Tutorial: BwUniCluster 3.0/HoreKa
Large Eddy Simulation (LES) in OpenFOAM

Course material developed at SCC
Scientific Centre for Computing
Karlsruhe Institute of Technology

Contents

1 Introduction

In this tutorial, we will learn how to set up a Large Eddy Simulation (LES) case in OpenFOAM uisng pimpleFoam solver. The tutorial covers essential aspects including inlet boundary condition specification, numerical discretization considerations, and post-processing techniques.

2 Case Setup for pimpleFoam

2.1 Directory Structure

The simulation parameters in pimpleFoam are configured through a collection of text files organized in a directory structure called a simulation case, which defines all aspects of the numerical setup including initial/boundary conditions (0/), physical properties (constant/), and solver controls (system/).

A typical pimpleFOAM case has the following directory structure:

```
case/
|-- 0/                          # Initial and boundary conditions
|   |-- U                       # Velocity field
|   |-- p                       # Pressure field
|   |-- (other files: k, nut, etc.) # Turbulence fields (k, epsilon, etc.)
|-- constant/                   # Mesh and physical properties
|   |-- polyMesh/               # Mesh files
|   |-- transportProperties     # Fluid properties
|-- system/                     # Solver settings and controls
    |-- controlDict             # Time control and output options
    |-- fvSchemes               # Discretization schemes
    |-- fvSolution              # Solution methods and tolerances
```

2.2 Initial and Boundary Conditions (0/ Directory)

The `0/` directory contains files that define the initial and boundary conditions for the simulation. For a pimpleFOAM case, you typically need to define conditions for velocity (`U`), pressure (`p`), and turbulence quantities if applicable.

2.2.1 Velocity Field (U)

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version      2.0;
format       ascii;
class        volVectorField;
object       U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
```

```
inlet
{
type              fixedValue;
value             uniform (0 0 1);
}
outlet
{
type              zeroGradient;
//use advective for LES -> See: https://doc.openfoam.com/2306/tools/processing/
    boundary-conditions/rtm/derived/outlet/advective/}
}
walls
{
type              noSlip;

//or:
//fixedValue;
//value       uniform (0 0 0);
}
// For other boundaries like cyclic, symmetry, etc.
}
// ************************************************************************* //
```

2.2.2 Pressure Field (p)

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version      2.0;
format       ascii;
class        volScalarField;
object       p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions        [0 2 -2 0 0 0 0];
internalField     uniform 0;
boundaryField
{
inlet
{
type              zeroGradient;
}
outlet
{
type              fixedValue;
value             uniform 0;  //or: $internalField;
}
walls
{
type              zeroGradient;
}
// other boundaries like cyclic, symmetry, etc are available.
}
// ************************************************************************* //
```

2.2.3 Turbulence Properties (e.g. nut, k (and epsilon for k-epsilon) model)

If using a turbulence model, you would also need initial and boundary conditions for *nut* (turbulent viscosity), k (turbulent kinetic energy), ϵ (turbulent dissipation rate), etc.

Example *nut* file:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -1 0 0 0 0];
internalField   uniform 0.0;  //In dynamic LES models, nut will be calculated by
    the model, and initial and boundary values are not important.
boundaryField
{
    inlet
    {
        type            calculated;
        value           uniform 0;
    }
    outlet
    {
        type            calculated;
        value           uniform 0;
    }
    walls
    {
        type            calculated;
        value           uniform 0;
}
```

Example k file (It is subgrid-scale (k_{SGS}) in LES)

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0.0;  // Set a small value for it in LES. But If you use
    RANS, set it based on desired turbulence intensity.
boundaryField
{
    inlet
```

```
    {
        type            fixedValue;
        value           uniform 0.0;  // Set a small value for it in LES. But If
    you use RANS, set it based on desired turbulence intensity.
    }
    outlet
    {
        type            zeroGradient;
    }
    walls
    {
        type            fixedValue;
        value           uniform 0;

        // For RANS use wall functions:
        // type            kqRWallFunction;
        // value           uniform 0.375;
    }
}
```

Example *epsilon* file (We do not need it in LES models)

