OpenFOAM solver openInjMoldSim version conversion

João Romero

March 27, 2018

Abstract

Description and explanation of steps required to compile the solver openInjMoldSim in OpenFOAM 5. Solver was written for OpenFOAM 3.0.1.

mojHeRhoThermo.C

When compiling the code with no changes this is the first error shown in the log:

```
./crossWlfInSorodno/mojHeRhoThermo.C:37:50: error: binding const Foam::
    Field<double> to reference of type Foam::scalarField& {aka Foam::Field
    <double>&} discards qualifiers
      scalarField& TCells = this->T_.internalField();
```

My first tendency based on this error message was to turn TCells into a variable of type scalarField instead of scalarField& (a reference to scalarField). Doing so compiled this line and similar ones in mojHeRhoThermo.C with no warning. However, TCells must be a reference variable otherwise the solver is unable to copy its values further ahead in the code. I was able to compile the whole solver unaware of this mistake, which took me a long time to find out.

The file mojHeRhoThermo.C is in fact heavily based on file thermophysicalModels/basic/rhoThermo/heRhoThermo.C from the OpenFOAM 3.0.1 source code. This file is also present in OpenFOAM 5 at the same location. Thus, a comparison between the two versions indicates all the changes required in mojHeRhoThermo.C. In particular, constant reference variables should be initialized without calling any method. So

```
const scalarField& pCells = this->p_.internalField();
const scalarField& strigCells = this->strig_.internalField();
```

becomes

```
const scalarField& pCells = this->p_;
const scalarField& strigCells = this->strig_;
```

Non-constant references are called with method primitiveFieldRef. So

```
scalarField& TCells = this->T_.internalField();
scalarField& psiCells = this->psi_.internalField();
scalarField& rhoCells = this->rho_.internalField();
scalarField& muCells = this->mu_.internalField();
scalarField& alphaCells = this->alpha_.internalField()
```

becomes

```
scalarField& TCells = this->T_.primitiveFieldRef();
scalarField& psiCells = this->psi_.primitiveFieldRef();
scalarField& rhoCells = this->rho_.primitiveFieldRef();
scalarField& muCells = this->mu_.primitiveFieldRef();
scalarField& alphaCells = this->alpha_.primitiveFieldRef();
```

References to boundary fields are called with method boundaryFieldRef() instead of boundaryField(). In the OpenFOAM 5 version of source file heRhoThermo.C the fields are initialized to a third object first, likely in order to reduce method calling overhead. So, if we look at the pressure field for example,

```
forAll(this->T_.boundaryField(), patchi)
{
        fvPatchScalarField& pp = this->p_.boundaryField()[patchi];
        ...
```

becomes

```
volScalarField::Boundary& pBf = this->p_.boundaryFieldRef();
...
forAll(this->T_.boundaryField(), patchi)
{
        fvPatchScalarField& pp = pBf[patchi];
        ...
```

With these changes mojHeRhoThermo.C is successfully compiled.

Thermodynamic variables units changes

After the changes to mojHeRhoThermo.C the first error on the log becomes

```
./mojThermo/mojThermoI.H:224:26: error: const class Foam::species::
    mojThermo<Foam::hConstThermo<Foam::polymerPVT<Foam::specie> >, Foam::
    sensibleInternalEnergy> has no member named cp
      return this->cp(p, T)/this->W();
```

Class mojThermo is from the custom solver and it is defined in the three files of folder mojThermo/. It has no member named cp but it inherits members from its template parameter class Thermo. From the declaration of mojThermo at mojThermo.H:

```
template<class Thermo, template<class> class Type>
class mojThermo
:
    public Thermo,
    public Type<mojThermo<Thermo, Type> >
```

From the compile log we see that class Thermo is class hConstThermo in the particular instance where the error occurs. Class hConstThermo is defined in the source code of both versions of OpenFOAM at folder thermophysicalModels/specie/thermo/hConst/.

