Presentation

Chalmers University of Technology

CFD with OpenSource software, Project

conjugateHeatFoam with explanational tutorial together with a buoyancy driven flow tutorial

Developed for OpenFOAM-1.5-dev
Requires: A computer

October 17, 2010

Outline

- conjugateHeatFoam solver - with simple example
- A buoyantSimpleFoam tutorial
- Develop a tool to calculate Nusselt number, `NusseltCalc`

conjugateHeatFoam - with simple example

- conjugateHeatFoam - standard implemented in OpenFoam 1.5-dev
- Used for heat problems with multiple regions
- Restricted to laminar, incompressible flow

The solver is found in: `$FOAM_SOLVERS/conjugate/conjugateHeatFoam`

conjugateHeatFoam.C

```
#include "fvCFD.H"
#include "coupledFvMatrices.H"
#include "regionCouplePolyPatch.H"

int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createFluidMesh.H"
#   include "createSolidMesh.H"
#   include "createFields.H"
#   include "createSolidFields.H"
#   include "initContinuityErrs.H"
```

conjugateHeatFoam.C - inside the time-loop

```
    Info<< "\nStarting time loop\n" << endl;
    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "solveFluid.H" // Simplified Navier stokes
#       include "solveEnergy.H" // Energy equation

        runTime.write();

        Info<< "ExecutionTime = "
            << runTime.elapsedCpuTime()
            << " s\n\n" << endl;
    }
    Info<< "End\n" << endl;
    return(0);
}
```

solveFluid.H & solveEnergy.H

- Navier-Stokes, assumed laminar incompressible flow

$$\frac{\partial u}{\partial t} + \nabla(\phi u) - \nabla(\nu \nabla u) = -\nabla p \tag{1}$$

- Energy equation solved in coupled manner on both domains

$$\frac{\partial T}{\partial t} + \nabla(\phi T) - \nabla(\alpha \nabla T) = 0 \tag{2}$$

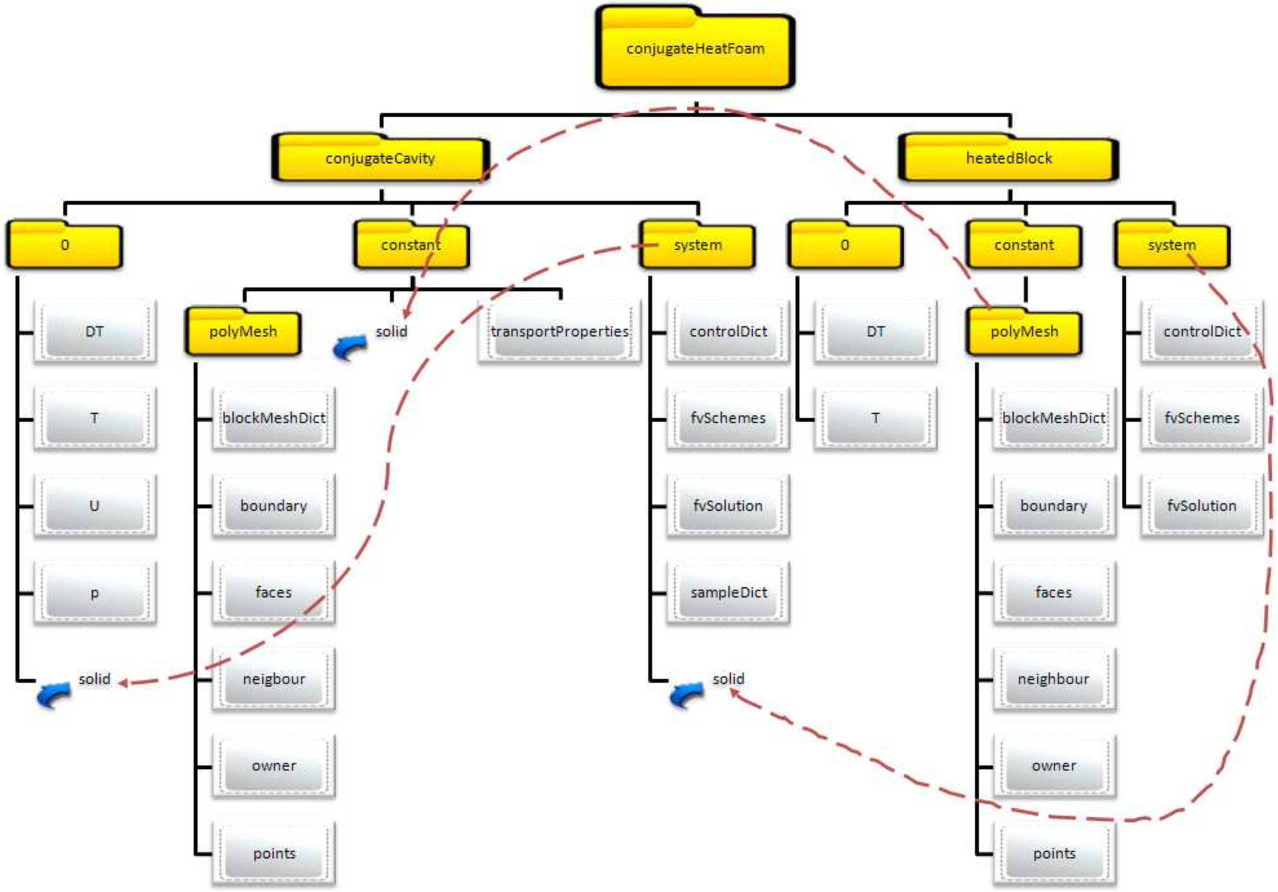
$$\frac{\partial T_{solid}}{\partial t} - \nabla(\alpha_{solid} \nabla T) = 0 \tag{3}$$

