CFD with OpenSource Software

A course at Chalmers University of Technology

Taught by

Håkan Nilsson

Project work:

Implementation of a myinterFoamDiabatic Solver with OpenFOAM

Developed for OpenFOAM-1.7.x

Requires:

Author:
Qingming Liu

Peer reviewed by:
Ehsan Yasari

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying les.

November 22 , 2011

Contents

Chapter 1

Introduction

Multi-phase simulation has been a very important topic both in industry and academy. There are several multi-phase solvers available in OpenFOAM. interFoam is one of the widely used solver which can be applied for isothermal 2 incompressible flow calculation. However, as the other multi-phase solvers in OpenFOAM, it cannot be used to simulate non-isothermal phenomena due to the lack of energy equation implementation[1]. This project is aiming to implement a new non-isothermal solver called **myinterFoamDiabatic** based on the solver **interFoam** by introducing energy equation. A simulation tutorial on bubbly 2 phase flow through a mini channel is done as well.
Since the bubbly flow in microchannel is a diabatic process, we need to implement thermal equations which have not included in interFoam so far. We need to add new thermalphysical properties of 2 phase as well.
The tasks of this project can be summarized as follows:

- Implementing **"myInterFoamDiabatic"** based on **"interFoam"** which using VOF method for simulation of two phase isothermal flow by adding energy equation as well as implementing a heat flux boundary condition.
- Set up the bubbly flow case in OpenFOAM.
- Study the flow pattern and pressure drop of elongated bubble flow in micro-channels(with diameter 1.7mm).
- Future plan: adding source terms to the transport equations.

Chapter 2

Mathematical background

The governing equations in the new solver **myinterFoamDiabatic** are summarized as follows:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u}\right) = -\boldsymbol{\nabla}P + \boldsymbol{\nabla}(\mu(\boldsymbol{\nabla}\cdot\boldsymbol{u} + \boldsymbol{\nabla}\cdot\boldsymbol{u}^T)) + \rho g \quad (1)$$

$$\nabla\cdot\boldsymbol{u} = 0\ ; \quad (2)$$

$$\rho C_p\left(\frac{\partial \boldsymbol{T}}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{T}\right) = \boldsymbol{\Phi} + \boldsymbol{\nabla}\cdot(\mathrm{k}\boldsymbol{\nabla}\mathrm{T}) \quad (3)$$

$$\frac{\partial \alpha}{\partial t} + \boldsymbol{u}\nabla\alpha = 0\ ; \quad (4)$$

They are the momentum equation (equation 1) the mass equation (equation 2), the energy equation (equation 3) and the volume fraction (equation 4).
Except the energy equation(eq. 3), the rest of equations has been implemented in the based solver **interFoam** already, so all we need to do in this project is to implement this equation. However, for simplicity, the source term (**Φ**) is assuming to be 0 at this project.

- In eq.3, c_p and k denote the specific heat capacity and the thermal conductivity respectively. μ is the viscosity.

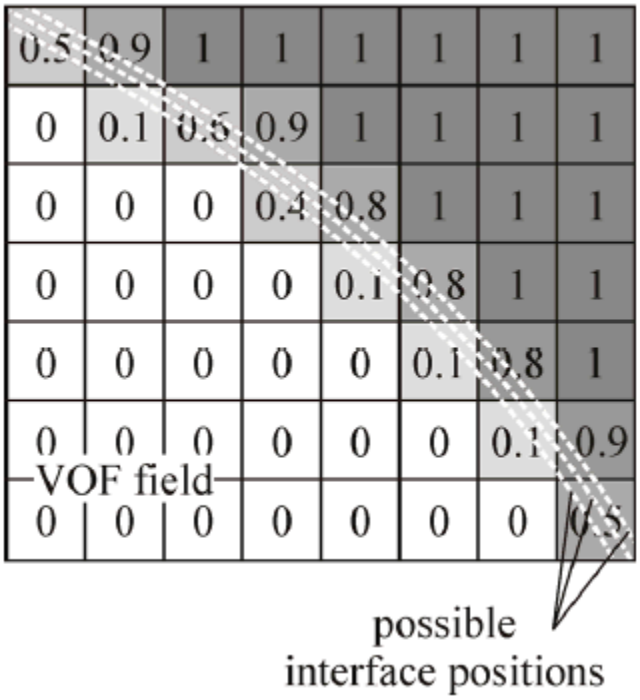


Figure 1 The VOF method[2]

The equation 4 is the volume fraction equation which will be used to capture the 2 phase interface by using a VOF method.
The volume fraction field(α) determines the liquid volume fraction in each cell. It has a value of 0 in all vapor cells, a value of 1 in all liquid cells and a value between 0 and 1 in cells that are cut by the liquid-vapor interface[3] [4](Figure 1).

Chapter 3

Implementation of myinterFoamDiabatic

The **myinterFoamDiabatic** application has a structure according to Figure 2. It contains a solver called **myinterFoamDiabatic** which called the shared library transportModels dynamically at runtime. Due to the introducing of thermal properties, the transportModels library should be modified to adapt the change. Hence we need to modify and compile the transportModels library independently before compiling the solver and setting up the case.

