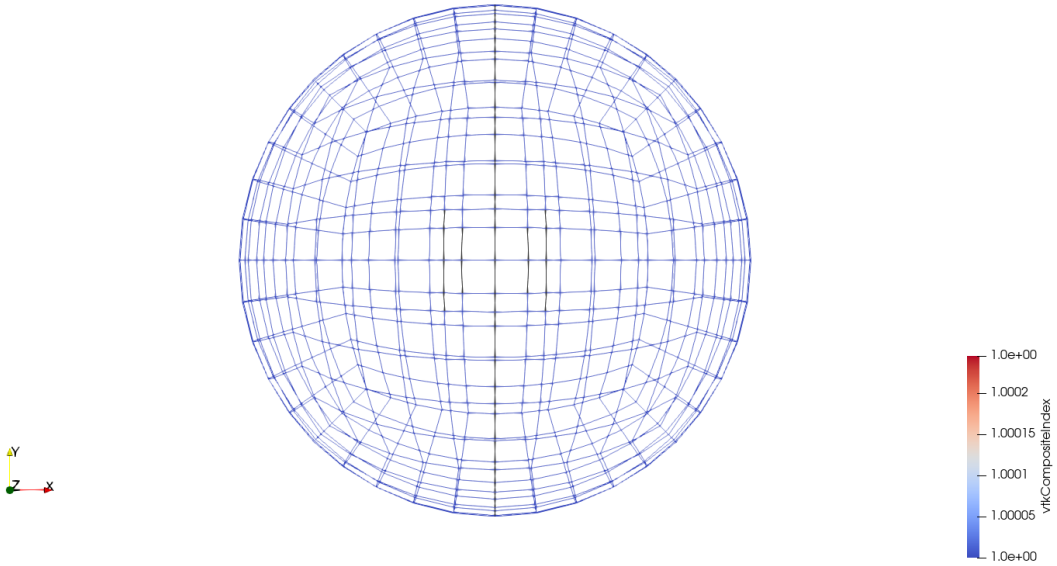

Figure 4:

2.4.5 Cylinder

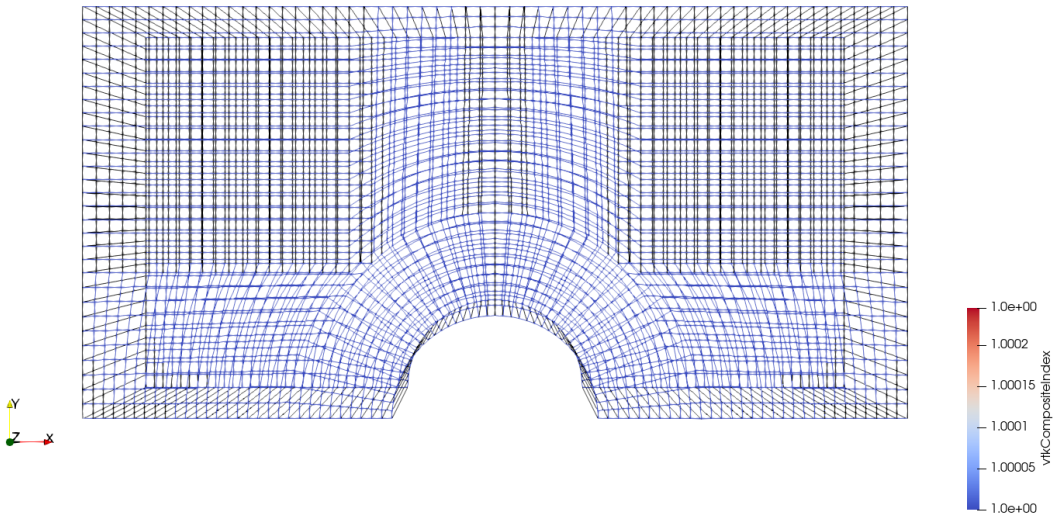

Figure 5:

2.4.6 Boundary Conditions

Appropriate boundary conditions are set for T to accurately represent the physical problem being modeled. These conditions might include fixed values, zero gradients, or more complex conditions.

```
dimensions      [0 0 0 1 0 0 0];
internalField   uniform 273;
boundaryField
{
    top{
        type            fixedValue;
        value           uniform 573;
    }

    bottom{
        type            fixedValue;
        value           uniform 273;
    }

    left{
        type            zeroGradient;
    }
    (...)
}
```

- **dimensions**: This line defines the physical dimensions of the field variable. The vector [0 0 0 1 0 0 0] indicates that the variable has the dimensions of temperature.
  - The dimensions in the `dimensions` entry of the OpenFOAM file are specified in the following order $[M\ L\ T\ \Theta\ N\ I\ J]$, where:
    - M is mass
    - L is length
    - T is time
    - Θ is temperature
    - N is quantity of substance
    - I is electric current
    - J is luminous intensity
- **internalField**: This defines the initial condition for the entire domain.
  - **uniform 273**: Sets the initial temperature to a uniform value of 273 Kelvin throughout the domain.
- **boundaryField**: This section specifies the boundary conditions for the field variable at the boundaries of the domain.

- **top**:
  - **type fixedValue**: Specifies that the boundary condition is a fixed value.
  - **value uniform 573**: Sets the temperature at the top boundary to a uniform value of 573 Kelvin.
- **bottom**:
  - **type fixedValue**: Specifies that the boundary condition is a fixed value.
  - **value uniform 273**: Sets the temperature at the bottom boundary to a uniform value of 273 Kelvin.
- **left, right, front, back**:
  - **type zeroGradient**: Specifies a zero gradient boundary condition, meaning there is no change in temperature across these boundaries. This effectively makes these boundaries insulating or adiabatic, preventing heat flux through them.

2.4.7 Physical properties

Every simulation of a physical object requires inputting the material properties of the simulated object, be it a fluid or a solid. In OpenFOAM the material properties for the simulation are defined in the `constant/transportProperties` file. In our case the only variable which can be changed in that file is thermal diffusivity, abbreviated as `DT`. It is a measure of how quickly temperature changes occur within a material when it is subjected to a thermal gradient. The corresponding file structure is as follows:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}

