Cite as: Sun, Y.: Description of interCondensatingEvaporatingFoam and implementation of SGS term into volume fraction equation . In Proceedings of CFD with OpenSource Software, 2022, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2022

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Description of interCondensatingEvaporatingFoam and implementation of SGS term into volume fraction equation

Developed for OpenFOAM-v2112

Author:
Yaquan Sun
Technical University of Darmstadt
sun@rsm.tu-darmstadt.de

Peer reviewed by:
Yatin Darbar
Mohammad Hossein Arabnejad Khanouki

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 files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 24, 2023

Learning outcomes

The main requirements of a tutorial in the course is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- How to use the `interCondensatingEvaporatingFoam` solver for two-phase flow cases

The theory of it:

- The theory of volume of fluid method
- The theory of the evaporation model
- The theory of the subgrid scale term in the α-equation

How it is implemented:

- How the subgrid scale term is implemented in OpenFOAM

How to modify it:

- How to modify the `interCondensatingEvaporatingFoam` solver by adding a source term to α-equation

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- Computational Fluid Dynamics
- Volume of Fluid method
- Basic C++ programming knowledge
- How to run standard tutorials in OpenFOAM

Contents

Chapter 1

Introduction

Atomization and evaporation process in a spray system plays a dominant role in many industrial applications, such as fuel injection in engines [1], gas turbines [2] and burners [3]. Thereby, the spray atomization which is a very complex phenomenon has been extensively investigated numerically over the past decades.

Focusing on the computational fluid dynamics (CFD), Large Eddy Simulation (LES) is computationally cheaper than Direct Numerical Simulation (DNS) but needs subgrid scale (SGS) models for the unresolved multiphase turbulence dynamics while the large turbulent structures are directly resolved. In LES framework, numerical simulations with interface capturing methods like level set (LS) [4], volume of fluid (VOF) [5], or a combination of LS and VOF [6] have been reported in the literature [7]. For the VOF-LES method, the presence of the interface and the resulting filtering across the discontinuity cause additional Subgrid scale (SGS) terms in the LES formulation. Without accounting for the small scale instabilities and the unresolved turbulence interface interaction, primary breakup in a-posteriori LES strongly depends on the mesh. Incorporating interfacial SGS deformations revealed the potential of improving the prediction of jet destabilization by means of LES. Therefore, the inclusion of SGS turbulent and interfacial effects is expected to be of crucial importance in order to predict the correct flow behavior in multiphase LES.

The aim for this report will be to provide a detailed description of the existing multi-phase solver `interCondensatingEvaporatingFoam` for predicting the phase change in OpenFOAM. Additionally, the SGS source term will be added to the volume fraction equation so that the influence of the SGS term on the volume fraction field will be studied.

Chapter 2

Theory

2.1 Volume of Fluid

The flow `interFoam` based solver is employed that utilizes VOF phase-fraction based interface capturing approach to simulate the atomization and evaporation process. A VOF-LES formulation [8] is derived by applying a LES low pass filter combined with a Favre filter

$$\bar{\psi}(\mathbf{x}) = \mathbf{G} \star \psi(\mathbf{x}) = \int_{-\infty}^{\infty} \psi(\mathbf{y})\mathbf{G}(\mathbf{x}-\mathbf{y},\mathbf{x})dy, \qquad \widetilde{\psi}(\mathbf{x}) = \frac{\overline{\rho\psi}}{\bar{\rho}} \tag{2.1}$$

where $\bar{()}$ denotes LES low pass filtered quantities and $\widetilde{()}$ denotes that term is calculated with filtered quantities. $\mathbf{G}$ is a filter kernel and ψ is any field such as velocity. The continuity equation in Favre-filtered form in this approach is given as:

$$\frac{\partial \bar{\rho}}{\partial t} + \nabla \cdot (\bar{\rho}\widetilde{\mathbf{U}}) = 0 \tag{2.2}$$

with time t, velocity $\mathbf{U}$, the density ρ is defined as $\rho = \alpha\rho_1 + (1-\alpha)\rho_2$, ρ_1 and ρ_2 are the densites of two fluids, respectively. The corresponding filtered momentum equation is expressed as:

$$\frac{\partial(\rho\widetilde{\mathbf{U}})}{\partial t} + \nabla \cdot (\bar{\rho}\widetilde{\mathbf{U}} \otimes \widetilde{\mathbf{U}} - \bar{\mu}\widetilde{\mathbf{D}} - \sigma\bar{\mathbf{n}}\bar{\kappa}\delta_{\mathrm{s}}) + \bar{p}\mathbf{I} = -\nabla \cdot (\tau_{\rho uu} + \tau_{\mu \mathbf{s}}) + \tau_{nn} + \bar{\rho}\mathbf{g} \tag{2.3}$$

p stands for the pressure, $\mathbf{I}$ for unit tensor, $\mu = \alpha\mu_1 + (1-\alpha)\mu_2$ for the viscosities of two fluids, μ_1 for the viscosity of the fluid 1, μ_2 for the viscosity of the fluid 2, $\mathbf{D} = (\nabla\mathbf{U}) + (\nabla^{\mathrm{T}}\mathbf{U})$ for the deformation rate, $\mathbf{g}$ for the gravitational acceleration, σ for the surface tension coefficient, $\mathbf{n} = \frac{\nabla\alpha}{|\nabla\alpha|}$ for the unit normal vector at the interface, $\kappa = \nabla \cdot \mathbf{n}$ for the curvature and $\delta_s \equiv |\nabla\alpha|$ for the mathematical delta function that equals infinity at the interface and zero elsewhere. The unresolverd subgrid scale (SGS) terms in the Eqn. 2.3 read

$$\tau_{\rho uu} = \overline{\rho\mathbf{U} \otimes \mathbf{U}} - \bar{\rho}\widetilde{\mathbf{U}} \otimes \widetilde{\mathbf{U}}, \tag{2.4}$$

$$\tau_{\mu s} = \overline{\mu\mathbf{D}} - \bar{\mu}\widetilde{\mathbf{D}}, \tag{2.5}$$

$$\tau_{nn} = \overline{\sigma\kappa\mathbf{n}\delta_s} - \sigma\bar{\kappa}\bar{\mathbf{n}}\delta_s. \tag{2.6}$$

The phase transport equation is stated as

$$\frac{\partial \overline{\alpha}}{\partial t} + \nabla \cdot (\overline{\alpha}\widetilde{\mathbf{U}}) = 0 \tag{2.7}$$

where the liquid volume fraction α is introduced to distinguish between the two phases:

$$\alpha = \begin{cases} 0 & \text{phase 2,} \\ 0 < \alpha < 1 & \text{interface,} \\ 1 & \text{phase 1 (tracked phase).} \end{cases} \tag{2.8}$$

2.2 Phase change

Since mass is transferred between liquid and vapor during evaporation and condensation, the right hand side (RHS) of Eqn. 2.7 is no longer zero. Thus the volume fraction equations for liquid and vapor are given by

$$\frac{\partial \overline{\alpha_l}}{\partial t} + \nabla \cdot (\overline{\alpha_l}\widetilde{\mathbf{U}}) = \frac{\dot{m}}{\rho_l} \tag{2.9}$$

$$\frac{\partial \overline{\alpha_v}}{\partial t} + \nabla \cdot (\overline{\alpha_v}\widetilde{\mathbf{U}}) = -\frac{\dot{m}}{\rho_v} \tag{2.10}$$

where ρ_l denotes liquid density. ρ_v stands for the vapor density, $\dot{m} = \overline{\alpha_l}\dot{m}_v + (1-\overline{\alpha_l})\dot{m}_c$ for mass transfer rate between the two phases, $\dot{m}_v$ for vaporization and $\dot{m}_c$ for condensation. By adding the LHS and RHS of Eqn. 2.9 and Eqn. 2.10 and applying the $\overline{\alpha_l} + \overline{\alpha_v} = 1$ we get

$$\nabla \cdot \widetilde{\mathbf{U}} = (\frac{1}{\rho_l} - \frac{1}{\rho_v})\dot{m} \tag{2.11}$$

