Universitat Politècnica de Catalunya
The School of Industrial Engineering of Barcelona
Department of Heat Engines

Research Assignment

Study of fluid-solid coupling for complex geometries in OpenFOAM

Katarzyna Połczyńska

Supervisor
Elisabet Mas de les Valls Ortiz

Barcelona, 2023

Abstract: „ Study of fluid-solid coupling for complex geometries in OpenFOAM”

The focus of this research will be the modelling of heat transfer with fluid-solid coupling for complex geometries in OpenFOAM. In the initial stage, selected aspects of the OpenFOAM toolkit will be introduced, including its fundamentals and usage. Afterwards, the available tools to model heat transfer in a solid wall in contact with a channel of a complex cross-section, through which the fluid flows will be explored. The outcome of this work will be an algorithm that could guide users in simulating a similar case.

Key words: OpenFOAM, heat transfer, fluid-solid coupling, complex geometry

Table of content

1. Introduction and scope of work 4
2. Fundamentals: Description of computational models 5
   2.1. Equations for solid 5
   2.2. Equations for fluid 5
   2.3. Boundary conditions 7
   2.4. Coupling between fluid and solid 7
3. Domain discretization 8
   3.1. Simple meshing (*blockMesh*) 8
   3.2. Multiregional meshing 9
4. Preliminary case: three solids 12
   4.1. Case setup 12
   4.2. Results 14
5. Final case: fluid-solid coupling 16
   5.1. Case setup in OpenFOAM 16
   5.2. Results 19
6. Conclusions 22
7. Bibliorgraphy 23
8. Appendix 24
   8.1. Preleminary case 24
   8.2. Final case 28

1. Introduction and scope of work

The aim of this work is to investigate and develop a comprehensive understanding of the coupled behaviour between the solid and fluid domains. To achieve this, the mathematical interpretation of physical phenomena is presented for both the solid and fluid domains. The coupling of these domains and the identification of necessary boundary conditions are thoroughly discussed, establishing a solid foundation for subsequent analyses.

Building upon the theoretical groundwork, the focus shifts towards exploring the available tools within OpenFOAM. OpenFOAM, an open-source software widely used for Computational Fluid Dynamics (CFD) purposes, offers remarkable flexibility and customization options. Its solver libraries and comprehensive pre- and post-processing tools provide a competitive edge compared to other similar products [1].

In this work, specific attention is given to investigating the applicability of OpenFOAM tools in handling complex geometries and multiregional structures. In the practical part of the research, an example case is presented that serves as a guide to effectively simulate similar cases and demonstrates the capabilities of OpenFOAM in this topic. By leveraging the tools and techniques presented in this work, users can gain hints and a useful approach for their own simulations.

Simulations of this kind play a crucial role in advancing renewable technologies by providing valuable insights into their performance, efficiency, and optimization. Through computational modelling, engineers and researchers can simulate and analyse various devices used in renewable energy systems. For example, solar panels or collectors can be modelled to optimize orientation and evaluate energy output. By using simulation tools, researchers can gain a clear overview of device behaviour, identify areas for improvement, and ultimately contribute to the development of more efficient and reliable renewable technologies.

2. Fundamentals: Description of computational models

To calculate heat transfer in coupled fluid-solid regions, a set of equations needs to be solved, considering heat conduction convection, and any internal heat generation. In computational solvers, one common approach is the partitioned method, where the equations are solved separately for each region. Afterward, at the interface between the regions, multiregional coupling is performed.

In the study the following assumptions were made: there are no extra source terms, the radiation is neglected, the fluid is incompressible, Newtonian and a continuous medium, with the density that is constant in all terms except those related to buoyancy (the Boussinesq approximation is used).

2.1. Equations for solid

To calculate heat transfer in a solid, the **energy conservation equation** needs to be solved, which takes the form of a differential equation as presented in formula (1) [2].

$$\frac{\partial(\rho_0 c_P T)}{\partial t} + \nabla(\rho_0 c_P u T) = \nabla(k \nabla T) \tag{1}$$

where: ρ_0 – density, c_P – constant-pressure specific heat, T– temperature, u – velocity field, k – thermal conductivity.

2.2. Equations for fluid

To analyse compressible flow behaviour, it is necessary to consider more than just the energy conservation equation. Mass transport and momentum equations also need to be taken into account, along with the equation of a state.

The **mass transport equation** or the continuity equation (2), expresses that mass is conserved within a fluid flow [2].

$$\frac{\partial \rho_0}{\partial t} + \nabla(\rho_0 u) = 0 \tag{2}$$

where: ρ_0 – density, u – velocity of fluid.

The **momentum equations** (3) consider the effects of pressure, viscous forces, and body forces on the fluid flow [2].

$$\frac{\partial(\rho_0 \vec{u})}{\partial t} + (\rho_0 \vec{u} \cdot \nabla)\vec{u} = -\nabla p + \nabla \cdot \tau + \rho \vec{g} \tag{3}$$

where: ρ_0 – density, u – velocity of fluid, p – pressure, τ – sum of viscose and turbulent stresses (depending on the kind of flow), ρ – variable density modelled using Boussinesq approximation, $\vec{g}$ – gravity.

The **energy equation** includes terms of heat conduction, viscous dissipation and work related to the fluid. Equation (4) represents a form of equation involving enthalpy, which is the combined value of internal energy and kinematic pressure. [3]

$$\frac{\partial(\rho_0 h)}{\partial t} + \nabla \cdot (\rho_0 \vec{u} h) + \frac{\partial(\rho_0 k)}{\partial t} + \nabla \cdot (\rho_0 \vec{u} k) = \frac{\partial p}{\partial t} + \nabla \cdot (\tau \cdot \vec{u}) - \rho_0 \vec{g} \cdot \vec{u} \qquad (4)$$

where: ρ_0 – density, h – enthalpy, u – velocity of fluid, k – kinetic energy, p – pressure, τ – sum of viscose and turbulent stresses (depending on the kind of flow), $\vec{g}$ – gravity.