$$\alpha = \frac{k}{\rho c_p} \tag{4}$$

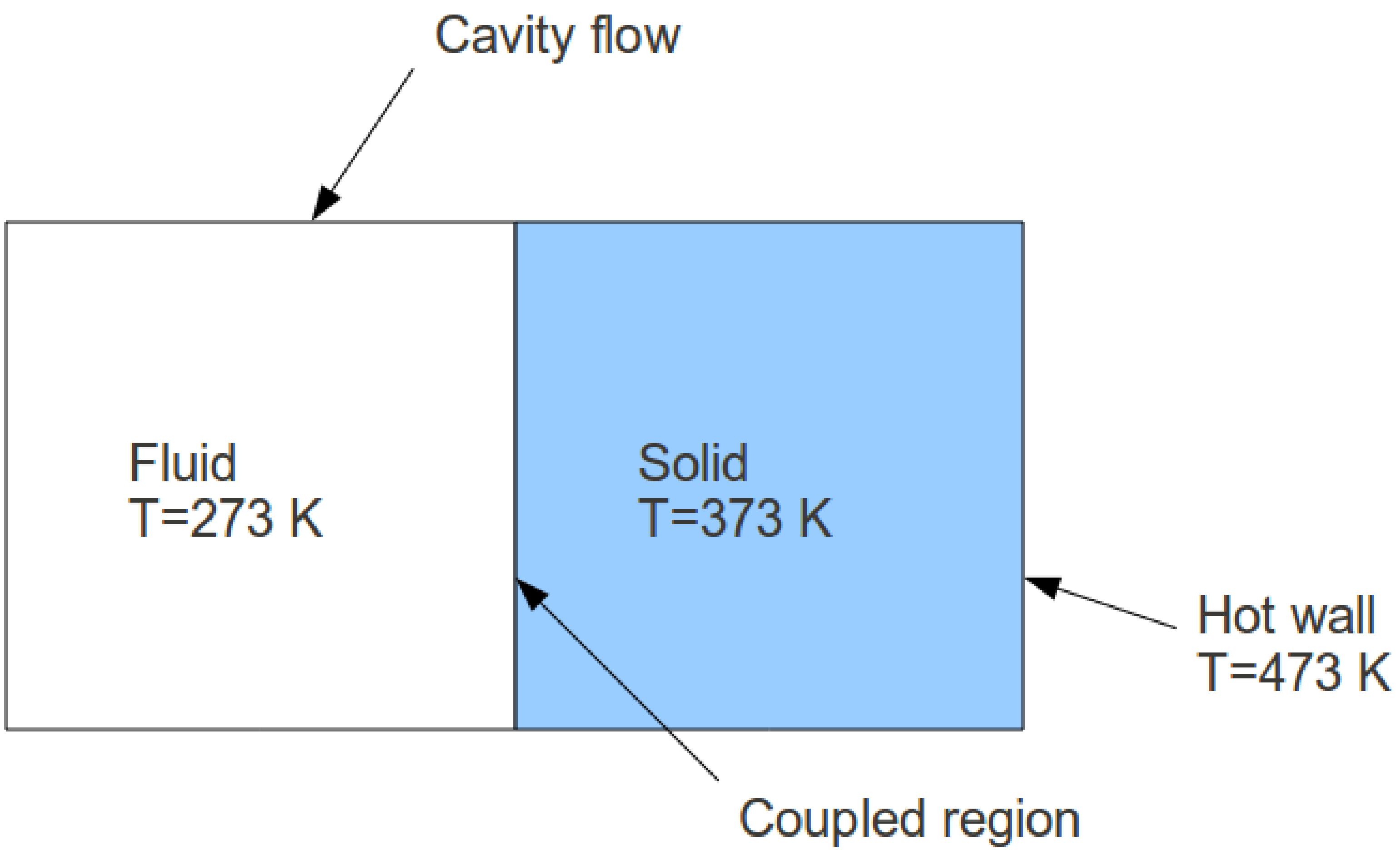
conjugateHeatFoam - simple example

Feel free to follow, copy the example to your `$FOAM_RUN` folder:

```
cp -r $FOAM_TUTORIALS/conjugateHeatFoam/ $FOAM_RUN
```



Problem specification



conjugateHeatFoam - simple example

- Create the both meshes by:
  `blockMesh -case conjugateCavity/`
  `blockMesh -case heatedBlock/`