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -3 0 0 0 0];
internalField   uniform 0.0256;  // Based on k value and turbulence length scale
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.0256;  // Based on k value and turbulence length
     scale
    }
    outlet
    {
        type            zeroGradient;
    }
    walls
    {
        type            epsilonWallFunction; // Use wall functions.
        value           uniform 0.0256;
    }
}
```

2.3 Physical Properties (constant/ Directory)

2.3.1 Transport Properties

The `transportProperties` file defines the fluid properties such as kinematic viscosity:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
transportModel  Newtonian;
nu              0.0001;
// ************************************************************************* //
```

2.3.2 Turbulence Properties

If your simulation includes turbulence modeling, you need to define the turbulence properties in the `turbulenceProperties` file:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType      LES; // options: laminar, LES, RANS

LES
{
    LESModel            dynamicKEqn;

    turbulence          on;

    printCoeffs         on;

    delta               cubeRootVol; // (Vc)^(1/3)

    dynamicKEqnCoeffs
    {
        filter simple;
    }

    cubeRootVolCoeffs
    {
        deltaCoeff      1; // delta = deltaCoeff * (Vc)^(1/3)
    }
}
// ************************************************************************* //
```

2.3.3 polyMesh Directory

The `constant/polyMesh/` directory contains all the files that define the computational grid (mesh) for the simulation. The polyMesh directory typically contains the following files:

```
polyMesh/
|-- points              # Coordinates of all mesh vertices
|-- faces               # List of faces defined by point labels
|-- owner               # Owner cell labels for each face
|-- neighbour           # Neighbour cell labels for each face
|-- boundary            # Boundary patch definitions
|-- (other files)       # Additional mesh information
```

There are several ways to generate the mesh for your OpenFOAM simulation:

blockMesh: OpenFOAM's built-in utility for creating simple parametric meshes. It's suitable for basic geometries such as channels, pipes, or rectangular domains. The mesh is defined in the `system/blockMeshDict` file:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112
   |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}

convertToMeters 0.25;

vertices
(
    (-0.65 0.65 0) //0
    (-1.414212 1.414212 0) //1
    (1.414212 1.414212 0) //2
    (0.65 0.65 0) //3
    (-0.65 0.65 25) //4
    (-1.414212 1.414212 25) //5
    (1.414212 1.414212 25) //6
    (0.65 0.65 25) //7

    (0.65 -0.65 0) //8
    (1.414212 -1.414212 0) //9
    (0.65 -0.65 25) //10
    (1.414212 -1.414212 25) //11

    (-0.65 -0.65 0) //12
    (-1.414212 -1.414212 0) //13
    (-0.65 -0.65 25) //14
    (-1.414212 -1.414212 25) //15
);

xcells 7;
ycells 7;
zcells 70;

xcells1 7;
```

```
ycells1 7;
zcells1 70;

stretch 1;

blocks
(
    //block0
     hex (0 3 2 1 4 7 6 5) ($xcells $ycells $zcells) simpleGrading (1 $stretch 1)
    //block1
    hex (3 8 9 2 7 10 11 6) ($xcells $ycells $zcells) simpleGrading (1 $stretch 1)
    //block2
    hex (8 12 13 9 10 14 15 11) ($xcells $ycells $zcells) simpleGrading (1
    $stretch 1)
    //block3
    hex (12 0 1 13 14 4 5 15) ($xcells $ycells $zcells) simpleGrading (1 $stretch
    1)
    //block4
    hex (0 12 8 3 4 14 10 7) ($xcells1 $ycells1 $zcells1) simpleGrading (1 1 1)
);

edges
(
    //block0 arc
    arc 1 2 (0 2 0)
    arc 5 6 (0 2 25)

    //block1 arc
    arc 2 9 (2 0 0)
    arc 6 11 (2 0 25)

    //block2 arc
    arc 9 13 (0 -2 0)
    arc 11 15 (0 -2 25)

    //block3 arc
    arc 1 13 (-2 0 0)
    arc 5 15 (-2 0 25)

    //block4 arc
    arc 0 3 (0 0.7 0)
    arc 0 12 (-0.7 0 0)
    arc 8 12 (0 -0.7 0)
    arc 8 3 (0.7 0 0)

    arc 4 7 (0 0.7 25)
    arc 4 14 (-0.7 0 25)
    arc 10 14 (0 -0.7 25)
    arc 10 7 (0.7 0 25)
);

boundary
(
    inlet
    {
    type patch;
    faces
    (
    (0 1 2 3)
    (2 3 8 9)
    (8 9 13 12)
    (13 12 0 1)
    (0 3 8 12)
```

```
    );
  }

  outlet
  {
    type patch;
    faces
    (
    (4 5 6 7)
    (6 7 10 11)
    (15 11 10 14)
    (15 14 4 5)
    (4 7 10 14)
    );
  }

  walls
  {
    type wall;
    faces
    (
    (1 5 6 2)
    (2 6 11 9)
    (9 11 15 13)
    (15 13 5 1)
    );
  }
);

mergePatchPairs
(
);
```

