Magnusson's Technology Center
Chalmers University of Technology

CFD with OpenSource software, Project

conjugateHeatFoam with explanational tutorial together with a buoyancy driven flow tutorial and a convective conductive tutorial

Developed for OpenFOAM-1.5-dev
Requires: A computer

Peer reviewed by:

Author:
Johan Magnusson

October 16, 2010

Contents

Tutorial conjugateHeatFoam

1 Solver conjugateHeatFoam

This section descriebes how the conjugateHeatFoam solver works. An important section to be able to continue with cases involving this solver. The solver is included in OpenFoam-1.5-dev and can be found here:

```
cd  $FOAM_SOLVERS/conjugate/conjugateHeatFoam
```

If the solver is missing it can be downloaded from openfoam-extend.wiki.sourceforge.net/ [1]

```
cd $FOAM_SOLVERS/
svn checkout http://openfoam-extend.svn.sourceforge.net/svnroot/openfoam-extend/
trunk/Core/OpenFOAM-1.5-dev/applications/solvers/conjugate/conjugateHeatFoam/
cd conjugateHeatFoam
wmake
```

Now, back to a deeper look into the solver. The conjugateHeatFoam solver is a solver for heat transfer problems where multiple regions occurs. For example how the free stream flow influences the cooling of a hot wall, heat interaction between water and steel and vice versa. These regions are all associated with its own mesh, in other words, one mesh for the fluid part and one mesh for the solid part.

The fluid part of the solver is based on the icoFoam solver and is therefore restricted to laminar, incompressible flow of Newtonian fluids. The flow equations is by this only solved for the fluid part of the domain. The energy equation is in the other hand solved for both the fluid and solid part. The energy equation is solved in a coupled manner at the matrix level.

To be able to change information between the solid part and the fluid part of the domain a speciel boundary condition, regionCouple is used. This boundary condition is active where the interaction between the two domains occurs.

1.1 Included parts in the conjugateHeatFoam solver

All code for the conjugateHeatFoam solver is found in its directory;
`$FOAM_SOLVERS/conjugate/conjugateHeatFoam`. Some of its part is already included in OpenFOAM by default and is therefore loaded into the solver by #include. Which external data needed by the solver is found in the main file, conjugateHeatFoam.C and is presented below.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#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"
```

Where the included files is found and what it is doing is briefly descreibed below:

- fvCFD.H - A standard file for the finite volume method in OpenFOAM.
  `\$FOAM_SRC/finiteVolume/cfdTools/general/include/`
- coupledFvMatrices.H - Header class for coupledFvMatrices. A coupled finite-volume member.
  `\$FOAM_SRC/coupledMatrix/coupledFvMatrices/`
- regionCouplePolyPatch.H - A region couple poly patch used particular for multi region conjugate simulations.
  `\$FOAM_SRC/OpenFOAM/meshes/polyMesh/polyPatches/constraint/regionCouple/`
- setRootCase.H - Checks the folder structure of the case. If problems occurs a "FATAL ERROR" is returned.
  `\$FOAM_SRC/OpenFOAM/include/`
- createTime.H - Checks runtime according to the controlDict and initiate the time variables.
  `\$FOAM_SRC/OpenFOAM/include/`
- createFluidMesh.H - Defines the fluid part of the domain.
  `\$FOAM_SOLVERS/conjugate/conjugateHeatFoam/`
- createSolidMesh.H -Defines the solid part of the domain.
  `\$FOAM_SOLVERS/conjugate/conjugateHeatFoam/`
- createFields.H - Creates the fields for the fluid part of the domain, i.e U, p, T, DT, phi.
  `\$FOAM_SOLVERS/conjugate/conjugateHeatFoam/`
- createSolidFields.H - Creates the fields for the solid part of the domain.
  `\$FOAM_SOLVERS/conjugate/conjugateHeatFoam/`
- initContinuityErrs.H - Declare and initialise the cumulative continuity error.
  `\$FOAM_SRC/finiteVolume/cfdTools/general/include/`

The second part of the conjugateHeatFoam.C file contains the solver loop. It is here the runTime field is set, which confirms the runtime and the equations are loaded and solved by the solver.

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

#       include "solveFluid.H"
#       include "solveEnergy.H"

        runTime.write();

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

The equations that is going to be solved during the process are external, `solveFluid.H` and `solveEnergy.H` files and are loaded into the solver separately.

1.2 solveFluid.H

```
{
    // Detach patches
#   include "detachPatches.H"
#   include "readPISOControls.H"
#   include "CourantNo.H"

    fvVectorMatrix UEqn
    (
        fvm::ddt(U)
      + fvm::div(phi, U)
      - fvm::laplacian(nu, U)
    );

    solve(UEqn == -fvc::grad(p));

    // --- PISO loop
    for (int corr = 0; corr < nCorr; corr++)
    {
        U = UEqn.H()/UEqn.A();
        U.correctBoundaryConditions();
        phi = fvc::interpolate(U) & mesh.Sf();
        for (int nonOrth = 0; nonOrth <= nNonOrthCorr; nonOrth++)
        {
            fvScalarMatrix pEqn
            (
                fvm::laplacian(1.0/UEqn.A(), p) == fvc::div(phi)
            );
            pEqn.setReference(pRefCell, pRefValue);
            pEqn.solve();
            if (nonOrth == nNonOrthCorr)
            {
                phi -= pEqn.flux();
            }
        }
#       include "continuityErrs.H"
        U -= fvc::grad(p)/UEqn.A();
        U.correctBoundaryConditions();
    }
}
```

It is the flow equations for the fluid part of the domain that are found in the solveFluid.H file. The equation that is solved is naver-stokes, simplified by assuming laminar incompressible flow.

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

After the flow equations are solved the iteration loop enters the PISO-loop and a corrector loop is initiated by;

```
for (int corr=0; corr<nCorr; corr++).
```

Also corrects the velocities according to the newest calculated pressurefield. In the end it calculate the continuity errors. The loop is done a prescribed number of times which is set in the fvSolution file found in /system/fvSolution [2].