- Set boundary types `gedit conjugateCavity/constant/polyMesh/boundary` and replace:

```
    right
    {
        type            wall;
        nFaces          10;
        startFace       200;
    }
```

```
right
{
    type regionCouple;
    nFaces 10;
    startFace 200;
    shadowRegion    solid;
    shadowPatch     left;
    attached        on;
}
```

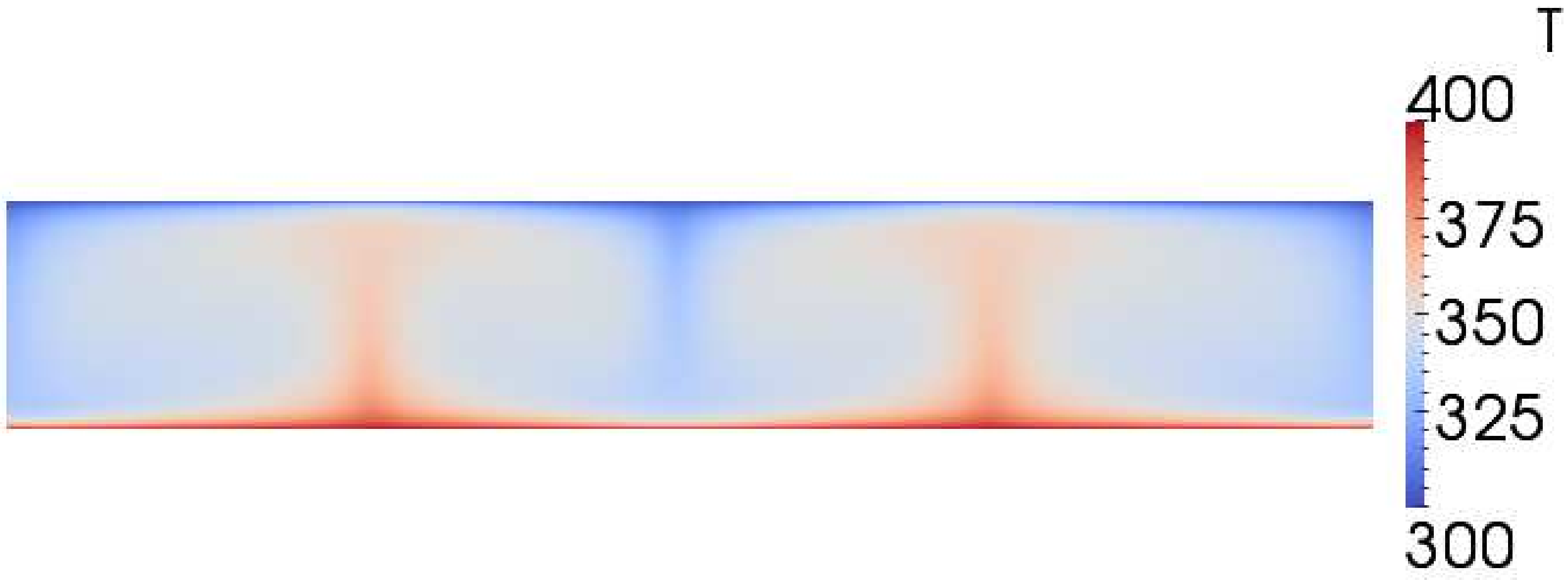
- Repeat for the `heatedBlock`, notice that it is now the left side!