In OpenFOAM 3.0.1 the member cp of class hConstThermo is declared at file thermophysicalModels/specie/thermo/hConst/hConstThermo.H:

```
//- Heat capacity at constant pressure [J/(kmol K)]
inline scalar cp(const scalar p, const scalar T) const;
```

This is where the solver originally got the method cp(p, T) at the line where the compilation error now occurs.

However, in OpenFOAM 5 class hConstThermo has no member cp but instead has a member called Cp, declared in hConstThermo.H as well:

```
//- Heat capacity at constant pressure [J/(kg K)]
inline scalar Cp(const scalar p, const scalar T) const;
```

Although both define the heat capacity at constant pressure of the thermophysical properties model (in this case the hConst model) they do so in different units and the method's different name (uppercase C instead of lower case c) is meant to reflect that.

So, the issue in this case is that in OpenFOAM 5 thermophysical properties such as the heat capacity, the enthalpy and the entropy are defined in units of kg^{-1} instead of $kmol^{-1}$. The change in units is given by the species molecular weight W, so that for example $Cp = \frac{cp}{W}$.

Class mojThermo has its methods defined in terms of the fundamental properties in OpenFOAM 3.0.1 thermophysical properties models: cp, ha, hs, hc and s. For OpenFOAM 5 the fundamental proprieties are Cp, Ha, Hs, Hc and S which are the same but in different units. Thus, the methods in mojThermo at files mojThermo.H and mojThermoI.H must be changed to reflect that. The class is heavily based on OpenFOAM's default class thermo at folder thermophysicalModels/specie/thermo/thermo/. Thus comparing files thermo.H and thermoI.H in that folder between both versions of OpenFOAM shows the changes needed in mojThermo.H and mojThermoI.H to reflect the new fundamental proprieties of OpenFOAM 5.

After the changes to the mojThermo class the first error is now

```
./crossWlfInSorodno/crossWLFTransportI.H:209:23: error: const class Foam
    ::crossWLFTransport<Foam::species::mojThermo<Foam::hTabularThermo<Foam
    ::polymerPVT<Foam::specie> >, Foam::sensibleInternalEnergy> > has no
    member named Cp
      return kappa(p, T)/this->Cp(p, T);
```

Class crossWLFTransport is declared in file crossWlfInSorodno/crossWLFTransport.H as does not have any definition of the heat capacity Cp/cp. However, the class does inherit members from its template parameter class.

```
template<class Thermo>
class crossWLFTransport
:
    public Thermo
```

In this case class Thermo is class mojThermo which originally had a member Cp defined based on the fundamental propriety cp. Conforming to OpenFOAM 5 it now it has a member cp defined based on the fundamental propriety Cp. So, what is missing is the definition of Cp in class hTabularThermo which defines a custom thermophysical properties model.

In hTabularThermo.H and hTabularThermoI.H members cp, hs and hc must be replaced by Cp, Hs and Hc respectively, with the appropriate units changes. Member cp returned $cpTable(T,p) * this->W()$; and Cp must now return only $cpTable(T,p)$. Member hs returned $hc() * this->W()$; and Hs must now return $Hc() * this->W()$. hc returned $Hf\_$ and Hc must return $Hf\_/this->W()$. This

should make the results equal to those from the original version of the solver although there is a lack of consistency with the units. Hs and Hc have supposedly the same units ($J \cdot kmol^{-1}$) yet Hs is equal to Hc multiplied by the molecular weight, which has units $kg \cdot kmol^{-1}$. This was the case in the original version of the solver as well. Class hTabularTHermo is based (although loosely) on class hPolynomialThermo in OpenFOAM source code at folder thermophysicalModels/specie/thermo/hPolynomial/.

