JKU
JOHANNES KEPLER
UNIVERSITY LINZ

Introduction to open source CFD

Session 4/4

Gijsbert Wierink
gijsbert.wierink@jku.at

Christian Doppler Laboratory on Particulate Flow Modelling
Johannes Kepler University | Linz | Austria

Disclaimer

This course is offered non-commercially as part of course work at the Department of Particulate Flow Modelling (PFM) at the Johannes Kepler University (JKU) in Linz, Austria. This offereing is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD ® trade marks.

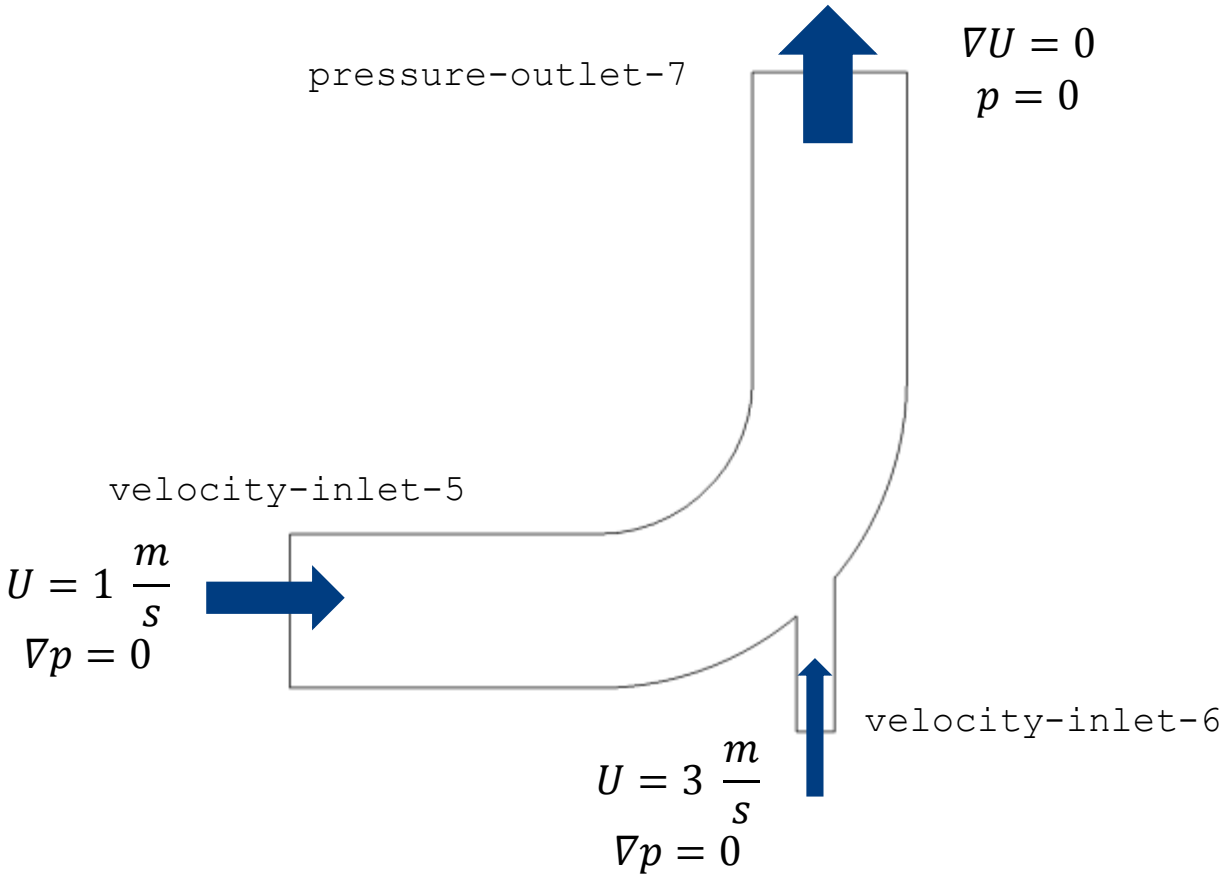
Introduction to open source CFD
Outline

- Creating your own physics library
  - Run a base case
  - Create a modelling library
  - Create a solver
  - Create a test case
  - Online sampling (function object)
- Project work (Wed 18.02.2015 presentation)
- (Miscillaneous)

CREATING A PHYSICS LIBRARY

Creating a physics library
Initial case

- We will use a standard OpenFOAM tutorial: mixing elbow
- Incompressible, transient (`icoFoam`)



Creating a physics library
Initial case

- Go to your run directory

```
> run
```

- Copy the tutorial

```
> cp -r $FOAM_TUTORIALS/incompressible/icoFoam/elbow .
```

- Change directory into the case

```
> cd elbow
```

Creating a physics library
Initial case

- The case contains a Fluent/Ansys mesh

```
> ls -1
0
Allrun
Allclean
constant
elbow.msh
system
```