The set of equations mentioned above needs to be complemented by an **equation of state** that establishes a relationship between density, pressure, and temperature. As it was mentioned before in this analysis the Boussinesq approximation for density model is supposed. This approach provides a density changes in response to a change in temperature at constant pressure. The variable density in the buoyant term is modelled using following equation [4]:

$$\rho \approx \rho_0[1 - \beta\,(T - T_0)] \qquad (5)$$

where: ρ_0 – density of the fluid at temperature T_0, β – volumetric thermal expansion coefficient of fluid, T_0– reference temperature.

2.2.1. PISO algorithm

The PISO algorithm (Pressure Implicit with Splitting of Operators) is a pressure-based algorithm designed for calculation of unsteady compressible flows. This procedure is one of the most effective ways to solve the Navier-Stokes equations.

The solver uses an iterative approach to solve the momentum and so-called pressure equations (obtained by combining both continuity and momentum), utilizing a fractional step method. In this method, the velocity and pressure fields are decoupled and solved independently in separate steps. A predictor-corrector scheme is used, where an initial estimation of the velocity field is obtained, and then the pressure equation is solved to update and correct the velocity and pressure fields. This iterative process continues until convergence is reached, ensuring accurate results [5].

2.3. Boundary conditions

A solution of described equations depends on the physical conditions at the boundaries of the medium and on conditions in the medium at some initial time (for time dependent scenarios). There are three kinds of boundary conditions commonly encountered in heat transfer:

1) Dirichlet condition: specifies values along the boundary of the domain. For example, a constant temperature on a surface $T(x_s, t) = T_s$, where T_S is the specified temperature.
2) Neumann condition: represents existence of a fixed or constant heat flux at the surface. For example given boundary heat flux $-k\frac{\partial T}{\partial x}|_{xS} = q_s^{"}$, where k is the thermal conductivity and $q_s^{"}$ is the specified heat flux.
3) Robin condition: mixed condition, consisting parts from previous types. For example, a boundary heat flux can be specified via a heat transfer coefficient and the temperature of the surrounding fluid.

Depending on the order of the equation appropriate boundary conditions have to be considered [4].

2.4. Coupling between fluid and solid

The two regions are being related by coupling of corresponding values.

1) Temperatures at the interface between solid and fluid are equal
$$T_s = T_f$$
2) Values of heat fluxes are coincident: At the interface heat flux leaving one region is equal to flux entering the other region.
$$Q_s = -Q_f$$

3. Domain discretization

The first stage of pre-processing in every CFD software involves defining the geometry. The process of mesh generation involves dividing the computational domain into smaller, non-overlapping sub-domains, resulting in a grid or mesh of cells, control volumes, or elements [5]. While meshing a single region using OpenFOAM is a relatively straightforward operation, creating a mesh for coupled regions can be more challenging.

3.1. Simple meshing (*blockMesh*)

In OpenFOAM, a basic approach to declare the geometry is by specifying a list of vertex coordinates, along with a scaling factor, description of edge types, and a list of patches in the blockMeshDict file. This script simultaneously creates a mesh. In case of discussed software mesh is created by dividing of the domain geometry into multiple three-dimensional, hexahedral blocks, by specifying the number of cells in each direction. The blocks edges may be straight lines, arcs or splines [6].

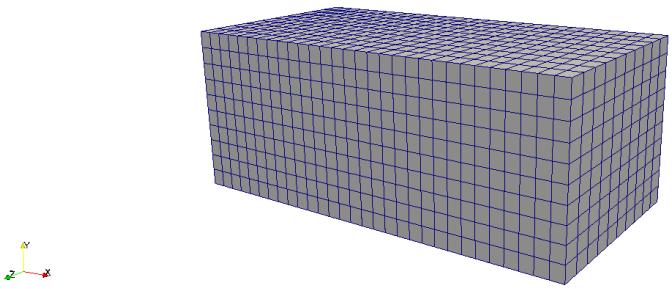

Figure 1. Simple mesh example with simpleGrading

The utility *blockMesh* in OpenFOAM also allows for mesh refinement, or grading, using the cell expansion ratio in each direction. There are two types of this option:

- *simpleGrading*: this method utilizes three expansion ratios, each corresponding to one orthogonal direction (x_1, x_2, x_3).
- *edgeGrading*: in this method, the number of expansion ratios matches the number of edges in the declared geometry.

The geometry can also be created using multiple blocks, allowing for the creation of a slightly more complex mesh with different zones. For instance, using a cylindrical block can lead to a grid that is more structured compared to a regular one (Figure 2). However, this approach requires a more extensive and intricate code implementation. Additionally, a limitation to consider is that each edge can only have one expansion ratio, resulting in the contacting blocks having the same split at the contact edge. *BlockMesh* it may not be the most appropriate approach,

especially if the desired model have complex geometry or require unstructured meshes. This tool alone does not allow for creating multiple regions.

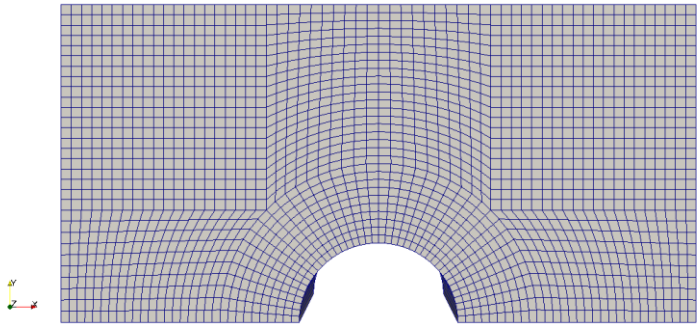

Figure 2. Mesh created with multiple blocks from OpenFOAM tutorial – Cylinder [7]

3.2. Multiregional meshing

Creating a multiregional model case requires additional tools that assist in separating the regions from each other. In OpenFOAM, there are various methods available to achieve this purpose.

3.2.1. topoSet

The toposet utility allows to specify regions in geometry created by *blockMesh* tool. By specifying the location and extent of each region using code, such as defining a box with vertex coordinates, users can specify regions within the geometry. Once the regions are defined, the next step involves running the *splitMeshRegion* utility.

