Implementing a non-isothermal interPhaseChangeFoam solver with a thermodynamic cavitation model

Keivan Afshar Ghasemi

Department of Marine Technology (IMT)/ Combustion Kinetics Research Group (ComKin)
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway

January 16, 2024

Contents

Background

A brief exploration of essential background topics for the commencement:

- Cavitation phenomenon
- Numerical methods for cavitation modeling
  - Homogeneous Equilibrium Methods (HEM)
  - Interface Tracking Methods (ITM)
    - Volume of Fluid (VOF)
- Built-in solvers in OpenFOAM for cavitation modeling
  - `cavitatingFoam` solver
  - `interPhaseChangeFoam` (`iPCF`) solver
    - `Kunz` cavitation model
    - `Merkle` cavitation model
    - `SchnerrSauer` cavitation model
- A non-isothermal built-in phase change solver in OpenFOAM
  - `interCondensatingEvaporatingFoam` (`iCEF`) solver
- `thermalInterPhaseChangeFoam` (`tIPCF`) solver
  - Considering the superposed effect of the mechanical and thermal impacts

Comparing the equations of iPCF and iCEF solvers

- Transport equations of iPCF solver
  - Mass conservation

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_j) = 0$$

  - Momentum conservation

$$\frac{\partial}{\partial t}(\rho u_j) + \frac{\partial}{\partial x_j}(\rho u_i u_j) = -\frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}\left((\mu + \mu_t)(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\ )\right) + \rho g$$

  - Volume fraction

$$\frac{\partial(\rho_l \alpha_l)}{\partial t} + \frac{\partial(\rho_l \alpha_l u_j)}{\partial x_j} = \dot{m}^+ + \dot{m}^-$$

- Additional transport equation of iCEF solver
  - Energy conservation

$$\frac{\partial(\rho c_p T)}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_j c_p T) = \frac{\partial}{\partial x_j}(\kappa \frac{\partial T}{\partial x_j}) - (\dot{m}_C + \dot{m}_E)L$$

iPCF and iCEF phase change models

- Cavitation and boiling models
- iPCF solver and its corresponding cavitation models
  - `Kunz`

$$\begin{cases} \dot{m}_{\mathrm{V}} = \dfrac{C_{\mathrm{V}}\rho_{\mathrm{v}}}{(\frac{1}{2}\rho_{\mathrm{l}}U_{\infty}^{2})t_{\infty}} \min\left[p - p_{\mathrm{sat}}, 0\right] \\ \dot{m}_{\mathrm{C}} = \dfrac{C_{\mathrm{C}}\rho_{\mathrm{v}}}{t_{\infty}}\alpha_{l}^{2}\dfrac{\max\left[p - p_{\mathrm{sat}}, 0\right]}{\max\left[p - p_{\mathrm{sat}}, 0.01p_{\mathrm{sat}}\right]} \end{cases}$$

  - `Merkle`

$$\begin{cases} \dot{m}_{\mathrm{V}} = \dfrac{C_{\mathrm{V}}\rho_{\mathrm{l}}}{(\frac{1}{2}\rho_{\mathrm{v}}U_{\infty}^{2})t_{\infty}} \min\left[p - p_{\mathrm{sat}}, 0\right] \\ \dot{m}_{\mathrm{C}} = \dfrac{C_{\mathrm{C}}}{(\frac{1}{2}U_{\infty}^{2})t_{\infty}} \max\left[p - p_{\mathrm{sat}}, 0\right] \end{cases}$$

  - `SchnerrSauer`

$$\begin{cases} \dot{m}_{\mathrm{V}} = C_{\mathrm{V}}(1 + \alpha_{\mathrm{nuc}} - \alpha_{\mathrm{l}})\alpha_{\mathrm{l}}\dfrac{3\rho_{\mathrm{l}}\rho_{\mathrm{v}}}{\rho R_{\mathrm{B}}}\sqrt{\dfrac{2}{3\rho_{\mathrm{l}}}\dfrac{1}{|p - p_{\mathrm{sat}} + 0.01p_{\mathrm{sat}}|}} \min\left[p - p_{\mathrm{sat}}, 0\right] \\ \dot{m}_{\mathrm{C}} = C_{\mathrm{C}}\alpha_{\mathrm{l}}^{2}\dfrac{3\rho_{\mathrm{l}}\rho_{\mathrm{v}}}{\rho R_{\mathrm{B}}}\sqrt{\dfrac{2}{3\rho_{\mathrm{l}}}\dfrac{1}{|p - p_{\mathrm{sat}} + 0.01p_{\mathrm{sat}}|}} \max\left[p - p_{\mathrm{sat}}, 0\right] \end{cases}$$

iPCF and iCEF phase change models

- iCEF solver and its corresponding boiling models
  - `constant`

$$\begin{cases} \dot{m}_{\mathrm{E}} = -C_{\mathrm{E}}\rho_{\mathrm{l}}\max\left[T - T_{\mathrm{sat}}, 0\right] \\ \dot{m}_{\mathrm{C}} = C_{\mathrm{C}}\rho_{\mathrm{v}}\max\left[T_{\mathrm{sat}} - T, 0\right] \end{cases}$$