- Convert the fluent mesh

```
> fluentMeshToFoam elbow.msh
```

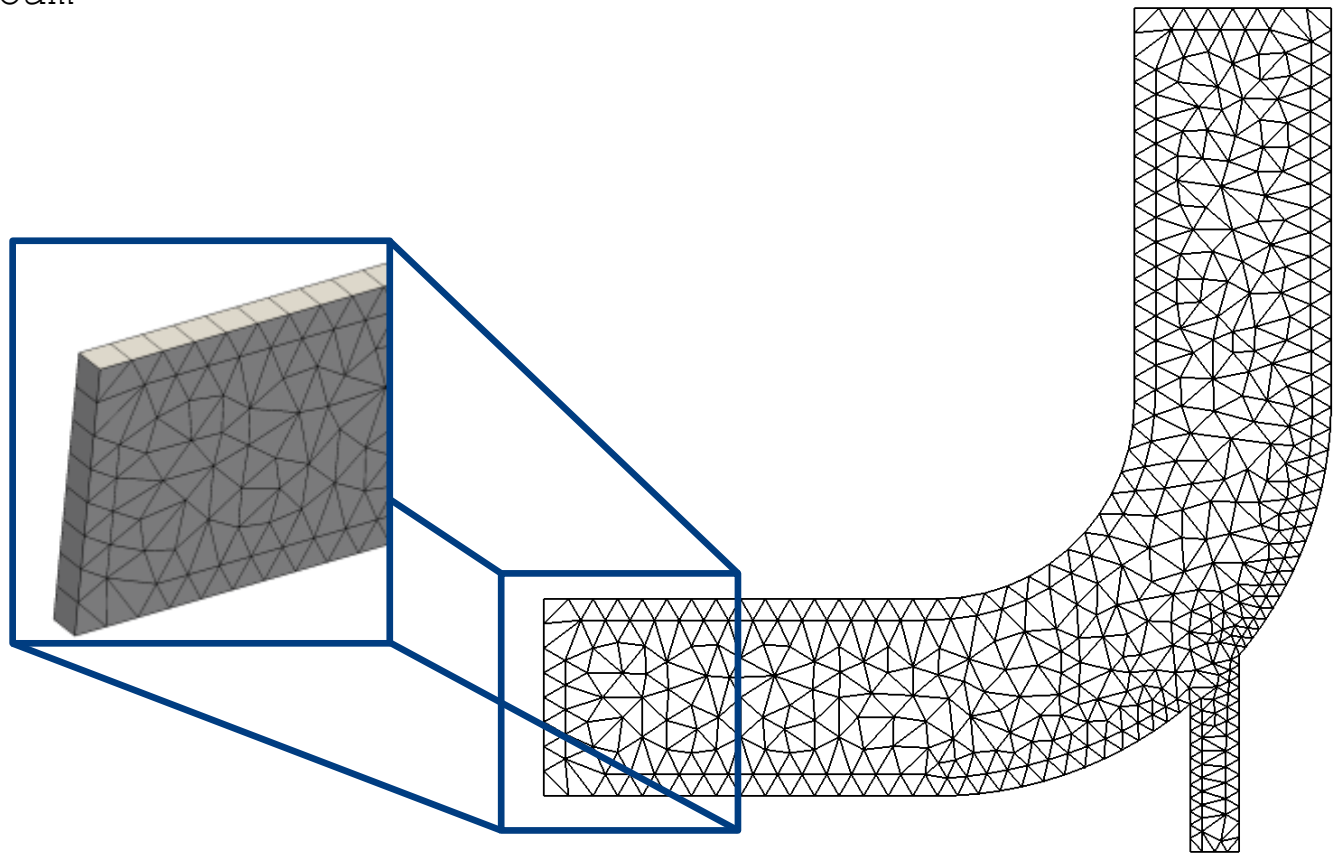
- Other converters can be found in the utilities directory

```
> ls $FOAM_UTILITIES/mesh/conversion
```