This expresses that the mass transfer effect on the divergence of velocity is not null in general. Then, a reorganization of the volume fraction equation is applied to couple the flux obtained through velocity with the flux of the liquid volume equation. Adding and subtracting $\overline{\alpha_l}\nabla \cdot \widetilde{\mathbf{U}}$ from the right hand side of Eqn. 2.9 and using Eqn. 2.11 we get

$$\frac{\partial \overline{\alpha_l}}{\partial t} + \nabla \cdot (\overline{\alpha_l}\widetilde{\mathbf{U}}) = (\frac{1}{\rho_l} - \overline{\alpha_l}(\frac{1}{\rho} - \frac{1}{\rho_v}))\dot{m} + \overline{\alpha_l}\nabla \cdot \widetilde{\mathbf{U}} \tag{2.12}$$

Here we define $\dot{V}' = \frac{1}{\rho_l} - \overline{\alpha_l}(\frac{1}{\rho_l} - \frac{1}{\rho_v})$ and because $\dot{m} = \overline{\alpha_l}\dot{m}_v + (1-\overline{\alpha_l})\dot{m}_c = \overline{\alpha_l}(\dot{m}_v - \dot{m}_c) + \dot{m}_c$ we get

$$\frac{\partial \overline{\alpha_l}}{\partial t} + \nabla \cdot (\overline{\alpha_l}\widetilde{\mathbf{U}}) = \dot{V}'(\overline{\alpha_l}(\dot{m}_v - \dot{m}_c) + \dot{m}_c) + \overline{\alpha_l}\nabla \cdot \widetilde{\mathbf{U}} \tag{2.13}$$

Then $\dot{V}_v = \dot{V}'\dot{m}_v$ and $\dot{V}_c = \dot{V}'\dot{m}_c$ and substituting them into Eqn. 2.13 and reorganizing we get

$$\frac{\partial \overline{\alpha_l}}{\partial t} + \nabla \cdot (\overline{\alpha_l}\widetilde{\mathbf{U}}) - \overline{\alpha_l}\nabla \cdot \widetilde{\mathbf{U}} = \overline{\alpha_l}\dot{V}_v - \overline{\alpha_l}\dot{V}_c + \dot{V}_c \tag{2.14}$$

Eqn. 2.14 is implemented in OpenFOAM (`alphaEqn.H` file) and the code of the equation will be explained in detail.

Mass transfer term is calculated by Lee's phase change model [9] which is one of the most popular phase change models for evaporation and condensation process in which the phase change is driven by the difference of interfacial temperature from saturated temperature. This model assumes that the mass is transferred at constant pressure and quasi-thermo-equilibrium state. The mass transfer rate per unit volume is calculated as

$$\dot{m}_c = r_c\overline{\alpha_v}\rho_v\frac{T_{sat} - \widetilde{T}}{T_{sat}} \qquad \widetilde{T} < T_{sat}(condensation) \tag{2.15}$$

$$\dot{m}_v = r_v\overline{\alpha_l}\rho_l\frac{\widetilde{T} - T_{sat}}{T_{sat}} \qquad \widetilde{T} > T_{sat}(vaporization) \tag{2.16}$$

Where r_c and r_v are the empirical coefficient named the mass transfer intensity factor with the unit of s^{-1}, which can be interpreted as the reciprocal of the relaxation time. $\widetilde{T}$ represents the temperature and T_{sat} for saturation temperature.

The energy equation is introduced to model the effect of heat transfer. The source term in the energy equation is the heat transferred due to mass transfer during phase change.

$$\frac{\partial \rho C_p\widetilde{T}}{\partial t} + \nabla \cdot (\rho\widetilde{\mathbf{U}}C_p\widetilde{T}) = \nabla \cdot (K\nabla\widetilde{T}) + \Delta h_v\dot{m} \tag{2.17}$$

where C_p is specific heat, K is thermal conductivity and h_v is the enthalpy of vaporization.

2.3 Subgrid scale term

As stated earlier, we will add a SGS term into the volume fraction equation to study the influence of the term on volume fraction field. Therefore, Eqn. 2.9 and Eqn. 2.10 can be written as

$$\frac{\partial \overline{\alpha_l}}{\partial t} + \nabla \cdot (\overline{\alpha_l}\widetilde{\mathbf{U}}) = \frac{\dot{m}}{\rho_l} + \tau_{u\alpha} \tag{2.18}$$

$$\frac{\partial \overline{\alpha_v}}{\partial t} + \nabla \cdot (\overline{\alpha_v}\widetilde{\mathbf{U}}) = -\frac{\dot{m}}{\rho_v} + \tau_{u\alpha} \tag{2.19}$$

$\tau_{u\alpha}$ is defined as

$$\tau_{u\alpha} = \widetilde{\mathbf{U}} \cdot \nabla\overline{\alpha} - \overline{\mathbf{U} \cdot \nabla\alpha} \tag{2.20}$$

A functional closure model is applied to close the SGS term in the volume fraction equation by the gradient approximation [8] in the Eqn. 2.20

$$\tau_{u\alpha} = \frac{\nu_{sgs}}{\sigma_{t,u\alpha}}\nabla^2\bar{\alpha}, \tag{2.21}$$

$\sigma_{t,u\alpha}$ is a turbulent Schmidt number, which is approximately equal to unity.

In the wall-adapting linear eddy-viscosity model (WALE) [10] which is applied in the present report, the subgrid scale viscosity is expressed as

$$\nu_{sgs} = (\mathrm{C}_W\Delta)^2 \frac{(\mathbf{S}^d_{ij}\mathbf{S}^d_{ij})^{3/2}}{(\bar{\mathbf{D}}_{ij}\bar{\mathbf{D}}_{ij})^{5/2} + (\mathbf{S}^d_{ij}\mathbf{S}^d_{ij})^{5/4}} \tag{2.22}$$

where C_W is the model coefficient and $\mathbf{S}^d_{ij}$ is the traceless symmetric part of the square of the velocity gradient tensor

$$\mathbf{S}^d_{ij} = \frac{1}{2}(\bar{L}^2_{ij} + \bar{L}^2_{ji}) - \frac{1}{3}\delta_{ij}\bar{L}^2_{kk}, \tag{2.23}$$

where $\bar{L}^2_{ij} = \bar{L}_{ik}\bar{L}_{jk}$ is the square of the velocity gradient tensor. The WALE model coefficient C_W is considered to be a true constant of $C_W = \sqrt{10.6}C_S \approx 0.5$ and it is usually not determined by means of a dynamic procedure. C_s stands for Smagorinsky constant.