DT              0.01;
```

3 Simulation results

This section consists of various tests and cases, some with analytical solutions. Let's start with geometry of block presented in Figure 6. Upper side of the block is kept at 573 K while lower part is kept at 273 K. We also include "wind" in the Y direction that will slightly modify our solution. We assume, that the wind is uniform in whole block. Without the wind one expects steady state that

$$\nabla^2 T = 0, \tag{7}$$

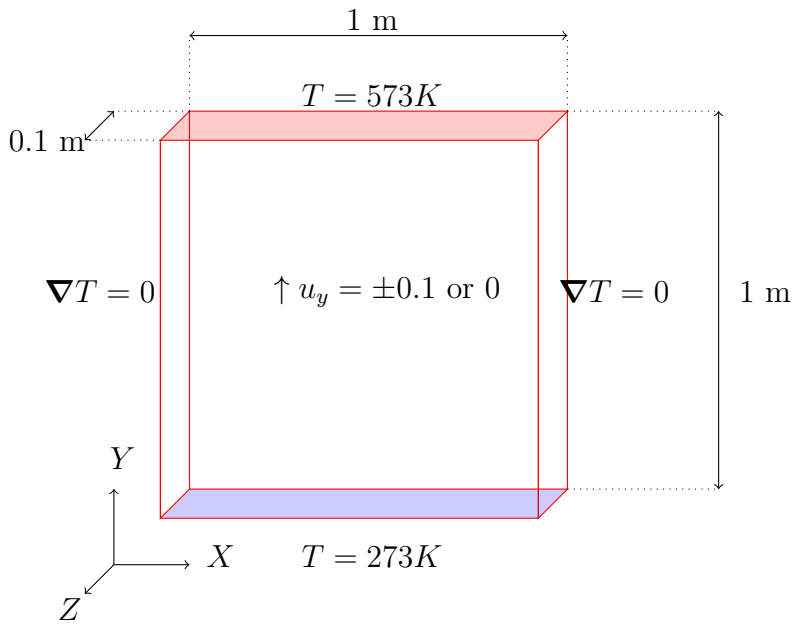

Figure 6: Diagram of the block used in the simulations

which in our case gives us linear rise from lower to upper value of temperature. This image slightly changes if we include wind. We have than

$$\frac{\partial^2 T}{\partial y^2} + \frac{u_y}{D_T}\frac{\partial T}{\partial y} = 0. \tag{8}$$

This solution has slightly different form with exponential rise/fall, with characteristic length $\lambda = \frac{D_T}{u_y}$.

Three separate cases are considered in total: no wind case and two cases with moderate velocity up and down the hill. In our simulations we assumed $D_T = 0.01$ and $u_y = \pm 0.1$ in wind cases. In order to simulate cases PBiCGStab solver with DILU preconditioner were used. All numerical schemes used simple linear interpolations together with Euler time scheme. Comparison between numerical and analitical solutions is presented in Figure 7. Paraview visulization of upwind case is presented in Figure 8. In case of zero velocity we are recovering simple linear form. In case of positive value of flow we are getting very steep temperature profile as the velocity "blocks" flow of the temperature. On the other hand if velocity is in other direction velocity is "spreading" temperature. In all cases we are getting around 0.5 K error despite only using $N = 51$ cells in Y direction.

In our case, we applied the transport + diffusion equation to a simple case of a 3d block and based on the parameters under volVectorField U, surfaceScalarField phi and flux, we run the simulations. The resultant simulation based on the choosen time steps can be seen as in the next page.

