Combination of reactingFoam and chtMultiRegionFoam as a first step toward creating a multiRegionReactingFoam, suitable for solid/gas phase reactions

Seyed Morteza Mousavi

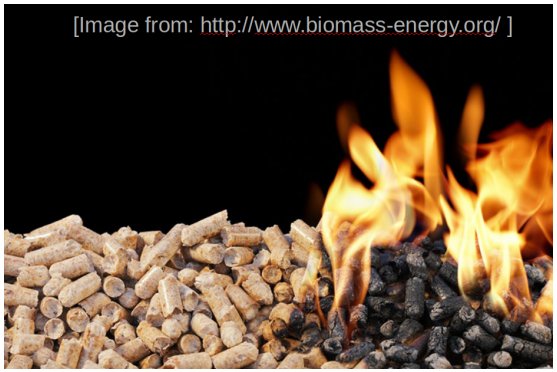
Department of Energy Sciences, Division of Fluid Dynamics,
Lund University,
Lund, Sweden

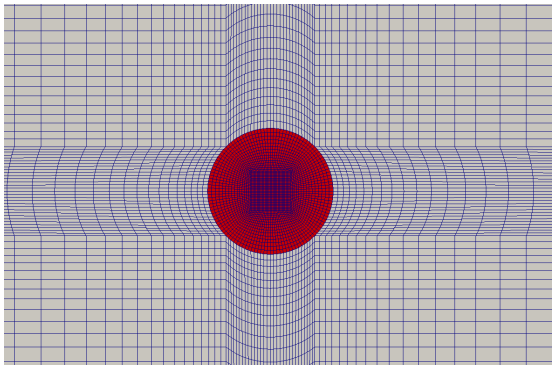
2019-11-27

Contents:

Background, solid fuel combustion

- Combustion mainly happens in gas phase
- Solid fuel is converted to gas first
- Intraparticle gradients depend on size
  - Small particles ($Bi << 1$)
  - **Large particles** ($Bi > 1$)





Motivation for multiRegionReactingFoam

- First step to model the flow and reactions inside a reacting porous particle
- reactingFoam is available for fluid region
- chtMultiRegionFoam is available, which couples multiple regions
- Several tasks to obtain a model for solid fuel combustion:
  - **Add a solid region to the reactingFoam solver**
  - **Solve energy equation for solid**
  - **Appropriate BC to couple E-equations**
  - Solve pyrolysis and Darcy equations inside porous media
  - Appropriate BC to couple U, p, Y_i equations

Fluid region governing equations

continuity equation

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho U) = 0 \tag{1}$$

momentum equation

$$\frac{\partial}{\partial t}(\rho U) + \nabla.(\rho U U) = -\nabla p + \nabla.\tau \tag{2}$$

species transport equation

$$\frac{\partial}{\partial t}(\rho Y_i) + \nabla.(\rho U Y_i) = \nabla \mu_{eff} \nabla Y_i + \dot{R}_i \tag{3}$$

and energy equation

$$\frac{\partial}{\partial t}(\rho h) + \nabla(\rho U h) + \frac{\partial}{\partial t}(\rho K) + \nabla(\rho U K) - \frac{\partial p}{\partial t} = \nabla \alpha_{eff} \nabla h + \dot{R}_{heat} \tag{4}$$

Solid region governing equations

Solid energy equation (heat conduction)

$$\frac{\partial}{\partial t}(\rho h) = \nabla \alpha_{eff} \nabla h \tag{5}$$

Boundary condition for energy equation

We need to satisfy these at the patch:

$$T_{p,1} = T_{p,2} = T_p, \tag{6}$$

$$-k_1 \frac{\partial T}{\partial n}\bigg|_{side1} = -k_2 \frac{\partial T}{\partial n}\bigg|_{side2} \tag{7}$$

Which leads to:

$$T_p = \frac{k_1 \Delta_1 T_{c,1} + k_2 \Delta_2 T_{c,2}}{k_1 \Delta_1 + k_2 \Delta_2} \tag{8}$$

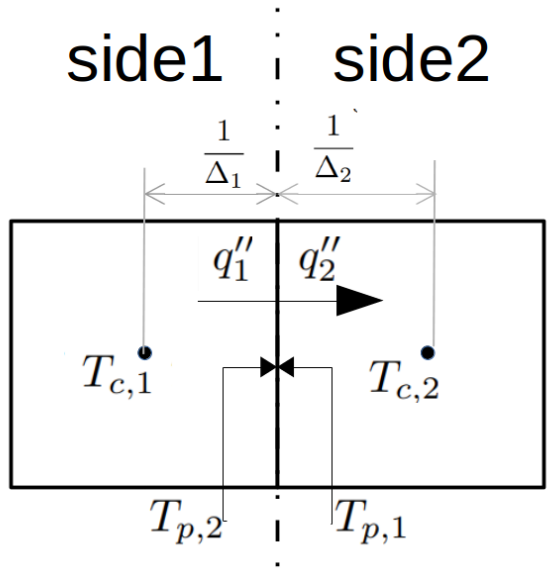


Figure: Schematic of two cells at the interface

DLR_A_LTS tutorial case

```
.
├── 0
│   ├── alphat
│   ├── CH4
│   ├── epsilon
│   ├── G
│   ├── H2
│   ├── H2O
│   ├── k
│   ├── N2
│   ├── nut
│   ├── O2
│   ├── p
│   ├── T.orig
│   ├── U
│   └── Ydefault
├── Allclean
├── Allrun
├── chemkin
│   ├── grimech30.dat
│   ├── thermo30.dat
│   └── transportProperties
├── constant
│   ├── chemistryProperties
│   ├── chemistryProperties.test
│   ├── combustionProperties
│   ├── g
│   ├── reactionsGRI
│   ├── thermo.compressibleGasGRI
│   ├── thermophysicalProperties
│   └── turbulenceProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── decomposeParDict
    ├── fvSchemes
    ├── fvSolution
    └── setFieldsDict

4 directories, 33 files
```

Figure: The contents of the DLR_A_LTS tutorial case.