  - `interfaceHeatResistance`

$$\begin{cases} \dot{m}_{\mathrm{E}} = -\dfrac{A_{\mathrm{interface}}R}{L(\alpha_{\mathrm{l}} + \mathrm{SMALL})}\max\left[T - T_{\mathrm{sat}}, 0\right] \\ \dot{m}_{\mathrm{C}} = \dfrac{A_{\mathrm{interface}}R}{L(\alpha_{\mathrm{v}} + \mathrm{SMALL})}\max\left[T_{\mathrm{sat}} - T, 0\right] \end{cases}$$

Theory of a thermodynamic cavitation model

- Rayleigh-Plesset bubble dynamics equation

$$\frac{p_\mathrm{v} - p}{\rho_\mathrm{l}} = R\frac{\mathrm{d}^2 R_\mathrm{B}}{\mathrm{d}t^2} + \frac{3}{2}\left(\frac{\mathrm{d}R_\mathrm{B}}{\mathrm{d}t}\right)^2 + \frac{4\nu_\mathrm{l}}{R_\mathrm{B}}\frac{\mathrm{d}R_\mathrm{B}}{\mathrm{d}t} + \frac{2\sigma}{\rho_\mathrm{l} R_\mathrm{B}}$$

- Simplifying the Rayleigh-Plesset equation
  - If $We = \rho_\mathrm{l} U^2 l/\sigma \gg 1$ and $Re = \rho_\mathrm{l} U/\nu > Re_\mathrm{laminar}$

$$\frac{p_\mathrm{v} - p}{\rho_\mathrm{l}} = R\frac{\mathrm{d}^2 R_\mathrm{B}}{\mathrm{d}t^2} + \frac{3}{2}\left(\frac{\mathrm{d}R_\mathrm{B}}{\mathrm{d}t}\right)^2$$

  - If $Kn = \lambda/l = \nu/l\sqrt{\pi M/2\Re T} \ll 1$

$$\frac{p_\mathrm{v} - p}{\rho_\mathrm{l}} = \frac{3}{2}\left(\frac{\mathrm{d}R_\mathrm{B}}{\mathrm{d}t}\right)^2 \rightsquigarrow \frac{\mathrm{d}R_\mathrm{B}}{\mathrm{d}t} = \sqrt{\frac{2}{3}\frac{p_\mathrm{v} - p}{\rho_\mathrm{l}}}$$

Theory of a thermodynamic cavitation model

- Zwart cavitation model

$$\begin{cases} \dot{m}_{\mathrm{V}} = C_{\mathrm{V}} \dfrac{3\alpha_{\mathrm{nuc}}\rho_{\mathrm{v}}}{R_{\mathrm{B}}} \sqrt{\dfrac{2}{3\rho_{\mathrm{l}}} \dfrac{1}{|p - p_{\mathrm{sat}} + 0.01 p_{\mathrm{sat}}|}} \min\left[p - p_{\mathrm{sat}}, 0\right] \\ \dot{m}_{\mathrm{C}} = C_{\mathrm{C}} \dfrac{3\rho_{\mathrm{v}}}{R_{\mathrm{B}}} \sqrt{\dfrac{2}{3\rho_{\mathrm{l}}} \dfrac{1}{|p - p_{\mathrm{sat}} + 0.01 p_{\mathrm{sat}}|}} \max\left[p - p_{\mathrm{sat}}, 0\right] \end{cases}$$

- Plesset-Zwick bubble dynamics equation

$$\frac{\mathrm{d}R_{\mathrm{B}}}{\mathrm{d}t} = \sqrt{\frac{3}{\pi}} \frac{\rho_{\mathrm{l}} c_{\mathrm{p}} (T - T_{\mathrm{sat}})}{\rho_{\mathrm{v}} L} \sqrt{\frac{a_{\mathrm{l}}}{t}}$$

- ZwartExtended cavitation model (developed by Zhang Yao et al.)

$$\begin{cases} \dot{m}_{\mathrm{V}} = C_{\mathrm{V}} \dfrac{3\alpha_{\mathrm{nuc}}\rho_{\mathrm{v}}}{R_{\mathrm{B}}} \left(\sqrt{\dfrac{2}{3} \dfrac{1}{\rho_{\mathrm{l}} |p - p_{\mathrm{sat}} + \mathrm{SMALL}|}} \min\left[p - p_{\mathrm{sat}}, 0\right] - \sqrt{\dfrac{3}{\pi}} \dfrac{\rho_{\mathrm{l}} c_{\mathrm{p}}}{\rho_{\mathrm{v}} L} \sqrt{\dfrac{a_{\mathrm{l}}}{t}} \max\left[T - T_{\mathrm{sat}}, 0\right] \right) \\ \dot{m}_{\mathrm{C}} = C_{\mathrm{C}} \dfrac{3\rho_{\mathrm{v}}}{R_{\mathrm{B}}} \left(\sqrt{\dfrac{2}{3} \dfrac{1}{\rho_{\mathrm{l}} |p - p_{\mathrm{sat}} + \mathrm{SMALL}|}} \max\left[p - p_{\mathrm{sat}}, 0\right] + \sqrt{\dfrac{3}{\pi}} \dfrac{\rho_{\mathrm{l}} c_{\mathrm{p}}}{\rho_{\mathrm{v}} L} \sqrt{\dfrac{a_{\mathrm{l}}}{t}} \max\left[T_{\mathrm{sat}} - T, 0\right] \right) \end{cases}$$