The *splitMeshRegion* utility reads a *regionProperties* file that contains the criteria for dividing the mesh into multiple regions. These criteria can include geometric specifications, cell sets, point sets, or field values that define the boundaries or characteristics of each region. An example was shown in Figure 3.

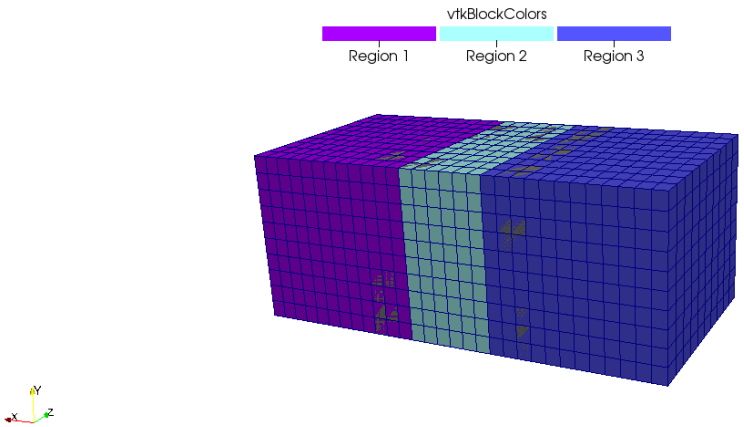

Figure 3. Multiregional mesh created using topoSet utility

This process enables the creation of multiple regions within the original mesh. However, it's important to note that this course of action may not be sufficient for simulations requiring grid refinement for example at the interface of regions.

3.2.2. *snappyHexMesh*

A way to achieve a more complex mesh in OpenFOAM is by utilizing the powerful utility called *snappyHexMesh*. Its main role is creating the mesh refining close to the edges, which is the significant advantage as it allows for more accurate simulations of complex geometries.

The snappyHexMesh utility generates three-dimensional meshes that contain hexahedra (hex) and split-hexahedra (split-hex) blocks. The mesh can be gradually shaped to match the surface of the geometry. A simple mesh is modified by adding more blocks, and their shape is adjusted to fit the surface. Another function of this utility is the ability to obtain a smaller grid or create additional layers. The geometry is imported from surface file in the Stereolithography (STL) format[1] [8]. An example of mesh created using this tool is shown in Figure 4.

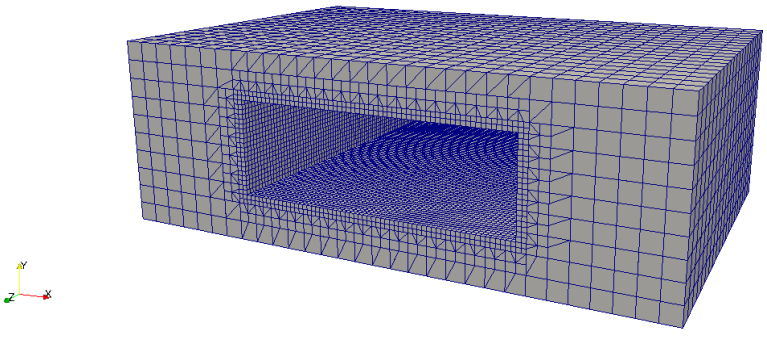

Figure 4. Mesh with refinement on wall of rectangular channel

[1] Since CAD systems save files in millimeters, a very useful function is *surfaceConvert*. It allows for rescaling an STL file to be compatible with the unit system used by OpenFOAM, which is meters. (*surfaceConvert* in.stl out.stl -clean -scale 0.001).

snappyHexMesh has three key functions:

1. Castellation: Allows to erase cells that are outside region indicated by the point, that operation allows to create more complex geometry. The declared surface in the STL format is being cut from predefined block.
2. Snapping: Reconstructs the cells to move the edges from inside the region to the required boundary
3. Layering: Creates additional layers of cells in the boundary region. These layers help for example capture the boundary behaviour more accurately.

Creating a multiregional model is achievable by specifying multiple regions in the *snappyHexMexhDict* file using segment *locationsInMesh*. Each region is indicated by a specific point. By defining these regions, and subsequently running the *splitMeshRegion* utility, separate regions can be created [9].

4. Preliminary case: three solids

In this chapter, with the help of the topoSet utility a heat transfer case in three separate solid regions with simple geometry will be performed. This case will provide necessary information about declaring boundary conditions for multiregional cases. To simplify the heat transfer phenomenon the initial temperature is equal in all the regions except one warmer wall (Figure 5), the thermophysical properties of each region are declared as equal and radiation in this scenario is neglected.

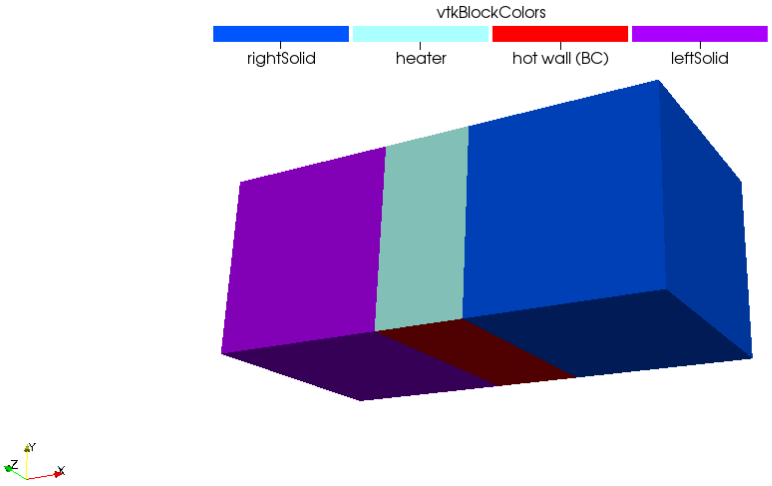

Figure 5. Three solids case with boundary condition marked

4.1. Case setup

In OpenFOAM there are three main directories (`0`, `constant`, `system`) in each case folder. In `0` directory the initial conditions are stored. Each file represents one property, for instance, file named T has information about initial temperature. The `constant` directory includes information about properties, mesh data and information about created geometry, such as patches and faces. The `system` directory stores information about solver, its settings and additional tools settings (in file usually named *tool_name*Dict).