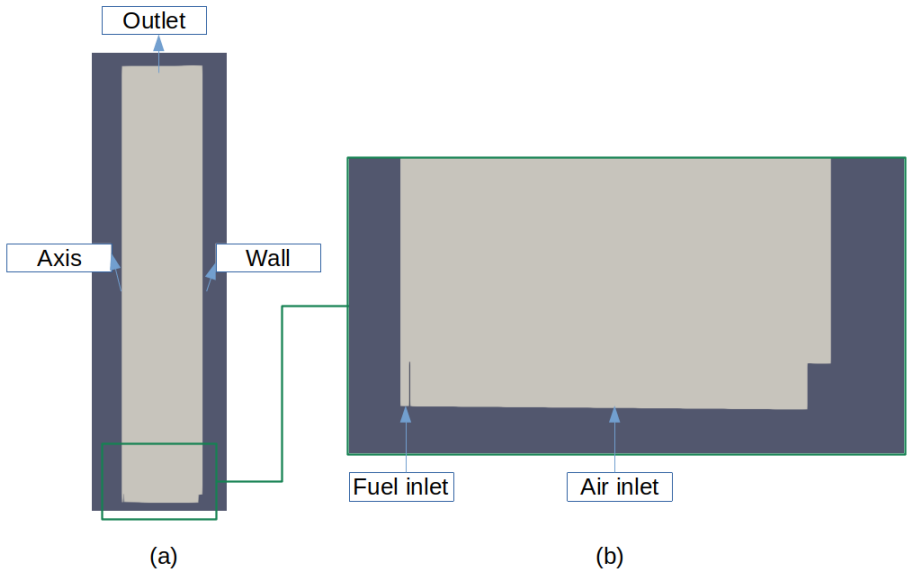


Figure: Geometry of the DLR_A_LTS case.

Initialize and run the case

Get the case:

```
run
cp -r $FOAM_TUTORIALS/combustion/reactingFoam/RAS/DLR_A_LTS/ .
cd DLR_A_LTS
```

Get a global reaction mechanism:

```
mkdir constant/BACKUP

mv constant/{chemistryProperties,thermophysicalProperties,\
reactionsGRI,thermo.compressibleGasGRI} constant/BACKUP

cp $FOAM_TUTORIALS/combustion/reactingFoam/RAS/membrane/constant/{chemistryProperties,\
thermophysicalProperties,reactions,thermo.compressibleGas} constant/

sed -i s/"chemistry .*"/"chemistry on;"/g constant/chemistryProperties
```

Run the case:

```
./Allrun
```

Some results

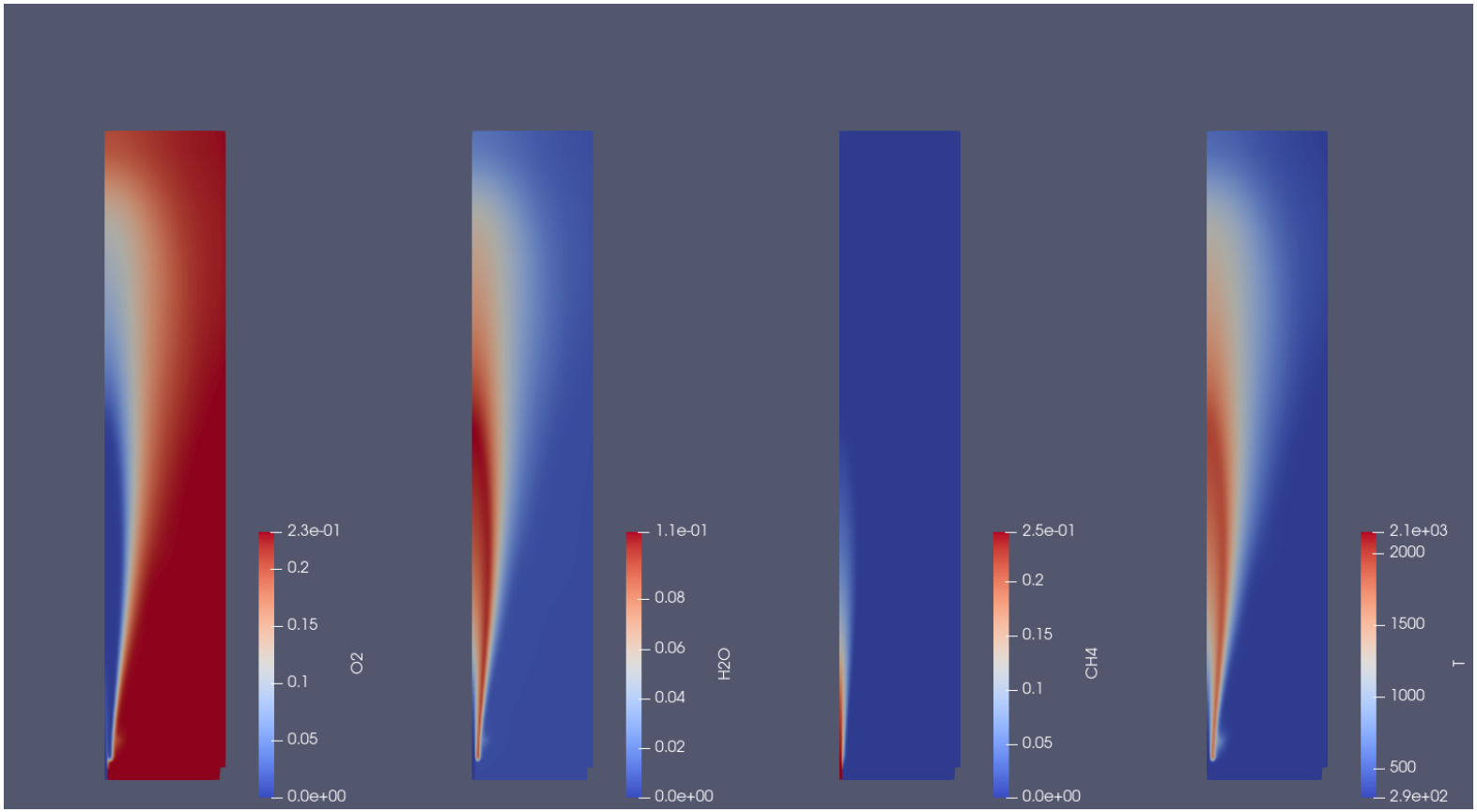


Figure: Final results of the DLR_A_LTS case using a simple global reaction mechanism. From left to right, O2 mass fraction, H2O mass fraction, CH4 mass fraction, and temperature distribution.

