Adding heat transfer to InterFoam©
(OpenFoam©2.3.0)

Damiano Natali

Wolf Dynamics Inc.

May 19, 2014
This offering is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OpenFOAM© and OpenCFD© trade marks.
Acknowledgements

These slides and the tutorials presented are based upon personal experience, OpenFOAM© source code, OpenFOAM© user guide, OpenFOAM© programmer’s guide, and presentations from previous OpenFOAM© training sessions and OpenFOAM© workshops. We gratefully acknowledge the following OpenFOAM© users/developers for their consent to use their material:

- Hrvoje Jasak. Wikki Ltd,
- Hakan Nilsson. Department of Applied Mechanics, Chalmers University of Technology,
1. Why editing an existing solver?

2. Editing the library

3. Editing the solver

4. Setting up the case
It may happen there is no native solver in OpenFOAM© that suits entirely our needs, so we may consider editing an existing one. In this tutorial we will modify the interFoam solver to take into account also heat fluxes, i.e. to solve the momentum, continuity and the volume fraction equation along with the energy equation

\[ \rho \left( \frac{\partial u}{\partial t} + u \cdot \nabla u \right) = -\nabla p + \mu \nabla^2 u \]

\[ \nabla \cdot u = 0 \]

\[ \frac{\partial \alpha}{\partial t} + u \cdot \nabla \alpha = 0 \]

\[ \rho c_p \left( \frac{\partial T}{\partial t} + u \cdot \nabla T \right) = \nabla \cdot (k \nabla T) + q \]

The new solver will be called interTempFoam.
If we had considered a one-phase case, our task would just consist in implementing the fourth equation in the solver source code and adding two additional space-independent parameters (the heat capacity $c_p$ and the thermal diffusivity $k$). The presence of two different phases not only double the number of parameters, but also impose to consider the interface position, as it already happens for the density $\rho$ or the viscosity $\mu$.

To handle the parameter spatial location InterFoam calls at runtime the shared library incompressibleTwoPhaseMixture, that now needs to be modified to adapt the change.

Thermal diffusivity $k$ will not be defined directly, but will be calculated using the heat capacity $c_p$ and the Prandtl number $Pr$:

$$ k = \rho \frac{c_p}{Pr} $$
First of all we have to modify the `incompressibleTwoPhaseMixture` library. The spirit of OpenFOAM© for customized applications is to place them in a different path from the original ones:

- standard applications: `OpenFOAM-2.3.0/applications`
- user-edited applications: `<username>-2.3.0/applications`

To make things easier, OpenFOAM© provides the environmental variable `$WM_PROJECT_USER_DIR` that corresponds to the user folder `<username>-2.3.0/` and `$FOAM_APP` that goes to the standard application folder. So the first thing to do is to copy the original folder:

- `cd $WM_PROJECT_USER_DIR`
- `mkdir -p src/transportModels/`
- `cd src/transportModels/`
- `cp -rp $FOAM_SRC/transportModels/incompressible .`
Why editing an existing solver?

Editing the library

Editing the solver

Setting up the case

mv incompressibleTwoPhaseMixture
myIncompressibleTwoPhaseMixture

cd myIncompressibleTwoPhaseMixture (from now on this will be our “root” directory)

Now we can rename both the .c and the .h files. The header file contains included files and declarations. We first have to modify the declaration of the class incompressibleTwoPhaseMixture to include both the definition of \( c_p \) and \( Pr \) for both phases:

dimensionedScalar rho1_;  
dimensionedScalar rho2_;  
dimensionedScalar cp1_;  
dimensionedScalar cp2_;  
dimensionedScalar Pr1_;  
dimensionedScalar Pr2_;
Then the original solver defines several member functions in order to return private attributes of the `incompressibleTwoPhaseMixture` class:

```cpp
const dimensionedScalar& rho2() const
{
    return rho2_;}

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

and analogously for `cp2_`, `Pr1_` and `Pr2_`. The underscore "_" is a common practice of the OpenFOAM© programming.
Let us turn to the .c file. First of all rename the included .H filename. Then, accordingly to the definition in the .H file, we need to modify both the constructor and the read() function defined to initialize the fields. So in the constructor section add:

```cpp
rho2_("rho", dimDensity, nuModel2_->viscosityProperties().lookup("rho")),
cp1_("cp", dimensionSet(0, 2, -2, -1, 0, 0, 0),
nuModel1_->viscosityProperties().lookup("cp")),
```
And in the `read()` function at the end of the file add:

```cpp
rho2_("rho", dimDensity, nuModel2_->viscosityProperties().lookup("rho")),
cp1_("cp", dimensionSet(0, 2, -2, -1, 0, 0, 0),
nuModel1_->viscosityProperties().lookup("cp")),
```

So the library will look for the keyword `cp` and will assign it to the variable `cp1_` with dimension given by the construct `dimensionSet`. 
Once we know $c_p$ and $Pr$ we have to calculate $k$ on the faces of every cell in order to evaluate the thermal flux $k \nabla T$, so let us add the function `kappaf()` after the similar `muf()` and `nuf()` functions:

```cpp
Foam::tmp<Foam::surfaceScalarField>
Foam::incompressibleTwoPhaseMixture::kappaf() const
{
    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())
            )
        )
    );
}
```
So far we have modified the `incompressibleTwoPhaseMixture` to take into account $c_p$ and $Pr$ (and indirectly also $k$). In order to make out customized library usable we need to compile it and store the output file in a directory known by OpenFOAM©. To do so let’s edit the path to our library in the `../Make/files` and the destination folder of the shared object:

```
myIncompressibleTwoPhaseMixture/myIncompressibleTwoPhaseMixture.C
```

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

Then we can compile the library with the command `wclean` (to clear previous compilations) and `wmake libso` from the `$WM_PROJECT_USER_DIR/src/transportModels/incompressible` folder.
Now we can focus on the solver. As previously done for the library, let’s make our work copy of the original solver:

- cd $WM_PROJECT_USER_DIR
- mkdir -p applications/solvers/multiphase/
- cd applications/solvers/multiphase/
- cp -rp $FOAM_APP/solvers/multiphase/interFoam .
- cd interFoam (from now on this will be our “root” directory)
- mv interFoam interTempFoam

Now we can rename both the .C and the .H files.
Before editing the solver itself, we have to edit the `createFields.H` file in order to create the dimensioned scalar $c_p$ and fields $T$ and $\rho c_p$ and the flux $\rho \phi c_p$, all involved in the energy equation.

Copy the `p_rgh` object in the object $T$:

```cpp
Info<< "Reading field T\n" << endl;
volScalarField T
(
    IOobject
    ("T",
     runTime.timeName(),
     mesh,
     IOobject::MUST_READ,
     IOobject::AUTO_WRITE
    ),
    mesh
);
```
Read the \( \text{cp1} \) and \( \text{cp2} \) objects from the \texttt{twoPhaseProperties} object, in the same way it has been done for \( \text{rho1} \) and \( \text{rho2} \):

\[
\begin{align*}
\text{const dimensionedScalar}&\quad \text{rho1} = \texttt{twoPhaseProperties.rho1}(); \\
\text{const dimensionedScalar}&\quad \text{rho2} = \texttt{twoPhaseProperties.rho2}(); \\
\text{const dimensionedScalar}&\quad \text{cp1} = \texttt{twoPhaseProperties.cp1}(); \\
\text{const dimensionedScalar}&\quad \text{cp2} = \texttt{twoPhaseProperties.cp2}();
\end{align*}
\]
Now let's create the $\rho c_p$ field needed to create the temporal term $\frac{d}{dt}(\rho c_p, T)$ of the energy equation. To do that, copy the similar object $\rho$ (needed to compute $\frac{d}{dt}(\rho, U)$).

```cpp
volScalarField rhoCp
(
    IOobject
    (
        "rhoCp",
        runTime.timeName(),
        mesh, // I0object
        I0object::READ_IF_PRESENT
    ),//
    alpha1*rho1*cp1 + alpha2*rho2*cp2,
    alpha1.boundaryField().types()
);
rhoCp.oldTime();
```
Now we need $\rho c_p \phi$ in order to compute the heat fluxes between faces $ddt(\rho \phi, T)$ for the energy equation. To do that, copy the mass flux $\rho \phi$ (needed for the momentum equation) into the $\rho c_p \phi$ object.