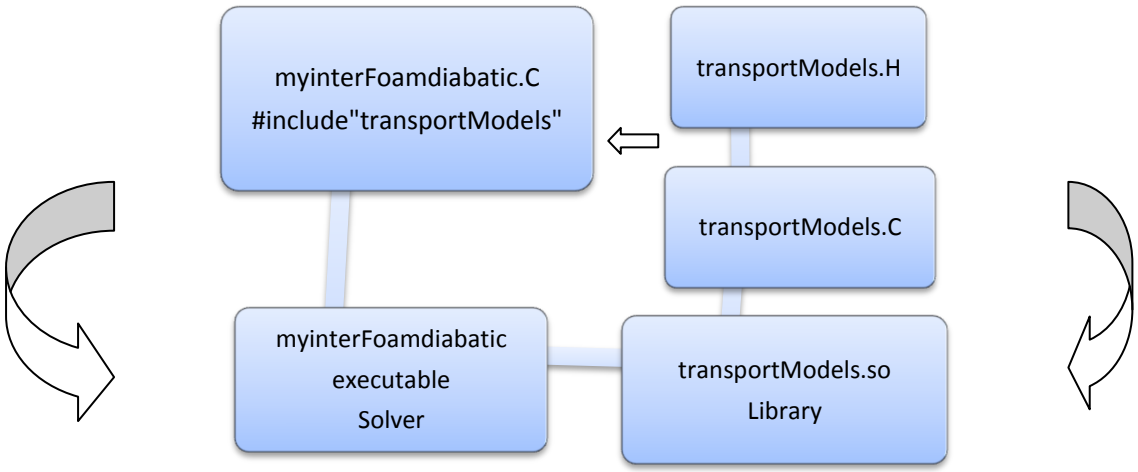


Figure 2 The tree of transportModel

3.1 Implementation of a new transport model

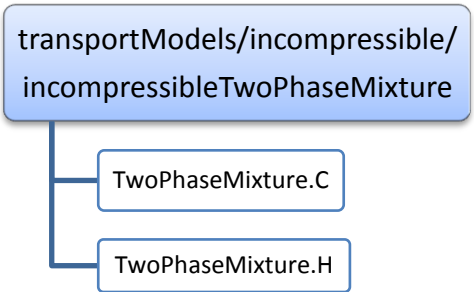


Figure 3 The tree of transportModel

Before implementing the solver, we should modify the transportModels library which is calling at run time by adding thermal properties for each phase and mixture.

First, copy the model to our own directory by the following command:

```
cp -r $WM_PROJECT_DIR/src/transportModels    $WM_PROJECT_USER_DIR/src
cd $WM_PROJECT_USER_DIR/src/transportModels/incompressible/
wclean
cd  incompressibleTwoPhaseMixture
```

Then we need to add the specific heat capacity(cp1, cp2)and Prandtl number (Pr1, Pr2)for each phase. Moreover, the thermal conductivity for the mixture is calculated according to the following expression:

$$k = \alpha\rho_1 \frac{c_{p1}}{Pr_1} + (1-\alpha)\rho_2 \frac{c_{p2}}{Pr_2} \qquad (5)$$

Open the file twoPhaseMixture.H and declaring the specific heat capacity(cp1, cp2)and Prandtl number s(Pr1, Pr2)for each phase by adding the following lines after line 67:

```
dimensionedScalar rho2_;           // line 67
//-----------------------Modified---------------------//
dimensionedScalar cp1_;
dimensionedScalar cp2_;
dimensionedScalar Pr1_;
dimensionedScalar Pr2_;
//-------------------------End-----------------------//
```

Modified lines after line 141 as follows:

```
const dimensionedScalar& rho2() const
{
return rho2_;
};                                    // line 141
//-------------------------Modified---------------------//
        const dimensionedScalar& Pr1() const
        {
        return Pr1_;
        }

        const dimensionedScalar& Pr2() const
        {
        return Pr2_;
        };

        const dimensionedScalar& cp1() const
        {
        return cp1_;
        }

        const dimensionedScalar& cp2() const
        {
        return cp2_;
        };
//--------------------------End-----------------------//
```

Add the thermal conductivity, specific heat capacity and Prandlt number as follows:

```
            tmp<surfaceScalarField> nuf() const;
//------------------------Modified---------------------//
    tmp<surfaceScalarField> kappaf() const;
//------------------------Modified---------------------//
```

```
rho2_(nuModel1_->viscosityProperties().lookup("rho2_"));
//---------------------------------------Modified-----------------------------------//
cp1_(nuModel1_->viscosityProperties().lookup("cp")),
cp2_(nuModel2_->viscosityProperties().lookup("cp")),
Pr1_(nuModel1_->viscosityProperties().lookup("Pr")),
Pr2_(nuModel2_->viscosityProperties().lookup("Pr")),
//---------------------------------------End-----------------------------------//
```

```
(
alpha1f*rho1*fvc::interpolate(nuModel1_->nu())
+ (scalar(1) - alpha1f)*rho2_*fvc::interpolate(nuModel2_->nu())
) /(alpha1f*rho1_ + (scalar(1) - alpha1f)*rho2_)
)
);
}

//---------------------------------------Modified-----------------------------------//
tmp<surfaceScalarField> twoPhaseMixture::kappaf() const
{
surfaceScalarField alpha1f =
min(max(fvc::interpolate(alpha1_), scalar(0)), scalar(1));

return tmp<surfaceScalarField>
(
new surfaceScalarField
(
"kappaf",
alpha1f*rho1_*cp1_*(1/Pr1_)*fvc::interpolate(nuModel1_->nu())
+ (scalar(1) - alpha1f)*rho2_*cp2_*(1/Pr2_)*fvc::interpolate(nuModel2_->nu())
)
);
}
//---------------------------------------End-----------------------------------//
```