multiRegionHeater tutorial case

```
.
├── 0.orig
│   ├── epsilon
│   ├── k
│   ├── p
│   ├── p_rgh
│   ├── T
│   └── U
├── Allclean
├── Allrun
├── Allrun.pre
├── constant
│   ├── bottomWater
│   ├── g
│   ├── heater
│   ├── leftSolid
│   ├── regionProperties
│   ├── rightSolid
│   └── topAir
├── README.txt
└── system
    ├── blockMeshDict
    ├── bottomWater
    ├── controlDict
    ├── decomposeParDict
    ├── fvSchemes
    ├── fvSolution
    ├── heater
    ├── leftSolid
    ├── README
    ├── rightSolid
    ├── topAir
    ├── topoSetDict
    └── vtkWrite

13 directories, 20 files
```

Figure: Contents.

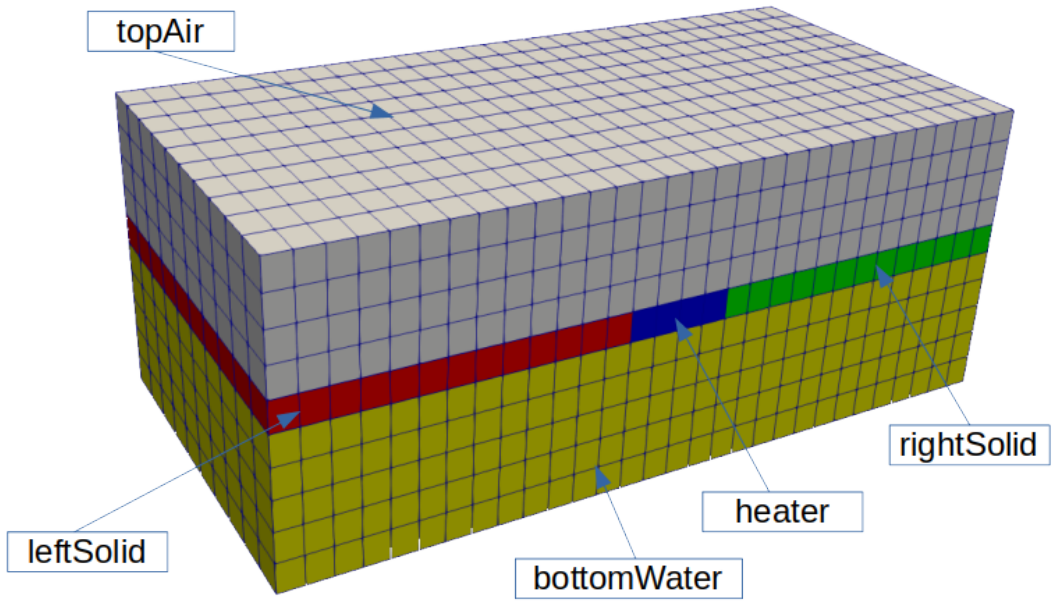


Figure: Geometry of the multiRegionHeater case.

multiRegionHeater tutorial case

$0/heater/T$

```
heater_to_leftSolid
{
    type compressible::turbulent\
       TemperatureCoupledBaffleMixed;
    value           uniform 300;
    Tnbr            T;
    kappaMethod     solidThermo;
    thicknessLayers ( 0.001 );
    kappaLayers     ( 0.0005 );
}
```

$constant/regionProperties$

```
regions
(
    fluid (bottomWater topAir)
    solid (heater leftSolid rightSolid)
);
```

$system/topoSetDict$

```
actions
(
    ...
    // leftSolid
    {
        name    leftSolidCellSet;
        type    cellSet;
        action  new;
        source  boxToCell;
        box     (-100 0 -100) (-0.01001 0.00999 100);
    }
    {
        name    leftSolid;
        type    cellZoneSet;
        action  new;
        source  setToCellZone;
        set     leftSolidCellSet;
    }
    ...
```

Initialize and run the case

Get the case and run:

```
run
cp -r $FOAM_TUTORIALS/heatTransfer/chtMultiRegionFoam/multiRegionHeater/ .
cd multiRegionHeater
./Allrun
```

Allrun script :

```
runApplication ./Allrun.pre
# Decompose
runApplication decomposePar -allRegions
# Run
runParallel $(getApplication)
# Reconstruct
runApplication reconstructPar -allRegions
```

Allrun.pre script :

```
runApplication blockMesh
runApplication topoSet

# Restore initial fields
restore0Dir

runApplication splitMeshRegions -cellZones -overwrite
# Remove fluid fields from solid regions
# (important for post-processing)
...
```