Comparing the source codes of iPCF and iCEF solvers

- Common files between iPCF and iCEF solvers
  - UEqn.H
  - alphaEqn.H
  - alphaContronls.H
  - alphaEqnSubCycle.H
- Transport equations
  - pEqn.H

phiHbyA definition - iCEF/pEqn.H

```
surfaceScalarField phiHbyA
    (
        "phiHbyA",
        (fvc::interpolate(HbyA) & mesh.Sf())
      + fvc::interpolate(rho*rAU)*fvc::ddtCorr(U, phi)
    );
```

phiHbyA definition - iPCF/pEqn.H

```
surfaceScalarField phiHbyA
    (
        "phiHbyA",
        fvc::flux(HbyA )
      + fvc::interpolate(rho*rAU)*fvc::ddtCorr(U, phi)
    );
```

Comparing the source codes of iPCF and iCEF solvers

- Transport equations
  - pEqn.H

p_rghEqn definition - iCEF/pEqn.H

```
fvScalarMatrix p_rghEqn
    (
        fvc::div(phiHbyA)
      - fvm::laplacian(rAUf, p_rgh)
      ==
        vDotv + vDotc
    );
```

p_rghEqn definition - iPCF/pEqn.H

```
fvScalarMatrix p_rghEqn
    (
        fvc::div(phiHbyA) - fvm::laplacian(rAUf, p_rgh)
      - (vDotvP - vDotcP)*(mixture->pSat() - rho*gh)
      + fvm::Sp(vDotvP - vDotcP, p_rgh)
    );
```

Comparing the source codes of iPCF and iCEF solvers

- Main algorithms
  - createFields.H files

p definition - iCEF/createFields.H

```
1 // Creating e based thermo
2 autoPtr<twoPhaseMixtureEThermo> thermo
3 (
4     new twoPhaseMixtureEThermo(U, phi)
5 );
6
7 ...
8
9 volScalarField& p = thermo->p();
```

p definition - iPCF/createFields.H

```
1  volScalarField p
2  (
3      IOobject
4      (
5          "p",
6          runTime.timeName(),
7          mesh,
8          IOobject::NO_READ,
9          IOobject::AUTO_WRITE
10     ),
11     p_rgh + rho*gh
12 );
```

Comparing the source codes of iPCF and iCEF solvers

- Phase change models and their directories

```
temperaturePhaseChangeTwoPhaseMixtures
    ├── Make
    ├── "A Boiling Model"
    ├── temperaturePhaseChangeTwoPhaseMixtures
    ├── thermoIncompressibleTwoPhaseMixture
    └── twoPhaseMixtureEThermo
```

```
phaseChangeTwoPhaseMixtures
    ├── Make
    ├── "A Cavitation Model"
    └── phaseChangeTwoPhaseMixture
```

  - `temperaturePhaseChangeTwoPhaseMixtures` VS. `phaseChangeTwoPhaseMixture`
  - `thermoIncompressibleTwoPhaseMixture` VS. `incompressibleTwoPhaseMixture`
  - `twoPhaseMixtureEThermo` VS. `basicThermo`

Preparation of the base directory

Preparing the base directory for implementing the new solver and its cavitation model

```
foam
cp -r --parents "applications/solvers/multiphase/interCondensatingEvaporatingFoam/"
    "$WM_PROJECT_USER_DIR"
ufoam
cd applications/solvers/multiphase
mv interCondensatingEvaporatingFoam thermalInterPhaseChangeFoam
cd thermalInterPhaseChangeFoam
mv interCondensatingEvaporatingFoam.C thermalInterPhaseChangeFoam.C
sed -i s/interCondensatingEvaporatingFoam/thermalInterPhaseChangeFoam/g
    thermalInterPhaseChangeFoam.C
sed -i s/interCondensatingEvaporatingFoam/thermalInterPhaseChangeFoam/g Make/files
sed -i s/FOAM_APPBIN/FOAM_USER_APPBIN/g Make/files
cd temperaturePhaseChangeTwoPhaseMixtures
rm -r interfaceHeatResistance
mv constant ZwartExtended
mv ZwartExtended/constant.C ZwartExtended/ZwartExtended.C
mv ZwartExtended/constant.H ZwartExtended/ZwartExtended.H
sed -i s/constant/ZwartExtended/g ZwartExtended/ZwartExtended.H
sed -i s/constant/ZwartExtended/g ZwartExtended/ZwartExtended.C
sed -i s/constant/ZwartExtended/g Make/files
sed -i s/interfaceHeatResistance.*//g Make/files
sed -i s/libphaseTemperatureChangeTwoPhaseMixtures/libthermalPhaseChangeTwoPhaseMixtures/g
    Make/files
sed -i s/FOAM_LIBBIN/FOAM_User_LIBBIN/g Make/files
cd ..
```