and following modification:

```
nuModel1_->viscosityProperties().lookup("cp") >> rho2_;
//-----------------------Modified---------------------//
nuModel1_->viscosityProperties().lookup("cp") >> cp1_;
nuModel2_->viscosityProperties().lookup("cp") >> cp2_;
nuModel1_->viscosityProperties().lookup("Pr") >> Pr1_;
nuModel2_->viscosityProperties().lookup("Pr") >> Pr2_;
//-------------------------End----------------------//
```

Usually it is better to compile the new library in a new directory (for instance, the user directory rather than the default OpenFOAM **src** directory) . So in this case the new library was complied to the user directory $FOAM_USER_LIBBIN by modifying the **Make/files** as follows:

```
LIB = $(FOAM_USER_LIBBIN)/libmyincompressibleTransportModels
```

Compiling the the new library by type command:

wmake libso

3.2 Implementation of the solver myinterFoamDiabatic

The structure trees of the solvers of both based solver **interFoam** and derived solver **myinterFoamDiabatic** are shown in Figure 4. As it indicate, there is a need to create a TEqn.H file and modify several other files(alphaEqnSubCycle.H, interFoam.C, alphaEqn.H, CreateFields.H, Make directory) .

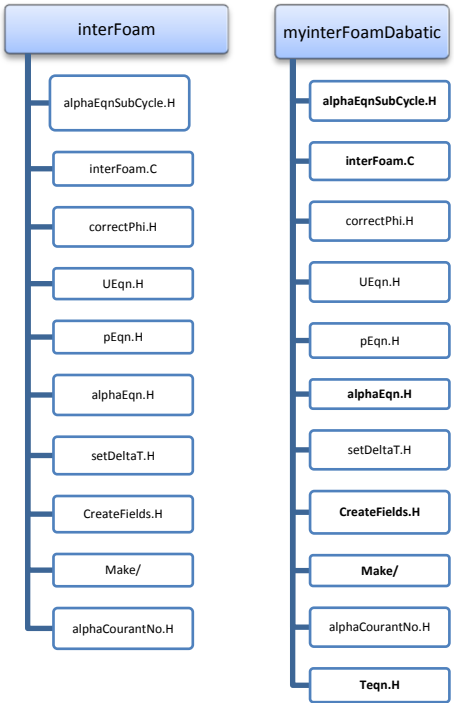


Figure 4 The tree of solver interFoam and myinterFoamDiabatic

Copy the solver to user directory

```
cp -r $FOAM_APP/solvers/multiphase/interFoam/    $WM_PROJECT_USER_DIR \
/applications/solvers/multiphase/myinterFoamDiabatic/
```

In order to add specific heat capacity Cp and Prandlt number Pr for each phase, the first file which should be modified is createFields.H, Then the Cp value should be added to alphaEqnSubCycle.H and alphaEqn.H. Finally, a Temperature equation is created and added to the main code interFoam.C.
We also need to the mixture specific heat capacity $\rho * c_p$as well as the heat flux $\rho * \varphi * c_p$ which are defined as equation (6) and (7) respectively.

$$\rho * c_p = \alpha\rho_1 c_{p1} + (1-\alpha)\rho_2 c_{p2} \quad (6)$$

$$\rho * \varphi * c_p = \alpha * \varphi * \rho_1 c_{p1} + (1-\alpha)\rho_2 \varphi c_{p2} \quad (7)$$

Here we do the instruction step by step:
First of all, we create temperature field 'T' in createFields.H by adding the following lines after creation of volume fraction alpha1:

```
//------------------Modified----------------------//
Info<< "Reading field T\n" << endl;
  volScalarField T
  (
     IOobject
     (
        "T",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
     ),
     mesh
  );
//----------------------End----------------------//
```

Open createFields.H and create the thermal properties we specified in the models before by adding following lines after line 68 and line 148 respectively:

```
const dimensionedScalar& rho2 = twoPhaseProperties.rho2();    // line 68
//------------------Modified----------------------//
const dimensionedScalar& cp1 = twoPhaseProperties.cp1();
const dimensionedScalar& cp2 = twoPhaseProperties.cp2();


//----------------------End----------------------//
```

```
                                                    // ……. line 148
//------------------Modified----------------------//
Info<< "Reading / calculating rho*cp\n" << endl;
volScalarField rhoCp
(
IOobject
(
"rho*Cp",
runTime.timeName(),
mesh,
IOobject::NO_READ,
IOobject::NO_WRITE
),
alpha1*rho1*cp1 + (scalar(1) – alpha1)*rho2*cp2,
alpha1.boundaryField().types()
);
rhoCp.oldTime();

Info<< "Reading / calculating rho*phi*cp\n" << endl;
surfaceScalarField rhoPhiCpf
(
IOobject
(
"rho*phi*cpf",
runTime.timeName(),
mesh,
IOobject::NO_READ,
IOobject::NO_WRITE
),
rhoPhi*cp1
);
//----------------------End-----------------------//
```