2.4 `interCondensatingEvaporatingFoam`

`interCondensatingEvaporatingFoam` solver is a multiphase solver for two incompressible, non-isothermal immiscible fluids with phase change using VOF phase-fraction based interface capturing method. Since the `interCondensatingEvaporatingFoam` solver is similar to `interFoam`, here we only discuss the most important features of the `interCondensatingEvaporatingFoam`. The source code is located at

`$FOAM_PROJECT_DIR/applications/solvers/multiphase/interCondensatingEvaporatingFoam`

In the source code folder, there are following files:

- `Make`
- `createFields.H`
- `interCondensatingEvaporatingFoam.C`
- `alphaCourantNo.H`
- `TEqn.H`

- pEqn.H
- temperaturePhaseChangeTwoPhaseMixtures

In the source code folder, we can not find files for calculating α and U. But if we check Make/options, we find that the files for solving α equation, U equation and other relevant files are in other directories. We will copy these files to the interCondensatingEvaporatingFoam source code folder in the process of modifying the solver. In createFields.H file, physical phase models are utilized to initialize the transport properties, as shown in the following code in line 32:

Initialization of transport properties

```
31 // Creating e based thermo
32 autoPtr<twoPhaseMixtureEThermo> thermo
33 (
34     new twoPhaseMixtureEThermo(U, phi)
35 );
```

This model takes care of parsing and updating the transport properties, like temperature, internal energy and thermal diffusivity. twoPhaseMixtureEThermo in line 32 is a subclass of basicThermo and thermoIncompressibleTwoPhaseMixture and new member data and functions are added in this class. Now we take a look at this class. Go into the temperaturePhaseChangeTwoPhaseMixtures in the source code folder, there is the twoPhaseMixtureEThermo class, two files in the folder: H file for the declaration of the twoPhaseMixtureEThermo class and C file for the definition of the twoPhaseMixtureEThermo class. In the H file, the following code shows that the saturated temperature in the phase change model is declared:

Declaration of saturation Temperature properties

```
63     //- Saturation Temperature
64     dimensionedScalar TSat_;
```

Additionally, many virtual member functions for the calculation of the thermal properties are declared here, like heat capacity, thermal diffusivity and so on. For example, the thermal diffusivity for temperature of the mixture within the H file is as follows:

Declaration of the thermal diffusivity for temperature of the mixture

```
248 virtual tmp<volScalarField> kappa() const;
```

In the C file, the definition of the thermal diffusivity for temperature of the mixture is as follows:

Definition of the thermal diffusivity for temperature of the mixture

```
338 Foam::tmp<Foam::volScalarField> Foam::twoPhaseMixtureEThermo::kappa() const
339 {
340     const volScalarField limitedAlpha1
341     (
342     min(max(alpha1_, scalar(0)), scalar(1))
343     );
344
345     return tmp<volScalarField>
346     (
347         new volScalarField
348         (
349         "kappa",
350         limitedAlpha1*kappa1() + (scalar(1) - limitedAlpha1)*kappa2()
351        )
352     );
353 }
```

In line 340, the limitedAlpha1 is built to keep the value positive and between 0 and 1. In line 350, the thermal diffusivity for temperature of the mixture Kappa is calculated from the thermal diffusivity of phase 1 and phase 2. Besides, the correct function and read function are declared in this class as well:

Correct function and read function

```
292     //- Correct the thermo fields
293     virtual void correct();
294     //- Return the name of the thermo physics
295     virtual word thermoName() const;
296     //- Read properties
297     virtual bool read();
```

In the definition of the `read` function in C file, we see the phase change properties is read by this function. In `createFields.H` file, the following code in line 39 takes care of the two phase mixture:

Correct function and read function

```
37 // Create mixture and
38 Info<< "Creating temperaturePhaseChangeTwoPhaseMixture\n" << endl;
39 autoPtr<temperaturePhaseChangeTwoPhaseMixture> mixture =
40     temperaturePhaseChangeTwoPhaseMixture::New(thermo(), mesh);
```

Here `temperaturePhaseChangeTwoPhaseMixture` is a class for the phase change model (located in `temperaturePhaseChangeTwoPhaseMixtures` folder in the source code folder). There are many pure virtual member functions for the mass transfer rate in the class. For example, `mDotAlphal` function return the mass condensation and vaporisation rates.

mDotAlphal function

```
135 virtual Pair<tmp<volScalarField>> mDotAlphal() const = 0;
```

T source function for the source term in the temperature equation in TEqn.H:

T source function

```
146 virtual tmp<fvScalarMatrix> TSource() const = 0;
```

and the `correct` function for correcting the phase change model:

correct function

```
158 virtual void correct() = 0;
```

However, if we check `temperaturePhaseChangeTwoPhaseMixtures.C` file, we can not find the definition of the above three functions. This is because they are pure virtual member functions and they need to be defined by the specific phase change model, like constant model (Lee's phase change model) or interfaceHeatResistance model in `temperaturePhaseChangeTwoPhaseMixtures` folder. We will go to the details of the constant model. After the variables, phases, and transport and turbulence models are initiated in `createFields.H` file, the solution algorithm can proceed. Since there is very little difference between `interCondensatingEvaporatingFoam.C` and `interFoam.C`, we only explain the different parts. First of all, there are three new header files which are included in `interCondensatingEvaporatingFoam.C`:

New header files

```
52 #include "interfaceProperties.H"
53 #include "twoPhaseMixtureEThermo.H"
54 #include "temperaturePhaseChangeTwoPhaseMixture.H"
```

The first file is located in `$FOAM SRC/transportModels/interfaceProperties` which contains the interface properties, like the compression coefficient `cAlpha`, surface tension `sigmaPtr` and Stabilisation for normalisation of the interface normal `deltaN`. The second file and the third file are headers we explained in `temperaturePhaseChangeTwoPhaseMixtures` folder in the source code folder. The second different part is line 91:

T scalar field

```
91 volScalarField& T = thermo->T();
```

Here the reference of the temperature T is created as a volScalarField and T() function is called by the thermo object. And the next different part is the following code:

divU field

```
107 volScalarField divU("divU", fvc::div(fvc::absolute(phi, U)));
```

This piece of code is to store divU from the previous mesh so that it can be mapped and used in correctPhi to ensure the corrected phi has the same divergence. And the definition of the density is added in line 181:

density field

```
181 rho = alpha1*rho1 + alpha2*rho2;
```

2.4.1 Volume fraction equation

The volume fraction was derived in Section 2.2, now we go to the details of the calculation of volume fraction equation. As mentioned earlier, α equation file is not in the source code directory. After checking `Make/options` we find the `alphaEqn.H` needed for the solver is in the `interPhaseChangeFoam`. The following piece of code is for the calculation of the Eqn. 2.14 we derived in Section 2.2.

Alpha equation

```
17 fvScalarMatrix alpha1Eqn
18         (
19             fv::EulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
20           + fv::gaussConvectionScheme<scalar>
21             (
22                 mesh,
23                 phi,
24                 upwind<scalar>(mesh, phi)
25             ).fvmDiv(phi, alpha1)
26           - fvm::Sp(divU, alpha1)
27          ==
28             fvm::Sp(vDotvmcAlphal, alpha1)
29           + vDotcAlphal
30         );
```

The definition of `divU` can be found in `alphaEqnSubCycle.H` in the source code folder:

Definition of divU

```
12 volScalarField divU
13     (
14         mesh.moving()
15         ? fvc::div(phi + mesh.phi())
16         : fvc::div(phi)
17     );
```

As we can see, `divU` is the divergence of U from line 16. From the following piece of code, the `vDotmcAlpha` equals to `vDotvAlpha - vDotcAlpha`. The definition of `vDotvmcAlpha`, `vDotvAlpha` and `vDotcAlpha` can be found in the `alphaEqn.H` file:

Definition of vDotvmcAlphal

```
9  volScalarField divU
10 const volScalarField& vDotcAlphal = vDotAlphal[0]();
11 const volScalarField& vDotvAlphal = vDotAlphal[1]();
12 const volScalarField vDotvmcAlphal(vDotvAlphal - vDotcAlphal);
```

In line 10 and 11, `vDotAlpha` function is defined in `temperaturePhaseChangeTwoPhaseMixture.C`. Now we can take a look of this function in `temperaturePhaseChangeTwoPhaseMixture.C`.