After the changes to hTabularThermo the first error is now

```
/opt/openfoam5/src/thermophysicalModels/specie/lnInclude/hConstThermoI.H
    :98:37: error: Cp is not a member of Foam::polymerPVT<Foam::specie>
     return Cp_ + EquationOfState::Cp(p, T);
```

This line is on the source code of OpenFOAM itself. It's inside the definition of method hConstThermo::Cp. The equivelent line in OpenFOAM 3.0.1 is thermophysicalModels/specie/lnInclude/hConstThermoI.H:109 inside the definition of method hConstThermo::cp. It simply wields

```
return Cp_;
```

So, the term EquationOfState::Cp(p, T), which raises the error, is simply not present in OpenFOAM 3.0.1.

polymerPVT is a class that defines a new equation of state (the tait equation). If one compares any equation of state class between the two versions of OpenFOAM (for example, the perfect gas equation at thermophysicalModels/specie/equationOfState/perfectGas) there are two methods defined for each equation of state in OpenFOAM 5 that do not exist in OpenFOAM 3.0.1. For example, at perfectGas.H in OpenFOAM 5:

```
//- Return enthalpy departure [J/kg]
inline scalar H(const scalar p, const scalar T) const;

//- Return Cp departure [J/(kg K]
inline scalar Cp(scalar p, scalar T) const;
```

These methods should be added to polymerPVT.H and polymerPVTI.H.

These functions are not the enthalpy and the heat capacity as a whole, but their departure functions. Their values seem to always appear as an added value to the respective property. Since the original code did not include these functions I returned 0 in both of them. I am unsure if the variable Cp_ in the original OpenFOAM 3.0.1 code already includes these departure values but so far the solutions obtained in the tested cases are equal. However, this issue should be investigated further.

Besides adding these two methods there is again an unit change. In files polymerPVT.H and polymerPVTI.H methods s and cpMcv (cp - cv) should be changed to S and CpMCv (Cp - Cv) respectively. The molecular weight W is the factor difference in both cases. Any equation of state in folder thermophysicalModels/specie/equationOfState can be used as a base of comparison between the two versions, to help modify class polymerPVT correctly.

mojBasicThermo.C

After the changes to class polymerPVT the first error in the log is now

```
crossWlfInSorodno/mojBasicThermo.C:52:27: error: GeometricBoundaryField
    in Foam::volScalarField {aka class Foam::GeometricField<double, Foam::
    fvPatchField, Foam::volMesh>} does not name a type
```

```
const volScalarField::GeometricBoundaryField& tbf =
```

This error is once again related to the differences in the methods of class GeometricField between the two OpenFOAM versions. In OpenFOAM 5 the boundary field is given by member Boundary instead of GeometricBoundaryField. mojBasicThermo.C is heavilly based on file thermophysicalModels/basic/basicThermo/basicThermo.C in the source code of OpenFOAM. Comparing this file between the two OpenFOAM versions once again gives insight to the changes that must be made in mojBasicThermo.C. In particular, GeometricBoundaryField& should be replaced by Boundary& and dimensionedInternalField() should be replaced by internalField().

headerOk

Next, the first error becomes

```
createFields.H:205:20: error: class Foam::IOobject has no member named
    headerOk
  if (!sldDictIO.headerOk())
```

From the source code for both versions at OpenFOAM/db/IOobject/IOobject.H it can verified that the method headerOK() has been replaced with the template method typeHeaderOk(const bool checkType = true). Since it is a template method, it requires a template parameter as well. So, sldDictIO.headerOk() should be replaced by sldDictIO.typeHeaderOk<IOdictionary>() at both occurences of headerOK() in createFields.H and at the occurrence in file openInjMoldSim.C.