Open file alphaEqnSubCycle.H and add the following line after line 37:

```
 rho == alpha1*rho1 + (scalar(1) – alpha1)*rho2;
//--------------------Modified----------------------//
rhoCp == alpha1*rho1*cp1 + (scalar(1) – alpha1)*rho2*cp2;

//----------------------End-----------------------//
```

Open file alphaEqn.H and add the following line after line 31:

```
 rhoPhi= phiAlpha*(rho1 - rho2) + phi*rho2;
//--------------------Modified----------------------//
rhoPhiCpf = phiAlpha*(rho1*cp1 - rho2*cp2) + phi*rho2*cp2;
//----------------------End-----------------------//
```

Create file TEqn.H and add the following line:

```
surfaceScalarField kappaf = twoPhaseProperties.kappaf();

fvScalarMatrix TEqn
(
fvm::ddt(rhoCp, T)
+ fvm::div(rhoPhiCpf, T)
- fvm::laplacian(kappaf, T)
);

TEqn.solve();
```

add Temperature equation to interfoam.C.
open interFoam.C and add the following lines after line 85:

```
#include "UEqn.H"
//-------------------Modified--------------------//
#include "TEqn.H"
//----------------------End--------------------//
```

After doing all above, it is time to rename the old interfoam.C to myinterFoamDiabatic.C and modifying Make/files as

```
myinterFoamDiabatic.C
EXE = $(FOAM_USER_APPBIN)/myinterFoamDiabatic
```

Since the library of transportModels were modified and compiled to a new directory ($WM_PROJECT_USER_DIR) instead of the original one. We need to tell the compiler to connect to the new library.
Open Make/options and add the following lines:

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels \
    -I$(WM_PROJECT_USER_DIR)/src/transportModels/incompressible/lnInclude \
     …….