Preparation of the base directory

Solver main directory - Make/options

```
interPhaseChangeFoam = $(FOAM_SOLVERS)/multiphase/interPhaseChangeFoam
interFoam = $(FOAM_SOLVERS)/multiphase/interFoam
VoF = $(FOAM_SOLVERS)/multiphase/VoF

EXE_INC = \
    -I$(interPhaseChangeFoam) \
    -I$(interFoam) \
    -I$(VoF) \
    -ItemperaturePhaseChangeTwoPhaseMixtures/lnInclude \
    -I$(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$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude \
    -I$(LIB_SRC)/phaseSystemModels/reactingEuler/saturationModels/lnInclude
```

Preparation of the base directory

Solver main directory - `Make/options`

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

Implementing ZwartExtended cavitation model

- twoPhaseMixtureEThermo class

twoPhaseMixtureEThermo.H

```
1  ...
2
3  #include "saturationModel.H"
4
5  ...
6
7  protected:
8
9      // Protected Data
10
11         //- Saturation model
12         autoPtr<saturationModel> saturationModelPtr_;
13
14         //- Critical pressure [Pa]
15         dimensionedScalar Pc_;
16
17         //- Critical temperature [K]
18         dimensionedScalar Tc_;
19
20         //- Triple-point pressure [Pa]
21         dimensionedScalar Pt_;
22
23         //- Triple-point temperature [K]
24         dimensionedScalar Tt_;
```

Implementing ZwartExtended cavitation model

- twoPhaseMixtureEThermo class

twoPhaseMixtureEThermo.H

```
1  ...
2
3          // Access to thermodynamic state variables
4
5              //- Return the saturation temperature based on the chosen model
6              virtual tmp<volScalarField> Tsat
7              (
8                  const volScalarField& p
9              ) const;
10
11             //- Return the saturation pressure based on the chosen model
12             virtual tmp<volScalarField> pSat
13             (
14                 const volScalarField& T
15             ) const;
```

Implementing ZwartExtended cavitation model

- twoPhaseMixtureEThermo class

twoPhaseMixtureEThermo.H

```
            //- Return const-access to the critical pressure
            const dimensionedScalar& Pc() const
            {
                return Pc_;
            }

            //- Return const-access to the critical temperature
            const dimensionedScalar& Tc() const
            {
                return Tc_;
            }

            //- Return const-access to the triple-point pressure
            const dimensionedScalar& Pt() const
            {
                return Pt_;
            }

            //- Return const-access to the triple-point temperature
            const dimensionedScalar& Tt() const
            {
                return Tt_;
            }
...
```

Implementing ZwartExtended cavitation model

- twoPhaseMixtureEThermo class

twoPhaseMixtureEThermo.C

```
1  ...
2
3  // * * * * * * * * * * * * * * * * * * Constructor * * * * * * * * * * * * * * * * * * //
4
5  Foam::twoPhaseMixtureEThermo::twoPhaseMixtureEThermo
6  (
7      const volVectorField& U,
8      const surfaceScalarField& phi
9  )
10 :
11     basicThermo(U.mesh(), word::null),
12     thermoIncompressibleTwoPhaseMixture(U, phi),
13
14     saturationModelPtr_
15     (
16         saturationModel::New
17         (
18             static_cast<const basicThermo&>(*this).subDict("saturationModel"),
19             U.mesh()
20         )
21     ),
22 ...
```

Implementing ZwartExtended cavitation model

- twoPhaseMixtureEThermo class

twoPhaseMixtureEThermo.C

```
1  ...
2
3  // * * * * * * * * * * * * * * * * * * Constructor * * * * * * * * * * * * * * * * * * //
4
5  Foam::twoPhaseMixtureEThermo::twoPhaseMixtureEThermo
6  (
7      const volVectorField& U,
8      const surfaceScalarField& phi
9  )
10 :
11     basicThermo(U.mesh(), word::null),
12     thermoIncompressibleTwoPhaseMixture(U, phi),
13
14     ...
15
16     Pc_("Pc", dimPressure, static_cast<const basicThermo&>(*this)),
17     Tc_("Tc", dimTemperature, static_cast<const basicThermo&>(*this)),
18     Pt_("Pt", dimPressure, static_cast<const basicThermo&>(*this)),
19     Tt_("Tt", dimTemperature, static_cast<const basicThermo&>(*this))
20 {}
```

Implementing ZwartExtended cavitation model

- twoPhaseMixtureEThermo class

twoPhaseMixtureEThermo.C

```
1  ...
2
3  // * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * //
4
5  ...
6
7  Foam::tmp<Foam::volScalarField> Foam::twoPhaseMixtureEThermo::Tsat
8  (
9      const volScalarField& p
10 ) const
11 {
12     return saturationModelPtr_->Tsat(p);
13 }
14
15
16 Foam::tmp<Foam::volScalarField> Foam::twoPhaseMixtureEThermo::pSat
17 (
18     const volScalarField& T
19 ) const
20 {
21     return saturationModelPtr_->pSat(T);
22 }
23
24 ...
```

Implementing ZwartExtended cavitation model

- twoPhaseMixtureEThermo class

twoPhaseMixtureEThermo.C

```
1  // * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * //
2
3  ...
4
5  bool Foam::twoPhaseMixtureEThermo::read()
6  {
7      if (basicThermo::read() && thermoIncompressibleTwoPhaseMixture::read())
8      {
9          basicThermo::readEntry("Pc", Pc_);
10         basicThermo::readEntry("Tc", Tc_);
11         basicThermo::readEntry("Pt", Pt_);
12         basicThermo::readEntry("Tt", Tt_);
13
14         return true;
15     }
16
17     return false;
18 }
19
20
21 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

Implementing ZwartExtended cavitation model

- temperaturePhaseChangeTwoPhaseMixture class

temperaturePhaseChangeTwoPhaseMixture.H

```
    // Member Functions

        ...