Definition of vDotAlphal

```
69 Foam::Pair<Foam::tmp<Foam::volScalarField>>
70 Foam::temperaturePhaseChangeTwoPhaseMixture::vDotAlphal() const
71 {
72     volScalarField alphalCoeff
73     (
74         1.0/mixture_.rho1() - mixture_.alpha1()
75         *(1.0/mixture_.rho1() - 1.0/mixture_.rho2())
76     );
77
78     Pair<tmp<volScalarField>> mDotAlphal = this->mDotAlphal();
79
80     return Pair<tmp<volScalarField>>
81     (
82     alphalCoeff*mDotAlphal[0],
83     alphalCoeff*mDotAlphal[1]
84     );
85 }
```

In the definition of the `vDotAlpha` function, the alphalCoeff which is $\dot{V}'$ in Eqn. 2.13 is defined here as $\frac{1}{\rho_l} - \overline{\alpha_l}(\frac{1}{\rho_l} - \frac{1}{\rho_v})$ from line 74 to line 75. The values returned in vDotAlpha function are `alphalCoeff*mDotAlphal[0]` and `alphalCoeff*mDotAlphal[1]`, which are $\dot{V}_v = \dot{V}'\dot{m}_v$ and $\dot{V}_c = \dot{V}'\dot{m}_c$ in Eqn. 2.14, respectively. So far all the variables in Eqn. 2.14 have been clarified. The mass flow rate term `mDotAlphal` will be explained in the phase change model.

2.4.2 Pressure correction equation

As stated in Eqn. 2.11, the divergence of velocity equals to the mass transfer term instead of 0, so the contribution of mass transfer term is added to the pressure correction equation. We can get the definition of the pressure correction equation in the `pEqn.H` in the source code file as

P equation

```
30 while (pimple.correctNonOrthogonal())
31     {
32         fvScalarMatrix p_rghEqn
33         (
34             fvc::div(phiHbyA)
35           - fvm::laplacian(rAUf, p_rgh)
36           ==
37             vDotv + vDotc
38         );
```

We will not go through the details of the two terms which are derived from continuity equation and momentum equation on the left hand side (LHS) of the above pressure correction equation. If the reader is interested in this, please refer to A. Asnaghi [11]. In line 37, we see that mass transfer term $(\frac{1}{\rho_l} - \frac{1}{\rho_v})\dot{m}$ is split to `vDotv` $(\frac{1}{\rho_l} - \frac{1}{\rho_v})\dot{m}_v$ and vDotc $(\frac{1}{\rho_l} - \frac{1}{\rho_v})\dot{m}_c$ and are introduced on the right hand side of the pressure correction equation. The definition of `vDotv` and `vDotc` can be found in `pEqn.H` as well:

vDotv and vDotc

```
88 const volScalarField& vDotc = vDot[0]();
89 const volScalarField& vDotv = vDot[1]();
```

`vDotc` is the value of `vDot[0]()` and `vDotv` is the value of `vDot[1]()`. Similarly, we find their definition in `temperaturePhaseChangeTwoPhaseMixture.C`:

Definition of vDot function

```
69 Foam::Pair<Foam::tmp<Foam::volScalarField>>
70 Foam::temperaturePhaseChangeTwoPhaseMixture::vDot() const
71 {
72     dimensionedScalar pCoeff(1.0/mixture_.rho1() - 1.0/mixture_.rho2());
```

```
73      Pair<tmp<volScalarField>> mDot = this->mDot();
74
75      return Pair<tmp<volScalarField>>(pCoeff*mDot[0], pCoeff*mDot[1]);
76  }
```

In line 72, the `pCoeff` is defined here as $\frac{1}{\rho_l} - \frac{1}{\rho_v}$. This is the coefficient of the mass transfer term in Eqn. 2.11. Therefore, the return values `pCoeff*mDot[0]` and `pCoeff*mDot[1]` in line 75 are $(\frac{1}{\rho_l} - \frac{1}{\rho_v})\dot{m}_v$ and $(\frac{1}{\rho_l} - \frac{1}{\rho_v})\dot{m}_c$, respectively. The `mDot` in the return value in the mass flow rate term will be explained in the phase change model.

2.4.3 Energy equation

The energy equation Eqn. 2.17 is defined and calculated in `TEqn.H` in the source code folder.

T equation

```
1   {
2       tmp<volScalarField> tcp(thermo->Cp());
3       const volScalarField& cp = tcp();
4
5       const dimensionedScalar Cp1 = thermo->Cp1();
6       const dimensionedScalar Cp2 = thermo->Cp2();
7
8       rhoCp = rho*cp;
9
10      kappaEff = thermo->kappa() + rho*cp*turbulence->nut()/Prt;
11
12      const surfaceScalarField rhoCpPhi
13      (
14          "rhoCpPhi",
15          rhoPhi*(Cp1 - Cp2) + phi*rho2*Cp2
16      );
17
18      fvScalarMatrix TEqn
19      (
20          fvm::ddt(rhoCp, T)
21        + fvm::div(rhoCpPhi, T)
22        - fvm::Sp(fvc::ddt(rhoCp) + fvc::div(rhoCpPhi), T)
23        - fvm::laplacian(kappaEff, T)
24        + mixture->TSource()
25      );
26
27
28      TEqn.relax();
29      TEqn.solve();
30
31      Info<< "min/max(T) = " << min(T).value() << ", "
32          << max(T).value() <<endl;
33  }
```

We see the definition of the energy equation from line 18 to line 25. The `rhoCp` of the energy equation is defined in line 8, the `rhoCpPhi` is defined from line 12 to line 16 and `kappaEff` is defined in line 10. Similarly, the mass transfer term is also introduced to energy equation in line 24. We will go through this source term in the next subsection in the phase change model.

2.4.4 Phase change term

So far we have explained the volume fraction equation, pressure correction equation and energy equation in the code. The mass transfer term `mDotAlpha` in the volume fraction equation, the `mDot` in the pressure correction equation and `TSource()` in energy equation have not been explained. If we check the `constant` folder in `temperaturePhaseChangeTwoPhaseMixtures`, we find their declarations and definitions.

The declaration of the member data and member functions of the evaporation model

```
namespace Foam
{
namespace temperaturePhaseChangeTwoPhaseMixtures
{

/*---------------------------------------------------------------------------*\
                           Class constant
\*---------------------------------------------------------------------------*/

class constant
:
    public temperaturePhaseChangeTwoPhaseMixture
{
    // Private data

        //- Condensation coefficient [1/s/K]
        dimensionedScalar coeffC_;

        //- Evaporation coefficient [1/s/K]
        dimensionedScalar coeffE_;


public:

    //- Runtime type information
    TypeName("constant");


    // Constructors

        //- Construct from components
        constant
        (
            const thermoIncompressibleTwoPhaseMixture& mixture,
            const fvMesh& mesh
        );


    //- Destructor
    virtual ~constant() = default;


    // Member Functions

        //- Return the mass condensation and vaporisation rates as a
        //  coefficient to multiply (1 - alphal) for the condensation rate
        //  and a coefficient to multiply  alphal for the vaporisation rate
        virtual Pair<tmp<volScalarField>> mDotAlphal() const;

        //- Return the mass condensation and vaporisation rates as coefficients
        virtual Pair<tmp<volScalarField>> mDot() const;

        //- Return the mass condensation and vaporisation rates as a
        //  coefficient to multiply (Tsat - T) for the condensation rate
        //  and a coefficient to multiply  (T - Tsat) for the vaporisation rate
        virtual Pair<tmp<volScalarField>> mDotDeltaT() const;

        //- Source for T equarion
        virtual tmp<fvScalarMatrix> TSource() const;

        //- Correct the constant phaseChange model
        virtual void correct();

        //- Read the transportProperties dictionary and update
        virtual bool read();
};
```

```
111
112
113 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
114
115 } // End namespace temperaturePhaseChangeTwoPhaseMixtures
116 } // End namespace Foam
```