EXE_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lmyincompressibleTransportModels \
     …….
```

Compile the file by command >>wmake .

3.3 Setup the case in OpenFOAM

3.3.1 Mesh Generation

The mesh was generated by two different ways in this study, namely: directly by blockMesh and indirectly by ANSYS FLUENT. In the second way, the mesh generated in ANSYS FLUENT need to be converted to OpenFOAM by utilities **fluent3DMeshToFoam**. However, in this tutorial only the first method has been presented.

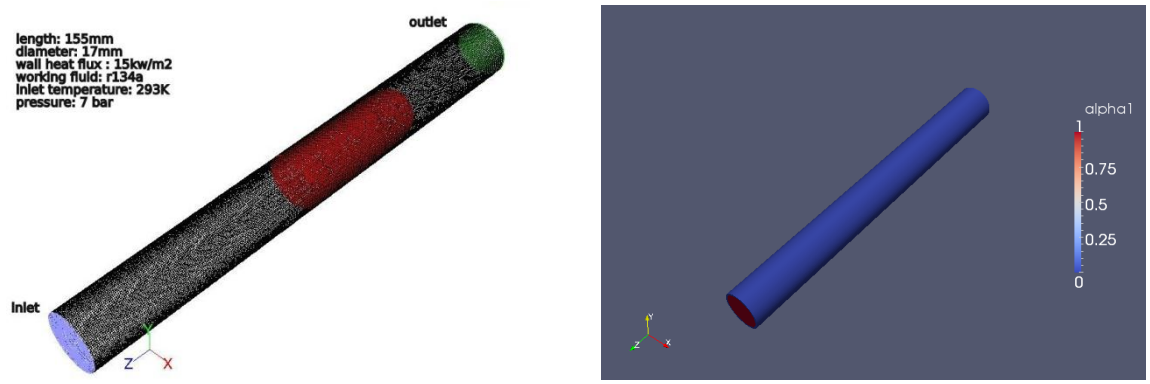


Figure 5 Mesh in ANSYS FLUENT (left) and OpenFoam(right)

The parameters of the case have been given as follows:

Table 1 The parameters of the case

Length ***mm***	**Diameter** ***mm***	**Wall heat flux** ***w/m2***	**Working fluid**	**Inlet temperature** ***K***	**Pressure: bar**
15.5	1.7	1500	R134a	293	7

The mesh generation using an O-grid according to the ERCOFTAC conical diffuser test case[5]. One main advantage of the O-grid method is that the mesh is refined in the near-wall places. The refining process has 2 steps. First , the cross-section of the channel was separated to two parts, namely: square-like inner and outer. As it is shown in Figure 6, the refines parameter 'rRelA' is defined as the ratio of the diagonal of the square-like inner to the diameter of the cross –section circle while 'rRelAc' denotes the ratio of the distance between mid-point of the arcs and the circle center to 'rRelA'.

'rRelAc=1' , the arc become a part of the circle with diameter 'rRelA'.

'rRelAc=$\frac{\sqrt{2}}{2}$', the 4 arcs become 4 straight lines and form an inner square with diagonal 'rRelA'.

Usually a value of 'rRelAc' between 1 and $\frac{\sqrt{2}}{2}$ is chosen in order to reduce the distortion of the mesh as much as possible. In this project, a value of 0.8 is set.

Then, the mesh near the wall can be refined by set relative large 'tNumberOfCells'. In order to do this, we need to create the file 'blockMesh.m4' under the directory constant/polyMesh/ and set the parameters according to Table 1. The detail code can be found in the code package. After that, we go the directory constant/polyMesh/ and type command >> m4 -P blockMeshDict.m4 > blockMesh .(the

case including the m4 file can be download from homepage:
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/).

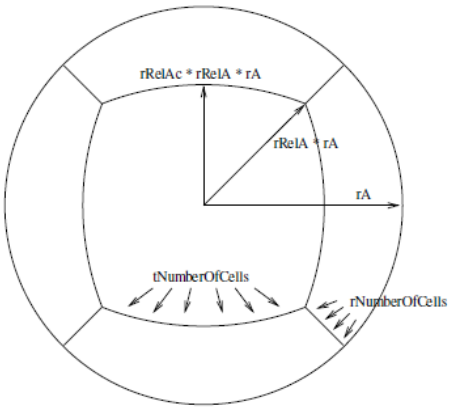
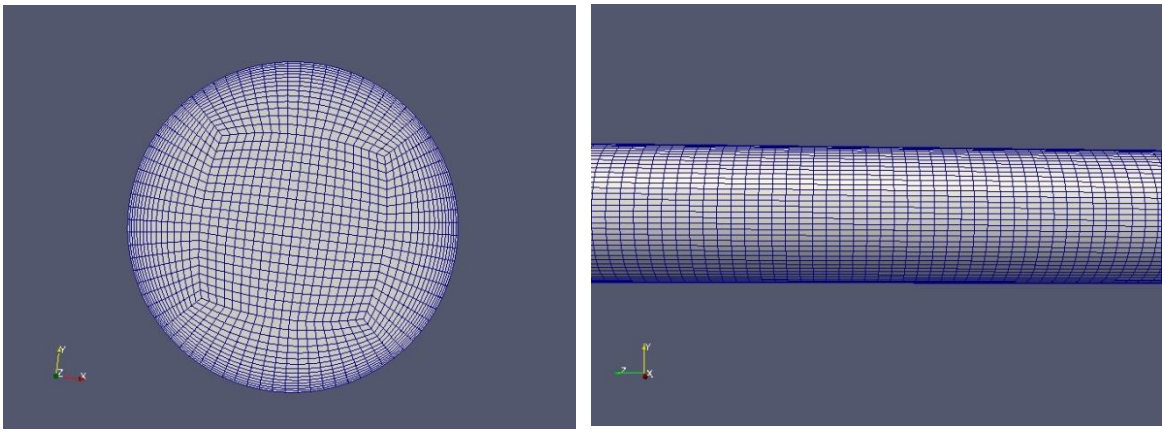


Figure 6 O grid method (H.Nilsson 2008)



a) b)

Figure 7 The mesh a) cross-section; b) side

3.3.2 Setup the case

As we can see from Figure 8, the case directory contains 3 sub-directory: constant / ,system/ and initial conditions 0/. New input parameters such Cp, Pr number, thermal conductivities as well as numerical schemes should be added to the case files in order to setup the case properly. This section we will do this step by step.

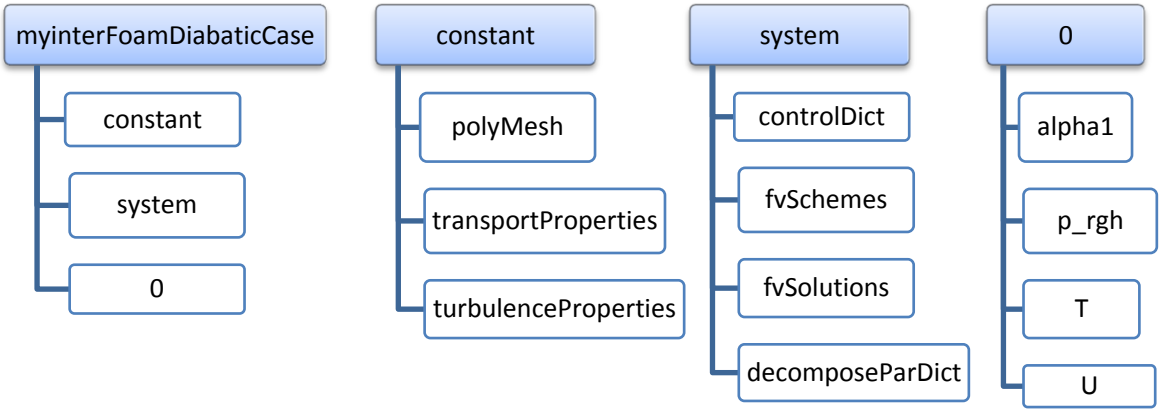


Figure 8 the tree structure of case directory

First of all, download the case file from the course homepage:
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/
The modification which had been done by the author can be summarized as follows:
The Cp and Prandlt number have added in the constant/transportProperties file as following lines:

```
phase1
{
  ....
  Pr          Pr [0 0 0 0 0 0 0] 3.366;
  cp          cp [0 2 -2 -1 0 0 0]  1433.4;
}
phase2
{
  ....
  Pr          Pr [0 0 0 0 0 0 0] 0.87;
  cp           cp [0 2 -2 -1 0 0 0]   1045.2;
}
```

Tell the code to use myinterFoamDiabatic solver and the new libraries by opening system/controlDict and add the following lines:

```
application          myinterFoamDiabatic;
...
libs("libmyincompressibleTransportModels.so");
```

A new finite divergence schemes for Temperature was added in system/ fvSchemes as follows:

```
divSchemes

{
   div(rho*phi,U)  Gauss limitedLinearV 1;
    ....
   div(rho*phi*cpf, T)     Gauss upwind;
    .....
}
```

Anew finite volume solution method for Temperature was added in system/ fvSolution as follows:

```
T
 {
  solver    BICCG;

  preconditioner   DILU;

  tolerance   1e-7;

  relTol   0;

 }
```

A Temperature field file was created under 0/ directory in 0/T as following lines:

```
 dimensions      [0 0 0 1 0 0 0];
internalField   uniform 300;
boundaryField
{
   inlet
   {
      type       fixedValue;
      value      uniform 300;
   }
   wall
   {
      type       fixedValue;
      value      uniform 300;
   }
  outlet
   {
      type       zeroGradien;
   }
}
```

As there is no constant heat flux boundary in OpenFOAM so far, we need to set this boundary condition indirectly. According to Fouriers' law, heat flux can be expressed as the following equation:

$$q = k\nabla T$$

So here we set a Temperature gradient which available in OpenFOAM instead of heat flux.

$$\nabla T = \frac{q}{k} = 183{,}765$$

```
wall
   {
      type       fixedGradient;
      value      uniform 183.765;
   }

 outlet
   {
      type       zeroGradient;
   }
```

3.3.3 SetField

Now we need to initialized a gas bubble in the inlet of the channel by the help of the utility called **funkySetFields** which is part of **swak4foam.** This utility can set the value of a scalar or a vector field depending on an expression that can be entered via the command line or a dictionary. It can also be used to set the value of fields on selected patches. It's like the **setFields** utility[6].

It can be used to set non-uniform initial-conditions without programming.

Since it is very difficult to initialize the bubble shape accurately, here a simplified cylinder bubble is set as the initial condition. The diameter of the bubble is set slightly smaller than the channel(1.7mm) to 1.5mm and the length is set to 0.35mm.

The expression of this initial bubble cylinder in Cartesian coordinate is as follows :

$$x^2 + y^2 \leq 0.0075^2 \;\&\&\; 0< z <0.003 \qquad (7)$$

```
cd $WM_PROJECT_USER_DIR/src
svn checkout https://openfoam-extend.svn.sourceforge.net/svnroot/openfoam-extend/trunk/Breeder_1.7/libraries/swak4Foam
```

Since only the funkySetFields utility is needed in this project, we do not compile the whole package. Instead, we do:

```
cd $WM_PROJECT_USER_DIR/src/swak4Foam/Utilities/funkySetFields
wmake
```

Then, we go to directory constant/polyMesh and set the initial bubble by funkySetFields utility according to the following command

```
funkySetFields -field alpha1 -expression 1 -time 0 -keepPatches -condition "pow(pos().x,2) + pow(pos().y,2) < pow(0.00075,2)  &&  pos().z <0.003"
```

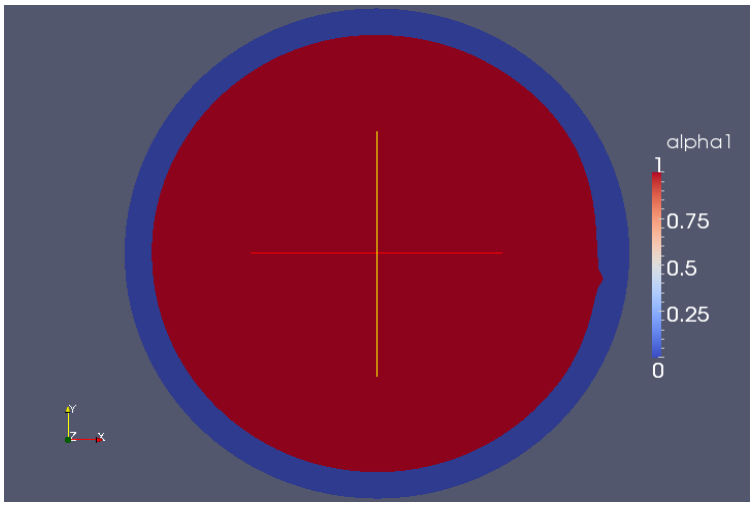


Figure 9 The initialized bubble

Decompose the case

Because of the big size of the mesh, it is better to run the case in parallel. In order to do this, we have to decompose the mesh to 4 parts by using the decomposeParDict. Open the file system/decomposeParDict and edit the contents as follows:

```
numberOfSubdomains 4;

method simple;

simpleCoeffs

{

n                (2 2 1);

delta           0.001;