        //- Return the pressure term of the mass condensation and vaporisation
        //  rates as coefficients to multiply (p - pSat)
        virtual Pair<tmp<volScalarField>> mDotP() const = 0;

        //- Return the temperature term of the mass condensation and
        //  vaporisation rates as coefficients
        virtual Pair<tmp<volScalarField>> mDotT() const = 0;

        ...

        //- Return the pressure term of the volumetric condensation and
        //  vaporisation rates as coefficients
        virtual Pair<tmp<volScalarField>> vDotP() const;

        //- Return the temperature term of the volumetric condensation and
        //  vaporisation rates as coefficients
        virtual Pair<tmp<volScalarField>> vDotT() const;

...
```

Implementing ZwartExtended cavitation model

- temperaturePhaseChangeTwoPhaseMixture class

temperaturePhaseChangeTwoPhaseMixture.C

```
// * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * * //

...

Foam::Pair<Foam::tmp<Foam::volScalarField>>
Foam::temperaturePhaseChangeTwoPhaseMixture::vDotP() const
{
    dimensionedScalar pCoeff(1.0/mixture_.rho1() - 1.0/mixture_.rho2());
    Pair<tmp<volScalarField>> mDotP = this->mDotP();

    return Pair<tmp<volScalarField>>(pCoeff*mDotP[0], pCoeff*mDotP[1]);
}

Foam::Pair<Foam::tmp<Foam::volScalarField>>
Foam::temperaturePhaseChangeTwoPhaseMixture::vDotT() const
{
    dimensionedScalar pCoeff(1.0/mixture_.rho1() - 1.0/mixture_.rho2());
    Pair<tmp<volScalarField>> mDotT = this->mDotT();

    return Pair<tmp<volScalarField>>(pCoeff*mDotT[0], pCoeff*mDotT[1]);
}

...
```

Implementing ZwartExtended cavitation model

- ZwartExtended class

ZwartExtended.H

```
"Look at the actual script!"
```

ZwartExtended.C

```
"Look at the actual script!"
```

Implementing thermalInterPhaseChangeFoam solver

- Pressure equation modification

pEqn.H

```
// Update the pressure BCs to ensure flux consistency
    constrainPressure(p_rgh, U, phiHbyA, rAUf);

    Pair<tmp<volScalarField>> vDotP = mixture->vDotP();
    const volScalarField& vDotcP = vDotP[0]();
    const volScalarField& vDotvP = vDotP[1]();

    Pair<tmp<volScalarField>> vDotT = mixture->vDotT();
    const volScalarField& vDotcT = vDotT[0]();
    const volScalarField& vDotvT = vDotT[1]();

    volScalarField limitedT (min(max(T, thermo->Tt()), thermo->Tc()));

    while (pimple.correctNonOrthogonal())
    {
        fvScalarMatrix p_rghEqn
        (
            fvc::div(phiHbyA) - fvm::laplacian(rAUf, p_rgh)
          - (vDotvP - vDotcP)*(thermo->pSat(limitedT) - rho*gh)
          + fvm::Sp(vDotvP - vDotcP, p_rgh)
          ==
            vDotvT + vDotcT
        );
    ...
```

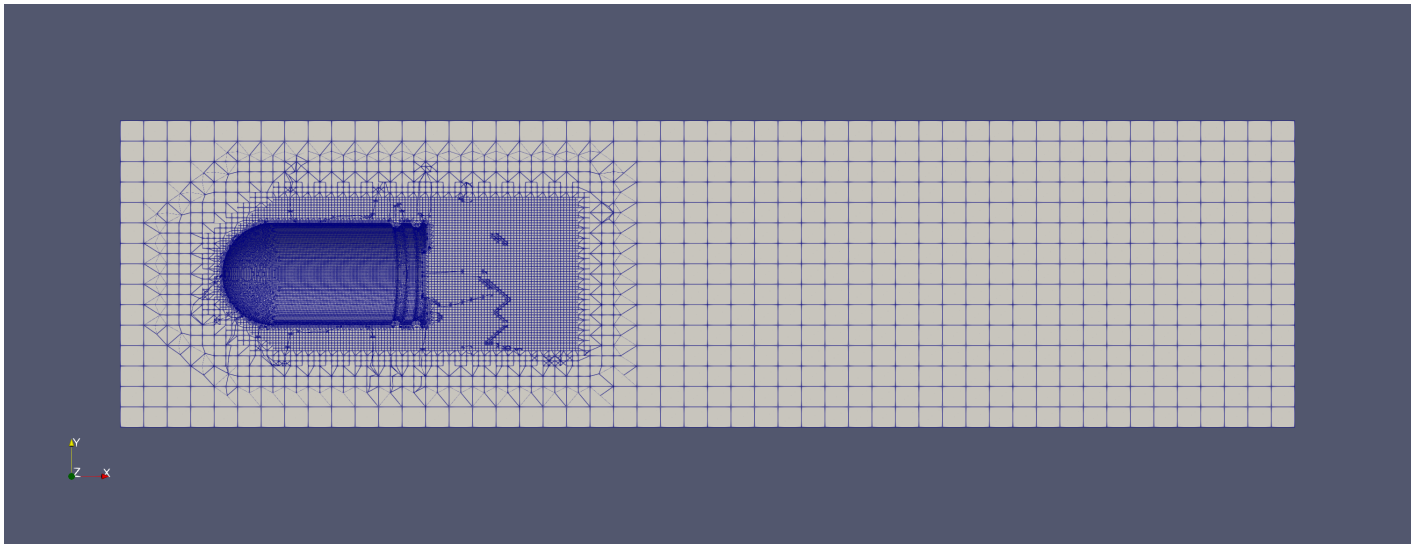
Compilation process

Compiling the new solver and its cavitation model