For discussed case the file structure looks as follows:

```
multiRegion_3_layers
└  0
   └  p
   └  T
└  constant
   └  g
   └  regionProperties
   └  directory for each region
      └  radiationProperties
      └  thermophysicalProperties
└  system
      └  blockMeshDict
      └  controlDict
      └  fvSchemes
      └  fvSolution
      └  topoSetDict
      └  directory for each region
         └  changeDictionaryDict
         └  fvSchemes
         └  fvSolution
```

4.1.1. Geometry and mesh

For the geometry model constructed in 3.2.1 *topoSet* was used, shown in Figure 3. It consists of three solids achieved by the *blockMesh* tool with a simple mesh (*simpleGrading)* and the *topoSet* tool, that specifies areas of the three regions: leftSolid, heater and rightSolid. The `topoSetDict` file code can be found in Appendix: chapter 8.1.1

4.1.2. Boundary conditions

The initial and boundary conditions are stored in directories named with number (a number corresponds with the time). As it was mentioned before the initial conditions are stored in `0` directory, we can specify there properties for boundaries created using *blockMesh*. The other directories are created in the main directory after running the simulation.

After splitting the model into three different regions the initial and boundary conditions have to be set in relevant folders in `0` directory. A tool that allows to do it automatically is called *changeDictionary*. That utility results in copying the information for each region from relevant directories in the `system` directory. The file that stores the information is called `changeDictionaryDict`. Codes for each region can be found in Appendix: chapters 8.1.2-8.1.4.

In the discussed case it is imposed that the initial temperature is 300 K and one wall of the heater region has constant temperature over a time and equal 500 K. Initially the internal field temperature is set as 300 K and the Neumann boundary condition for boundary field is declared equal also 300 K (*zeroGradient*). Afterwards for the one of the heater's walls the temperature is changed to 500 K - the Dirichlet boundary condition for temperature (*fixedValue*) can be found in `system/heater/changeDictionaryDict`. Then for the walls in contact the boundary condition for temperature coupling between two regions in applied (*compressible::turbulentTemperatureRadCoupledMixed*).

4.1.3. Utilities summary

The summary of all utilities used the case is presented in a Table 1.

Table 1. Utilities summary for preliminary case

The geometry and mesh	`blockMesh`
	`topoSet`
	`splitMeshRegions -cellZones -overwrite`
The boundary conditions in `0` directory for each region	`changeDictionary -region region_name`
Solver	`chtMultiRegionFoam`

4.2. Results

The obtained simulation results are in line with the expectations:

- the temperature of the wall of the middle block remains constant over time as it was declared in the assumptions,
- the components are heating up symmetrically on either side of the model, that corresponds with the fact that the right and left solids have equal physical properties.

The distribution of temperature for two different times was presented in Figure 6 and Figure 7.

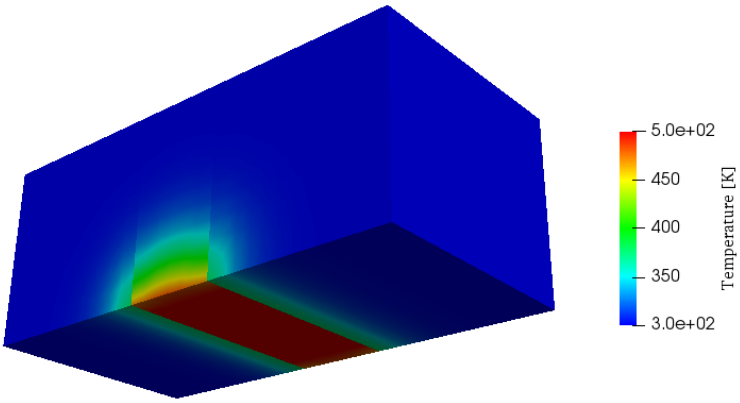

Figure 6. The temperature distribution after one time step (10 s)

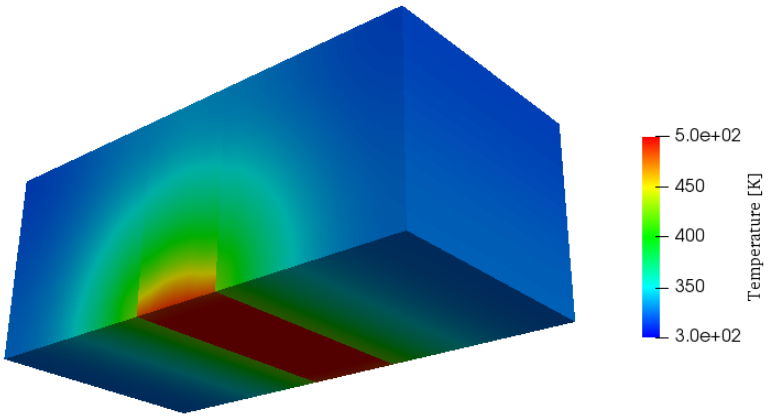

Figure 7. The temperature distribution after ten time steps (100 s)

Observing the results of this case, it can be concluded that the boundary conditions have been adequately defined. The analysed multi-regional case works properly, so based on its experience, a model with a more complex geometry can now be addressed.

5. Final case: fluid-solid coupling

In this chapter, the applicability of previously described OpenFOAM tools will be investigated. The final case will be presented in a frame of handling complex geometries and multiregional structures. Modelled case will be a heat transfer in a solid with a channel of a star-shaped cross-section, through which the fluid flows (Figure 8).