Creating a physics library
Initial case

- Now the mesh can be viewed in ParaView

```
> paraFoam
```



Creating a physics library
Initial case

- Run the solver

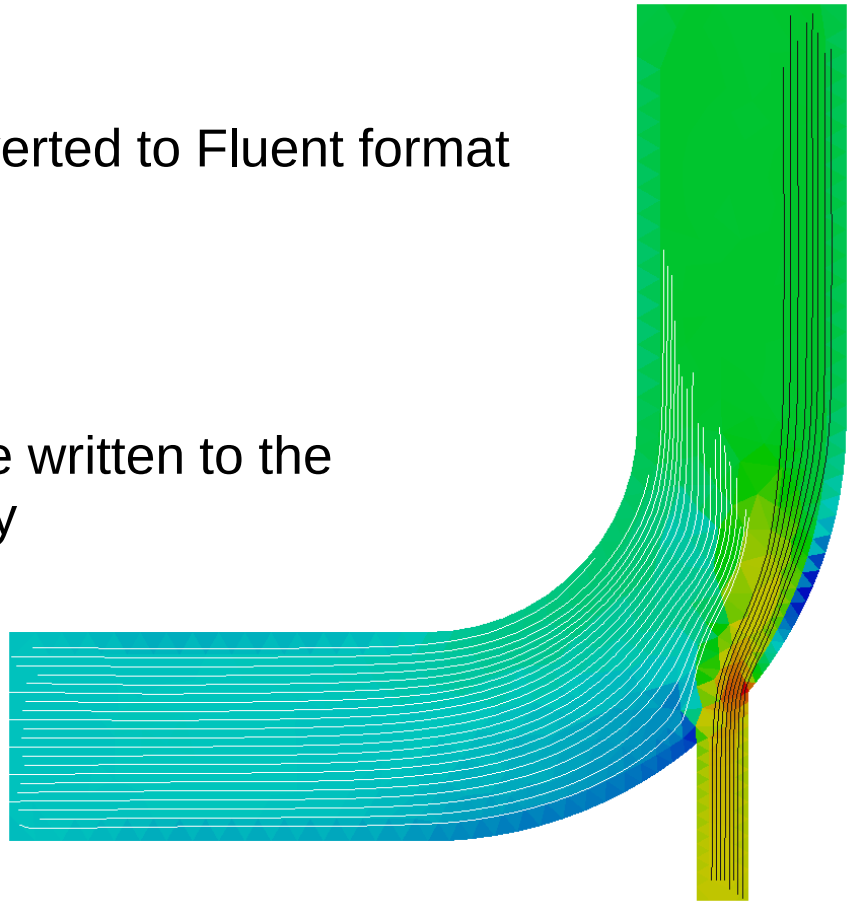
  ```
  > icoFoam
  ```

- Finally, the case can be converted to Fluent format

  ```
  > foamMeshToFluent
  ```

  ```
  > foamDataToFluent
  ```

- The fluent mesh and data are written to the `fluentInterface` directory



Creating a physics library
Modelling library

- Modelling libraries in OpenFOAM® are compiled as shared objects
  - run-time selectable models
  - modular, less duplication
  - shared objects can be linked to multiple executables

- Compiling
  - an executable (e.g. solver): `wmake`
  - a shared library (e.g. physical model): `wmake libso`

- Examples of libraries include
  - turbulence models
  - thermodynamics

Creating a physics library
Modelling library

- Typically three instances of implementation of library object
  - Create

```
autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, laminarTransport)
);
```

  - Use

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  + turbulence->divDevReff(U)
);
```

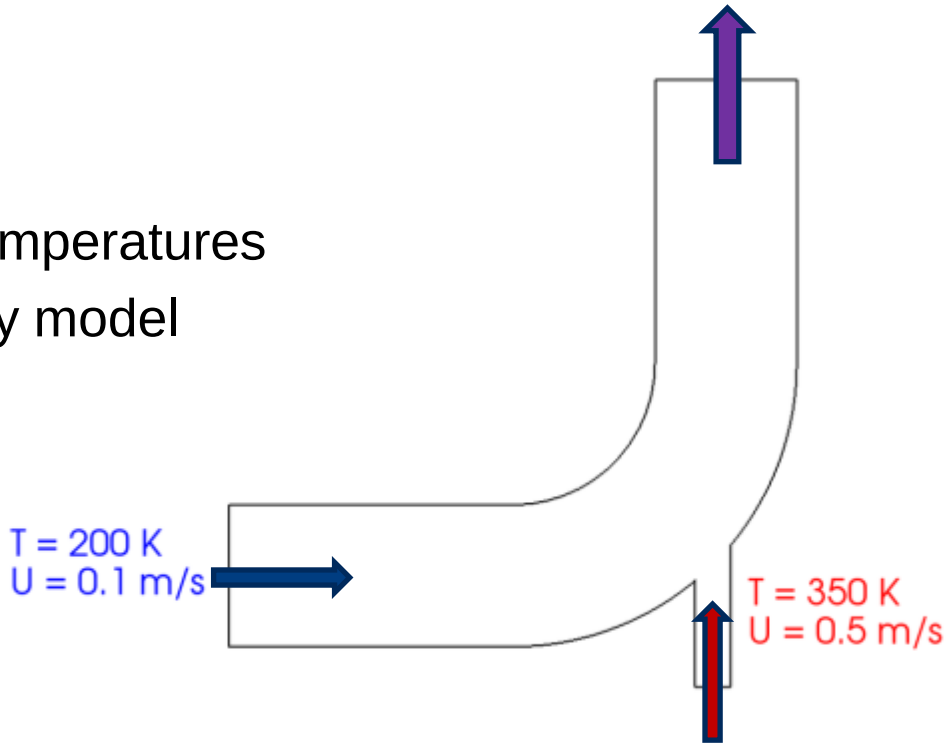
  - Update