```
ufoam
cd applications/solvers/multiphase/thermalInterPhaseChangeFoam
wclean
wclean temperaturePhaseChangeTwoPhaseMixtures
wmake temperaturePhaseChangeTwoPhaseMixtures
wmake
```

Test cases

- Original `interPhaseChangeFoam` tutorial
  - `cavitatingBullet`



Cross-sectional view: computational domain of the original tutorial

- Setting up the higher temperature test case for `iPCF` solver
  - Preparing the base case

Copying the original tutorial

```
1 run
2 cp -r "$WM_PROJECT_DIR/tutorials/multiphase/interPhaseChangeFoam/cavitatingBullet/"
      "cavitatingBullet_HighTemperature"
```

Test cases

- Setting up the higher temperature test case for `iPCF` solver
  - Modifying the saturation pressure and transport properties of the phases

`constant/transportProperties` dictionary

```
pSat            69783;   // Saturation pressure at 363 K

sigma           0.061;   // Water surface tension at 363 K

water // Saturated liquid water properties at 363 K (CoolProp)
{
    transportModel  Newtonian;
    nu              3.26e-07;
    rho             965.396;
}

vapour // Saturated vapour water properties at 363 K (CoolProp)
{
    transportModel  Newtonian;
    nu              2.817e-05;
    rho             0.422;
}
...
```

Test cases

- Setting up the higher temperature test case for `iPCF` solver
  - Changing the number of subdomains for parallel simulation

`system/decomposeParDict` dictionary

```
numberOfSubdomains 8;
method          simple;

coeffs
{
    n           (1 2 4);
}
```

  - Changing the `Allrun` script for running in parallel mode

`Allrun` file

```
...
runApplication decomposePar

# Run the solver
# runApplication $(getApplication)
runParallel $(getApplication)
#------------------------------------------------------------------------
```

Test cases

- Setting up the test case for `tIPCF` solver
  - Preparing the base case

Preparing the original tutorial

```
run
cp -r "$WM_PROJECT_DIR/tutorials/multiphase/interPhaseChangeFoam/cavitatingBullet/"
     "thermalCavitatingBullet"
cd thermalCavitatingBullet
cp 0.orig/p_rgh 0.orig/p
cp 0.orig/alpha.water 0.orig/T
touch constant/phaseChangeProperties
touch constant/thermophysicalProperties
```

Test cases

- Setting up the test case for `tIPCF` solver
  - Pressure field dictionary

0.orig/p dictionary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * //

dimensions      [ 1 -1 -2 0 0 0 0 ];

internalField   uniform 1e05;

boundaryField
{
    inlet
    {
        type            calculated;
        value           $internalField;
    }
```

0.orig/p dictionary

```

    outlet
    {
        type            calculated;
        value           $internalField;
    }

    walls
    {
        type            symmetry;
    }

    bullet
    {
        type            calculated;
        value           $internalField;
    }
}

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

Test cases

- Setting up the test case for `tIPCF` solver
  - Temperature field dictionary

0.orig/T dictionary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 1 0 0 0];

internalField   uniform 363;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           $internalField;
    }
```

0.orig/T dictionary

```

    outlet
    {
        type            inletOutlet;
        inletValue      $internalField;
    }

    walls
    {
        type            symmetry;
    }

    bullet
    {
        type            zeroGradient;
    }
}


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

Test cases

- Setting up the test case for `tIPCF` solver
  - Phase change properties

`constant/phaseChangeProperties` dictionary

```
1  FoamFile
2  {
3      version     2.0;
4      format      ascii;
5      class       dictionary;
6      object      phaseChangeProperties;
7  }
8  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
9
10 phaseChangeTwoPhaseModel ZwartExtended;
11
12 ZwartExtendedCoeffs
13 {
14     Cc          2e-03;
15     Cv          10;
16     alphaNuc    5e-04;
17     rNuc        1e-06;
18     tG          5e-03;
19     TInf        363;
20 }
21
22 // ************************************************************************* //
```

Test cases

- Setting up the test case for `tIPCF` solver
  - Thermophysical properties

`constant/thermophysicalProperties` dictionary

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

//- Water critical- and triple-point pressures and temperatures (CoolProp)
Pc    2.2064e7;
Tc    647.096;
Pt    611.655;
Tt    273.16;

saturationModel
{
    type    Antoine;
    A       23.196;
    B       -3816.447;
    C       -46.13;
}
// ************************************************************************* //
```