- Set boundary conditions for `U, p, DT T` for both domains.

conjugateHeatFoam - simple example

- Continue by set the solver settings, `fvSolution`, `fvSchemes`, `controlDict`
- `controlDict` in `heatedBlock` not used.
- Run the case by
  `conjugateHeatFoam -case conjugateCavity`
- Evaluate

Buoyancy driven flow - tutorial



- Simulate buoyancy driven flow inside a enclosure
- Hot lower wall and a cooler upper wall, what do we need?

Buoyancy driven flow - tutorial

- Find a proper solver that handles:

  1. Buoyancy flows
  2. Flow can be considered incompressible but since the flow is dependent on gravity, a compressible solver is needed
  3. Steady-state turbulent flow

- Matches: Standard solver `buoyantSimpleFoam`

The solver solves for the momentum equation:

$$\nabla(\phi U) - (\nabla\phi)U\nabla\mu_{eff}\nabla U - \nabla(\mu_{eff}(\nabla U)T) = -\nabla pd - (\nabla\rho)gh \tag{5}$$

The energy equation:

$$\nabla(\phi h) - (\nabla\phi)h\nabla\alpha\nabla h = \nabla(\frac{\phi}{\rho p}) - p\nabla(\frac{\phi}{\rho}) \tag{6}$$

Beyond the pressure and flux is calculated by:

$$\nabla\rho(rUA)\nabla pd = \nabla\phi \tag{7}$$

And a correction of the velcities is made by:

$$U = rUA(\nabla pd + (\nabla\rho)gh \tag{8}$$

Buoyancy driven flow - Boundary conditions

- Solver using `basicThermo.H`, therefore set `thermophysicalProperties`

	Type	Number of moles	Mol weight	c_p	Heat fusion	μ	Pr
mixture	air	1	28.97	1009	0	$208.2 * 10^{-7}$	0.700

- Create a `refValues` for the Nusselt number, should containt the data needed for the Nusselt number. (More information later)
  Set k, length scale and the temperatures.

- Set boundary conditions
  Good to calculate initiate values for k and ϵ. First guess U in y-direction (1 m/s), assume turbulence intensity of 10% .

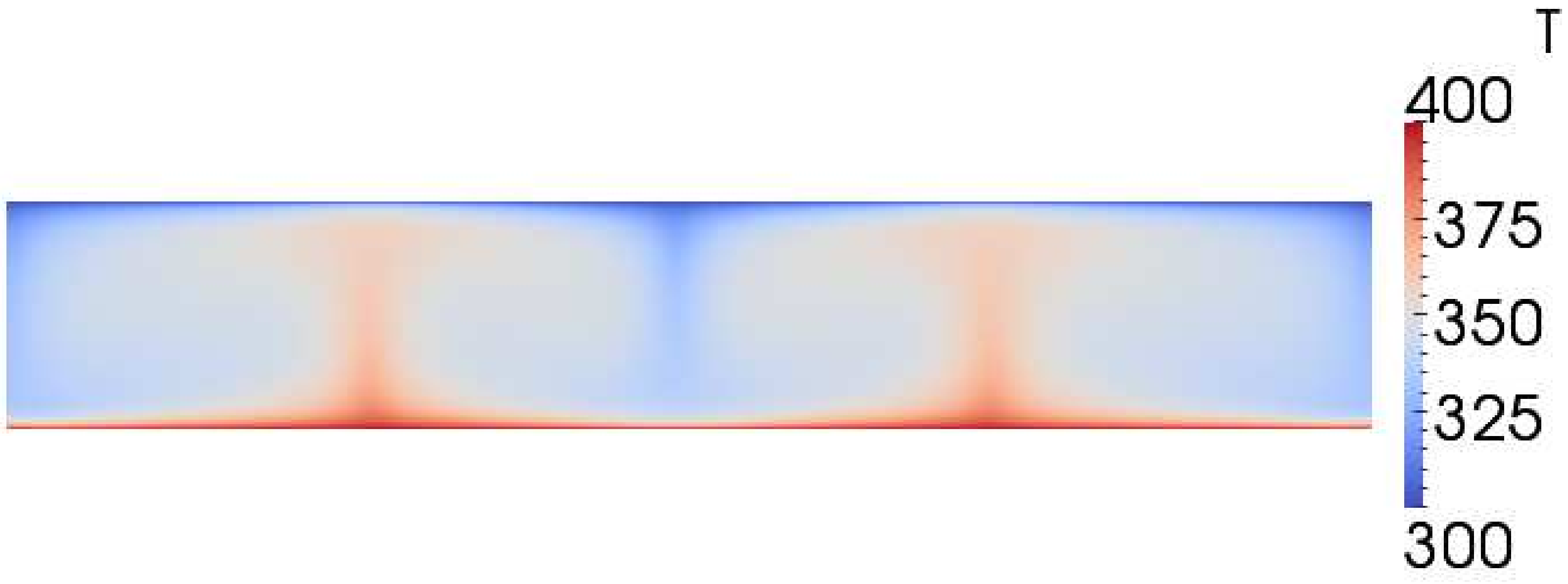
$$k = \frac{3}{2}(u * turbintensity)^2 = \frac{3}{2}(1 * 0.1)^2 = 0.015 \tag{9}$$