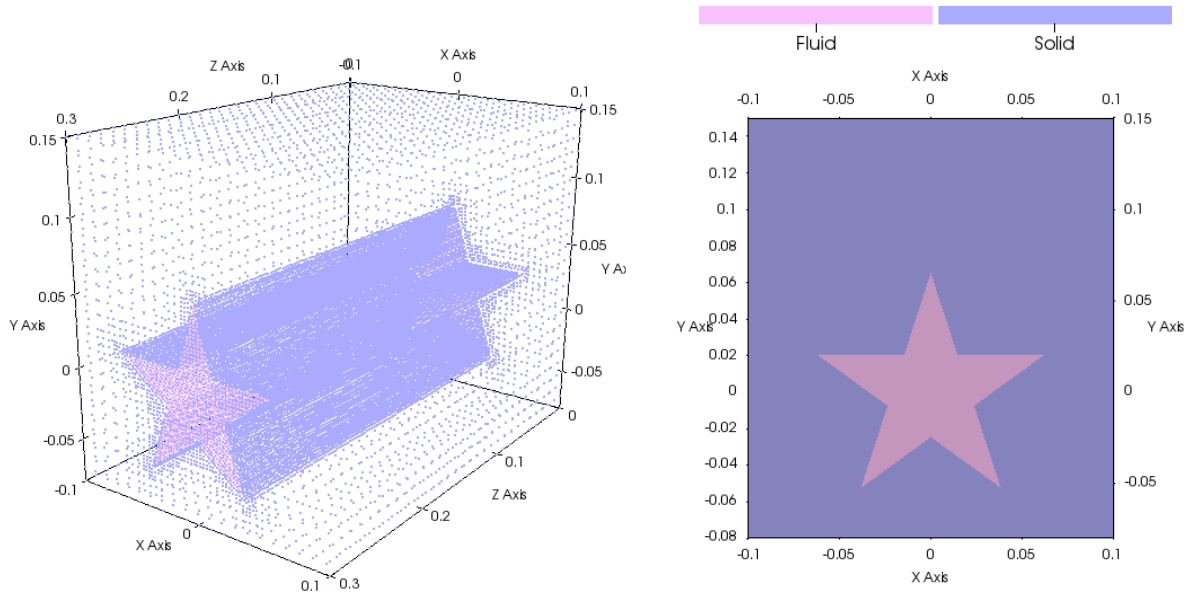

Figure 8. Model for fluid-solid coupling case

5.1. Case setup in OpenFOAM

To determine the flow velocity corresponding to turbulent flow, the hydraulic diameter was calculated based on the dimensions of the channel cross-section:

$$D_h = \frac{4 \cdot A}{P} \approx 6.36 \tag{6}$$

where: A – cross-sectional area of the channel, P – wetted perimeter of the cross-section.

As mentioned earlier, the fluid flow in this case is assumed to be turbulent. For the analysis, at the inlet a turbulent Reynolds number of 400 000 was imposed. Using this turbulent Reynolds number, the flow velocity was calculated to be U = 0.063 m/s. The formula used to calculate the Reynolds number is as follows [4]:

$$Re = \frac{U \cdot D_h}{\nu} \tag{7}$$

where: U – flow velocity, D_h – hydraulic diameter, ν – kinematic viscosity.

In the simulation, I have chosen the RAS (Reynolds-Averaged Navier-Stokes) turbulence model and specifically the k-omega SST (Shear Stress Transport) model. Within the RAS framework, the k-omega SST model is selected. The k-omega SST

model is a two-equation turbulence model that provides accurate predictions for a wide range of turbulent flows. It combines the benefits of the k-epsilon and k-omega models by using the k-omega model in the near-wall region and the k-epsilon model in the free-stream region [5].

Since there is a fluid present in analysed case, additional files are required: `turbulenceProperties` file that specifies type of flow and parameter value files. The set of files depends on the used turbulence model, for k-omega SST following files are necessary: `alphat`, `k`, `nut`, `omega` and `U`. Also, in comparison to the Preliminary case: three solids there is additional directory **`triSurface`** that stores surface geometry for *snappyHexMesh*.

The file structure looks as follows:

multiRegion star
- **0**
  - alphat
  - k
  - nut
  - omega
  - p
  - p_rgh
  - T
  - U
- **constant**
  - **triSurface**
    - *geometry.stl*
  - **g**
  - regionProperties
  - ***directory for each region***
    - radiationProperties
    - thermophysicalProperties
    - turbulenceProperties
- **system**
  - blockMeshDict
  - controlDict
  - fvSchemes
  - fvSolution
  - snappyHexMeshDict
  - ***directory for each region***
    - changeDictionaryDict
    - fvSchemes
    - fvSolution

5.1.1. Geometry and mesh

To achieve the described model, the combination of *blockMesh* and *snappyHexMesh* tools is used. Firstly, a simple mesh with cubic cells is created, then a prepared surface file with star-shaped cross-section is used to create a mesh refinement in the neighbourhood of the edges of the channel using *snappyHexMesh.* Additionally, to help extract the sharp edges from surface file the *surfaceFeatures* utility may be used prior *snappyHexMesh.* The `snappyHexMeshDict` file code can be found in Appendix: chapter 8.2.1.

For this case the level of surface refinement was chosen by trial and error: the mesh presented on Figure 9 has too few cells in edge regions area and for comparison the mesh from Figure 10 shows significant improvement.

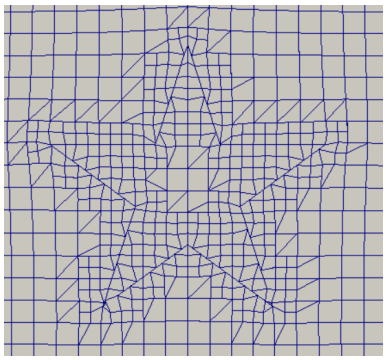

Figure 9. Mesh with refinementSurfaces level 1

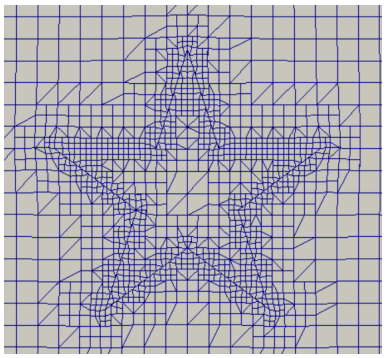

Figure 10. Mesh with refinementSurfaces level 2

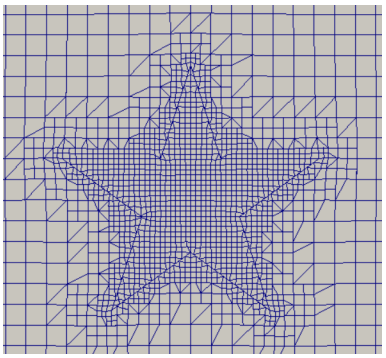

Figure 11. Mesh with refinementSurfaces level 1 and refinementRegion

To improve the mesh in the middle of the fluid region additionally a refinement region was declared in *snappyHexMesh,* the results are presented on Figure 11. Cylinder region with carefully selected diameter allows to create detailed mesh in the fluid region, however not to generate excessive number of cells in solid region.