Test cases

- Setting up the test case for `tIPCF` solver
  - Transport properties

`constant/transportProperties` dictionary

```
phases          (water vapour);

sigma           0.061; // Water surface tension at 363 K

water // Saturated liquid water properties at 363 K (CoolProp)
{
    transportModel  Newtonian;
    nu              3.26e-07;
    rho             965.396;

    Cp              4205.135;
    Cv              3821.188;
    kappa           0.673; // Thermal conductivity [J/s/m/K]
    hf              376408.297;
}

...
```

Test cases

- Setting up the test case for `tIPCF` solver
  - Transport properties

constant/transportProperties dictionary

```
...

vapour // Saturated vapour water properties at 363 K (CoolProp)
{
    transportModel  Newtonian;
    nu              2.817e-05;
    rho             0.422;

    Cp              2042.423;
    Cv              1531.255;
    kappa           0.024;
    hf              2659285.627;
}

Prt             0.7;

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

Test cases

- Setting up the test case for `tIPCF` solver
  - Changing the default solver and adding two new entries in control dictionary

`system/controlDict` dictionary

```
1  ...
2
3  application     thermalInterPhaseChangeFoam;
4
5  ...
6
7  maxAlphaCo      1.0;
8
9  maxDeltaT       1e-02;
10
11 // ************************************************************************* //
```

Test cases

- Setting up the test case for `tIPCF` solver
  - Including the required numerical schemes for the temperature field

`system/fvSchemes` dictionary

```
...

divSchemes
{
...
    div(rhoCpPhi,T)     Gauss linearUpwind grad(T);
...
    div((muEff*dev(T(grad(U))))  Gauss linear;
    div((interpolate(cp)*rhoPhi),T) Gauss linearUpwind grad(T);
}

...