The mesh created by running the command:

```
$ blockMesh
```

snappyHexMesh: A more advanced OpenFOAM utility for generating complex 3D meshes based on triangulated surface geometries (STL files). It creates predominantly hexahedral meshes that conform to complex geometries through a series of operations including castellation, snapping, and layer addition. It's configured through the `system/snappyHexMeshDict` file and is typically run after blockMesh:

```
$ snappyHexMesh
```

External mesh generators: For complex geometries, it's often more convenient to use specialized external mesh generation software and convert the mesh to OpenFOAM format. OpenFOAM provides several conversion utilities:

`fluent3DMeshToFoam`: Converts Fluent .msh mesh files to OpenFOAM format

```
$ fluent3DMeshToFoam fluentMesh.msh
```

`cfx4ToFoam`: Converts CFX mesh files `gambitToFoam`: Converts Gambit mesh files `ideasUnvToFoam`: Converts I-DEAS .unv mesh files `star4ToFoam`: Converts STAR-CD mesh files `gmshToFoam`: Converts Gmsh mesh files

After generating the mesh, it's good practice to check its quality using:

```
$ checkMesh
```

This will report various mesh quality metrics such as non-orthogonality, skewness, and aspect ratio, which can help identify potential issues before running the simulation. Poor mesh quality can lead to numerical instability and inaccurate results.

2.4 Solver Settings (system Directory)

2.4.1 Control Dictionary (controlDict)

The `controlDict` file controls the simulation execution parameters:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     pimpleFoam;
startFrom       latestTime; // Options: startTime, latestTime, etc.
startTime       0;
stopAt          endTime;
endTime         100;
deltaT          0.02; // Time step size
writeControl    timeStep; // Options: timeStep, runTime, adjustableRunTime
writeInterval   10; // Write results every 10 time steps
purgeWrite      0; // Keep all time directories
writeFormat     ascii; // Options: ascii, binary
writePrecision  6;
writeCompression on;
timeFormat      general;
timePrecision   6;
runTimeModifiable false; // Allow modifications during run time

adjustTimeStep  yes;        // Enable adaptive time stepping
maxCo           0.4;        // Maximum Courant number

functions
{
    // Function objects for post-processing can be added here
}
// ************************************************************************* //
```

2.4.2 Finite Volume Schemes (fvSchemes)

The `fvSchemes` dictionary specifies the discretization schemes:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
object      fvSchemes;
}
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
    default         Euler; // First-order time derivative scheme: Euler and Second
    -order: backward or CrankNicolson <coeff>
}
gradSchemes
{
    default         Gauss linear;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linearUpwind grad(U);
    div(phi,k)      Gauss limitedLinear 1;
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
}
// ************************************************************************* //
```

2.4.3 Finite Volume Solution (fvSolution)

The `fvSolution` dictionary specifies solver settings, tolerances, and PIMPLE algorithm parameters:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.01;
        smoother        GaussSeidel;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 1000;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }

    pFinal
```

```
    {
        $p;
        smoother        DICGaussSeidel;
        tolerance       1e-06;
        relTol          0;
    }

    "(U|k)"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-06;
        relTol          0;
        minIter         1;
    }