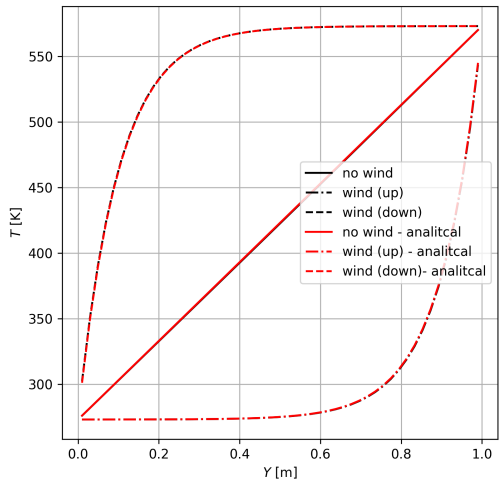

Figure 7: Comparison of velocity profile with and without velocity field (up stands for the positive u_y).

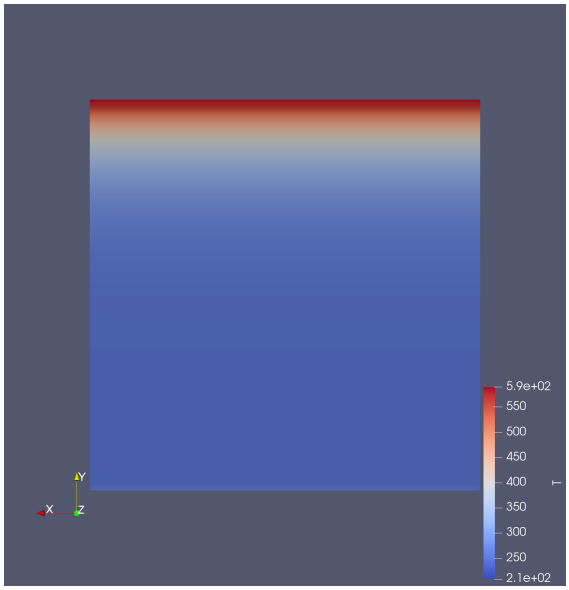

Figure 8: Temperature profile in the Y direction after reaching steady state with nonzero upflow velocity.

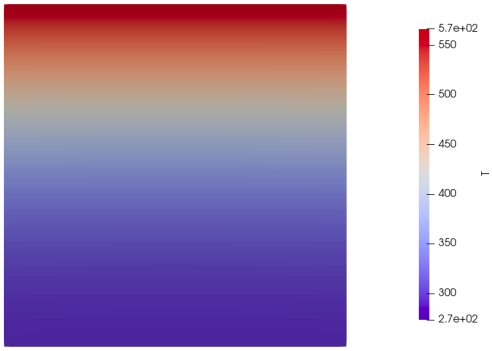

Figure 9: 100 s

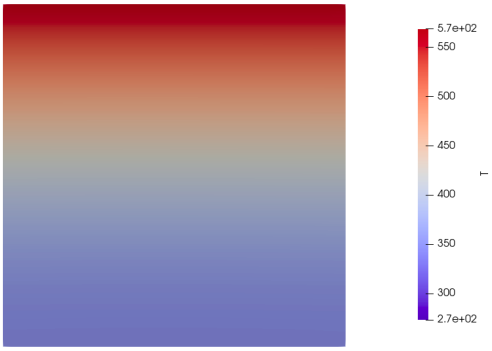

Figure 10: 200 s

Figure 9, 10, 11 and 12 showcases the variation of temperature across the cube, as governed by the transport and diffusion equation. We can see how with increasing time steps, the temperature finally saturates throughout the surface as shown in figure 12.

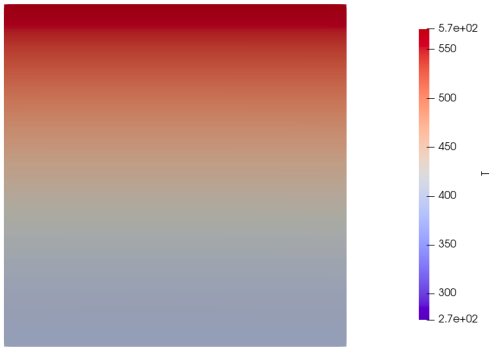

Figure 11: 300 s

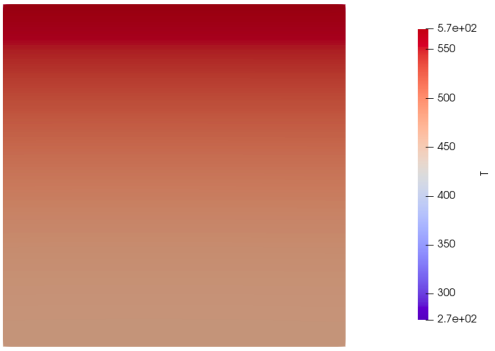

Figure 12: 500 s

4 Couette flow

4.1 Physics

Now let's move to the Coutte flow in pipe. Here, a viscous laminar flow will be considered with incompressible solver called `icoFoam`. There are two variables $\mathbf{u}$ and p and two equations

$$\boldsymbol{\nabla} \cdot \mathbf{u} = 0, \tag{9}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} = \nu \nabla^2 \mathbf{u} - \boldsymbol{\nabla} p. \tag{10}$$

$$\tag{11}$$

Here, pressure p represents in fact pressure divided by the density (pressure density). Incompressible flow is a good approximation as long as considered velocities are smaller than local speed of sound, which is important to keep in mind. We are considering flow in pipe, which is purely in Z direction. Moreover, we are assuming flow is symmetrical and hence depends only on r variable. It's possible to prove in such case, that velocity profile inside pip should be parabolic. Here is where the things get a little bit tricky. Usually in such cases we are assuming, that we have translational symmetry. This usually can be realized in case of very long system. Here, we can employ a different approach. OpenFoam provides us with a **cyclic** boundary condition. Those allow for realizing various for of translational boundaries.

4.2 Simulation setup

First we need to start with `blockMeshDict`.