```
turbulence->correct();
```

Creating a physics library
Modelling library

- Create a non-Newtonian viscosity model, solver and test case
  - Create a non-Newtonian viscosity **model**
  - Create (modify) a non-Newtonian **solver**
  - Create a **case** to test the solver

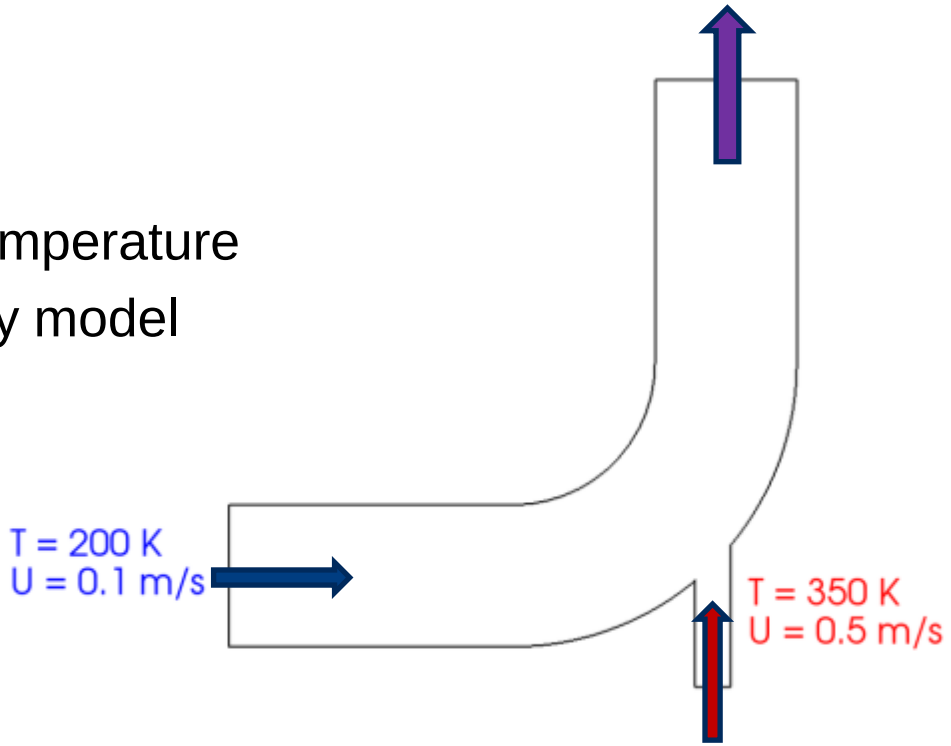
- The test case
  - Elbow junction
  - Acetone at different temperatures
  - Vogel thermal viscosity model



Creating a physics library
Creating the modelling library

- Create a non-Newtonian viscosity model, solver and test case
  - **Create a non-Newtonian viscosity model**
  - Create (modify) a non-Newtonian solver
  - Create a case to test the solver

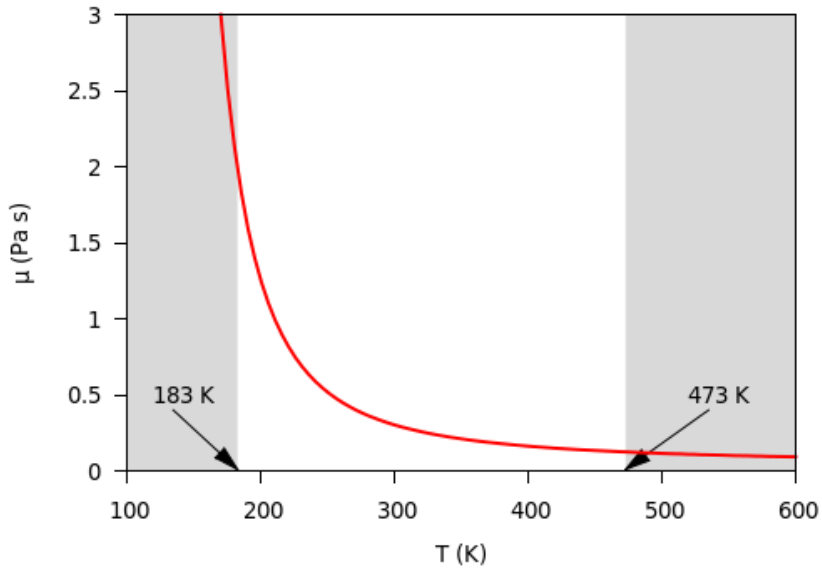
- The test case
  - Elbow junction
  - Acetone at different temperature
  - Vogel thermal viscosity model



Creating a physics library
Creating the modelling library

- The Vogel non-Newtonian viscosity model[1]:

$$\nu = \frac{1000}{\rho*} \exp\left(a + \frac{b}{c + T}\right)$$



- Create a shared object library to couple the viscosity model to a solver

[1] See DDBST: http://ddbonline.ddbst.de/VogelCalculation/VogelCalculationCGI.exe?component=Acetone

Creating a physics library
Creating the modelling library

- The Vogel non-Newtonian viscosity model:

$$\nu = \frac{1000}{\rho *} \exp\left(a + \frac{b}{c + T}\right)$$

- Create a shared object library to couple the viscosity model to a solver

- Copy an existing model and modify

```
> mkdir -p \
    $WM_PROJECT_USER_DIR/src/transportModels/incompressible/viscosityModels
> cd !$
> export VM=$FOAM_SRC/transportModels/incompressible/viscosityModels
> cp -r $VM/HerschelBulkley Vogel
> cd Vogel
> rename 's/HerschelBulkley/Vogel/g' *.*
> sed -i 's/HerschelBulkley/Vogel/g' *.*
```

Creating a physics library
Creating the modelling library

- Copy a `Make` directory into the Vogel library

```
> cp -r $FOAM_SRC/transportModels/incompressible/Make .
```

- Modify `Make/files` as:

```
1 Vogel.C
2
3 LIB = $(FOAM_USER_LIBBIN)/libvogelViscosityModel
```

- … and `Make/options` as:

```
1 EXE_INC = \
2     -I$(LIB_SRC)/finiteVolume/lnInclude \
3     -I$(LIB_SRC)/transportModels/incompressible/lnInclude
4
5 LIB_LIBS = \
6     -lfiniteVolume \
7     -lincompressibleTransportModels
```

Creating a physics library
Creating the modelling library

- Before making any changes it is a good idea to compile