    "(U|k)Final"
    {
        solver          PBiCG;
        preconditioner  DILU;
        tolerance       1e-06;
        relTol          0;
        minIter         1;
    }
}
PIMPLE
{
    nOuterCorrectors 1;
    nCorrectors      2;
    nNonOrthogonalCorrectors 0;
    // when you do not have a boundary with known pressure:
    // pRefCell         0;
    // pRefValue        0;
}
// ************************************************************************* //
```

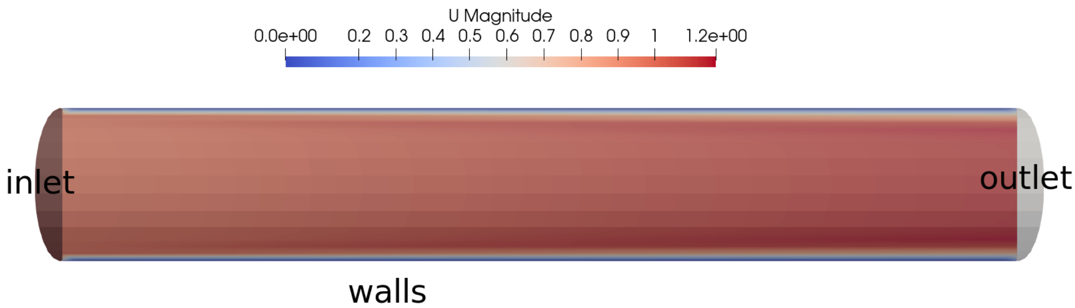

Figure 1: Result of *testCase − simple*.

We used an LES model, but we couldn't resolve any flow structures. Why?

3 Initial and Inlet Boundary Condition for LES

3.1 Importance of Turbulent Inlet Conditions

In Reynolds-Averaged Navier-Stokes (RANS) simulations, the effects of turbulence are modeled using turbulence models, which are based on empirical relationships between the mean flow properties and turbulence quantities. These models assume that the turbulence is statistically steady and homogeneous, which means that the turbulence structures do not vary significantly in space and time. As a result, generating turbulent structures at the inlet is not necessary in RANS simulations because the turbulence models are designed to simulate the averaged effects of turbulence on the mean flow.

In contrast, for Large Eddy Simulation (LES) or Direct Numerical Simulation (DNS) of fluid flows, it is important to accurately capture the turbulent structures present in the flow. In order to capture these turbulent structures, it is necessary to specify appropriate boundary conditions at the inlet of the computational domain. This is because turbulence is an unsteady and chaotic process, and the statistical properties of the turbulence vary in both space and time.

3.2 Approaches for Generation of Turbulent Fluctuations in OpenFOAM

3.2.1 Synthetic Turbulence Generation

Divergence-Free Synthetic Eddy Method (turbulentDFSEMInlet) is a velocity boundary condition including synthesized eddies for use with DNS, LES, and DES turbulent flows. It can be used as:

```
inlet
{
    type            turbulentDFSEMInlet;
    delta           1;                    // Characteristic length scale
    U               uniform (0 0 1);      // Mean velocity
    R               uniform (0.2 0 0 0.2 0 0.2); // Reynolds stress: <Rxx> <Rxy> <
    Rxz> <Ryy> <Ryz> <Rzz>
    L               uniform 0.4;          // Integral length scale
    nCellPerEddy    1;                    // Minimum eddy length in units of number
    of cells
    value           uniform (0 0 1);
}
```

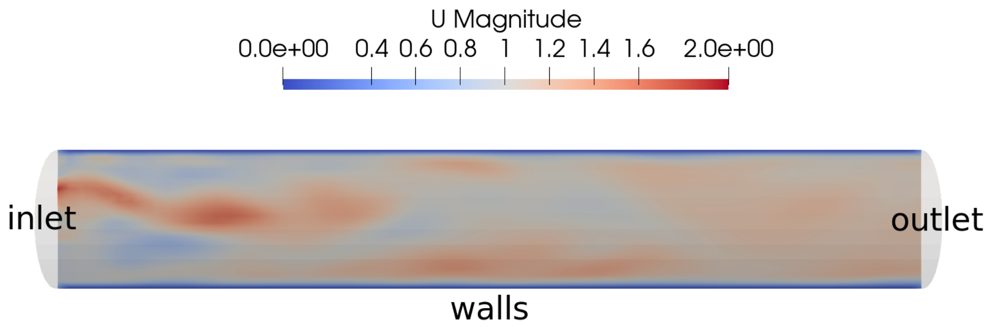

Figure 2: Result of $testCase - inflow - generator$.

It is possible to use a field for U, R, and L in turbulentDFSEMInlet. To do that, first use codedFixedValue to generate the velocity field in the inlet and write the data just for a time step. Then this generated field can be pasted in turbulentDFSEMInlet boundary condition. Below is an example of codedFixedValue boundary condition:

```
inlet
{
    type            codedFixedValue;
    value           uniform (0 0 0);
    name            myInlet;
    code
```

```
    #{
        scalar U_max = 2;
        const fvPatch& boundaryPatch = this->patch();
        const vectorField& Cf = boundaryPatch.Cf();
        vectorField& field = *this;
        forAll(boundaryPatch, i)
        {
            scalar r = sqrt(Cf[i].y()*Cf[i].y() + Cf[i].x()*Cf[i].x())/0.5;
            field[i] = vector(0, 0, U_max*Foam::pow(1.0-r, 1.0/7.0));
        }
    #};
}
```

One of the main problems of the turbulentDFSEMInlet is that it needs additional data for Reynolds stresses and integral length scale, which is not available in many cases. Moreover, the generated flow in the inlet is not completely physical.

3.2.2 Recycling-Method (Mapped Boundary Condition)

This approach involves extending the domain upstream and extracting turbulent velocities (and other fields if needed) from the interior domain.

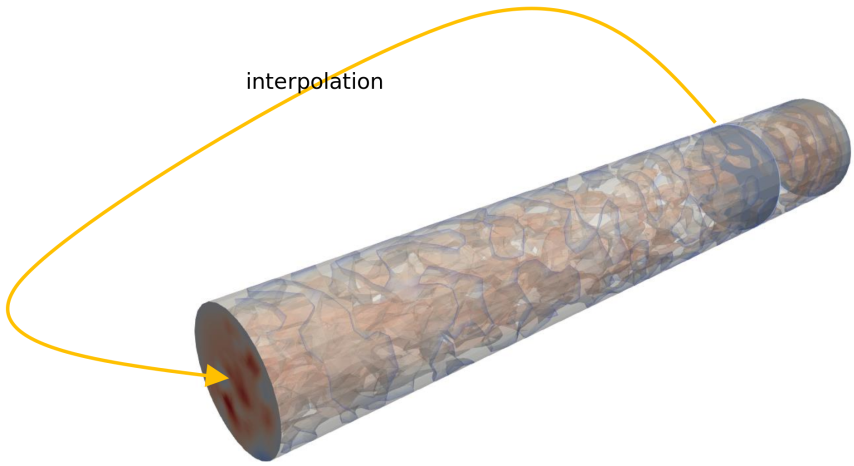

Figure 3: Schematic of the Recycling method for inlet boundary conditions.

To use this method, the boundary file in the polyMesh directory should be modified in the following way:

```
inlet
{
    type            mappedPatch;    // modified
    nFaces          245;
    startFace       50225;
    sampleMode      nearestCell;    // added
    samplePatch     none;           // added
    sampleRegion    region0;        // added
    offsetMode      uniform;        // added
    offset          (0 0 5);        // added
}
```

Then the boundary condition is set for U and k as below:

For velocity (U):

```
inlet
{
```

```
    type              mapped;
    value             uniform (1 0 0);
    interpolationScheme cell;
    setAverage        true;
    average           (1 0 0);
}
```

For turbulent kinetic energy (k):