Some results

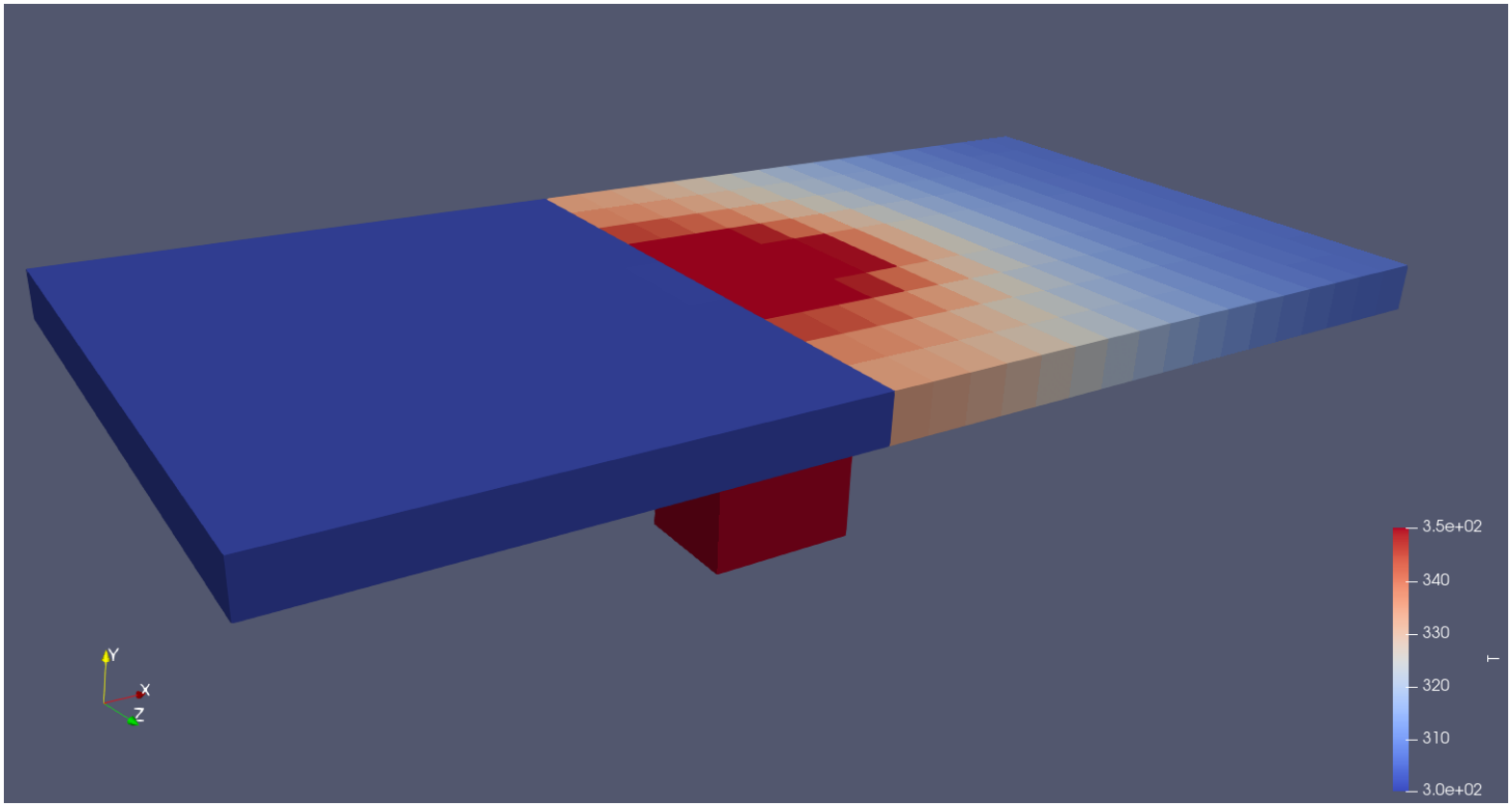


Figure: The final temperature of the solid regions in the multiRegionHeater tutorial case.

Look into reactingFoam.C

Go to case directory

```
sol
cd combustion/reactingFoam
```

Included libraries

```
#include "fvCFD.H"
#include "turbulentFluidThermoModel.H"
#include "psiReactionThermo.H"
#include "CombustionModel.H"
#include "multivariateScheme.H"
#include "pimpleControl.H"
#include "pressureControl.H"
#include "fvOptions.H"
#include "localEulerDdtScheme.H"
#include "fvcSmooth.H"
```

Look into reactingFoam.C

To use LTS in a case, we need

```
ddtSchemes
{
    default         localEuler;
}
```

Back to the reactingFoam.C

```
int main(int argc, char *argv[])
{
    argList::addNote
    (
        "Solver for combustion with chemical reactions"
    );

    #include "postProcess.H"

    #include "addCheckCaseOptions.H"
    #include "setRootCaseLists.H"
```

Look into reactingFoam.C

```
#include "createTime.H"              // runTime
#include "createMesh.H"              // mesh
#include "createControl.H"           // pimple
#include "createTimeControls.H"      // adjustTimeStep, maxCo, maxDeltaT
#include "initContinuityErrs.H"      // cumulativeContErr
#include "createFields.H"
#include "createFieldRefs.H"
```

Look into createFields.H

```
#include "createRDeltaT.H" \\ bool LTS = fv::localEulerDdt::enabled(mesh);

Info<< "Reading thermophysical properties\n" << endl;
autoPtr<psiReactionThermo> pThermo(psiReactionThermo::New(mesh));
psiReactionThermo& thermo = pThermo();
thermo.validate(args.executable(), "h", "e");

basicSpecieMixture& composition = thermo.composition();
PtrList<volScalarField>& Y = composition.Y();

const word inertSpecie(thermo.get<word>("inertSpecie"));
if (!composition.species().found(inertSpecie))
{
    FatalIOErrorIn(args.executable().c_str(), thermo)
        << "Inert specie " << inertSpecie << " not found in available species "
        << composition.species() << exit(FatalIOError);
}
```

Look into createFields.H

The ρ, p, U, and ϕ variables are created similar to other solvers. Then:

```
autoPtr<compressible::turbulenceModel> turbulence
(
    compressible::turbulenceModel::New
    (
        rho,
        U,
        phi,
        thermo
    )
);
autoPtr<CombustionModel<psiReactionThermo>> reaction
(
    CombustionModel<psiReactionThermo>::New(thermo, turbulence())
);
```

Look into createFields.H

```
volScalarField Qdot
(
    IOobject
    (
        "Qdot",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    mesh,
    dimensionedScalar(dimEnergy/dimVolume/dimTime, Zero)
);

#include "createDpdt.H"
#include "createK.H"
#include "createMRF.H"
#include "createFvOptions.H"
```

Look into createFieldRefs.H

```
const volScalarField& psi = thermo.psi();
const volScalarField& T = thermo.T();
const label inertIndex(composition.species()[inertSpecie]);
```

(Why these references are separated from others?)