In line 61 and 64, the two coefficients `coeffC_` for condensation and `coeffE_` for evaporation are declared here. But the `coeffC_` is not r_c in Eqn. 2.15 and the `coeffE_` is not r_e in Eqn. 2.16. From the comments in line 60 and 63, the unit of `coeffC_` and `coeffE_` is 1/s/K, thus the `coeffC_` represents for r_c/T_{sat} and `coeffE_` for r_e/T_{sat} . This is why we can not see the temperature denominators in Eqn. 2.15 and Eqn. 2.16 from the code. We will explain this again when we go through the definition of mass transfer term in the code. In line 92, `mDotAlphal()` for $\dot{m}_v$ and $\dot{m}_c$ in Eqn. 2.13 is declared. `mDot()` for $\dot{m}_v$ and $\dot{m}_c$ in the pressure correction equation is declared in line 95. The source term of the temperature equation is declared in line 103. `mDotDeltaT()` function is not discussed here, as this function was not used in the momentum, pressure correction and temperature equation. Then, the `correct()` for the correction of phase change model and `read()` for reading the the `transportProperties` dictionary and updating the functions are declared here. Next, we will go through the member functions in the `constant.H` file. Open `constant.C` file, firstly we see `mDotAlphal()` function:

Definition of mDotAlphal function

```
76 Foam::Pair<Foam::tmp<Foam::volScalarField>>
77 Foam::temperaturePhaseChangeTwoPhaseMixtures::constant::mDotAlphal() const
78 {
79     const volScalarField& T = mesh_.lookupObject<volScalarField>("T");
80
81     const twoPhaseMixtureEThermo& thermo =
82         refCast<const twoPhaseMixtureEThermo>
83         (
84             mesh_.lookupObject<basicThermo>(basicThermo::dictName)
85         );
86
87     const dimensionedScalar& TSat = thermo.TSat();
88
89     const dimensionedScalar T0(dimTemperature, Zero);
90
91     return Pair<tmp<volScalarField>>
92     (
93         coeffC_*mixture_.rho2()*max(TSat - T, T0),
94         -coeffE_*mixture_.rho1()*max(T - TSat, T0)
95     );
96 }
```

In line 79, the `volScalarField T` are constructed. From line 81 to 87, the `thermo` object is constructed to get the saturation temperature from `thermophysicalProperties` in `constant` file. In line 89, `T0` is constructed to keep the difference between `TSat` and `T` positive. The return values of the `mDotAlphal()` function are the RHS of Eqn. 2.15 and Eqn. 2.16 for the phase change model but without $\overline{\alpha_v}$ and $\overline{\alpha_l}$. $\overline{\alpha_v}$ and $\overline{\alpha_l}$ have been multiplied in the definition of `alphalCoeff` in Section 2.4.1 as discussed before. Then we go through the definition of `mDot()` function:

Definition of mDot function

```
99  Foam::Pair<Foam::tmp<Foam::volScalarField>>
100 Foam::temperaturePhaseChangeTwoPhaseMixtures::constant::mDot() const
101 {
102
103     volScalarField limitedAlpha1
104     (
105         min(max(mixture_.alpha1(), scalar(0)), scalar(1))
106     );
107
108     volScalarField limitedAlpha2
```

```
109     (
110         min(max(mixture_.alpha2(), scalar(0)), scalar(1))
111     );
112 
113     const volScalarField& T = mesh_.lookupObject<volScalarField>("T");
114 
115     const twoPhaseMixtureEThermo& thermo =
116         refCast<const twoPhaseMixtureEThermo>
117         (
118             mesh_.lookupObject<basicThermo>(basicThermo::dictName)
119         );
120 
121     const dimensionedScalar& TSat = thermo.TSat();
122 
123     const dimensionedScalar T0(dimTemperature, Zero);
124 
125     volScalarField mDotE
126     (
127         "mDotE", coeffE_*mixture_.rho1()*limitedAlpha1*max(T - TSat, T0)
128     );
129     volScalarField mDotC
130     (
131         "mDotC", coeffC_*mixture_.rho2()*limitedAlpha2*max(TSat - T, T0)
132     );
133 
134     if (mesh_.time().outputTime())
135     {
136         mDotC.write();
137         mDotE.write();
138     }
139 
140     return Pair<tmp<volScalarField>>
141     (
142         tmp<volScalarField>(new volScalarField(mDotC)),
143         tmp<volScalarField>(new volScalarField(-mDotE))
144     );
145 }
```

`mDot()` function is the mass transfer term in pressure correction equation. In line 103 and 108, two `volScalarField limitedAlpha1` and `limitedAlpha2` are constructed. `min()` and `max()` functions are applied here to keep α between 0 and 1. The return values of `mDot()` function are `mDotC` and `-mDotE`. In line 125 and 129, `mDotE` and `mDotC` are the mass transfer term in Eqn. 2.15 and Eqn. 2.16. Then we go through the source term in the temperature equation.

Definition of Tsource()

```
179 Foam::tmp<Foam::fvScalarMatrix>
180 Foam::temperaturePhaseChangeTwoPhaseMixtures::constant::TSource() const
181 {
182 
183     const volScalarField& T = mesh_.lookupObject<volScalarField>("T");
184 
185     tmp<fvScalarMatrix> tTSource
186     (
187         new fvScalarMatrix
188         (
189             T,
190             dimEnergy/dimTime
191         )
192     );
193 
194     fvScalarMatrix& TSource = tTSource.ref();
195 
196     const twoPhaseMixtureEThermo& thermo =
197         refCast<const twoPhaseMixtureEThermo>
198         (
199             mesh_.lookupObject<basicThermo>(basicThermo::dictName)
```

```
200         );
201
202     const dimensionedScalar& TSat = thermo.TSat();
203
204     dimensionedScalar L = mixture_.Hf2() - mixture_.Hf1();
205
206     volScalarField limitedAlpha1
207     (
208         min(max(mixture_.alpha1(), scalar(0)), scalar(1))
209     );
210
211     volScalarField limitedAlpha2
212     (
213         min(max(mixture_.alpha2(), scalar(0)), scalar(1))
214     );
215
216     const volScalarField Vcoeff
217     (
218         coeffE_*mixture_.rho1()*limitedAlpha1*L*pos(T - TSat)
219     );
220     const volScalarField Ccoeff
221     (
222         coeffC_*mixture_.rho2()*limitedAlpha2*L*pos(TSat - T)
223     );
224
225     TSource =
226         fvm::Sp(Vcoeff, T) - Vcoeff*TSat
227       + fvm::Sp(Ccoeff, T) - Ccoeff*TSat;
228
229     return tTSource;
230 }
```