```
> wmake libso
```

- Modify private member data in `Vogel.H`

```
 1 class Vogel
 2 :
 3 public viscosityModel
 4 {
 5     // Private data
 6
 7     dictionary VogelCoeffs_;
 8
 9     dimensionedScalar a_;
10     dimensionedScalar b_;
11     dimensionedScalar c_;
12     dimensionedScalar rhoStar_;
13
14     volScalarField nu_;
```

Creating a physics library
Creating the modelling library

- Implement the Vogel model as a member function in `Vogel.C`

$$\nu = \frac{1000}{\rho*} \exp\left(a + \frac{b}{c + T}\right)$$

```
50 Foam::tmp<Foam::volScalarField>
51     Foam::viscosityModels::Vogel::calcNu() const
52 {
53     const volScalarField& T =
54         U_.db().lookupObject<volScalarField>("T");
55
56     return
57     (
58         scalar(1000)/rhoStar_*exp(a_ + (b_/(c_ + T)))
59     );
60 }
```

Creating a physics library
Creating the modelling library

- Add the new model variables to the constructor in `Vogel.C`

```
64   Foam::viscosityModels::Vogel::Vogel
65   (
66       const word& name,
67       const dictionary& viscosityProperties,
68       const volVectorField& U,
69       const surfaceScalarField& phi
70   )
71   :
72       viscosityModel(name, viscosityProperties, U, phi),
73       VogelCoeffs_(viscosityProperties.subDict(typeName + "Coeffs")),
74       a_(VogelCoeffs_.lookup("a")),
75       b_(VogelCoeffs_.lookup("b")),
76       c_(VogelCoeffs_.lookup("c")),
77       rhoStar_(VogelCoeffs_.lookup("rhoStar")),
78       nu_
79       (
80           ...
```

Creating a physics library
Creating the modelling library

- Add the new model variables to the read function in `Vogel.C`

```
 95  bool Foam::viscosityModels::Vogel::read
 96  (
 97      const dictionary& viscosityProperties
 98  )
 99  {
100      viscosityModel::read(viscosityProperties);
101
102      VogelCoeffs_ = viscosityProperties.subDict(typeName + "Coeffs");
103
104      VogelCoeffs_.lookup("a") >> a_;
105      VogelCoeffs_.lookup("b") >> b_;
106      VogelCoeffs_.lookup("c") >> c_;
107      VogelCoeffs_.lookup("rhoStar") >> rhoStar_;
108
109      return true;
110  }
```

Creating a physics library
Creating the modelling library

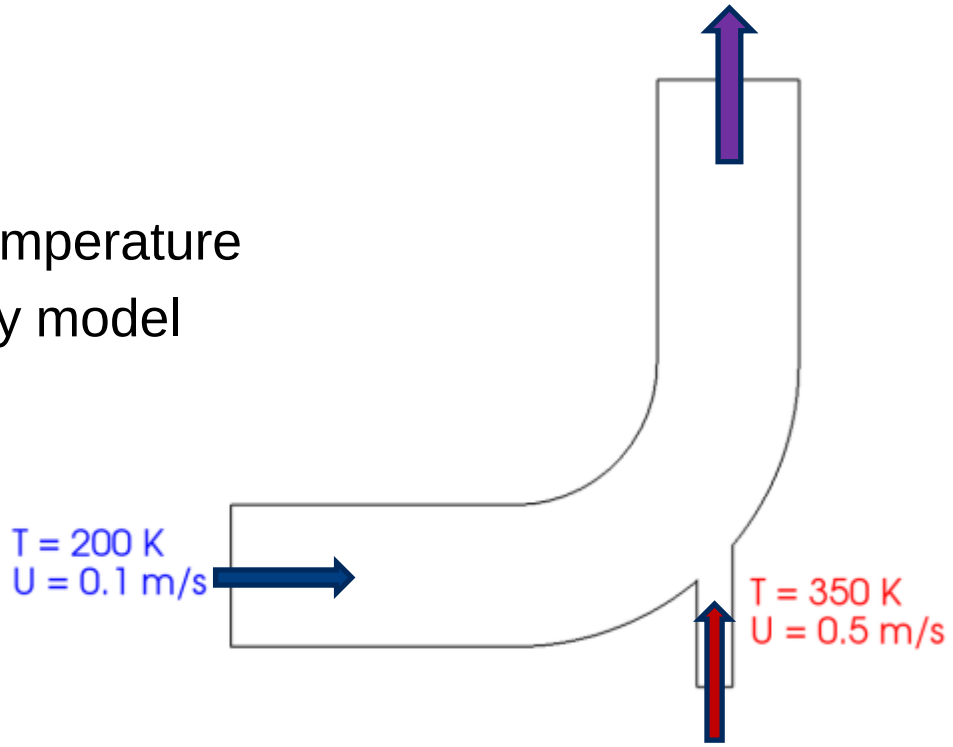
- Finally, compile the library

```
> wmake libso
```

Creating a physics library
Create a thermal non-Newtonian solver

- Create a non-Newtonian viscosity model, solver and test case
  - Create a non-Newtonian viscosity model
  - **Create (modify) a non-Newtonian solver**
  - Create a case to test the solver

- The test case
  - Elbow junction
  - Acetone at different temperature
  - Vogel thermal viscosity model



Creating a physics library
Create a thermal non-Newtonian solver

- Start from `nonNewtonianIcoFoam` solver and implement the thermal tracer