```
boundary
(
    inlet
    {
        type  cyclic;
```

```
            neighbourPatch outlet;
            faces
            (
                <faces>
            );
        }
        outlet
        {
            type  cyclic;
            neighbourPatch inlet;
            faces
            (
                <faces>
            );
        }
    <pipe>
    )
```

We are writing down a different type of boundary called `cyclic`. We also need to specify boundary condition in our fields, here in case of velocity **u**

```
boundaryField
{
    pipe
    {
        type noSlip;
    }
    inlet
    {
        type cyclic;
        neighbourPatch outlet;
        transform translational;
    }
    outlet
    {
        type cyclic;
        neighbourPatch inlet;
        transform translational;
    }
}
```

This guaranties that our case will have translational symmetry and hence we do not need a long pipe.

`icoFoam` solvers is a little bit more elaborate than laplacian solver and detailed description of it is beyond the scope of the report. As it is one of most popular solvers available, we won't delve into the details of `fvSchemes` and `fvSolution` files. Unfortunately fluid simulations

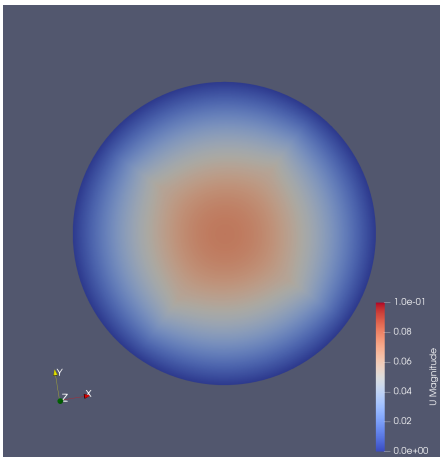

Figure 13: Paraview rendering of velocity in block.

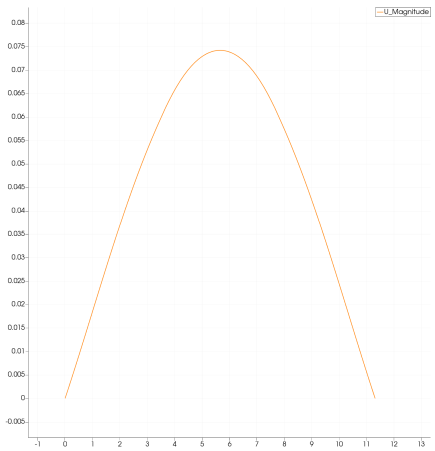

Figure 14: Velocity profile inside the pipe.

are quite slow compared to the simple laplacian solver so simulations were accelerated using OpenFoam MPI interface.

4.3 Results

In simulations, viscosity $\nu = 0.01$ was used with initial uniform velocity profile, with velocity $u_z = 0.1$ m/s. After 400 s final velocity profile was plotted and is presented with Paraview block rendering in Figure 13 and Figure 14. One can clearly see velocity profile is parabolic in nature just like the way the equation predicts.

5 Discussion

The simulations conducted using OpenFOAM provided valuable insights into the behavior of fluid flow under different conditions. Two main cases were considered: a 3D block for advection-diffusion simulation and a Couette flow in a pipe for incompressible flow analysis. The results obtained align well with theoretical predictions, demonstrating the effectiveness of OpenFOAM in handling complex CFD problems.

- Advection-Diffusion Simulation: The simulation results for the advection-diffusion equation in a 3D block highlight the influence of velocity on temperature distribution. Without the wind, a steady-state linear temperature gradient was observed, which is expected due to the direct diffusion from the hot to the cold boundary. This is consistent with the theoretical behavior of advection-dominated flows, where the wind can either enhance or oppose the natural diffusion process.

- Couette Flow in Pipe: The Couette flow simulation using the icoFoam solver successfully demonstrated the development of a parabolic velocity profile, a hallmark of

laminar viscous flow between two surfaces. The use of cyclic boundary conditions effectively modeled the translational symmetry, allowing the simulation to achieve realistic steady-state behavior in a relatively short pipe length.

-

References

[1] © OpenCFD Ltd 2024. OpenFoam. https://www.openfoam.com/, 2024.

[2] University of Washington. Physics Across Oceanography: Fluid Mechanics and Waves. https://uw.pressbooks.pub/ocean285/chapter/advection-diffusion-equation/, 2020.

[3] © OpenCFD Ltd 2024. 4.3 Mesh generation with the blockMesh utility . https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.3-mesh-generation-with-the-blockmesh-utility, 2024.

[4] wolfdynamics. blockMesh. http://www.wolfdynamics.com/wiki/meshing_OF_blockmesh.pdf, 2016.