Unlike `mDotAlphal()` and `mDotAlphal()`, `fvScalarMatrix` is constructed for `TSource()`, which means it will contribute to the coefficient of the matrix for the temperature equation. In line 204, a `dimensionedScalar L` is created as the difference between the latent heat of the two phases. The return value of `Tsource()` function is actually the `TSource` in line 225. The definition of `TSource` from line 225 to line 227 shows that the `Vcoeff*T` and `Ccoeff*T` are discretized implicitly and the `Vcoeff*TSat` and `Ccoeff*TSat` are discretized explicitly. In line 216 and line 220, we see the mass transfer term is a little bit different from it in the volume fraction and the pressure correction equation. The difference between the latent heat of the first phase and the second phase is multiplied. Then, the `correct()` function for the correction of phase change model and `read()` function for reading the `coeffC` and `coeffE` are defined:

Definition of correct() and read()

```
233 void Foam::temperaturePhaseChangeTwoPhaseMixtures::constant::correct()
234 {
235 }
236
237
238 bool Foam::temperaturePhaseChangeTwoPhaseMixtures::constant::read()
239 {
240     if (temperaturePhaseChangeTwoPhaseMixture::read())
241     {
242         subDict(type() + "Coeffs").readEntry("coeffC", coeffC_);
243         subDict(type() + "Coeffs").readEntry("coeffE", coeffE_);
244
245         return true;
246     }
247
248     return false;
249 }
```

Until now the Lee's phase change model has been explained in detail. In the source code folder in `temperaturePhaseChangeTwoPhaseMixtures` folder, there is an `interfaceHeatResistance` folder. This is the other phase change model, which is not discussed in this report.

Chapter 3

Implementation of SGS term into `interCondensatingEvaporatingFoam`

The implementation of the SGS term starts by copying the `interCondensatingEvaporatingFoam` solver, changing its name to `myInterCondensatingEvaporatingFoam`, and renaming its files and functions accordingly. As described in Chapter 3, α-equation is solved using MULES algorithm in `alphaEqn.H`. To add the source terms to α-equation, we need to modify `alphaEqn.H` file. This section will provide the steps in the implementation of the SGS term named, `myInterCondensating EvaporatingFoam`. Start by copying the `interCondensatingEvaporatingFoam` solver from the OpenFOAM installation as `myInterCondensatingEvaporatingFoam` with these commands:

```
cd $WM_PROJECT_USER_DIR
mkdir -p applications/solvers/multiphase
cd applications/solvers/multiphase
cp -r $FOAM_SOLVERS/multiphase/interCondensatingEvaporatingFoam .
mv interCondensatingEvaporatingFoam myInterCondensatingEvaporatingFoam
cd myInterCondensatingEvaporatingFoam
mv interCondensatingEvaporatingFoam.C myInterCondensatingEvaporatingFoam.C
```

As mentioned before, the files for solving α equation, U equation and other relevant files are in other directories. If we check `Make/options`, we find the files for solving α equation and U equation, the file(`alphaControls.H`) for the parameters of MULES and `alphaEqnSubCycle.H` are in the `interPhaseChangeFoam` source code folder. `setDeltaT.H` and `createAlphaFluxes.H` files are in the `VoF` folder. `correctPhi.H`, `initCorrectPhi.H` and `rhofs.H` are in the `interFoam` source code folder. So here we firstly copy the above necessary files from other folders:

```
cp $FOAM_SOLVERS/multiphase/interPhaseChangeFoam/alphaEqn.H .
cp $FOAM_SOLVERS/multiphase/interPhaseChangeFoam/UEqn.H .
cp $FOAM_SOLVERS/multiphase/interPhaseChangeFoam/alphaControls.H .
cp $FOAM_SOLVERS/multiphase/interPhaseChangeFoam/alphaEqnSubCycle.H .
cp $FOAM_SOLVERS/multiphase/interFoam/correctPhi.H .
cp $FOAM_SOLVERS/multiphase/interFoam/initCorrectPhi.H .
cp $FOAM_SOLVERS/multiphase/interFoam/rhofs.H .
cp $FOAM_SOLVERS/multiphase/VoF/setDeltaT.H .
cp $FOAM_SOLVERS/multiphase/VoF/createAlphaFluxes.H .
```

Rename within the files in Make directory to change the name and path of the excutable:

```
sed -i s/inter/myInter/g Make/files
sed -i s/APPBIN/USER_APPBIN/g Make/files
```

Then change the `interCondensatingEvaporatingFoam` to `myInterCondensatingEvaporatingFoam` in `myInterCondensatingEvaporatingFoam.C`:

```
sed -i s/interCon/myInterCon/g myInterCondensatingEvaporatingFoam.C
```

Rename the library to our library in the source code folder:

```
mv temperaturePhaseChangeTwoPhaseMixtures myTemperaturePhaseChangeTwoPhaseMixtures
```

Then we have to update the `Make/options` files in the source code folder and the `myTemperaturePhaseChangeTwoPhaseMixtures` folder, respectively. Firstly we delete the relative paths for `interPhaseChangeFoam`, `interFoam` solver and `VoF` in the `Make/options` files in the source code folder, since we already copy the needed files to the source folder. We also need to link to the modified library, we change the `Make/options` files as follows:

Make/options in the source code

```
EXE_INC = \
    -ImyTemperaturePhaseChangeTwoPhaseMixtures/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/fvOptions/lnInclude\
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/sampling/lnInclude \
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/twoPhaseMixture/lnInclude \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/interfaceProperties/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude

EXE_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lfiniteVolume \
    -lfvOptions \
    -lmeshTools \
    -lsampling \
    -ldynamicFvMesh \
    -lmyPhaseTemperatureChangeTwoPhaseMixtures \
    -ltwoPhaseMixture \
    -linterfaceProperties \
    -ltwoPhaseProperties \
    -lincompressibleTransportModels \
    -lturbulenceModels \
    -lincompressibleTurbulenceModels \
    -lfluidThermophysicalModels
```

Then rename within the files in the `Make` directory of `myTemperaturePhaseChangeTwoPhaseMixtures` to change the name and path of the library:

```
sed -i s/libphase/libmyPhase/g myTemperaturePhaseChangeTwoPhaseMixtures/Make/files
sed -i s/LIBBIN/USER_LIBBIN/g myTemperaturePhaseChangeTwoPhaseMixtures/Make/files
```

Now we have a local version of the interCondensatingEvaporatingFoam solver. Try to compile it to see that everything works as intended

```
wmake myTemperaturePhaseChangeTwoPhaseMixtures
wmake
```

Then we modify the `alphaEqn.H` to add the SGS source term to the volume fraction equation. First of all, we need to create a new file for SGS term. Create `alphaTUA_WALE.H` file and add the following piece of code for the SGS term to the file:

Code for the SGS term

```
0  //Get the mesh size
1  volScalarField V
2      (
3                IOobject
4            (
5                mesh.V().name(),
6                runTime.timeName(),
7                mesh,
8                IOobject::NO_READ,
9                IOobject::NO_WRITE,
10               false
11           ),
12           mesh,
13           dimensionedScalar(mesh.V().dimensions(), Zero),
14           calculatedFvPatchField<scalar>::typeName
15     );
16        V.ref() = mesh.V();
17 volScalarField delta = pow(V, 1./3);
18
19 //Create the variables needed by SGS term
20 volTensorField gradU(fvc::grad(U));
21 volSymmTensorField Sd(dev(symm(gradU & gradU)));
22 volScalarField magSqrSd(magSqr(Sd));
23 volScalarField nuSGS =
24           sqr(0.5*delta/1.0)*
25              sqrt(
26                  pow(magSqrSd, 3.0)
27                 /(
28                     sqr
29                     (
30                         pow(magSqr(symm(gradU)), 5.0/2.0)
31                       + pow(magSqrSd, 5.0/4.0)
32                     )
33                   + dimensionedScalar
34                     (
35                         "SMALL",
36                         dimensionSet(0, 0, -10, 0, 0),
37                         SMALL
38                     )
39                 )
40             );
41
42 //SGS term
43 volScalarField TUA
44 (
45       nuSGS*fvc::laplacian(alpha1)
46 );
```

Here we implement the SGS term in Eqn. 2.20, since the subgrid scale viscosity is modelled by WALE model. The above code from row 0 to row 40 is applied to create the subgrid scale viscosity in Eqn. 2.21 and the last four rows are for the Eqn. 2.22. Save the file and exit.