Back to the main file:

```
turbulence->validate();

if (!LTS)
{
    #include "compressibleCourantNo.H"
    #include "setInitialDeltaT.H"
}
```

Look into reactingFoam.C

```
Info<< "\nStarting time loop\n" << endl;

while (runTime.run())
{
    #include "readTimeControls.H"

    if (LTS)
    {
        #include "setRDeltaT.H"
    }
    else
    {
        #include "compressibleCourantNo.H"
        #include "setDeltaT.H"
    }

    ++runTime;

    Info<< "Time = " << runTime.timeName() << nl << endl;
```

Look into reactingFoam.C

```
#include "rhoEqn.H"
while (pimple.loop())
{
    #include "UEqn.H"
    #include "YEqn.H"
    #include "EEqn.H"
    // --- Pressure corrector loop
    while (pimple.correct())
    {
        if (pimple.consistent())
        {
            #include "pcEqn.H"
        }
        else
        {
            #include "pEqn.H"
        }
    }
    if (pimple.turbCorr())
    {
        turbulence->correct();
    }
}
rho = thermo.rho();
runTime.write();
runTime.printExecutionTime(Info);
```

Look into equations I

rhoEqn.H

```
fvScalarMatrix rhoEqn
(
    fvm::ddt(rho)
  + fvc::div(phi)
  ==
    fvOptions(rho)
);
```

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho U) = 0$$

UEqn.H

```
tmp<fvVectorMatrix> tUEqn
(
    fvm::ddt(rho, U) + fvm::div(phi, U)
  + MRF.DDt(rho, U)
  + turbulence->divDevRhoReff(U)
 ==
    fvOptions(rho, U)
);
...
if (pimple.momentumPredictor())
    solve(UEqn == -fvc::grad(p));
```

$$\frac{\partial}{\partial t}(\rho U) + \nabla.(\rho U U) = -\nabla p + \nabla.\tau$$

$$\tau = \mu\left[(\nabla U + \nabla U^T) - \frac{2}{3}\nabla.U I\right]$$

Look into equations II

YEqn.H

```
fvScalarMatrix YiEqn
(
    fvm::ddt(rho, Yi)
  + mvConvection->fvmDiv(phi, Yi)
  - fvm::laplacian(turbulence->muEff(), Yi)
 ==
    reaction->R(Yi)
  + fvOptions(rho, Yi)
);
```

$$\frac{\partial}{\partial t}(\rho Y_i) + \nabla.(\rho U Y_i) = \nabla \mu_{eff} \nabla Y_i + \dot{R}_i$$

Look into equations III

EEqn.H

```
fvScalarMatrix EEqn
(
    fvm::ddt(rho, he) + mvConvection->fvmDiv(phi, he)
  + fvc::ddt(rho, K) + fvc::div(phi, K)
  + (
        he.name() == "e"
      ? fvc::div
        (
            fvc::absolute(phi/fvc::interpolate(rho), U),
            p,
            "div(phiv,p)"
        )
      : -dpdt
    )
  - fvm::laplacian(turbulence->alphaEff(), he)
 ==
    Qdot
  + fvOptions(rho, he)
);
```

$$\frac{\partial}{\partial t}(\rho h) + \nabla(\rho U h) + \frac{\partial}{\partial t}(\rho K) + \nabla(\rho U K) - \frac{\partial p}{\partial t} = \nabla \alpha_{eff} \nabla h + \dot{R}_{heat}$$

Look into chtMultiRegionFoam.C

Go to the directory of the solver

```
sol
cd heatTransfer/chtMultiRegionFoam/
```

Among libraries:

```
#include "solidRegionDiffNo.H"
#include "regionProperties.H"
```

Inside regionProperties class:

```
// Member Functions
    //- Total count of all region names.
    label count() const;
    //- The region names. Sorted by region type.
    wordList names() const;
    //- The region names in sorted order.
    wordList sortedNames() const;
```

Inside the case directory: constant/regionProperties file

```
regions
(
    fluid (bottomWater topAir)
    solid (heater leftSolid rightSolid)
);
```

Look into chtMultiRegionFoam.C

Parts of code included in the main function:

```
int main(int argc, char *argv[])
{
    ...
    #define NO_CONTROL
    #define CREATE_MESH createMeshesPostProcess.H
    #include "postProcess.H"
    ...
    #include "createMeshes.H"                       // fluidRegions and solidRegions
    ...
    #include "readSolidTimeControls.H"              // maxDi
    #include "compressibleMultiRegionCourantNo.H"   // coNum
    #include "solidRegionDiffusionNo.H"             // DiNum
    #include "setInitialMultiRegionDeltaT.H"        // call to runTime.setDeltaT
```

Inside createMeshes.H

```
regionProperties rp(runTime);

#include "createFluidMeshes.H"
#include "createSolidMeshes.H"
```

Look into chtMultiRegionFoam.C

Right after setInitialMultiRegionDeltaTT, is the main loop:

```
while (runTime.run())
{
    #include "readTimeControls.H"
    #include "readSolidTimeControls.H"
    #include "readPIMPLEControls.H"

    #include "compressibleMultiRegionCourantNo.H"
    #include "solidRegionDiffusionNo.H"
    #include "setMultiRegionDeltaT.H"
    ...
```

How to couple EEqn in two regions?