```
PISO
{
    nCorrectors     2; // Number of times to run the PISO-loop corrector.
}
```

The PISO-loop is an efficient method to solve unsteady problems in the Navier-Stokes equation. Compared to the SIMPLE algorithm the PISO algorithm dont needs under-relaxation. It also allows the user to calculate the momentum corrector more than once. As it can be seen above it is done two times (`nCorrectors 2`).

1.3 solveEnergy.H

```
{
    // Decoupled patches
#   include "attachPatches.H"
    // Solid side
#   include "readSolidControls.H"
    for (int nonOrth = 0; nonOrth <= nNonOrthCorr; nonOrth++)
    {
        coupledFvScalarMatrix TEqns(2);
        // Add fluid equation
        TEqns.set
        (
            0,
            new fvScalarMatrix
            (
                fvm::ddt(T)
              + fvm::div(phi, T)
              - fvm::laplacian(DT, T)
            )
        );
        // Add solid equation
        TEqns.set
        (
            1,
            new fvScalarMatrix
            (
                fvm::ddt(Tsolid) - fvm::laplacian(DTsolid, Tsolid)
            )
        );
        TEqns.solve();
    }
}
```

The energy equation is solved for both domains and the communication between them is done by the previous mentioned boundary condition, regionCouple.

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

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

2 conjugateHeatFoam example

To get better understanding of the conjugateHeatFoam solver a simple case is going to be used and solved. The case is already built in as a tutorial in OpenFoam-1.5-dev and can be found here:

```
cd  $FOAM_TUTORIALS/conjugateHeatFoam
```

If the tutorial is missing it can be downloaded from openfoam-extend.wiki.sourceforge.net/:

```
run
svn checkout http://openfoam-extend.svn.sourceforge.net/svnroot/openfoam-extend/
trunk/Core/OpenFOAM-1.5-dev/tutorials/conjugateHeatFoam/
cd conjugateHeatFoam
```

A first look in the conjugateHeatFoam tutorial makes it easy to believe that there is two cases, but that is wrong, there is instead two "cases" or domains combined as mentioned before. The folders and files are structured in a specific way, described by the following figure. The conjugateCavity case contains links, as for example the `0` folder. This link, links to the information from the heatedBlock case. The solver is therefore started in the conjugateCavity case and the information from the heatedBlock case is taken through the links. How the information is transferred between the different domains is described below.

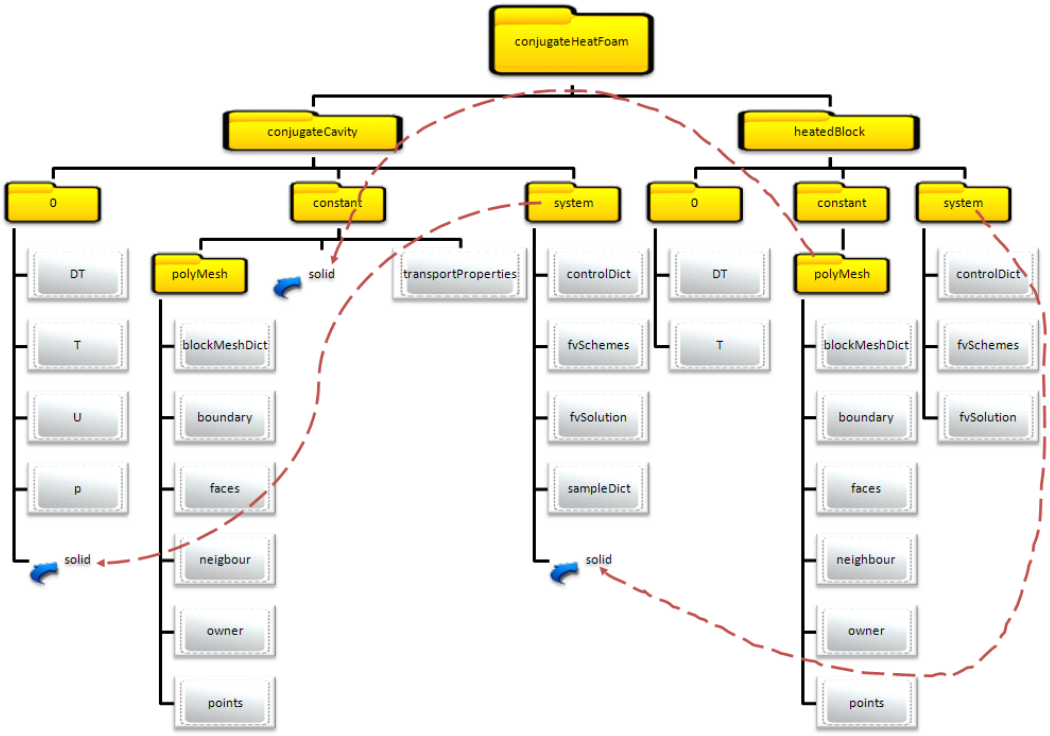

Figure 1: Folder-structure of a conjugateHeatFoam case

The information from the heatedBlock-case are linked to the conjugateCavity-case. The folder structure is read by the solver and the domains are solved separately. The information between them are as mentioned before transferred between the cases through the coupled boundary function, regionCoupling.

IMPORTANT: During the tutorial the root-folder will be mentioned as conjugateHeatFoam. As it can be seen in the figure over the folder structure the conjugateHeatFoam folder is at the top, and contains both cases. If `blockMesh` does not work the firs time, try a second time since its a common bug that a error message occur the first time.

2.1 Problem specification

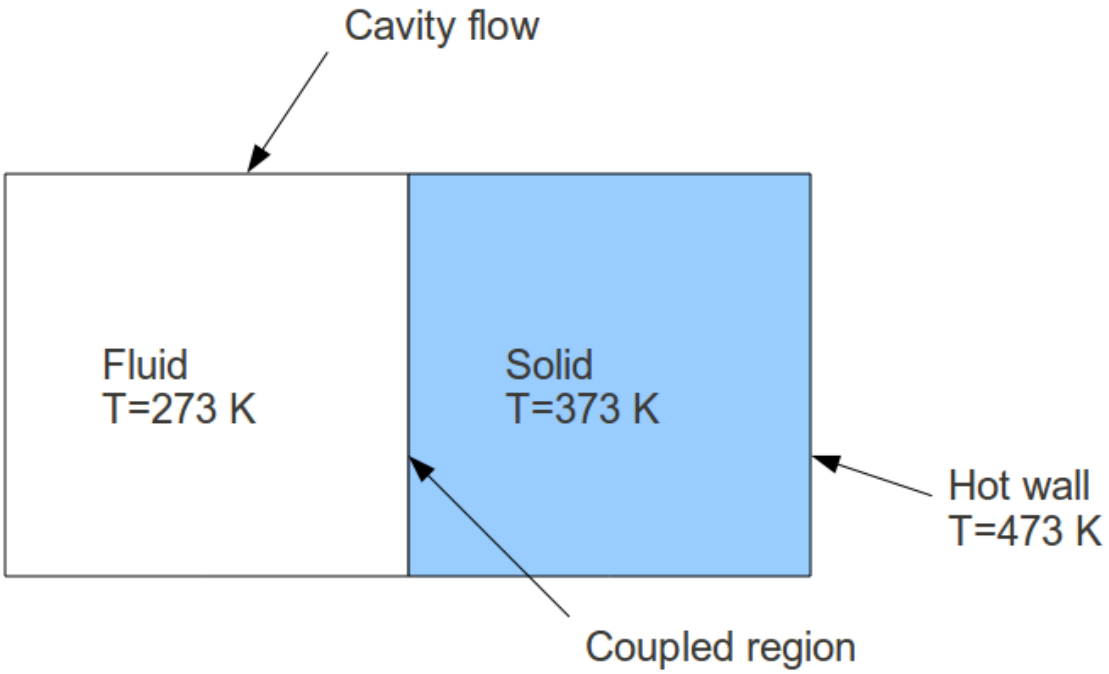

Figure 2: Domain, initial conditions

The geometry of this case consists of a two blocks, Figure 7, one of these blocks looks exactly the same as the cavity case $FOAM_TUTORIALS/icoFoam/cavity. This blocks is filled with air, with a 0.1x0.1 meter base and a deep of 0.01 meter. The second block has the same form as the first block but contains the solid part of the domain. Due to the small deep the case is treated as a 2D-case.

Velocity on the cavity side is 1 m/s in positive x-direction. The internal temperaturefield is 273 K. The solid part of the domain has an internal temperature of 373 K and a right boundary layer at 473 K. The coupled region is where the intersection between the two domain occurs.

Simple heat transfer knowledge tells that heat goes from hot to cold and a good assumption in this case is that the heat will travel to the left part of the solid, into the fluid part of the domain. Let's see if this assumption is correct!

2.2 Mesh generation

The solid part of the mesh is found in the heatedBlock folder and is edited by:

```
gedit heatedBlock/constant/polyMesh/blockMeshDict
```

The entries are as follows:

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
```

```
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.1;
vertices
(
    (1 0 0)
    (2 0 0)
    (2 1 0)
    (1 1 0)
    (1 0 0.1)
    (2 0 0.1)
    (2 1 0.1)
    (1 1 0.1)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (10 10 1) simpleGrading (1 1 1)
);
edges
(
);
patches
(
    patch topAndBottom
    (
        (3 7 6 2)
        (1 5 4 0)
    )
    patch left
    (
        (0 4 7 3)
    )
    patch right
    (
        (2 6 5 1)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

The solid mesh consists of a simple block with 10x10x1 cells. Case is symmetric in z-direction, according to the empty-patch. Create the mesh by typing:

```
blockMesh -case heatedBlock/
```

The fluid mesh is found in the conjugaCavity-folder. Edit the mesh parameters by:

```
gedit conjugateCavity/constant/polyMesh/blockMeshDict
```

The entries are as follows:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.1;
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (10 10 1) simpleGrading (1 1 1)
);
edges
(
);
patches
(
    wall movingWall
    (
        (3 7 6 2)
    )
```

```
    wall left
    (
        (0 4 7 3)
    )
    wall right
    (
        (2 6 5 1)
    )
    wall bottom
    (
        (1 5 4 0)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

The fluid mesh consists of a simple block of 10x10x1 cells. All sides exepts the symmtric ones are treated as walls. Finally, create the fluid mesh by typing:

```
blockMesh -case conjugateCavity
```

At last the intersection between the two domains has to be specified. By looking at the blockMeshDict in conjugateCavity case, it is the right patch that connects with the heatedBlock. At the heatedBlocked it is the left patch that connects with the conjugateCavity case. To specify this, open first the boundary in the conjugateCavity domain:

```
gedit conjugateCavity/constant/polyMesh/boundary
```

Replace the following lines:

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

With the following change:

```
right
{
    type regionCouple;
    nFaces 10;
    startFace 200;

    shadowRegion    solid;
    shadowPatch     left;
    attached        on;
}
```

The same procedure is done in the heatedBlock domain.

`gedit heatedBlock/constant/polyMesh/boundary`

Replace the following lines:

```
left
{
    type            patch;
    nFaces          10;
    startFace       200;
}
```

With the following change:

```
left
{
    type regionCouple;
    nFaces 10;
    startFace 200;
    shadowRegion    region0;
    shadowPatch     right;
    attached        on;
}
```

The shadowRegion tells the solver on which region it should get the neighbour information. In this case is the shadowRegion getting its information from the conjugateCavity case.

The shadowPatch tells the solver on which patch in this region the intersection occurs which in this case is the right patch on the conjugateCavity case. At last a specification if the surfaces should be attached or not can be made. The differences between on or off is that when it is on, non-zero velocity vectors is allowed at the surface. Off means that the velocity vectors is zero at the wall. In this case, use attached on since the vectors then are allowed to be non-zero.

2.3 Boundary conditions and initial fields

Now, both meshes are created and there's time to move on and set the boundary conditions. In this case we are required to set the initial and boundary fields for velocity U and pressure p in the fluid domain. Temperature T and the thermal diffusivities DT has to be set for both domains.

The pressure boundary are set to `zeroGradient` at all walls, except the front and back face which are set to `empty`.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "";
```

```
    local           "";

    class           volScalarField;
    object          p;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    walls
    {
        type            zeroGradient;
    }
    right
    {
        type            fixedValue;
        value           uniform 0;
    }
    left
    {
        type            zeroGradient;
    }
    cylinder
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}


// ************************************************************************* //
```

The boundaries for the velocity are set to zero velocity on the wall. The movingWall are set to moving boundary with a velocty in (1,0,0) m/s. Front and back are as pressure set to empty.

```
/*--------------------------------------------------------------------------*\
| =========                 |                                                |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  1.3                                  |
|   \\  /    A nd           | Web:      http://www.openfoam.org              |
|    \\/     M anipulation  |                                                |
\*--------------------------------------------------------------------------*/

FoamFile
{
```

```
    version         2.0;
    format          ascii;

    root            "";
    case            "";
    instance        "";
    local           "";

    class           volVectorField;
    object          U;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //


dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
    walls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    right
    {
        type            zeroGradient;
    }
    left
    {
        type            fixedValue;
        value           uniform (0.1 0 0);
    }
    cylinder
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}


// ************************************************************************* //
```

It is when it comes to the temperature field and the thermal diffusivity field that it can become a little bit tricky. First of all open the initial conditions for the temperature in the conjugateCavity case.

```
gedit conjugateCavity/0/T
```

Be sure that the right boundary is set-up correctly since this boundary has as neighbour the left

patch on the heatedBlock side:

```
    right
    {
        type            regionCoupling;
        value           uniform 273;
        remoteField     T;
    }
```

The remoteField is the oppsite patch, i.e the left on the heatedBlock case. Notice the higher uniform temperature on the heatedBlock domain. Set the temperatures regionCoupling for the heatedBlock case:

```
gedit heatedBlock/0/T
```

```
    left
    {
        type            regionCoupling;
        value           uniform 373;
        remoteField     T;
    }
```

The same procedure is repeated for the thermal diffusivity.

```
gedit conjugateCavity/0/DT
```

```
    right
    {
        type            regionCoupling;
        value           uniform 1e-3;
        remoteField     DT;
    }
```

Notice that on every patch where intersection occurs between the fields, the regionCoupling boundary condition is used, it doesnt matter if it is the temperature of thermal diffusivty. The intersection follows by a regionCoupling boundary condition. For the heatBlock the thermal diffusivity is set to 1e-3. The same on the conjugateCavity side.

```
gedit heatedBlock/0/DT
```

```
    left
    {
        type            regionCoupling;
        value           uniform 1e-3;
        remoteField     DT;
    }
```

2.4 Case control

When the boundaries are set there is time to specify the case controls. A look in the conjugateCavity/system directory shows controlDict, fvSolution and fvSchemes. By these files, all solver parameters can be set, for example which interpolation scheme, solver control and how large the time step should be.

2.4.1 controlDict

To change the timesteps and endtime of the calculation, it is enough to change this in the conjugateCavity/system/controlDict file. This due to that the conjugateHeatFoam solver reads this field

first. Open the controlDict file and set the solving endTime to 2s and the deltaT to 0.01s and writeInterval 5s:

```
gedit conjugateCavity/system/controlDict
```

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         2;

deltaT          0.01;

writeControl    timeStep;

writeInterval   5;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;
// ************************************************************************* //
```

Notice that it is written icoFoam on the applications line. The fluid part of the domain uses as mentioned before the icoFoam to solve the solution. By removing the regionCoupling boundary conditions and replace them with a icoFoam friendly boundary conditions allows the user to run the fluid part of the domain separately.

2.4.2 fvSolution

`gedit conjugateCavity/system/fvSolution`

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    p PCG
    {
        preconditioner   DIC;
        tolerance        1e-06;
        relTol           0;
    };
    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-05;
        relTol           0;
    };
    T+T BiCG
    {
        preconditioner
        {
            type                Cholesky;
        }
        minIter      0;
        maxIter      1000;
        tolerance    1e-6;
        relTol       0.0;
    };
}
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
// ************************************************************************* //
```

It is in the `fvSolution` that the PISO-algorithm, mentioned above in the report, is found.

The pressure is solved with the PCG (Preconditioned conjugate gradient) solver. The preconditioner uses an option called DIC, which is Diagonal incomplete-Cholesky. The velocity U is solved by the PBiCG solver. The PBiCG is the quite the as the PCG, the difference is that PCG uses symmetric asymmetric lduMatrices when the the PBiCG uses asymmetric lduMatrices. As option to this the DILU is used. DILU means Diagonal incomplete-LU. The temperature T+T is solved using the BiCG solver (Bi-conjugate gradient).

The T+T means that there are two temperature fields. If there had been more fields there had been more T:s too. Since the temperature in this case depends on two fields and that the fields can consist of differnt size of meshes. The temperature needs there for to be iterated to fit between both meshes. In this the maximum number of iterations is 1000.

The `tolerance` and `relTol` used when solving both U, p and T tells the specific solver when it should end the iteration for the specifik timestep. The initial value is based on the current value and the solver stops either of the residuals falls below the tolerance or the the ratio of current to initial residuals falls below the solver relative tolerance. Relative solver tolerance set to zero means that the solver is forced to stop when the tolerance criteria is reached [7].