```
> mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/incompressible
> cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible
> cp -r $FOAM_SOLVERS/incompressible/nonNewtonianIcoFoam \
      nonNewtonianIcoThermalFoam
> cd nonNewtonianIcoThermalFoam
> mv nonNewtonianIcoFoam.C nonNewtonianIcoThermalFoam.C
```

- Edit `Make/files`

```
1 nonNewtonianIcoThermalFoam.C
2
3 EXE = $(FOAM_USER_APPBIN)/nonNewtonianIcoThermalFoam.C
```

- Compile the solver

```
> wmake
```

Creating a physics library
Create a thermal non-Newtonian solver

- Add tracer properties to `createFields.H`, first read transportProperties

```
30      // Instantiated the constant/transportProperties dictionary.
31      Info<< "Reading transportProperties\n" << endl;
32
33      IOdictionary transportProperties
34      (
35          IOobject
36          (
37              "transportProperties",
38              runTime.constant(),
39              mesh,
40              IOobject::MUST_READ_IF_MODIFIED,
41              IOobject::NO_WRITE
42          )
43      );
```

Creating a physics library
Create a thermal non-Newtonian solver

- Add tracer properties to `createFields.H`, then instantiate `DT`

```
45      // Instantiated tracer diffusion coefficient DT, read from
46      // constant/transportProperties.
47      dimensionedScalar DT
48      (
49          transportProperties.lookup("DT")
50      );
```

Creating a physics library
Create a thermal non-Newtonian solver

- Add tracer properties to `createFields.H`, and create field `T`

```
52      // Create tracer field T.
53      Info<< "Reading field T\n" << endl;
54      volScalarField T
55      (
56          IOobject
57          (
58              "T",
59              runTime.timeName(),
60              mesh,
61              IOobject::MUST_READ,
62              IOobject::AUTO_WRITE
63          ),
64          mesh
65      );
```

Creating a physics library
Create a thermal non-Newtonian solver

- Solve tracer transport in `nonNewtonianIcoThermalFoam.C`

```
108         // Solve passive scalar transport equation for tracer T.
109         solve
110         (
111             fvm::ddt(T)
112           + fvm::div(phi, T)
113           + fvm::laplacian(DT, T)
114         );
```

Creating a physics library
Create a thermal non-Newtonian solver

- Adjust te header of `nonNewtonianIcoThermalFoam.C`

```
Application
    nonNewtonianIcoThermalFoam

Description
    Transient solver for incompressible, laminar flow of non-Newtonian
    fluids. A transport equation for massless species T is implemented
    and. The Vogel viscosity model adjusts the viscosity according to
    the local value of the T (temperature) field.
```

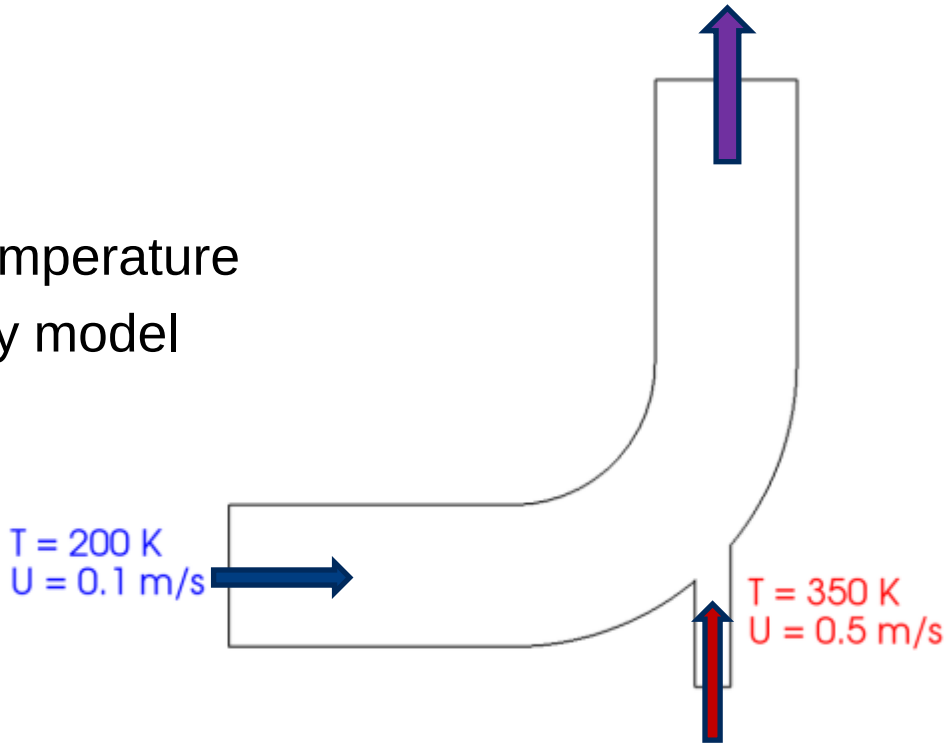
- Finally, compile the solver