5.1.2. Boundary conditions

In this case it is imposed that the initial temperature is 300 K beside the upper wall of solid region that has a constant temperature over a time and equal 500 K. For solid region the internal field temperature initially is set as 300 K and the boundary field is declared equal also 300 K (zeroGradient). Afterwards for the one of the walls – upper wall the temperature is changed to 500 K (fixedValue). Then for the surfaces in contact with fluid the boundary condition for temperature coupling between two regions in applied (compressible::turbulentTemperatureRadCoupledMixed).

For fluid initially internal and boundary field temperature is set as 300. Then for the contact surface between fluid-solid the boundary condition in applied (compressible::turbulentTemperatureRadCoupledMixed). All those conditions are assigned in system/*name of region*/changeDictionaryDict. Codes for each region can be found in Appendix: chapters 8.2.2-8.2.3.

It is important to note that before using changeDictionary utility the fluid fields have to be removed from solid regions: nut, alphat, omega, k, U, p_rgh.

5.1.3. Utilities summary

The summary of all utilities used the case is presented in a Table 2.

Table 2. Utilities summary for final case

The geometry and mesh	`blockMesh` `surfaceFeatureExtract`
	`snappyHexMesh -overwrite`
	`splitMeshRegions -cellZones -overwrite`
The boundary conditions in `0` directory for each region	`rm -f 0/`*`solid_region_name`*`/{nut, alphat, omega, k, U, p_rgh}` `changeDictionary -region`*`firstSolid`* `changeDictionary -region`*`fluid`*
Solver	`chtMultiRegionFoam`

5.2. Results

The obtained results are in line with the expectations:

- the temperature of the upper wall remains constant over time as it was declared in the assumptions,
- the solid heats up and transmits the heat to the fluid in the channel,
- the geometry of the channel and the flow velocity cause turbulence, numerous recirculations can be observed.

The distribution of temperature for analysed region was presented in Figure 12.

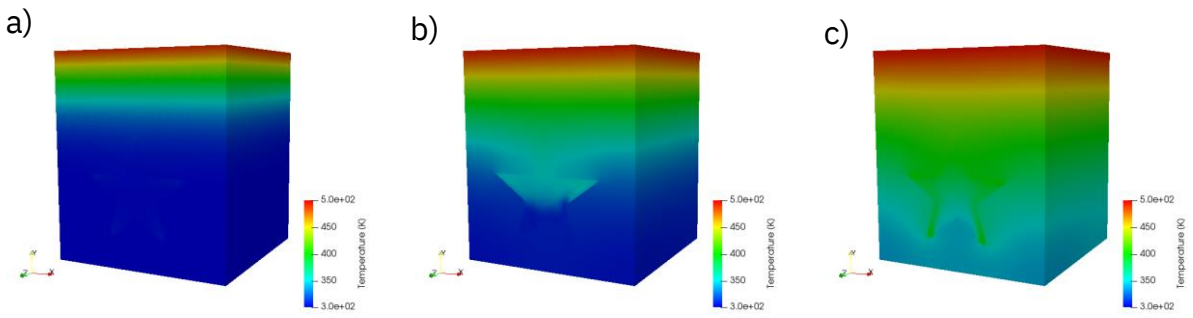

Figure 12. The distribution of temperature for times: a) 50 s, b) 150 s, c) 300 s

The temperature and velocity vector plot for a fluid in the normal section to the X axis (along the fluid flow direction) was presented on Figures 13-18.