Find the code for turbulentTemperatureCoupledBaffleMixed:

```
src
find . -iname "turbulentTemperatureCoupledBaffleMixed"
```

Inside turbulentTemperatureCoupledBaffleMixedFvPatchScalarField.H:

```
Description
    Mixed boundary condition for temperature, to be used for heat-transfer
    on back-to-back baffles. Optional thin thermal layer resistances can be
    specified through thicknessLayers and kappaLayers entries.
...
Example of the boundary condition specification:
    <patchName>
    {
        type            compressible::turbulentTemperatureCoupledBaffleMixed;
        Tnbr            T;
        thicknessLayers (0.1 0.2 0.3 0.4);
        kappaLayers     (1 2 3 4);
        kappaMethod     lookup;
        kappa           kappa;
        value           uniform 300;
    }
```

Look into turbulentTemperatureCoupledBaffleMixedFvPatchScalarField.H

```
class turbulentTemperatureCoupledBaffleMixedFvPatchScalarField
:
    public mixedFvPatchScalarField,
    public temperatureCoupledBase
{
    // Private data
        //- Name of field on the neighbour region
        const word TnbrName_;
        //- Thickness of layers
        scalarList thicknessLayers_;
        //- Conductivity of layers
        scalarList kappaLayers_;
        //- Total contact resistance
        scalar contactRes_;

public:
    //- Runtime type information
    TypeName("compressible::turbulentTemperatureCoupledBaffleMixed");
    // Constructors
    ...
    // Member functions
        //- Update the coefficients associated with the patch field
        virtual void updateCoeffs();
        //- Write
        virtual void write(Ostream&) const;
}
```

How mixed boundaries work in OF? (mixedFvPatchField.H)

$$x_p = wx + (1 - w)\left(x_c + \frac{\nabla_\perp x}{\Delta}\right)$$

where

variable	Access name to modify	Description
x_p	-	patch values
x_c	-	patch internal cell values
x	refValue	reference value
$\nabla_\perp x$	refGrad	reference gradient
w	valueFraction	weight field
Δ	-	inverse distance from face centre to internal cell centre

For instance:

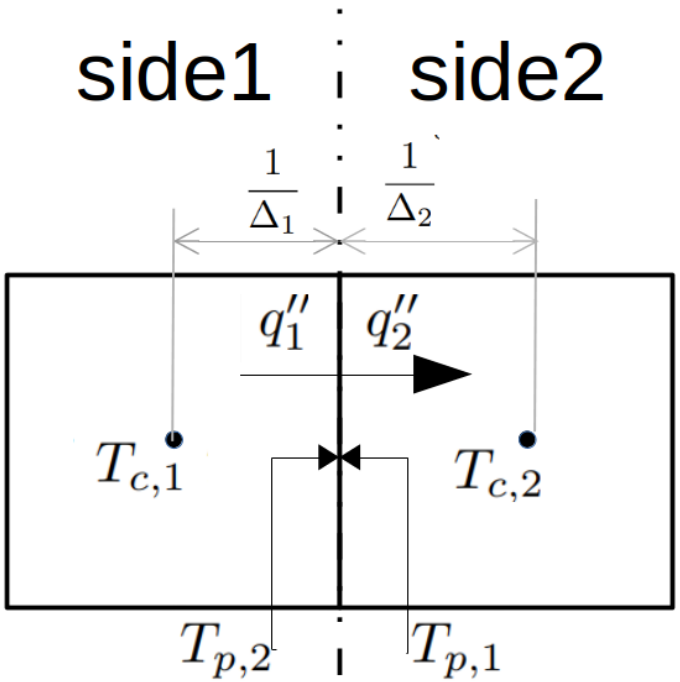
$w = 1$: fixedValue BC

$w = 0$: fixedGradient BC

Inside updateCoeffs()

```
this->refValue() = nbrIntFld();
this->refGrad() = 0.0;
this->valueFraction() = nbrKDelta()/(nbrKDelta() + myKDelta());

mixedFvPatchScalarField::updateCoeffs();
```



$$x = refValue = T_{c,2}$$

$$\nabla_\perp x = refGrad = 0.0$$

$$w = valueFraction = \frac{k_2\Delta_2}{k_1\Delta_1 + k_2\Delta_2}$$

Calculating the patch values

$$x_p = wx + (1 - w)\left(x_c + \frac{\nabla_\perp x}{\Delta}\right)$$

$$x = refValue = T_{c,2}$$

$$\nabla_\perp x = refGrad = 0.0$$

$$w = valueFraction = \frac{k_2\Delta_2}{k_1\Delta_1 + k_2\Delta_2}$$

Therefore we have:

$$T_p = \frac{k_2\Delta_2}{k_1\Delta_1 + k_2\Delta_2} T_{c,2} + \left(\frac{k_1\Delta_1}{k_1\Delta_1 + k_2\Delta_2}\right)(T_{c,1} + 0.0)$$

$$\rightarrow T_p = \frac{k_1\Delta_1 T_{c,1} + k_2\Delta_2 T_{c,2}}{k_1\Delta_1 + k_2\Delta_2}$$

Implementing multiRegionReactingFoam

Start from a fresh copy of reactingFoam.

```
foam
cp --parents -r applications/solvers/combustion/reactingFoam/ $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/applications/solvers/combustion/reactingFoam/
rm -r rhoReacting*
```

Change filenames and make sure you can compile.

```
cd ..
mv reactingFoam multiRegionReactingFoam
cd multiRegionReactingFoam
mv reactingFoam.C multiRegionReactingFoam.C
sed -i s/FOAM_APPBIN/FOAM_USER_APPBIN/g Make/files
sed -i s/reactingFoam/multiRegionReactingFoam/g Make/files
wclean
wmake
```

Implementing multiRegionReactingFoam

Add following libraries to multiRegionReactingFoam.C

```
// Solid region libraries
#include "solidRegionDiffNo.H"
#include "solidThermo.H"
```

In the main function, modify the .H files.

```
// To be able to use postProcess
#define CREATE_MESH createMeshes.H
#include "postProcess.H"

#include "addCheckCaseOptions.H"
#include "setRootCaseLists.H"
#include "createTime.H"
// To create more than one mesh
#include "createMeshes.H"
#include "createControl.H"
#include "createTimeControls.H"
// To read the max diffusion number
#include "readSolidTimeControls.H"
#include "initContinuityErrs.H"
#include "createFields.H"
#include "createFieldRefs.H"
```