```
> wmake
```

Creating a physics library
Create a test case

- Create a non-Newtonian viscosity model, solver and test case
  - Create a non-Newtonian viscosity model
  - Create (modify) a non-Newtonian solver
  - **Create a case to test the solver**

- The test case
  - Elbow junction
  - Acetone at different temperature
  - Vogel thermal viscosity model



Creating a physics library
Create a test case

- Go to the run directory and copy the elbow case

```
> run
> cp -r $FOAM_TUTORIALS/incompressible/icoFoam/elbow elbowThermal
> cd elbowThermal
```

- Add `T` to `system/fvSolution` and `system/fvSchemes` (e.g. "`div(phi,T) Gauss upwind;`")
- Adjust `0/U` so that patch `velocity-inlet-5` has `(0.1 0 0)` and `velocity-inlet-6` has `(0 0.5 0)` m/s velocity
- Add the `Vogel` library to `system/controlDict`

```
18    libs
19    (
20        "libOpenFOAM.so"
21        "libvogelViscosityModel.so"
22    );
```

Creating a physics library
Create a test case

- Create `0/T` (just copy `0/p`, change name, dimensions, and BCs)

```
17 dimensions      [0 0 0 1 0 0 0];
18
19 internalField   uniform 295;
20
21 boundaryField
22 {
23     wall-4
24     {
25         type            zeroGradient;
26     }
27
28     velocity-inlet-5
29     {
30         type            fixedValue;
31         value           uniform 200;
32     }
33
```

Creating a physics library
Create a test case

- Create `0/T` (just copy `0/p`, change name, dimensions, and BCs)

```
34          velocity-inlet-6
35          {
36              type            fixedValue;
37              value           uniform 350;
38          }
39
40          pressure-outlet-7
41          {
42              type            inletOutlet;
43              inletValue      uniform 295;
44              value           uniform 295;
45          }
```

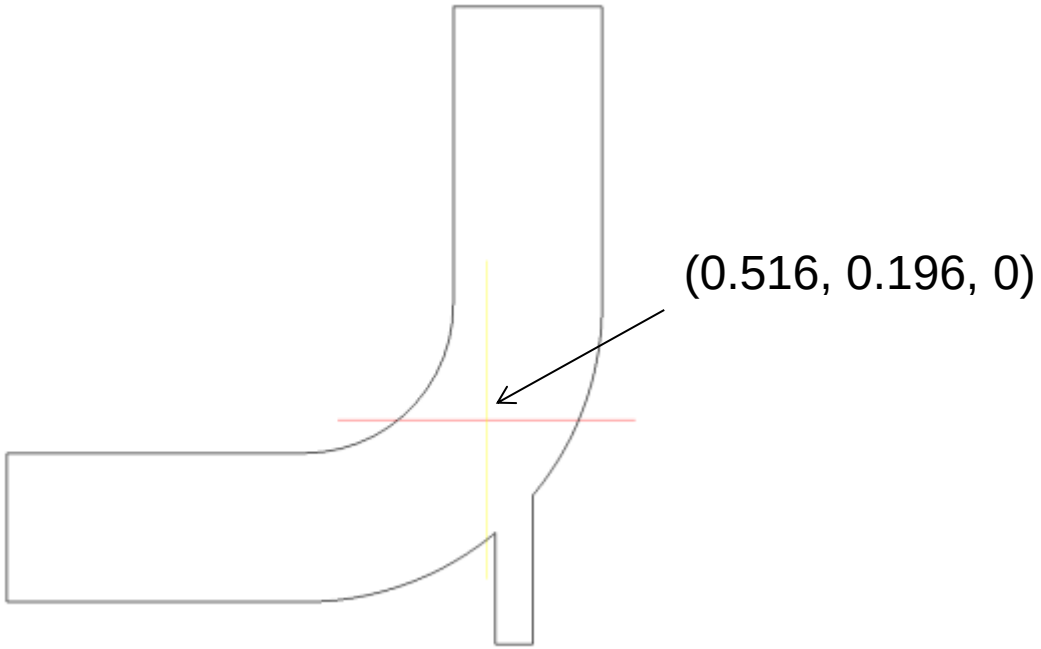
Creating a physics library
Create a test case

- Add the `Vogel` model to `constant/transportProperties`

```
18    transportModel  Vogel;
19
20    nu              nu [ 0 2 -1 0 0 0 0 ] 1;
21
22    DT              DT [ 0 2 -1 0 0 0 0 ] 1e-4;
23
24    VogelCoeffs
25    {
26        a               a [ 0 0 0 0 0 0 0 ] -3.4;
27        b               b [ 0 0 0 1 0 0 0 ] 553;
28        c               c [ 0 0 0 1 0 0 0 ] -47;
29        rhoStar         rhoStar [ 0 -2 1 0 0 0 0 ] 890;
30    }
```

Creating a physics library
Create a test case

- Let’s monitor the mixing zone by sampling viscosity and temperature



Creating a physics library
Create a test case

- Use a function object to sample online. Put the code below in `system/controlDict`:

```
54 functions
55 {
56     probes
57     {
58         type            probes;
59         functionObjectLibs ("libsampling.so");
60         enabled         true;
61         outputControl   timeStep;   // Sample every time step
62         outputInterval  1;          // Write data outputInterval*timeStep
63         fields ( T nu );            // Fields to sample
64         probeLocations
65         (
66             ( 0.516 0.196 0 )       // Sample location ( x y z )
67         );
68     }
69 }
```

Creating a physics library
Create a test case

- Use a function object to sample online. Put the code below in `system/controlDict`:

```
18 application     nonNewtonianIcoThermalFoam;
19
20 startFrom       latestTime;
21
22 startTime       0;
23
24 stopAt          endTime;
25
26 endTime         20;
27
28 deltaT          5e-4;
29
30 writeControl    runTime;
31
32 writeInterval   0.1;
33
34 purgeWrite      0;
```

Creating a physics library
Create a test case

- Before running convert the mesh. The original mesh is huge, so let's create a mesh of more sensible size

```
> fluentMeshToFoam -scale 0.01 elbow.msh
```

- Finally, run the solver

```
> nonNewtonianThermalFoam
```

Creating a physics library
Create a test case

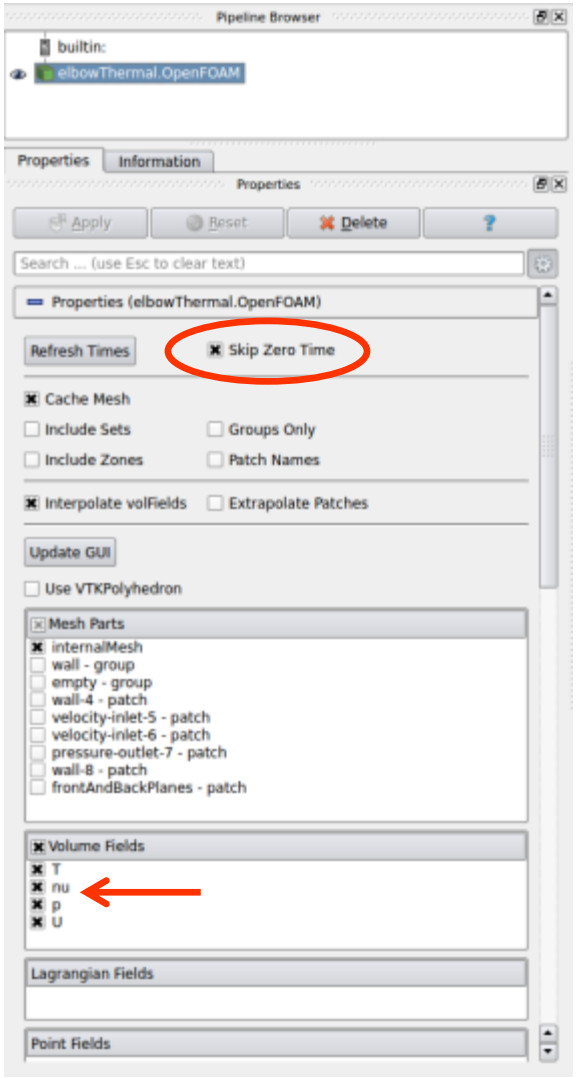
- Before running convert the mesh. The original mesh is huge, so let's create a mesh of more sensible size

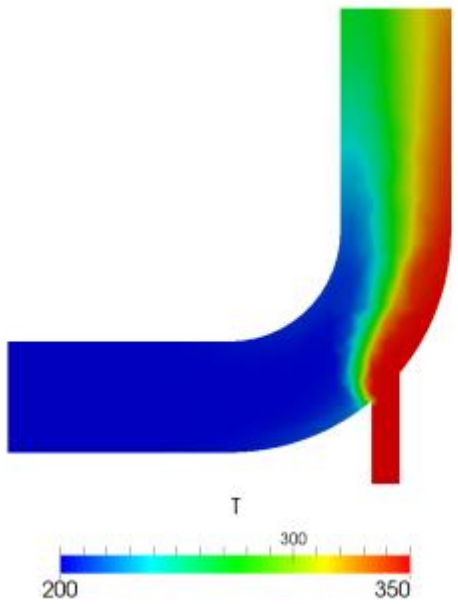
```
> fluentMeshToFoam -scale 0.01 elbow.msh
```

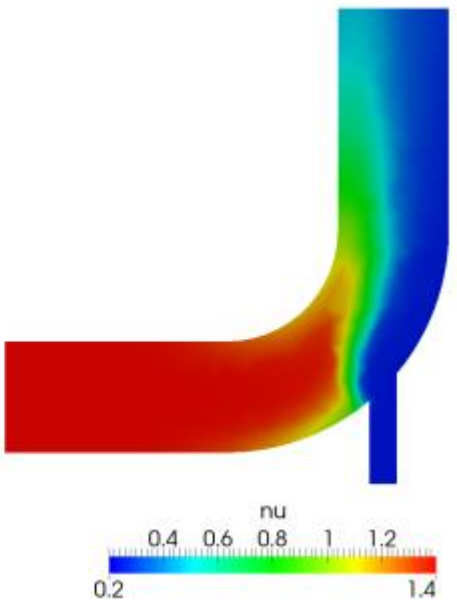
- Finally, run the solver

```
> nonNewtonianThermalFoam
```

Creating a physics library
Create a test case







Creating a physics library
Create a test case

- View the sampling results, e.g. using `gnuplot`

```
> gnuplot
gnuplot > plot postProcessing/probes/0/nu using 1:2 with lines notitle
```