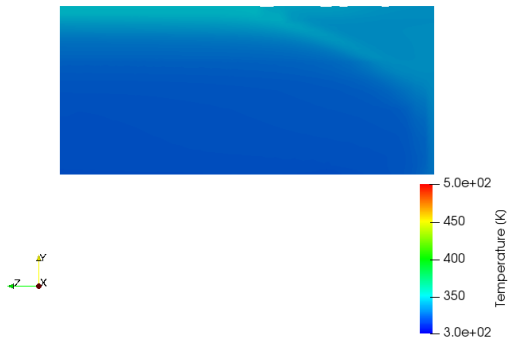

Figure 13. Fluid temperature, X-slice, 50 s

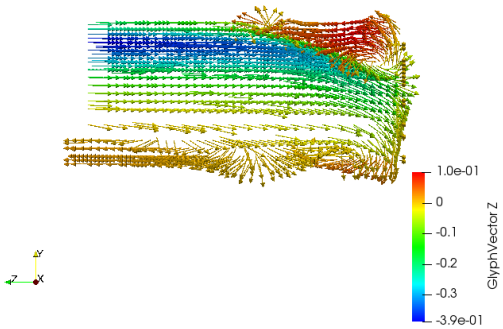

Figure 14. Velocity vector plot, X-silce, 50 s

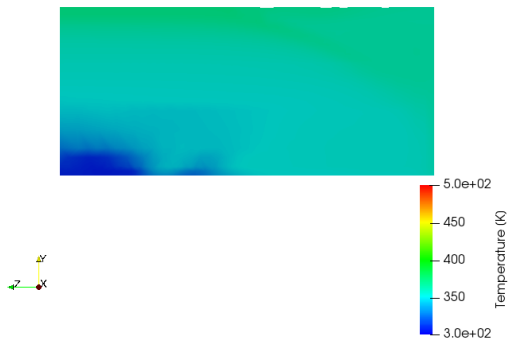

Figure 15. Fluid temperature, X-slice, 150 s

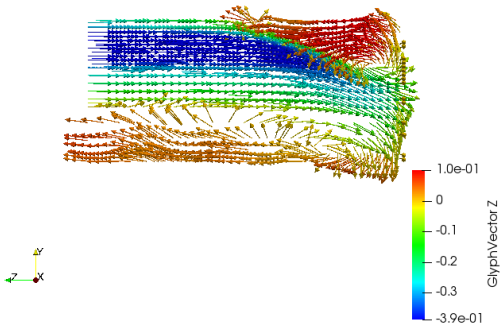

Figure 16. Velocity vector plot, X-silce, 150 s

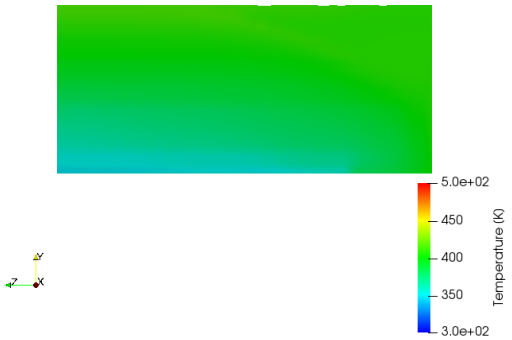

Figure 17. Fluid temperature, X-slice, 300 s

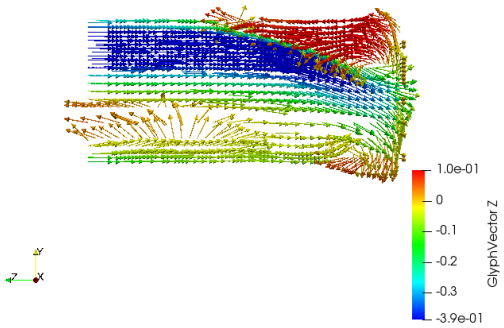

Figure 18. Velocity vector plot, X-silce, 300 s

The part of fluid located in the upper part of star's arm is strongly affected. The closer to the centre of the channel, the more closely the velocity profile resembles the set profile at the inlet.

6. Conclusions

Based on the conducted analysis, it can be concluded that OpenFOAM effectively addresses the issue of fluid-solid coupling for complex geometries. The software provides a suitable solver specifically designed for this problem. The key utilities required for creating a model, such as *snappyHexMesh, splitMeshRegions* and *changeDictionary*, have been thoroughly discussed in earlier sections, along with their specifications and application examples. Furthermore, the research findings demonstrate that OpenFOAM is highly capable of handling heat transfer cases involving advanced geometry with both fluids and solids, as evidenced by the example of the final case presented. Overall, these findings highlight OpenFOAM as a reliable and versatile tool for simulating complex heat transfer cases.

7. Bibliorgraphy

[1] OpenFOAM Overview, Available at: https://www.openfoam.com/governance/overview/ (Accessed: 25 May 2023)

[2] S. V. Patankar, *Numerical heat transfer and fluid flow*. Boca Ratón: Taylor & Francis, 1980.

[3] Energy Equation in OpenFOAM, *CFD Direct*, Available at: https://doc.cfd.direct/openfoam/energy-equation/ (Accessed: 3 June 2023)

[4] F. P. Incropera, Fundamentals of heat and mass transfer, 6th ed. New York, John Wiley & Sons, 2006

[5] H. Versteeg, W. Malalasekera, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*, Second Edition, 2007

[6] 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 (Accessed: 27 May 2023)

[7] Tutorial guide: Flow around a cylinder https://www.openfoam.com/documentation/tutorial-guide/2-incompressible-flow/2.2-flow-around-a-cylinder (Accessed: 1 June 2023)

[8] Mesh generation with the snappyHexMesh utility https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.4-mesh-generation-with-the-snappyhexmesh-utility (Accessed: 1 June 2023)

[9] F. Moukalled, L. Mangani, and M. Darwish, The Finite Volume Method in Computational Fluid Dynamics An Advanced Introduction with OpenFOAM® and Matlab, 1st ed. 2016. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-16874-6.

8. Appendix

In the following section most important files used for analysed cases were presented.

8.1. Preleminary case

8.1.1. topoSetDict

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

actions
(
    // Heater
    {
        name    heaterCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        box     (-0.02 -0.04 -0.05 )   (0.02 0.04 0.05);
    }

    {
        name    heater;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        set     heaterCellSet;
    }

    // leftSolid
    {
        name    leftSolidCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        box     (-0.1 -0.04 -0.05) (-0.02 0.04 0.05);
    }
    {
        name    leftSolid;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        set     leftSolidCellSet;
```

```
    }

    // rightSolid
    {
        name    rightSolidCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        box     (0.02 -0.04 -0.05) (0.1 0.04 0.05);
    }
    {
        name    rightSolid;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        set     rightSolidCellSet;
    }

);
// ************************************************************* //
```

8.1.2. changeDictionaryDict – heater

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

boundary
{
    minY
    {
        type            patch;
        inGroups        (coupleGroup);
    }
    minZ
    {
        type            patch;
    }
    maxZ
    {
        type            patch;
    }
}
```

```
T
{
    internalField   uniform 300;

    boundaryField
    {
        ".*"
        {
            type            zeroGradient;
            value           uniform 300;
        }
        "heater_to_.*"
        {
            type
compressible::turbulentTemperatureRadCoupledMixed;
            Tnbr            T;
            kappaMethod     solidThermo;
            value           uniform 300;
        }
        minY
        {
            type            fixedValue;
            value           uniform 500;
        }
    }
}
// ************************************************************* //
```

8.1.3. changeDictionaryDict – leftSolid

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

boundary
{
    minZ
    {
        type            patch;
    }
    maxZ
    {
        type            patch;
    }
}
```