The following solver scheme are used for the `heatedBlock` case:

```
gedit heatedBlock/system/fvSolution
```

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSolution;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    T               ICCG 1e-06 0;
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
}


// ************************************************************************* //
```

Compared to the conjugateCavity domain the heatedBlock case is solved using the SIMPLE-algoritm. Even if the heatedBlock doesnt have any pressure or velocity there needs to be a specified solution algoritm. The one that is used officially by the conjugateHeatFoam solver is the PISO algorithm which is descriebed above.

The temperature is solved using the incomplete Cholesky preconditioned conjugate gradient. It is derived from the general preconditioned conjugate gradient but with a choice of a pre-selected

preconditioner.

2.4.3 fvSchemes

The `fvSchemes` sets the numerical schemes to the case. It tells the solver in which manner the different terms should be solved. In an euler way as the ddtScheme means that the term is solved in first order, bounded and implicit way.

A Gauss linear way means that the discretisation using gauss. It needs also a interpolation scheme and the linear scheme means that it interpolate by second order. The upwind also occurs and this means interpolation in a first order upwind manner.

The laplacian schemes using corrected gauss linear schemes. The corrected addition means the the interpolation needs to be conservative.

The chosen interpolation schemes for the conjugateCavity case is showned below:

`gedit conjugateCavity/system/fvScheme`

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSchemes;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;

    div(phi,T)      Gauss upwind;
}
```

```
laplacianSchemes
{
    default         none;
    laplacian(nu,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;

    laplacian(DT,T)     Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
    interpolate(HbyA) linear;
}

snGradSchemes
{
    default         corrected;
}
fluxRequired
{
    default         no;
    p;
}


// ************************************************************************* //
```

The interpolation schemes used by the heatedBlock case is showned below:

`gedit heatedBlock/system/fvScheme`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*-------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSchemes;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default Euler;
}
```

```
gradSchemes
{
    default         Gauss linear;
    grad(T)         Gauss linear;
}

divSchemes
{
    default         none;
}

laplacianSchemes
{
    default         none;
    laplacian(DT,T)  Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
{
    default         no;
    T;
}


// ************************************************************************* //
```

2.5 Run the case and post process

To run the case, type:

```
conjugateHeatFoam -case conjugateCavity
```

This runs the case for the pre-set time. When the iteration process has finished. Open the case and study the result. Hopefully this gives a better a better understanding of how the conjugateHeatFoam solver works.

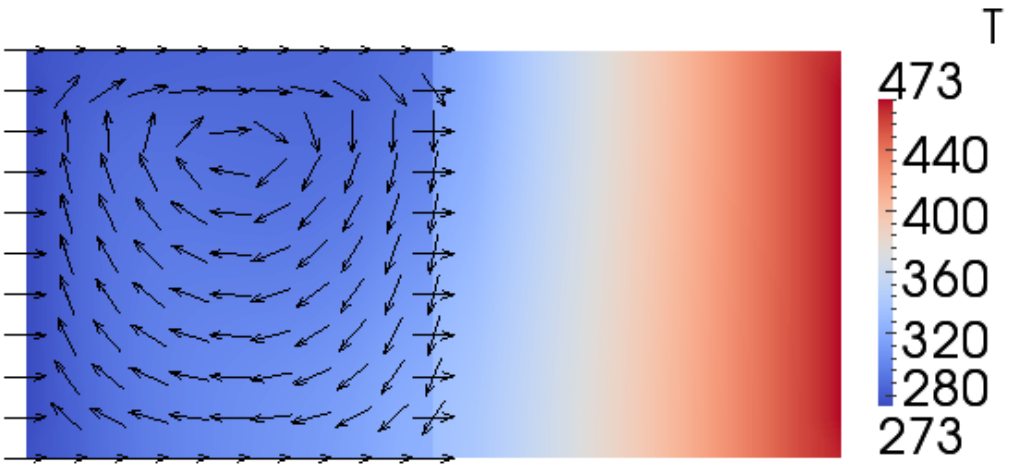

Figure 3: Temperature field with velocity vectors

3 2D free convection in enclosures

3.1 Problem specification

This tutorial allows the user to setup and analyse free convection in enclosure. The forced convection is assumed to ne negligible and the flow is therefore driven by the the temperature differences between the plates.

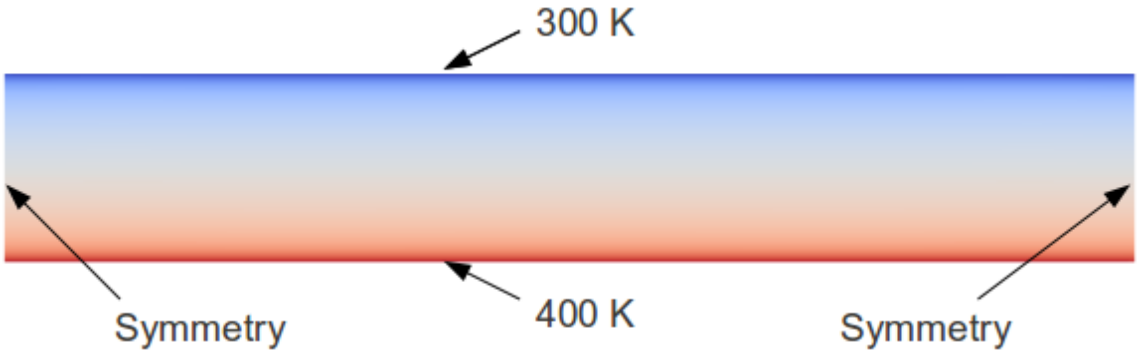

Figure 4: Problem specification of 2D free convection in enclosures

The enclosure is consists of a hot lower wall and a cold upper wall with Dirichlet boundary conditions. The dimensions for the enclosure is 6x1 dm. The hot lower wall has a temperature of 400 K and the upper wall has a temperature of 300 K. The gravity of the case is set to -9.82 m/s$\hat{2}$. The air in the enclosure has a temperature of 350 K. The change in Rayleigh number is made by changing the gravity.

All air properties for the case are taken at a temperature of 350K. The following values where used:

T	ρ	c_p	$\mu * 10^7$	$\nu * 10^6$	k	$\alpha * 10^6$	Pr
K	kg/m^3	$kJ/kg * K$	$N * s/m^2$	m^2/s	$W/m * K)$	m^2/s	
350	0.9950	1.009	208.2	20.92	0.030	29.9	0.700

3.2 Solver-selection

To solve the specified problem above a solver for OpenFOAM is needed. To make the choise easier, a list of demands is which the solver needs to satisfy:

- A solver that handles Buoyancy-flow
- The flow can be considers incompressible but due to buoyancy the gravity changes in the flow, therefore a compressible solver is needed.
- Steady-state solver is preferred.
- Solve for turbulent flow.

According to OpenFOAMs standard solvers there is one that satisfy the demands, in other words buoyantSimpleFoam [3].

Solver buoyantSimpleFoam Here are som brief notes about how the buoyantSimpleFoam solver solves the equations and which equations. The solver solves for the momentum equation:

$$\nabla(\phi U) - (\nabla\phi)U\nabla\mu_{eff}\nabla U - \nabla(\mu_{eff}(\nabla U)T) = -\nabla pd - (\nabla\rho)gh \tag{4}$$

The energy equation:

$$\nabla(\phi h) - (\nabla\phi)h\nabla\alpha\nabla h = \nabla(\frac{\phi}{\rho p}) - p\nabla(\frac{\phi}{\rho}) \tag{5}$$

Beyond the pressure and flux is calculated by:

$$\nabla\rho(rUA)\nabla pd = \nabla\phi \tag{6}$$

And a correction of the velcities is made by:

$$U = rUA(\nabla pd + (\nabla\rho)gh \tag{7}$$

The solver is using the `basicThermo.H` which handels the thermodynamic properties based on the perfect gas assumption.

3.3 Mesh generation

Lets get back to the case again. A solver is selected and it is time to create the mesh and set up the case. The mesh should have a domain of 6x1 dm. Open blockMeshDict and create the mesh.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.01;
```

```
vertices
(
    (0 0 0)
    (60 0 0)
    (60 10 0)
    (0 10 0)
    (0 0 1)
    (60 0 1)
    (60 10 1)
    (0 10 1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (120 60 1) simpleGrading (1 1 1)
);

edges
(
);

patches
(
    wall top
    (
        (3 7 6 2)
    )
    wall bottom
    (
        (1 5 4 0)
    )
    patch side
    (
        (2 6 5 1)
        (0 4 7 3)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);

mergePatchPairs
(
);

// ************************************************************************* //
```