DimensionedInternalField

After correcting createFields.H and openInjMoldSim.C the first error becomes

```
alphaEqns.H:9:9: error: DimensionedInternalField is not a member of Foam
    ::volScalarField {aka Foam::GeometricField<double, Foam::fvPatchField,
    Foam::volMesh>}
        volScalarField::DimensionedInternalField Sp
```

Again, this arises from the changes to class GeometricField in the two versions. DimensionedInternalField should be replaced by Internal in both occurrences in file alphaEqns.H

Boundary Field reference

Next, the first error becomes

```
pEqn.H:33:5: error: no matching function for call to setSnGrad(const Foam
    ::GeometricField<double, Foam::fvPatchField, Foam::volMesh>::Boundary
    &, Foam::tmp<Foam::FieldField<Foam::fvsPatchField, double> >)
  );
  ^
In file included from /opt/openfoam5/src/finiteVolume/lnInclude/fvCFD.H
    :17:0,
                 from openInjMoldSim.C:38:
```

```
/opt/openfoam5/src/finiteVolume/lnInclude/
    fixedFluxPressureFvPatchScalarField.H:168:17: note: candidate:
    template<class GradBC> void Foam::setSnGrad(Foam::GeometricField<
    double, Foam::fvPatchField, Foam::volMesh>::Boundary&, const Foam::
    FieldField<Foam::fvsPatchField, double>&)
      inline void setSnGrad
                  ^
/opt/openfoam5/src/finiteVolume/lnInclude/
    fixedFluxPressureFvPatchScalarField.H:168:17: note:   template
    argument deduction/substitution failed:
In file included from openInjMoldSim.C:121:0:
pEqn.H:28:28: note:   cannot convert p_rgh.Foam::GeometricField<Type,
    PatchField, GeoMesh>::boundaryField<double, Foam::fvPatchField, Foam::
    volMesh>() (type const Foam::GeometricField<double, Foam::fvPatchField
    , Foam::volMesh>::Boundary) to type Foam::GeometricField<double, Foam
    ::fvPatchField, Foam::volMesh>::Boundary&
         p_rgh.boundaryField(),
                             ^
In file included from /opt/openfoam5/src/finiteVolume/lnInclude/fvCFD.H
    :17:0,
                 from openInjMoldSim.C:38:
/opt/openfoam5/src/finiteVolume/lnInclude/
    fixedFluxPressureFvPatchScalarField.H:184:17: note: candidate:
    template<class GradBC> void Foam::setSnGrad(Foam::GeometricField<
    double, Foam::fvPatchField, Foam::volMesh>::Boundary&, const Foam::tmp
    <Foam::FieldField<Foam::fvsPatchField, double> >&)
      inline void setSnGrad
                  ^
/opt/openfoam5/src/finiteVolume/lnInclude/
    fixedFluxPressureFvPatchScalarField.H:184:17: note:   template
    argument deduction/substitution failed:
In file included from openInjMoldSim.C:121:0:
pEqn.H:28:28: note:   cannot convert p_rgh.Foam::GeometricField<Type,
    PatchField, GeoMesh>::boundaryField<double, Foam::fvPatchField, Foam::
    volMesh>() (type const Foam::GeometricField<double, Foam::fvPatchField
    , Foam::volMesh>::Boundary) to type Foam::GeometricField<double, Foam
    ::fvPatchField, Foam::volMesh>::Boundary&
         p_rgh.boundaryField(),
```

This error basically means that p_rgh.boundaryField() in pEqn.H:28 should be of type GeometricField::Boundary& and is instead of type const GeometricField::Boundary&. Checking OpenFOAM/-fields/GeometricFields/GeometricField/GeometricField.H is OpenFOAM 5 it can be concluded that in OpenFOAM 5 a non-constant reference to the boundary field is returned by the method boundaryFieldRef(). Thus, p_rgh.boundaryField() should be replaced by p_rgh.boundaryFieldRef()

tmp non-contant reference

Next, the first error becomes

```
pEqn.H:50:70: error: passing const Foam::fvMatrix<double> as this
    argument discards qualifiers [-fpermissive]
         deleteDemandDrivenData(p_rghEqnComp1().faceFluxCorrectionPtr());
```

This error comes from calling a non-constant method through a constant variable. The compiler complains because it has no idea if the method is going to change the variable or not.