Implementing multiRegionReactingFoam

Create a new file named createMeshes.H.

```
Foam::Info
    << "Create fluid mesh for time = "
    << runTime.timeName() << Foam::nl << Foam::endl;
Foam::fvMesh mesh
(
    Foam::IOobject
    (
        "fluid",
        runTime.timeName(),
        runTime,
        Foam::IOobject::MUST_READ
    )
);
Foam::Info
    << "Create pyrolysis mesh for time = "
    << runTime.timeName() << Foam::nl << Foam::endl;
Foam::fvMesh pyrolysisMesh
(
    Foam::IOobject
    (
        "pyrolysisRegion",
        runTime.timeName(),
        runTime,
        Foam::IOobject::MUST_READ
    )
);
```

Implementing multiRegionReactingFoam

Copy some files from chtMultiRegionFoam solver to solid sub-directory.

```
mkdir solid
cp $FOAM_SOLVERS/heatTransfer/chtMultiRegionFoam/solid/readSolidTimeControls.H solid/
cp $FOAM_SOLVERS/heatTransfer/chtMultiRegionFoam/solid/solidRegionDiffNo.H solid/
cp $FOAM_SOLVERS/heatTransfer/chtMultiRegionFoam/solid/solidRegionDiffNo.C solid/
cp $FOAM_SOLVERS/heatTransfer/chtMultiRegionFoam/solid/solidRegionDiffusionNo.H solid/
cp $FOAM_SOLVERS/heatTransfer/chtMultiRegionFoam/include/setMultiRegionDeltaT.H solid/
```

Back to main function in main file, replace the if (LTS) block with the following:

```
if (LTS)
{
    #include "setRDeltaT.H"
}
else
{
    #include "compressibleCourantNo.H"
    // To include solidRegion in decision over deltaT
    #include "solidRegionDiffusionNo.H"
    #include "setMultiRegionDeltaT.H"
}
```

Implementing multiRegionReactingFoam

Add the following lines to main loop, after EEqn.H.

```
// To solve the solid energy equation
#include "solidEEqn.H"
```

Create a new file named solidEEqn.H in the solid sub-directory containing the following code.

```
{
    volScalarField& h = pyroThermo.he();
    pyroRho = pyroThermo.rho();
    fvScalarMatrix hEqn
    (
     fvm::ddt(pyroRho, h)
      - fvm::laplacian(pyroThermo.alpha(), h)
      ==
     fvOptions(pyroRho, h)
    );
    hEqn.relax();
    fvOptions.constrain(hEqn);
    hEqn.solve();
    fvOptions.correct(h);
    pyroThermo.correct();
    Info<< "Solid Min/max T:" << min(pyroThermo.T()).value() << " "
        << max(pyroThermo.T()).value() << endl;
}
```

Implementing multiRegionReactingFoam

Create another empty file named createSolidFields.H in the solid sub-directory and add following.

```
Info<< "Reading solid thermophysical properties\n" << endl;
autoPtr<solidThermo> pPyroThermo(solidThermo::New(pyrolysisMesh));
solidThermo& pyroThermo = pPyroThermo();

volScalarField pyroRho
(
    IOobject
    (
        "solidRho",
        runTime.timeName(),
        pyrolysisMesh
    ),
    pyroThermo.rho()
);
```

Add the following two lines to the last line of the original createFields.H

```
// To create the solid fields
#include "createSolidFields.H"
```

Implementing multiRegionReactingFoam

Replace the contents of the solid/solidRegionDiffusionNo.H file with the following.

```
scalar DiNum = -GREAT;

{
    tmp<volScalarField> magKappa;
    magKappa = pyroThermo.kappa();

    tmp<volScalarField> tcp = pyroThermo.Cp();
    const volScalarField& cp = tcp();

    tmp<volScalarField> trho = pyroThermo.rho();
    const volScalarField& rho = trho();

    DiNum = solidRegionDiffNo
    (
        pyrolysisMesh,
        runTime,
        rho*cp,
        magKappa()
    );
}
```

Implementing multiRegionReactingFoam

Modify Make/files

```
solid/solidRegionDiffNo.C
multiRegionReactingFoam.C

EXE = $(FOAM_USER_APPBIN)/multiRegionReactingFoam
```

Add these to the Make/options

```
EXE_INC = \
    -I./solid \
    -I$(LIB_SRC)/thermophysicalModels/solidThermo/lnInclude \
    ...

EXE_LIBS = \
    -lsolidThermo \
    ...
```

Thats it, the code should be ready now! Try compiling it.