```
inlet
{
    type              mapped;
    value             uniform 0.0;
    interpolationScheme cell;
    setAverage        false;
}
```

One benefit of using this approach is that it does not require any parameters. However, it is important to note that the internal field needs to be agitated initially, as otherwise, it may take a significant amount of time for turbulent structures to form. Therefore, a possible solution is to utilize the turbulentDFSEMInlet method to generate vortices throughout the pipe (with a rough estimation of R and L) before switching to the mapped boundary condition.

4 Numerical Dissipation in LES

Numerical dissipation in Large Eddy Simulation (LES) refers to the artificial damping of the resolved turbulent scales due to the discretization of the governing equations on a numerical grid. Numerical dissipation arises from the truncation error in the numerical scheme used to solve the equations, and can lead to a loss of accuracy in the resolved scales.

In LES, the resolved turbulent scales are computed on a grid with finite resolution, which means that small-scale turbulent structures cannot be fully resolved and must be modeled using subgrid-scale (SGS) models. The numerical dissipation in the LES model can cause additional damping of the resolved scales, which can impact the accuracy of the subgrid-scale models.

In OpenFOAM, there are several discretization schemes available for the solution of the Navier-Stokes equations, each with different levels of numerical dissipation and accuracy. The choice of discretization scheme depends on the specific flow problem and the desired level of accuracy. However, central differencing schemes (Linear) are less diffusive than the upwind schemes, but they can introduce numerical oscillations in regions with strong gradients.

An example of a suitable discretization for LES is shown below:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         backward;
```

```
}

gradSchemes
{
    default         leastSquares;   // "Gauss linear" is more stable
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;   // use "LUST" For low quality grids
    div(phi,k)      Gauss linear;   // use "limitedLinear" For low quality grids
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}
// ************************************************************************* //
```

To examine the effect of discretization in LES, you can apply the following changes in fvSchemes of testCase2:

```
div(phi,U) Gauss linear; -> div(phi,U) Gauss linearUpwindV grad(U);
```

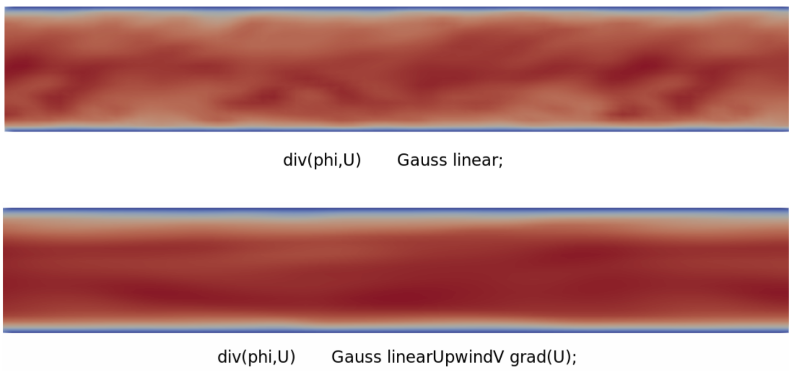

Figure 4: Effect of discretizations schemes

5 Post-processing

5.1 Field Averaging

The "fieldAverage" is a utility that is used to compute time-averaged scalar and vector fields from the transient data generated by OpenFOAM solvers. It can also compute the root-mean-square (RMS) values of the fluctuating components of the fields. The time-averaged fields can be used for further analysis, such as computing turbulence statistics, or for validation against experimental data.

To use this utility, the following code should be added in the controlDict:

```
functions
{
    myFieldAverage
    {
        type            fieldAverage;
        libs            (fieldFunctionObjects);
        writeControl    writeTime;
        fields
        (
            U
            {
                mean        on;
                prime2Mean  on;
                base        time;
            }
            p
            {
                mean        on;
                prime2Mean  on;
                base        time;
            }
        );
    }
}
```

5.2 Point Probes

The ”probes” utility in OpenFOAM is a diagnostic tool used to extract information about the flow field at a particular point or location during the simulation. It can be used to monitor the evolution of various flow parameters such as velocity, pressure, temperature, and turbulence at a given point or a set of points in the computational domain.

To use this utility, the following code should be added in the controlDict:

```
functions
{
    probes
    {
        type            probes;
        libs            (sampling);
        name            probes;
        writeControl    timeStep;
        writeInterval   1;
        fields
        (
            U
        );
        probeLocations
        (
            (0 0 5)
            (0.025 0 5)
            (0.05 0 5)
            (0.075 0 5)
            (0.1 0 5)
        );
    }
}
```

5.3 Surface Sampling

In OpenFOAM, the ”surfaces” utility can be used to perform surface sampling of various flow parameters such as velocity, pressure, and temperature on defined surfaces. To perform surface sampling using the surfaces utility, a user needs to first define the surface(s) of interest using a surface definition input. This definition specifies the location and geometry of the surface(s) in the computational domain.