p_rghEqnComp1 (and p_rghtEqnComp2) is a variable of class tmp<fvScalarMatrix>[1] since they are declared as such in pEqn.H

```
tmp<fvScalarMatrix> p_rghEqnComp1;
tmp<fvScalarMatrix> p_rghEqnComp2;
```

p_rghEqnComp1() calls method operator()() of class tmp which is declared in source file OpenFOAM/memory/tmp/tmp.H. In OpenFOAM 5 tmp.H has the following lines:

```
#ifdef NON_CONST_TMP
//- Deprecated non-const dereference operator.
//  Use ref() where non-const access is required
inline T& operator()();
#endif

//- Const dereference operator
inline const T& operator()() const;
```

Thus, the method operator()() that returns a non-constant only exists if NON_CONST_TMP is defined when compiling OpenFOAM. This is probably not the case otherwise we wouldn't get the compilation error above. Thus, one must use instead the method ref() as indicated in comments. In instances where a compilation error arises p_rghEqnComp1() and p_rghEqnComp2() should be replaced by p_rghEqnComp1.ref() and p_rghEqnComp2.ref() respectively.[2]

Turbulence models

After the previous changes there is only a single compilation error in the log

```
mojTurbulentFluidThermoModel/mojTurbulentFluidThermoModels.C:35:21: fatal
    error: laminar.H: Ficheiro ou directoria inexistente
compilation terminated.
```

In OpenFOAM 5 the class laminar from file TurbulenceModels/turbulenceModels/laminar/laminar.H has been replaced with class laminarModel from TurbulenceModels/turbulenceModels/laminar/laminarModel/laminarModel.H. However, laminar.H in OpenFOAM 3.0.1 implements a full laminar model, while laminarModel.H in OpenFOAM 5 is an abstract class from which the actual model will built upon. Laminar models in OpenFOAM 5 are implemented the same way RAS and LES

[1] fvScalarMatrix is just a different name for fvMatrix<double>

[2] I only made the substitution in instances where a compilation arises because the author might have meant a constant reference in other instances

models are implemented in both versions, and mojTurbulentFluidThermoModel implementation must reflect that.

All four files in folder mojTurbulentFluidThermoModel/ are heavily based on those in folder TurbulenceModels/compressible/turbulentFluidThermoModels in OpenFOAM's source code. Thus, a comparison between both versions of OpenFOAM will indicate the changes that need to be done in mojTurbulentFluidThermoModel/, in particular how to implement laminar models in OpenFOAM 5.

With the changes to the turbulent model openInjMoldSim successfully compiles in OpenFOAM 5.

mojLibForces

The mojForces library defines a function object for postProcessing results from openInjMoldSim. It is not necessary to compile mojForces to compile openInjMoldSim.

The library is heavily based on the forces function object from OpenFOAM's source code. In OpenFOAM 3.0.1 this is located in postProcessing/functionObjects/forces/ but in OpenFOAM 5 it is located in functionObjects/forces.

There are many differences between the two versions. However, mojLibForces presents very few changes from the OpenFOAM forces library. Thus, I think the best way to compile this library in OpenFOAM 5 is to delete all files and then copy the files in functionObjects/forces in OpenFOAM 5 source code and then do the following:

1. Rename forces/forces.C to forces/mojForces.C
2. In forces/mojForces.C replace #include "turbulentFluidThermoModel.H" with #include "mojTurbulentFluidThermoModel.H". Replace all occurrences of fluidThermo with mojFluidThermo and all occurrences of basicThermo with mojBasicThermo.
3. In Make/files replace forces.C with mojForces.C, add the following lines

   ```
   ../crossWlfInSorodno/mojBasicThermo.C
   ../crossWlfInSorodno/mojFluidThermo.C
   ```

   and replace -LIB = $(FOAM_LIBBIN)/libforces with +LIB = $(FOAM_USER_LIBBIN)/libmojforces01
4. In Make/options add to EXE_INC the lines

   ```
   -I.. \
   -I../crossWlfInSorodno \
   -I../mojTurbulentFluidThermoModel \
   ```

This is all that is required for mojLibForces to successfully compile in OpenFOAM 5.