}
```

After we run the decompose by type command >>decompsePar, we can get the 4 parts of the mesh as Figure 10.

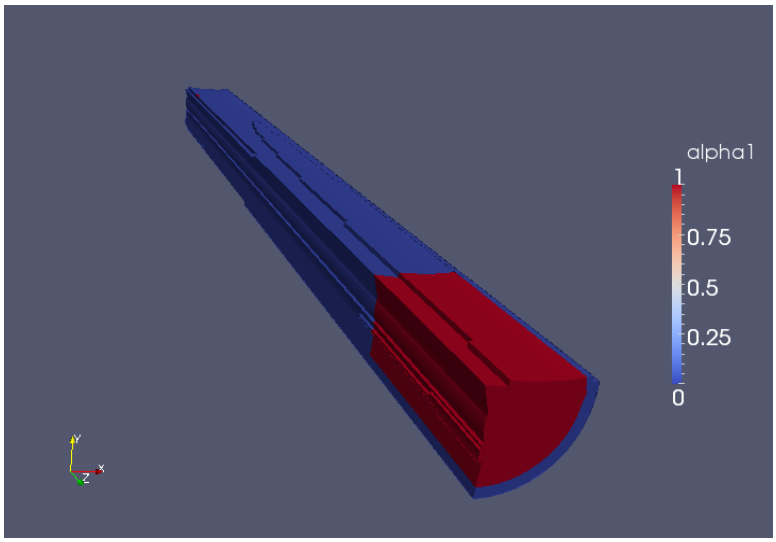


Figure 10 One of the 4 parts of channel

3.3.4 Ramped inlet conditions

Because these bubbles are generated periodically, we need to set a ramped phase inlet boundary condition. We want the gas volume fraction of inlet to ramp from the value $\alpha_1 = 1$ to the value $\alpha_2 = 0$ in the time interval between $t_1 = 0$ and $t_2 = 0.04s$. The ramp is simply defined by the extremal file: the interpolation routine will automatically provide the correct values for the times contained inside the interval specified above. So, we need to create a text file with the following format:

```
(
(t_0 value_0)
...
(t_n value_n)
)
```

First , we should create a 'Ramp' file(myinterFoamCase/ramp) which including the values of start time, ending time and gas volume fraction of the inlet under the case directory as following:

```
(
(0.0 1 )
(0.009999 1 )
(0.01 0 )
(0.019999  0)
(0.02   1)
(0.029999 1 )
(0.03 0 )
(0.039999  0)
(0.04   1)
)
```

Then we set the ramped boundary condition by choosing timeVaryingUniformFixedValue in the initial file(0/alpha1)

```
 boundaryField
…
   inlet
   {
      type          timeVaryingUniformFixedValue;
      fileName       "ramp";
      outOfBounds      clamp;
   }
}
```

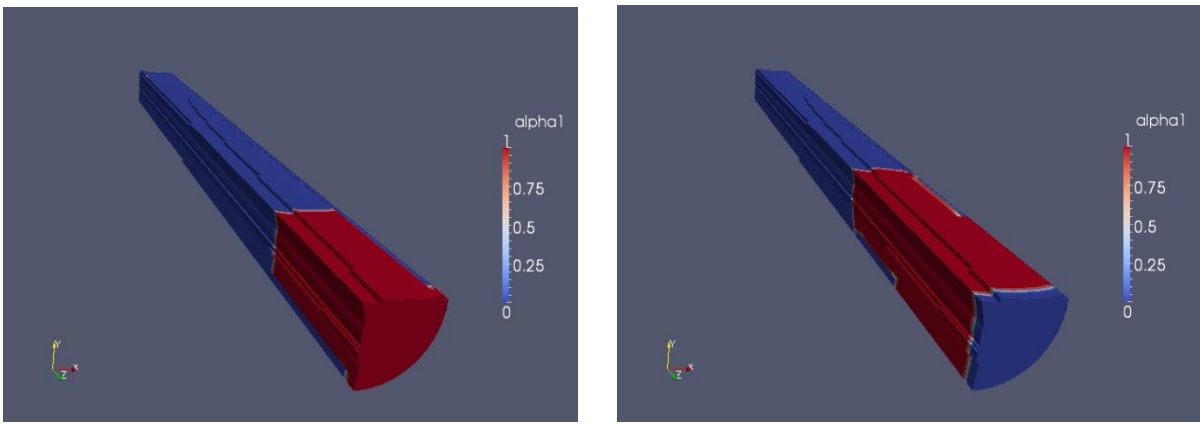
3.3.5 Running case

In the end, we set the simulation time, time step and interval and run our case in parallel according to the following parameters(System/controDict).