```
  T
  {
      internalField   uniform 300;

      boundaryField
      {
          ".*"
          {
              type            zeroGradient;
              value           uniform 300;
          }
          "leftSolid_to_.*"
          {
              type
compressible::turbulentTemperatureRadCoupledMixed;
              Tnbr            T;
              kappaMethod     solidThermo;
              value           uniform 300;
          }
      }
  }
  // ************************************************************* //
```

8.1.4. changeDictionaryDict – rightSolid

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

  boundary
  {
      minZ
      {
          type            patch;
      }
      maxZ
      {
          type            patch;
      }
  }

  T
  {
```

```
    internalField   uniform 300;

    boundaryField
    {
        ".*"
        {
            type            zeroGradient;
            value           uniform 300;
        }
        "rightSolid_to_.*"
        {
            type
compressible::turbulentTemperatureRadCoupledMixed;
            Tnbr            T;
            kappaMethod     solidThermo;
            value           uniform 300;
        }
    }
}
// ************************************************************* //
```

8.2. Final case

8.2.1. snappyHexMeshDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      snappyHexMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * ** * * * * * * //
castellatedMesh true;
snap            true;
addLayers       false;

geometry
{
    gwiazda-skala.stl
    {
        type triSurfaceMesh;
        name geom;
    }

    cylinder
   {
        type            searchableCylinder;
        point1          (0 0 0);
```

```
        point2          (0 0 0.3);
        radius          0.04;
    }
}

castellatedMeshControls
{
    maxLocalCells 100000;
    maxGlobalCells 2000000;
    minRefinementCells 10;
    nCellsBetweenLevels 2;

    features
    (
        {
            file "gwiazda-skala.extendedFeatureEdgeMesh";
            level 1;
        }
    );

       refinementSurfaces
    {
        geom
        {
             level (2 2);
        }
    }

    resolveFeatureAngle 30;

    refinementRegions
    {

       cylinder
       {
           mode inside;
           levels ((1 2));
       }
    }

    locationsInMesh
    (
        (( 0 0.1 0.0005522) firstSolid)
        ((0.000005 0.0000555 0.001) fluid)
    );

    allowFreeStandingZoneFaces false;
}
```

```
snapControls
{
    nSmoothPatch 3;
    tolerance 1.0;
    nSolveIter 30;
    nRelaxIter 5;
    nFeatureSnapIter 10;
}

addLayersControls
{
    relativeSizes true;
    layers
    {
        maxY
        {
            nSurfaceLayers 3;
        }
    }

    expansionRatio 1.3;
    finalLayerThickness 1;
    minThickness 0.1;
    nGrow 0;
    featureAngle 30;

    nRelaxIter 3;
    nSmoothSurfaceNormals 1;
    nSmoothNormals 3;
    nSmoothThickness 2;
    maxFaceThicknessRatio 0.5;
    maxThicknessToMedialRatio 1;
    minMedialAxisAngle 90;
    nBufferCellsNoExtrude 0;
    nLayerIter 50;
}

meshQualityControls
{
    #include "meshQualityDict"

    nSmoothScale 4;
    errorReduction 0.75;
}

mergeTolerance 1e-6;

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

8.2.2. changeDictionaryDict – fluid

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

boundary
{
    minZ
    {
        type    patch;
    }
    maxZ
    {
        type    patch;
    }
}

U
{
    internalField   uniform (0 0 0.063);

    boundaryField
    {
        ".*"
        {
            type            fixedValue;
            value           uniform (0 0 0);
        }
        minZ
        {
            type            fixedValue;
            value           uniform (0 0 0.063);
        }
        maxZ
        {
            type            zeroGradient;
            value           uniform (0 0 0);
        }
    }
}

T
```

```
    {
        internalField   uniform 300;

        boundaryField
        {
            ".*"
            {
                type            zeroGradient;
            }

            minZ
            {
                type            fixedValue;
                value           $internalField;
            }

            "fluid_to_.*"
            {
                type
compressible::turbulentTemperatureRadCoupledMixed;
                Tnbr            T;
                kappaMethod     fluidThermo;
                value           uniform 300;
            }
        }
    }

    omega
    {
        internalField   uniform 9.32; //0.1;

        boundaryField
        {
            ".*"
            {
                type            omegaWallFunction;
                value           $internalField;
            }

            minZ
            {
                type            fixedValue;
                value           $internalField;
            }
```

```
maxZ
        {
            type            zeroGradient;
            value           $internalField;
        }
    }

}

k
{
    internalField   uniform 5.11e-11; //0.1;

    boundaryField
    {
        ".*"
        {
            type            kqRWallFunction;
            value           $internalField;
        }
        minZ
        {
            type            fixedValue;
            value           $internalField;
        }
        maxZ
        {
            type            zeroGradient;
            value           $internalField;
        }
    }
}

p_rgh
{
    internalField   uniform 1e5;

    boundaryField
    {
        ".*"
        {
            type            fixedFluxPressure;
            value           uniform 1e5;
        }
```

```
minZ
        {
            type            zeroGradient;
            value           uniform 1e5;
        }

        maxZ
        {
            type            fixedValue;
            value           uniform 1e5;
        }
    }
}

p
{
    internalField   uniform 1e5;

    boundaryField
    {
        ".*"
        {
            type            calculated;
            value           uniform 1e5;
        }

        maxZ
        {
            type            calculated;
            value           uniform 1e5;
        }
    }
}

nut
{
  boundaryField
  {

    ".*"
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }
```

```
minZ
    {
        type            calculated;
        value           uniform 1.25e-7;
    }

    maxZ
    {
        type            calculated;
        value           uniform 1.25e-7;
    }

  }
}

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

8.2.3. changeDictionaryDict – firstSolid

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

boundary
{
    minX
    {
        type            patch;
    }
    maxX
    {
        type            patch;
    }
    minY
    {
        type            patch;
    }
    maxY
    {
        type            patch;
    }
```

```
    minZ
        {
            type            patch;
        }
        maxZ
        {
            type            patch;
        }
    }

    T
    {
        internalField   uniform 300;

        boundaryField
        {
            ".*"
            {
                type            zeroGradient;
                value           uniform 300;
            }
            maxY
            {
                type            fixedValue;
                value           uniform 500;
            }

             "firstSolid_to_.*"
            {
                type
compressible::turbulentTemperatureRadCoupledMixed;
                value           uniform 300;
                Tnbr            T;
                kappaMethod     solidThermo;
            }
        }
    }
    // ************************************************************************//
```