To use this utility, the following code should be added in the controlDict:

```
functions
{
    cuttingPlane
    {
        type            surfaces;
        libs            (sampling);
        writeControl    timeStep;
        writeInterval   5;
        surfaceFormat   vtk;
        fields          ( U );
        interpolationScheme cellPoint;
        surfaces
        {
            zNormal
            {
                type            cuttingPlane;
                planeType       pointAndNormal;
                pointAndNormalDict
                {
                    point       (0 0 0);
                    normal      (0 1 0);
                }
                interpolate     true;
            }
        }
    }
}
```

5.4 Q–Criterion and Iso–Surface Sampling

The Q–criterion is a scalar field used to identify vortical structures in a flow. It is defined as

$$Q = \tfrac{1}{2}\left(\|\mathbf{\Omega}\|^2 - \|\mathbf{S}\|^2\right)$$

where

$$\mathbf{S} = \tfrac{1}{2}\left(\nabla\mathbf{U} + (\nabla\mathbf{U})^T\right) \quad \text{and} \quad \mathbf{\Omega} = \tfrac{1}{2}\left(\nabla\mathbf{U} - (\nabla\mathbf{U})^T\right)$$

are the rate–of–strain tensor and the vorticity tensor, respectively. Positive values of Q indicate regions where rotation dominates over strain (vortical regions).

5.4.1 Computing Q with fieldFunctionObjects

Add the following to your `controlDict` under the `functions` block to compute Q:

```
functions
{
    Q1
    {
        type            Q;
        libs            (fieldFunctionObjects);
        writeControl    writeTime;    // write Q at every saved time
        // Alternatively, use:
        // writeControl    timeStep;
        // writeInterval   1;
    }
}
```

5.4.2 Extracting an Iso–Surface of Q

To sample an iso–surface at $Q = 1.0$, extend your `controlDict` functions block:

```
functions
{
    QisoSurface
    {
        type            surfaces;
        libs            (sampling);
        writeControl    timeStep;
        writeInterval   5;
        surfaceFormat   vtk;
        fields          ( U p );
        surfaces
        {
            iso
            {
                type            isoSurface;
                isoField        Q;
                isoValue        1.0;
            }
        }
    }
}
```

This produces VTK files at

`postProcessing/QisoSurface/<time>/iso_Q_1.0000.vtk`

every 5 time–steps.

5.4.3 Making animation of Results

It is possible to utilize a Python script to create an animation of the output files that have been generated. The "vtkAnim.py" can be downloaded from the following link:
https://openfoamwiki.net/index.php/VtkAnim

5.5 Post-processing in Python

While OpenFOAM provides built-in utilities for post-processing, it is also possible to export simulation data into formats compatible with Python-based scientific computing libraries such as PyTorch.

The following custom C++ utility, `FoamToGraph`, reads the velocity field **U** and mesh connectivity from an OpenFOAM case, and converts the data into tensors suitable for python using the LibTorch (C++ version of PyTorch) API.

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2016 OpenFOAM Foundation
    Copyright (C) 2018-2021 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
```

```
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    FoamToGraph

Description
    FoamToGraph By H.Tofighian

\*---------------------------------------------------------------------------*/
#include <torch/torch.h>
#include <torch/script.h>

#include "argList.H"
#include "timeSelector.H"
#include "volFields.H"

#include <stdio.h>
#include <stdlib.h>

using namespace Foam;

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

int main(int argc, char *argv[])
{
    timeSelector::addOptions();

    #include "setRootCase.H"
    #include "createTime.H"
    instantList timeDirs = timeSelector::select0(runTime, args);
    #include "createNamedMesh.H"


    // Create output directory in case directory
    fileName outputDir(runTime.rootPath()/runTime.globalCaseName()/"graph_data");
    Foam::mkDir(outputDir);
    Info<< "Saving graph data to: " << outputDir << nl << endl;

    forAll(timeDirs, timei)
    {
        runTime.setTime(timeDirs[timei], timei);

        Info<< "Time = " << runTime.timeName() << endl;
        Info<< "Reading field U\n" << endl;
        volVectorField U
        (
            IOobject
            (
                "U",
                runTime.timeName(),
                mesh,
                IOobject::MUST_READ,
                IOobject::NO_WRITE
            ),
```