```
application             myinterFoamDiabatic;
startFrom               latestTime;
startTime               0;
stopAt                 endTime;
endTime                 0.02;
deltaT                 2e-4;
writeControl            adjustableRunTime;
writeInterval           0.001;
purgeWrite              0;
writeFormat             ascii;
writePrecision          6;
writeCompression        uncompressed;
timeFormat              general;
timePrecision           6;
runTimeModifiable       yes;
adjustTimeStep          on;
maxCo                  0.5;
maxAlphaCo              0.5;
maxDeltaT               1;
libs (“libmyincompressibleTransportModels.so”);
```

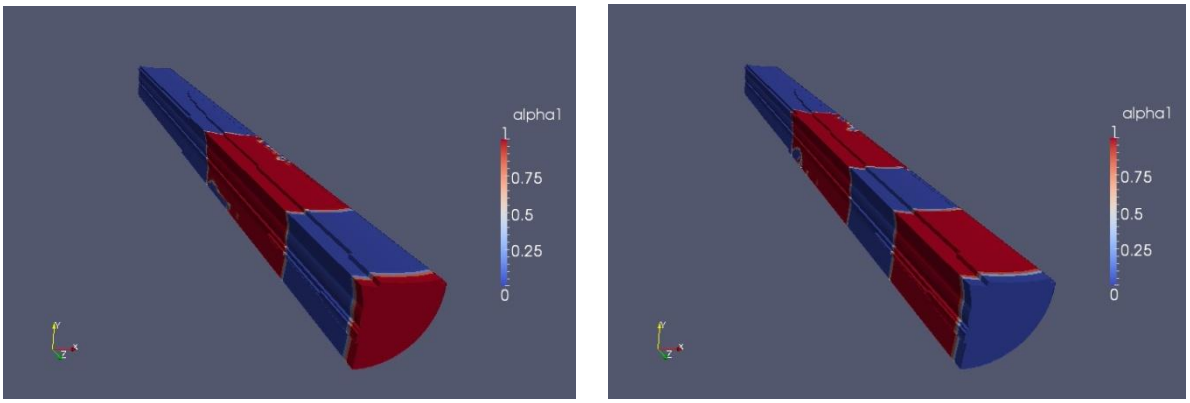
We just need to type the following command to run the case in parallel:

```
mpirun -np 4 myinterFoamDiabatic -parallel | tee log
```



0 s 0.0005s

Figure 11 Alpha1 gas bubble in the channel



0.015 s 0.02 s

Figure 12 gas bubble in the channel(at 0s, 0.005s, 0.015s, 0.02s)

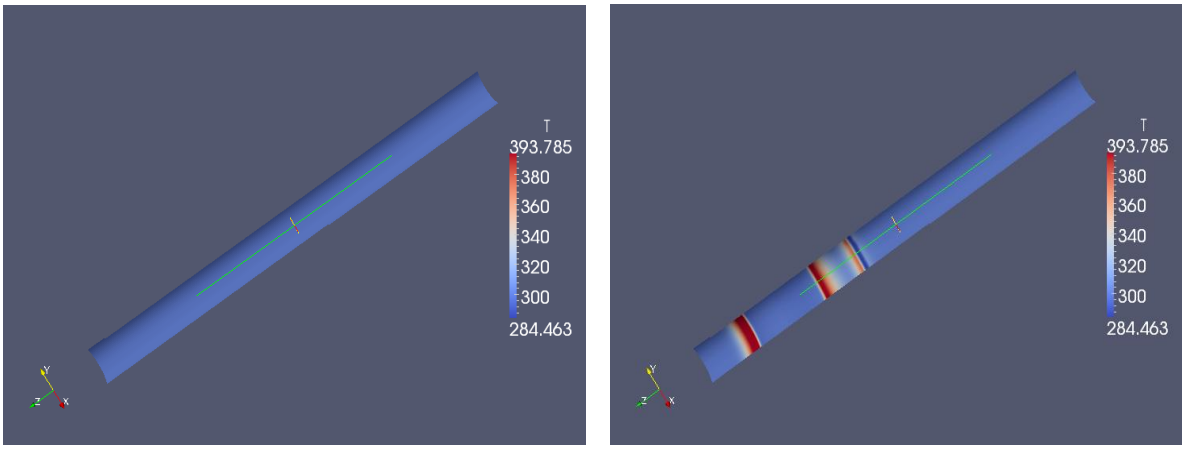


Figure 13 Temperature of the wall at 0s (left),0.02s(right)

From Figure11 and 12 the bubble going through the channel periodically according to the ramped boundary conditions. The temperature field in Figure 13 indicate that the heat was simulated

successfully in the case. A further study by adding source terms to mass and energy equation should be implement in order to investigate the phase change phenomena.

Reference

[1] OpenFoam User Guide
[2] Kunkelmann C, Stephan P. Modification and Extension of a Standard Volume-Of-Fluid Solver for Simulating Boiling Heat Transfer, European Conference on Computational Fluid Dynamics, Lisbon, Portugal, 14–17 June 2010
[3] Hirt, C.W.; Nichols, B.D. (1981), "Volume of fluid (VOF) method for the dynamics of free boundaries", Journal of Computational Physics39 (1): 201–225
[4] Pilliod, J.E. (1992), "An analysis of Piecewise Linear Interface Reconstruction Algorithms for Volume of Fluid Methods. Technical Report.", Technical Report, U.C. Davis
[5] Håkan Nilsson, Maryse Page, Martin Beaudoin, Bernhard Gschaider, Hrvoje Jasak
The OpenFOAM Turbomachinery Working Group, and Conclusions from the Turbomachinery Session of theThird OpenFOAM Workshop
[6] http://openfoamwiki.net