wallDist
{
    method          meshWave;
}
```

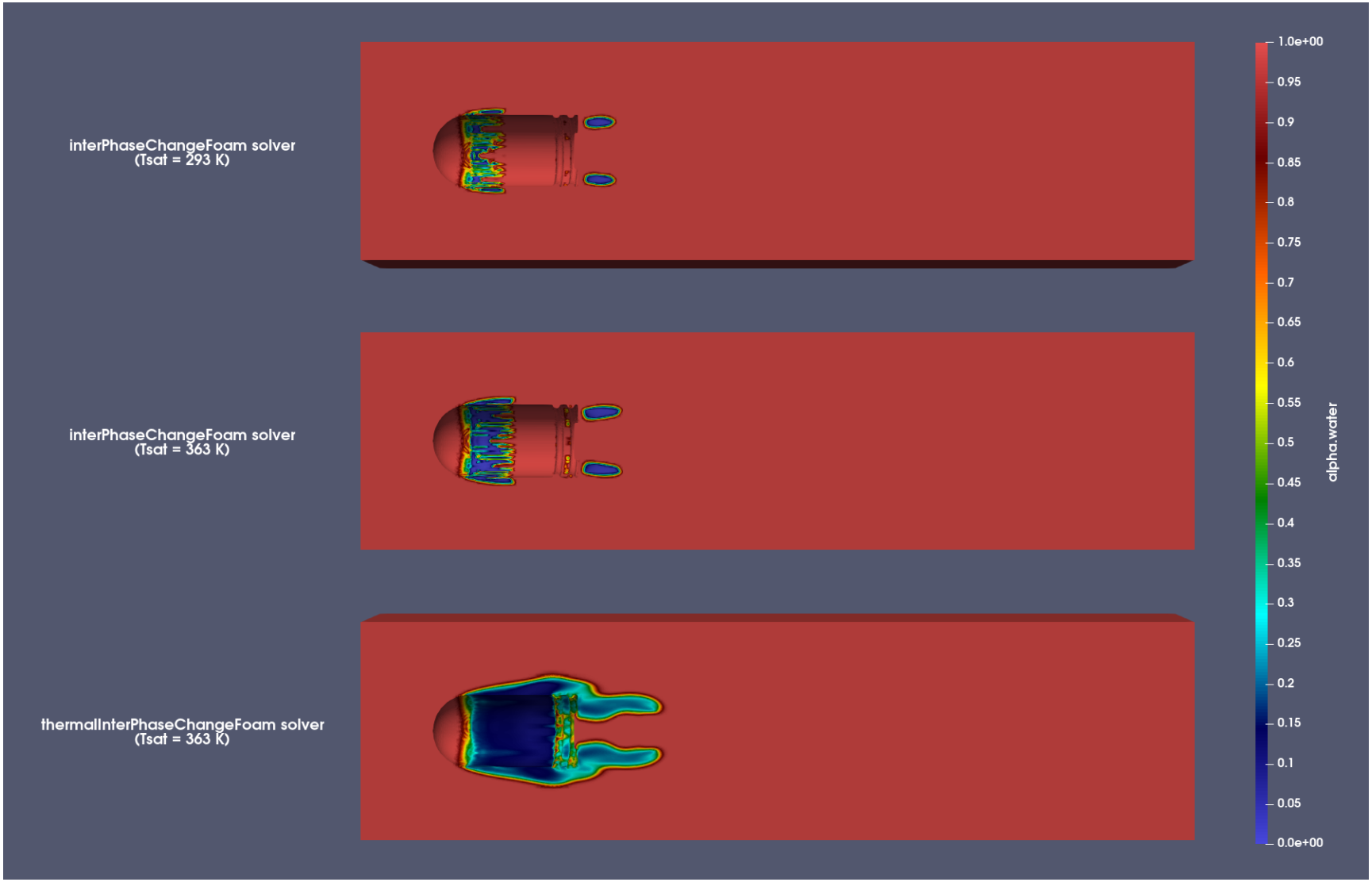
Test cases

- Setting up the test case for `tIPCF` solver
  - Including the required control entries for the energy equation

`system/fvSolution` dictionary

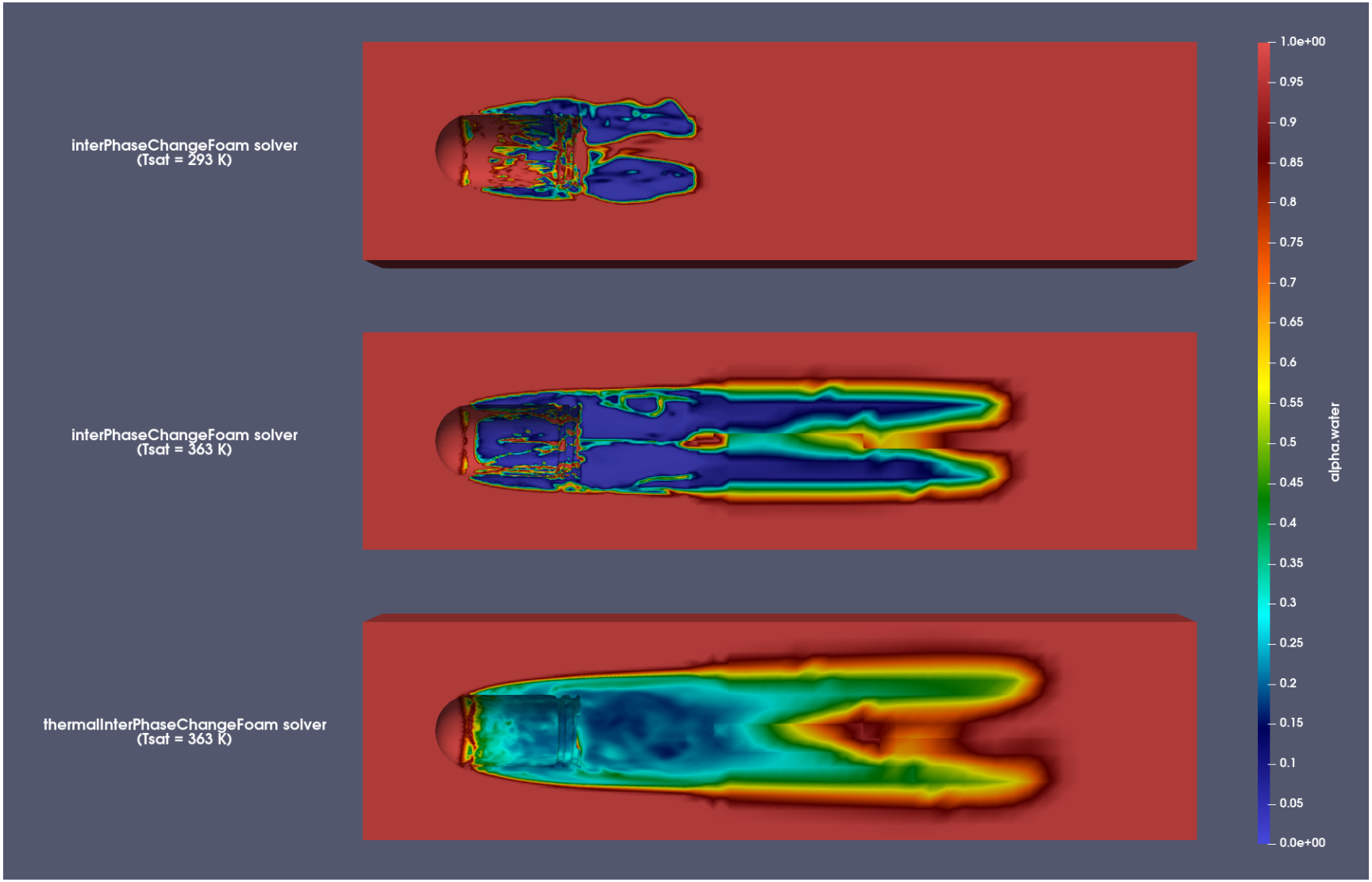
```
...
solvers
{
    ".*(rho|rhoFinal)"
    {
        solver          diagonal;
    }

...

    "T.*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-6;
        relTol          0.0;
    }

...

}
...
```

Results



Volume fraction of the carrier fluid at 1 ms with different solvers and under different conditions

Results



Volume fraction of the carrier fluid at 8 ms with different solvers and under different conditions