```cpp
surfaceScalarField rhoCpPhi
(
    IOobject
    (    "rhoCpPhi",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    fvc::interpolate(rhoCp)*phi
);```

Damiano Natali  Adding heat transfer to InterFoam©(OpenFoam©2.3.0)
After defining this quantities, we have to calculate them. This is done in alphaEqnSubCycle.H (that calls in turn alphaEqn.H). So in alphaEqn.H let us calculate rhoCpPhi

\[
\begin{align*}
\text{rhoPhi} & = \text{tphiAlpha}() \times (\text{rho1} - \text{rho2}) + \text{phi} \times \text{rho2}; \\
\text{rhoCpPhi} & = \text{tphiAlpha}() \times (\text{rho1} \times \text{cp1} - \text{rho2} \times \text{cp2}) + \text{phi} \times \text{rho2} \times \text{cp2};
\end{align*}
\]

And in alphaEqnSubCycle.H do the same for rhoCp

\[
\begin{align*}
\text{rho} & = \text{alpha1} \times \text{rho1} + \text{alpha2} \times \text{rho2}; \\
\text{rhoCp} & = \text{alpha1} \times \text{rho1} \times \text{cp1} + \text{alpha2} \times \text{rho2} \times \text{cp2};
\end{align*}
\]
Finally we can write the energy equation. As you see, this is the simplest part of the work. Like \texttt{UEqn.H}, this is done in a separate file that we will call \texttt{TEqn.H}:

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

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

TEqn.solve();
```

Note that at the beginning of the file we have first initialized the object \texttt{kappaf} with the value defined in \texttt{twoPhaseProperties}. As for \texttt{UEqn.H}, also this file needs to be included in the main \texttt{.C} file with \texttt{#include "TEqn.H"} after the pressure-velocity loop.
Then we can compile our customized solver with the command `wclean` (to clear previous compilations) and `wmake` from the $WM_PROJECT_USER_DIR/run/applications/solvers/multiphase/interTempFoam folder.
In order to test our solver, the easiest way is to customize an existing tutorial. We choose the famous "damBreak" tutorial. As for libraries and solvers, let us copy the case in our user folder. In this case, the alias run comes very handy.

- run
- cp -rp $FOAM_TUTORIALS/multiphase/
  - interFoam/laminar/damBreak .
- mv damBreak/ damTempBreak/
- cd damTempBreak/
In order to test our solver, the easiest way is to customize an existing tutorial. We choose the famous "damBreak" tutorial. As for libraries and solvers, let us copy the case in our user folder. In this case, the alias run comes very handy.

- run
- cp -rp $FOAM_TUTORIALS/multiphase/
  interFoam/laminar/damBreak .
- mv damBreak/ damTempBreak/
- cd damTempBreak/

To indicate what application should be run here, change the application field of the system/controlDict file in interTempFoam.
To set up the case, the first thing to do is to define the initial and boundary conditions of the $T$ field. So let us copy the 0/alpha file in the 0/T with the right dimensions and with all but the defaultFaces (empty) boundary conditions of type zeroGradient (next slide).
Why editing an existing solver?
Editing the library
Editing the solver
Setting up the case

dimensions [0 0 0 1 0 0 0];
internalField uniform 0;
boundaryField
{
    leftWall
    {
        type zeroGradient;
    }
    rightWall
    {
        type zeroGradient;
    }
    lowerWall
    {
        type zeroGradient;
    }
    atmosphere
    {
        type zeroGradient;
    }
    defaultFaces
    {
        type empty;
    }
}
Moreover, we have to set the numerical scheme used to evaluate the heat fluxes, so in the `system/fvSchemes` file we have to add:

\[
\text{div}(\rho \Phi, U) \quad \text{Gauss linearUpwind} \quad \text{grad}(U);
\]
\[
\text{div}(\rho C_p \Phi, T) \quad \text{Gauss linearUpwind} \quad \text{grad}(T);
\]

And the solver for the $T$ equation, that needs to be defined in `system/fvSolutions`

\[
T
\{
\begin{align*}
\text{solver} & \quad \text{BICCG}; \\
\text{preconditioner} & \quad \text{DILU}; \\
\text{tolerance} & \quad 1e-07; \\
\text{relTol} & \quad 0;
\end{align*}
\}
\]
If we want to set our initial condition with the `setFields` utility, we have to edit also the `system/setFieldsDict` file:

```plaintext
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
    volScalarFieldValue T 0
);
regions
(
    boxToCell
    {
        box (0 0 -1) (0.1461 0.292 1);
        fieldValues
        (            
            volScalarFieldValue alpha.water 1
        );
    }
    boxToCell
    {
        box (0.2 0.3 -1) (0.3 0.4 1);
        fieldValues
        (            
            volScalarFieldValue T 300
        );
    }
);
```