Now we have created the file for SGS term, the next step is to add this term to `alphaEqn.H` file. Firstly, we include the file in line 17 and then add the source term we created in line 30.

Code for alpha equation

```
15     if (MULESCorr)
16     {
17         #include "alphaTUA_WALE.H"
18         fvScalarMatrix alpha1Eqn
19         (
20             fv::EulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
21           + fv::gaussConvectionScheme<scalar>
22             (
23                 mesh,
24                 phi,
```

```
25                  upwind<scalar>(mesh, phi)
26              ).fvmDiv(phi, alpha1)
27            - fvm::Sp(divU, alpha1)
28           ==
29              fvm::Sp(vDotvmcAlphal, alpha1)
30            + vDotcAlphal + TUA
31          );
```

Then we have implemented our SGS source term into the volume fraction equation. Try to compile it to see that everything works as intended

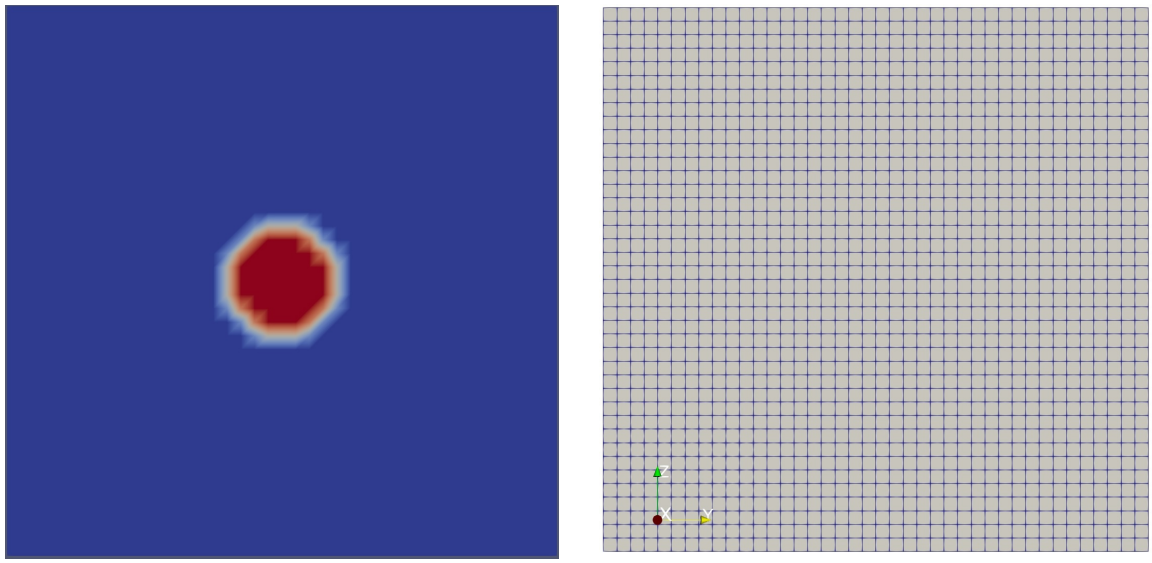
```
wclean
wmake
```

Chapter 4

Test case and results

4.1 Test case setting

In this section we will test the original solver `interCondensatingEvaporatingFoam` and the modified solver `myInterCondensatingEvaporatingFoam` by applying them to a liquid droplet in vapor in a box. An overview of the test case geometry and mesh are shown in Figure 4.1. The droplet is initialized with a specific diameter and position. The emphasis will be on qualitatively evaluating the evaporation in the phase change model and the SGS term in the volume equation. The test case can be found in the supplied material attached to this report, it is named `dropletEvaporation_testCase`. The saturated temperature of the droplet is set to 300K, and the temperature of the internal field is set to 400K. Therefore, the evaporation of the droplet will happen and the results will be shown in the next subsection. If further details of case setting are required by the reader, please refer to the case files.



(a) Computational domain (b) Mesh

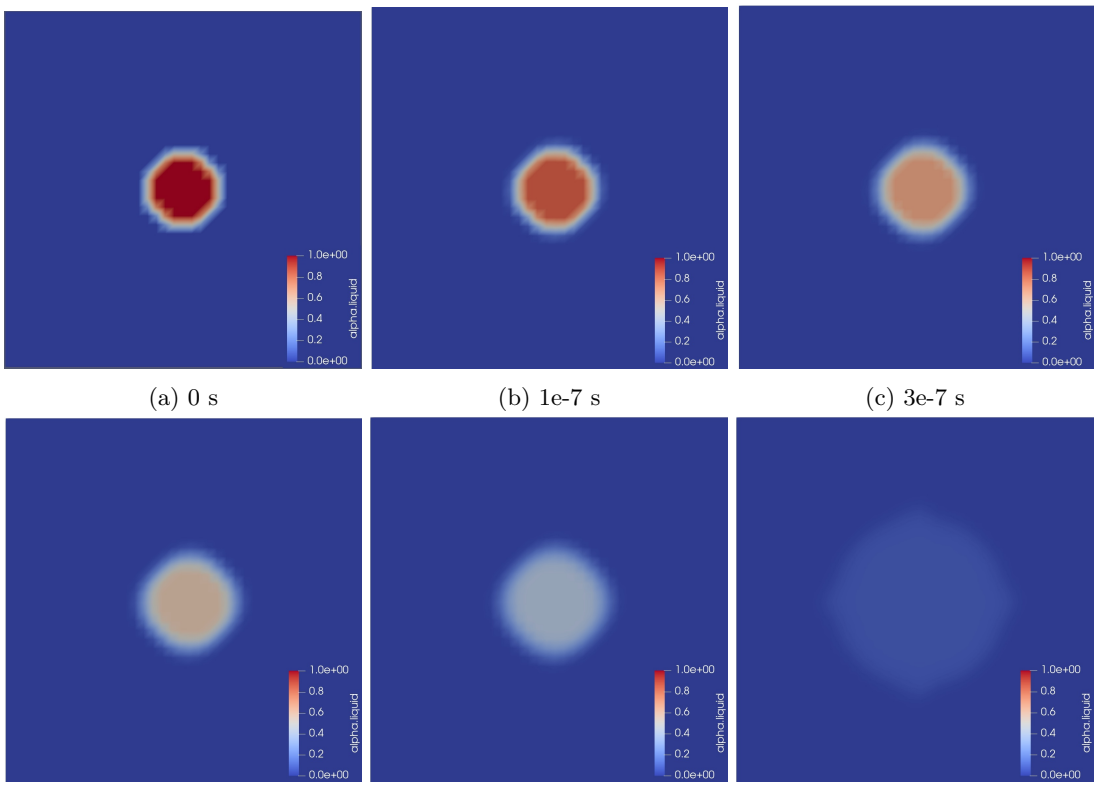
Figure 4.1: Computational domain and mesh

4.2 Results

In order to compare the simulation results without evaporation model and with evaporation model and to study the influence of SGS term on the volume fraction field. The droplet will be simulated by `interFoam`, `interCondensatingEvaporatingFoam` and `myInterCondensatingEvaporatingFoam`. All solvers are interface capturing method, the SGS term is implemented in `myInterCondensatingEvaporatingFoam`.