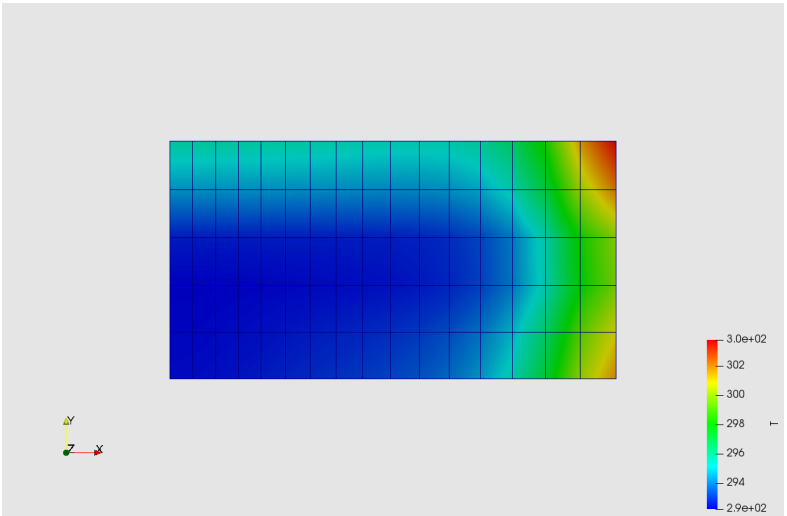
```
wclean
wmake
```

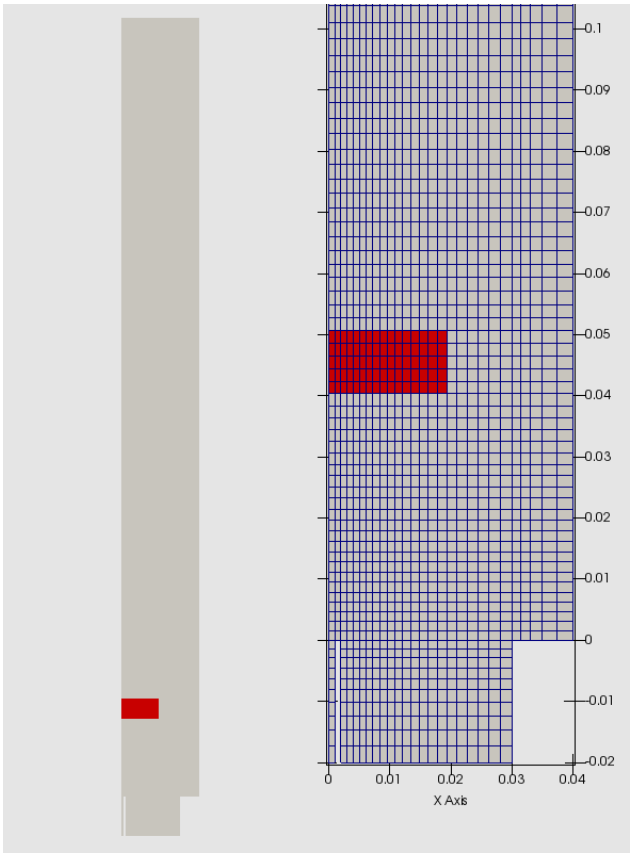
Simple tutorial for multiRegionReactingFoam

Simply go to the case directory and run it:

```
./Allrun
paraFoam &
```

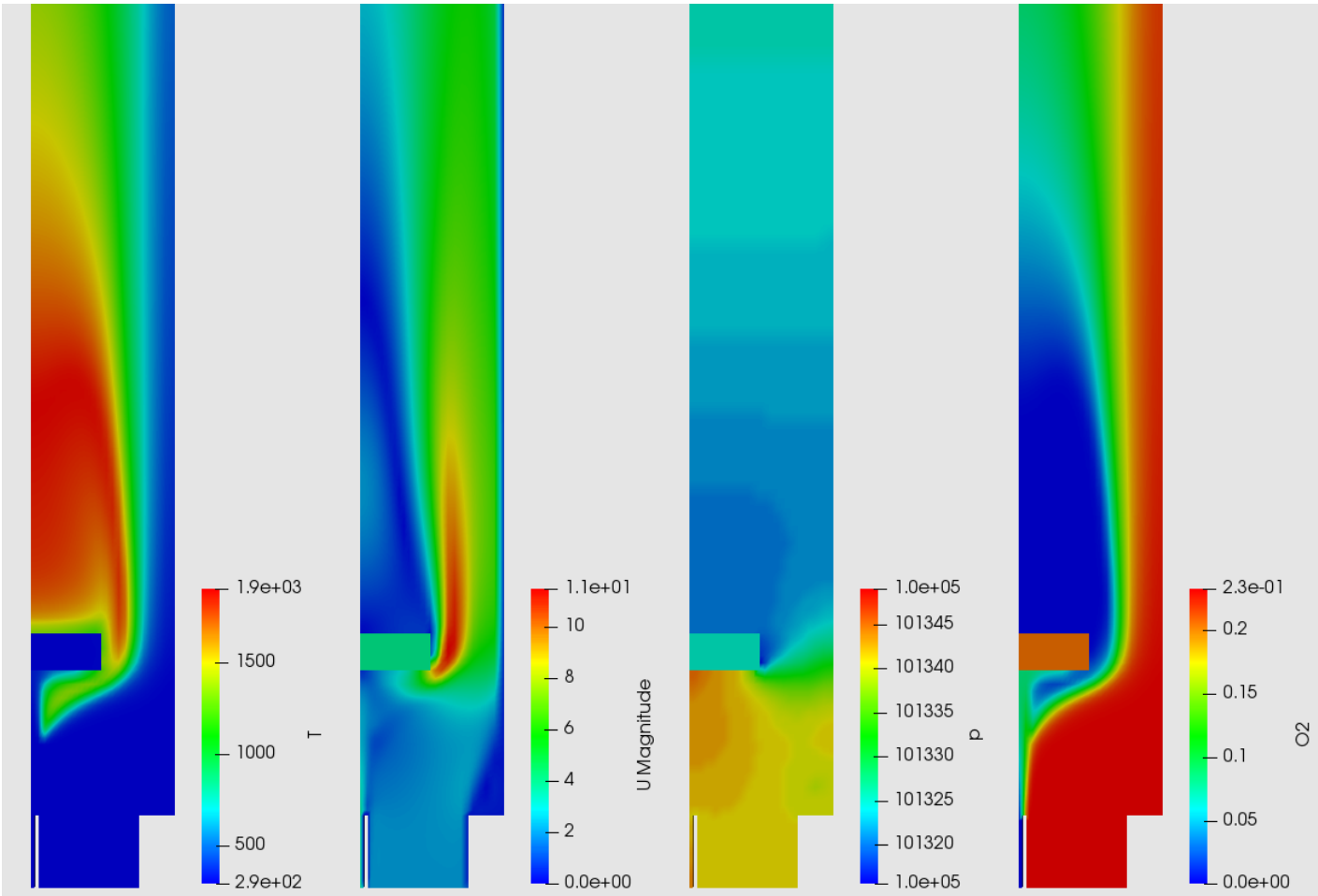
Final solid temperature at t=0.5s





Motivation for multiRegionReactingFoam

Some results after t=0.5s



Final Slide

Thank you for your time!

Good luck!