Assume lengthscale to 0.1:

$$\epsilon = \frac{C_\mu^{0.75} * k^{3/2}}{l} = \frac{0.09^0.75 * 0.015^{3/2}}{0.1} = 0.00302. \tag{10}$$

Buoyancy driven flow - Run

- Set the settings for the solver!
- Run and evaluate!



Create solver for Nusselt number, NusseltCalc

- Developed from standard postProcessing tool

```
run
cp -r $FOAM_APP/utilities/postProcessing/wall/wallHeatFlux/ .
```

- Change name of the files, `wallHeatFlux.C` to `NusseltCalc.C`.
- Change the `Make\files`:

```
NusseltCalc.C
EXE = $(FOAM_USER_APPBIN)/NusseltCalc
```

Create solver for Nusselt number, NusseltCalc

- Originally developed for combustion, change this to buoyancy by:

```
#include "basicThermo.H"
```

- Change in the `createFields.H` from combustion to buoyancy by.

```
sed -e "s/hCombustionThermo/basicThermo/g"  createFields.H > tmp.H
mv tmp.H createFields.H
```

Create solver for Nusselt number, NusseltCalc

- Nusselt number is calculated by

$$h = \frac{Q}{T_{hot} - T_{initial}} \tag{11}$$

$$Nu = \frac{h * l}{k} \tag{12}$$

- Need to add this equation to the solver!

Create solver for Nusselt number, NusseltCalc

```
volScalarField NusseltNumber
   (
       IOobject
       (
           "NusseltNumber",
           runTime.timeName(),
           mesh
       ),
       mesh,
       dimensionedScalar("NusseltNumber", heatFlux.dimensions(), 0.0)
   );

   forAll(NusseltNumber.boundaryField(), patchi)
   {
       NusseltNumber.boundaryField()[patchi] = length*
         patchHeatFlux[patchi]/((T_hot-T_initial)*k);
   }

   NusseltNumber.write();
```

Create solver for Nusselt number, NusseltCalc

- Tell the solver where to read the `k`, `T_hot`, `T_initial`, `length`. Create `readRefValues`.

```
Info << "\nReading refValues" << endl;
    IOdictionary refValues
    (
        IOobject
        (
            "refValues",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    scalar k (readScalar(refValues.lookup("k")));
Info << "Conductivity is:"<< k << endl;
    scalar T_initial(readScalar(refValues.lookup("T_initial")));
Info << "Initial temperature is:"<< T_initial << endl;
    scalar T_hot(readScalar(refValues.lookup("T_hot")));
Info << "Hot wall temperature:"<< T_hot << endl;
    scalar length(readScalar(refValues.lookup("length")));
Info << "Length scale is set to:"<< T_hot << endl;
```

Create solver for Nusselt number, NusseltCalc

- Include `readRefValues` in the `NusseltCalc` solver.

```
#include "readRefValues.H"
```

- If everything is done correct, go for `wmake`
- Calculate the Nusseltnumber on `buoyantSimpleFoam`-case!

Thank you!