4.2.1 interCondensatingEvaporatingFoam

Figure 4.2 presents the results by `interCondensatingEvaporatingFoam`. At 1e-7 s, it can be seen the droplet evaporated obviously and the liquid volume fraction decreased. At 1e-6 s, the size of the liquid droplet is becoming lager due to the evaporation. At 1e-5 s, the droplet almost fully evaporated.



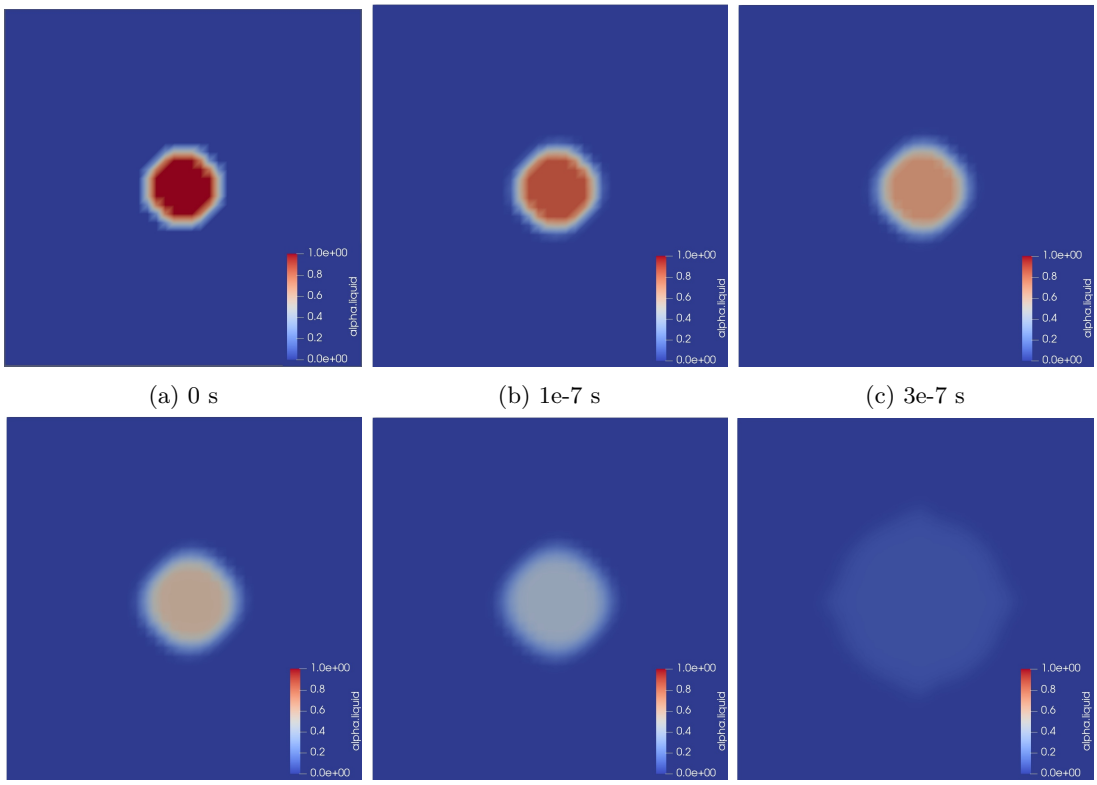
(a) 0 s (b) 1e-7 s (c) 3e-7 s

(d) 5e-7 s (e) 1e-6 s (f) 1e-5 s

Figure 4.2: Contours of liquid volume fraction from `interCondensatingEvaporatingFoam` at different time steps

4.2.2 myInterCondensatingEvaporatingFoam

Figure 4.3 presents the results by `myInterCondensatingEvaporatingFoam`. The difference between `interCondensatingEvaporatingFoam` and `myInterCondensatingEvaporatingFoam` is only the SGS term in the volume fraction equation. If we compare the results in Figure 4.2 (a) and Figure 4.3 (a), it can be seen both droplets have evaporated, the different is very slight at every time step. Overall, the influence of the SGS term is very small in the present case.



(a) 0 s (b) 1e-7 s (c) 3e-7 s

(d) 5e-7 s (e) 1e-6 s (f) 1e-5 s

Figure 4.3: Contours of liquid volume fraction from `myInterCondensatingEvaporatingFoam` at different time steps

Study questions

1. How does the phase interface be tracked in volume of fluid method?
2. What is the assumption of the Lee's phase change model?
3. Why is the mass transfer rate $\dot{m}$ decomposed into $\dot{m}_v$ and $\dot{m}_c$
4. How do you implement source terms into the volume fraction equation?

Bibliography

[1] T. M. Nguyen, R. N. Dahms, L. M. Pickett, and F. Tagliante, "The corrected distortion model for lagrangian spray simulation of transcritical fuel injection," *International Journal of Multiphase Flow*, vol. 148, p. 103927, 2022.

[2] I. Enagi, K. Al-attab, Z. Zainal, and Y. Teoh, "Palm biodiesel spray and combustion characteristics in a new micro gas turbine combustion chamber design," *Energy*, p. 124335, 2022.

[3] Y. Sun, M.-E. Clavel, L. Dressler, Y. Li, K. Nishad, G. Cabot, B. Renou, and A. Sadiki, "Investigation of liquid atomization in a pressure-swirl atomizer using realtime coupled volume of fluid and lagrangian particle tracking methods within large eddy simulation framework," *Proceedings of ILASS-Europe*, 2022.

[4] C. Shao, K. Luo, Y. Yang, and J. Fan, "Detailed numerical simulation of swirling primary atomization using a mass conservative level set method," *International Journal of Multiphase Flow*, vol. 89, pp. 57–68, 2017.

[5] F. Fröde, T. Grenga, V. Le Chenadec, M. Bode, and H. Pitsch, "A three-dimensional cell-based volume-of-fluid method for conservative simulations of primary atomization," *Journal of Computational Physics*, p. 111374, 2022.

[6] D. Zuzio, J.-L. Estivalèzes, and B. DiPierro, "An improved multiscale eulerian–lagrangian method for simulation of atomization process," *Computers & Fluids*, vol. 176, pp. 285–301, 2018.

[7] M. Herrmann, M. Arienti, and M. Soteriou, "The impact of density ratio on the liquid core dynamics of a turbulent liquid jet injected into a crossflow," *Journal of Engineering for Gas Turbines and Power*, vol. 133, no. 6, 2011.

[8] S. Ketterl and M. Klein, "A-priori assessment of subgrid scale models for large-eddy simulation of multiphase primary breakup," *Computers & Fluids*, vol. 165, pp. 64–77, 2018.

[9] W. H. Lee, "Pressure iteration scheme for two-phase flow modeling," *IN" MULTIPHASE TRANSPORT: FUNDAMENTALS, REACTOR SAFETY, APPLICATIONS"*., pp. 407–432, 1980.

[10] F. Nicoud and F. Ducros, "Subgrid-scale stress modelling based on the square of the velocity gradient tensor," *Flow, turbulence and Combustion*, vol. 62, no. 3, pp. 183–200, 1999.

[11] A. Asnaghi, *Developing computational methods for detailed assessment of cavitation on marine propellers*. Chalmers Tekniska Hogskola (Sweden), 2015.