```
        mesh
    );

    const label nCells = mesh.cells().size();

    // Extract node features: use the velocity components (x, y, z) from each cell.
    std::vector<float> node_feat;
    node_feat.reserve(3 * nCells);
    forAll(U, i)
    {
        node_feat.push_back(U[i].x());
        node_feat.push_back(U[i].y());
        node_feat.push_back(U[i].z());
    }
    // Create a LibTorch tensor for node features with shape [nCells, 3].
    torch::Tensor node_features = torch::from_blob(node_feat.data(), {static_cast<long>(nCells), 3}, torch::kFloat32).clone(); // Shape: [num_nodes, 3]

    // Build edge indices using mesh connectivity.
    // In OpenFOAM, each internal face connects two cells. The mesh provides
    // owner and neighbour lists; we add both directions for an undirected graph.
    const labelList& owner = mesh.owner();
    const labelList& neighbour = mesh.neighbour();
    const size_t numFaces = owner.size();
    const size_t numEdges = 2*numFaces;

    // Separate lists for source and target indices.
    std::vector<int64_t> edge_sources;
    std::vector<int64_t> edge_targets;

    // Reserve space for two directed edges per face.
    edge_sources.reserve(numEdges);
    edge_targets.reserve(numEdges);

    forAll(owner, i)
    {
        // Forward edge: owner -> neighbour.
        edge_sources.push_back(owner[i]);
        edge_targets.push_back(neighbour[i]);
        // Reverse edge: neighbour -> owner.
        edge_sources.push_back(neighbour[i]);
        edge_targets.push_back(owner[i]);
    }
    // Create a tensor for edge indices; stack two vectors.
    torch::Tensor edge_index = torch::stack({
        torch::from_blob(edge_sources.data(), {static_cast<long>(numEdges)}, torch::kInt64),
        torch::from_blob(edge_targets.data(), {static_cast<long>(numEdges)}, torch::kInt64)
    }).clone(); // Shape: [2, num_edges]

    // Extract node positions: cell centers (x, y, z) for each cell
    std::vector<float> node_pos;
    node_pos.reserve(3 * nCells);

    // Get cell centers
    const pointField& cellCenters = mesh.C();

    // Add each cell center coordinate to the positions vector
    forAll(cellCenters, i)
```

```
        {
            node_pos.push_back(cellCenters[i].x());
            node_pos.push_back(cellCenters[i].y());
            node_pos.push_back(cellCenters[i].z());
        }

        // Create a LibTorch tensor for node positions with shape [nCells, 3]
        torch::Tensor node_positions = torch::from_blob(node_pos.data(), {
    static_cast<long>(nCells), 3}, torch::kFloat32).clone(); // Shape: [num_nodes,
     3]



        // Save tensors
        // Save node features
        {
            fileName nodeFile(outputDir/"node_features_" + runTime.timeName() + ".
    pt");
            auto node_bytes = torch::pickle_save(node_features);
            std::ofstream fout(nodeFile, std::ios::out | std::ios::binary);
            fout.write(node_bytes.data(), node_bytes.size());
            fout.close();
            Info<< "Saved node features to " << nodeFile << endl;
        }

        // Save edge indices
        {
            fileName edgeFile(outputDir/"edge_index_" + runTime.timeName() + ".pt"
    );
            auto edge_bytes = torch::pickle_save(edge_index);
            std::ofstream fout(edgeFile, std::ios::out | std::ios::binary);
            fout.write(edge_bytes.data(), edge_bytes.size());
            fout.close();
            Info<< "Saved edge indices to " << edgeFile << endl;
        }
        // Save node positions
        {
            fileName posFile(outputDir/"node_positions_" + runTime.timeName() + ".
    pt");
            auto pos_bytes = torch::pickle_save(node_positions);
            std::ofstream fout(posFile, std::ios::out | std::ios::binary);
            fout.write(pos_bytes.data(), pos_bytes.size());
            fout.close();
            Info<< "Saved node positions to " << posFile << endl;
        }
    }

    Info << "Execution complete." << nl;

    return 0;
}


// ********************************************************************** //
```