Set the convertToMeters to 0.01. This scales the domain to the desired dimensions. The domain consists of 120 cells in x-direction and 60 cells in y-direction. When the mesh settings are done continue to create the mesh by `blockMesh`. Check the mesh with `checkMesh` to see that everything is OK.

The following step is to set the boundary conditions to imitate the problem specification. Open

the `boundary` and set the boundary conditions to the following:

Surface:	**Boundary type:**
top	wall
bottom	wall
side	wall
frontAndBack	empty

3.4 Boundary conditions and initial fields

According to information above the gravitaion should be set to -9.82 m/s^2. Specify this by open the `environmentalProperties` and set `g` to -9.82.

Next thing to do is to specify all the thermophysical data. This is done in the `thermophysicalProperties`. The `mixture`-line should be set to the following:

	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

At last set the reference pressure to 101325 Pa.

Check so the RASModel is set to `kEpsilon` and that turbulence is `on`. This is done in the `RASProperties`.

To be able to calculate the Nusselt number later we need to specify some reference values that is used in for the Nusselt number. These are specified in the `refValues`:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      refValues;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

k               0.030; // Conductivity

T_initial       350 // Initial temperature

T_hot           400 // Hot wall temperature

length          0.01 // Length scale

// ************************************************************************* //
```

Next stef is to set up all the boundary conditions for the different fields that the solvers needs, epsilon, k, p, U, pd, T.

The boundaries for the epsilon is set to:

Surface:	**Boundary condition:**
top	zeroGradient
bottom	zeroGradient
side	zeroGradient
frontAndBack	empty

The initial field is calculated by some simple assumption. First guess a maximum velocity in y-direction (1 m/s), assume turbulence intensity of 10% and calculated with this the kinetic energy:

$$k = \frac{3}{2}(u * turbintensity)^2 = \frac{3}{2}(1 * 0.1)^2 = 0.015 \tag{8}$$

Continue with assume that the lengthscale is the same as the distance between the two walls and calculate the dissipation.

$$\epsilon = \frac{C_\mu^{0.75} * k^{3/2}}{l} = \frac{0.09^0.75 * 0.015^{3/2}}{0.1} = 0.00302. \tag{9}$$

This is just initial guesses that through the iteration process will change. This gives
`internalField uniform 0.00302;` Continue with set the boundary conditions of k:
`internalField uniform 0.015;`

Surface:	**Boundary condition:**
top	zeroGradient
bottom	zeroGradient
side	zeroGradient
frontAndBack	empty

dp:
`internalField uniform 0;`

Surface:		**Boundary condition:**
top		fixedFluxBuoyantPressure
	value	uniform 0
bottom		fixedFluxBuoyantPressure
	value	uniform 0
side		fixedFluxBuoyantPressure
	value	uniform 0
frontAndBack		empty

U:
`internalField uniform (0 0 0);`

Surface:		**Boundary condition:**
top		fixedValue
	value	uniform (0 0 0)
bottom		fixedValue
	value	uniform (0 0 0)
side		fixedValue
	value	uniform (0 0 0)
frontAndBack		empty

p:
`internalField uniform 101325;`

Surface:	**Boundary condition:**
top	zeroGradient
bottom	zeroGradient
side	zeroGradient
frontAndBack	empty

T:

```
internalField   uniform 350;
```

Surface:		Boundary condition:
top		fixedValue
	value	uniform 300
bottom		fixedValue
	value	uniform 400
side		zeroGradient
frontAndBack		empty

All boundaries are now set.

3.5 Case control

The turbulence model are $k - \epsilon$ and it should solve the equations under steady-state conditions. Be sure that this is done under `ddtScheme`, it should show `steadyState`. This is found in the `fvSchemes`. The velocity-pressure coupling should be set to SIMPLE in the `fvSolution`. Also there is a use of under-relaxation to get the solution more stable during solving. Set the `endTime` to 8000, `deltaT` to 1 in the `controlDict`.

3.6 Run the case

Finally all setting are made and the case is ready to run, type:

```
buoyantSimpleFoam >& log&
```

The result should look something like this:

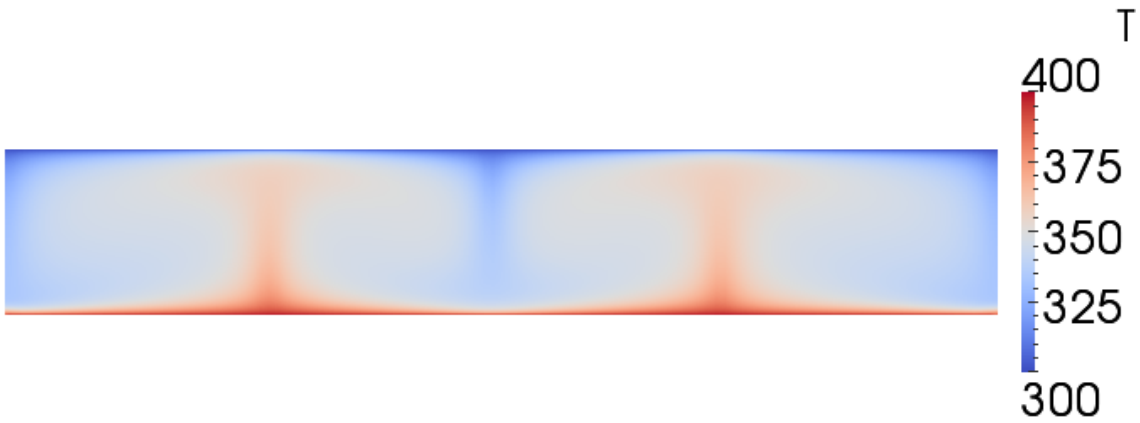

Figure 5: Buoyancy driven flow

3.7 Calculated Nusselt number using NusseltCalc

When the solver has finished it is time to calculated the heat flux (Q) and the Nusselt number. The nusselt number and heat flux will be calculated using NusseltCalc. How NusseltCalc is created and how whats happens in it is described in section NusseltCalc. Calculate nusselt number with:

```
NusseltCalc
```

The nusselt number and heat flux is now calculated and by open paraFoam, it can be plotted. They are only calculated on the patches, not on the hole field. To plot them, open `paraFoam` and the NusseltNumber and wallHeatFlux should be available in the list of cellArrays.

4 NusseltCalc

The NusseltCalc is a new tool developed to calculate the Nusselt number on the boundaries. It is a further development of the already existing postProcessing tool wallHeatFlux. The problem is that wallHeatFlux dont work for buoyancy driven flow so the first thing to do is convert it so it handles these kinds of flow. Copy the tool the run-folder:

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

Change the name of the `wallHeatFlux.C` to `NusseltCalc.C`. Then open `Make/files` and change content to:

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

This creates a link to the NusseltCalc tool and makes it executable from the terminal.

Continue by open the `NusseltCalc.C` file. This tool is from the beginning developed for combustion applications and now we have bounacy flow. So the first thing to do is to take away the `hCombustionThermo.H` and replace it with `basicThermo.H`. The basicThermo handles basic thermodynamic properties which is needed in buoyancy driven flows. The following step is to replace all parts containg `hCombustionThermo` in the `createFields.H` file with `basicThermo`. This is done by the following command:

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

The NusseltCalc is now prepared to calculate the heat flux on a buoyancy driven flow but to calculate the Nusselt number the equation for Nusselt number needs to be added to the solver loop in the `NusseltCalc.C`. The Nusselt number is calculated by using the previously calculated heat flux, Q, which is the heat flux for all patches at the boundary field of a volScalarField. The solver also prints the integrated heat flux for all wall patches by:

```
        Info<< "\nWall heat fluxes [W]" << endl;
        forAll(patchHeatFlux, patchi)
        {
            if (typeid(mesh.boundary()[patchi]) == typeid(wallFvPatch))
            {
                Info<< mesh.boundary()[patchi].name()
                    << " "
                    << sum
                       (
                           mesh.magSf().boundaryField()[patchi]
                          *patchHeatFlux[patchi]
                       )
                    << endl;
            }
        }
        Info<< endl;
```

The used equation for Nusselt number is:

$$h = \frac{Q}{T_{hot} - T_{initial}} \tag{10}$$

$$Nu = \frac{h * l}{k} \tag{11}$$

This is implemented in the `NusseltCalc.C` by including the following loop. Include the loop just below the `wallHeatFlux.write();` line:

```
    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();
```

Whats happening is that it creates a volScalarField with the name of "NusseltNumber". All values in this files will be scalars due do `dimensionedScalar`. The following procedure is loop through all boundary patches and calculate the Nusselt number. As mentioned above it is calculated from the heat flux on the boundaries. The heat flux is calculated from:

```
surfaceScalarField heatFlux =
           fvc::interpolate(RASModel->alphaEff())*fvc::snGrad(h);
```

To be able to full fill the calculation of the nusselt number the length, T_{hot}, $T_{initial}$ and k needs to be specified. These values should the NusseltCalc get from the refValues in the case folder. Therefore do we have to create a readRefValues thats reads the values from a dictionary. This is done by creating a file called `readRefValues.H`. The file should contain the following:

```
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;
```

The files starts by recognize the IOdictionary refValues. Then it continues to read all the needed scalars. For example the conductivity k is read by `readScalar(refValues.lookup("k"))`. The procedure starts by that it search for a line that starts with k, then it takes the scalar on this line and names it k. The procedure is then repeated for all scalars that are specified in the `readRefValues.H` file.

To let the NusseltCalc get these scalars, just include the file inside the `forAll(timeDirs, timeI)`-loop by:

```
#include "createFields.H"
```

Then is the NusseltCalc configuration done and it is time to compile it by `wmake` in the terminal.

5 2D convective and conductive heat transfer

The following tutorial are going to solve convective and conductive heat transfer with a heat generation in 2D. The solver that is used in the case is the conjugateHeatFoam solver.

During the tutorial the root-folder will be mentioned as the folder that contains the `conjugateDomain` and `heatedCylinder`. If `blockMesh` does not work the firs time, try a second time since its a common bug that a error message occur the first time.

5.1 Problem specification

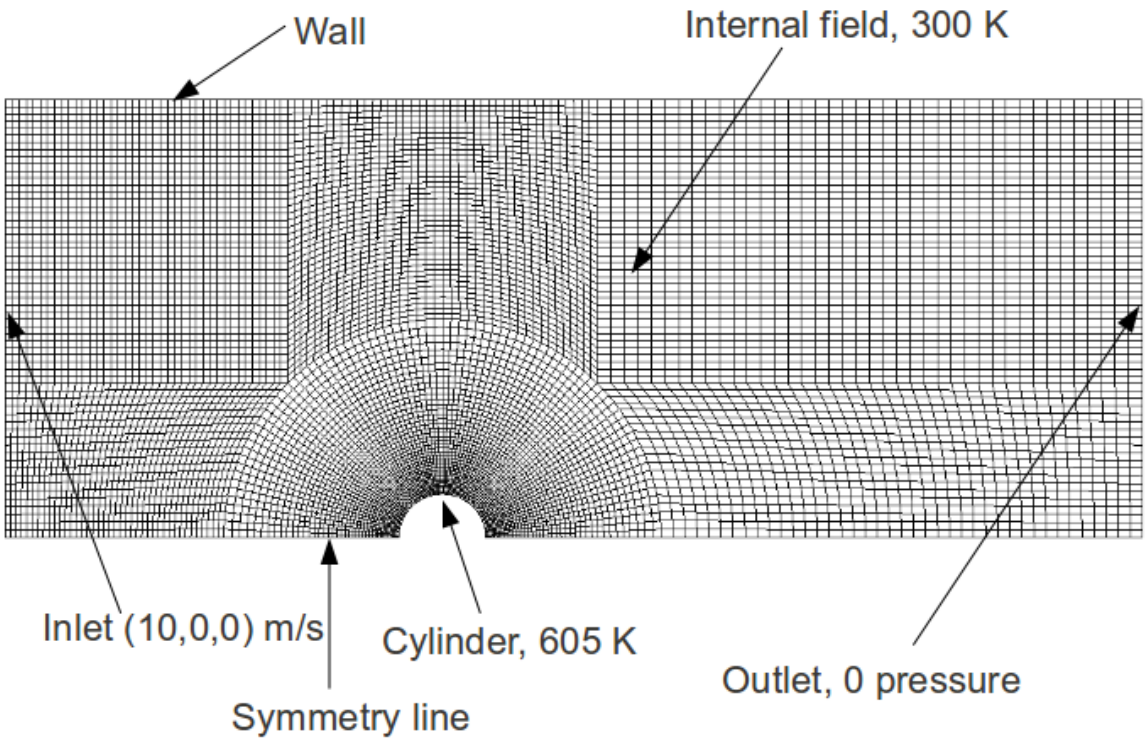

Figure 6: Convective conductive heat transfer, initial conditions

The problem consists of a cylinder in cross-flow. The domain is symmetric so only half of the problem is meshed. All walls in the channel are at 300 K exept the half cylinder which has a temperature of 605 K. The initial temperature of the air inside the channel are at 300 K too. Inlet velocity is set to 10 m/s in x-direction with a 0 pressure outlet boundary. This types of problems can be used to model heat exchanger problems.

The solver conjugateHeatFoam needs the thermal diffusivity to solve and this is specified as:

$$\alpha = \frac{k}{\rho c_p} \tag{12}$$

The data needed for the problem is:

Data:	Air		
Temp (K)	$\rho(kg/m^3)$	k (W/(m*K))	c_p $(J/(kg*K))$
300	1.1614	0.0263	1007
450	0.7740	0.03239	1021

By this data the film-temperature can be calculated and following the thermal diffusivity value for the film temperature:

$$T_{film} = \frac{T_{inf} + T_{wall}}{2} \tag{13}$$

Whichs gives:

$$\alpha_{film} = \frac{k_{film}}{\rho_{film} c_{p,film}} = \frac{0.03239}{0.7740 * 1021} = 0.000040987 \tag{14}$$

$$\alpha_{300K} = \frac{k_{300K}}{\rho_{300K} c_{300K}} = \frac{0.0263}{1.1614 * 1007} = 0.000022488 \tag{15}$$

5.2 Mesh generation

The mesh that is created for the fluid part of the domain is created from 8 boxes. A small grading is used for the cells that approaching the cylinder.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.01;
vertices
(
    (0.5 0 -0.5)
    (2.5 0 -0.5)
    (8 0 -0.5)
    (8 1.7677 -0.5)
    (1.7677 1.7677 -0.5)
    (0.353553 0.353553 -0.5)
    (8 5 -0.5)
    (1.7677 5 -0.5)
    (0 5 -0.5)
    (0 2.5 -0.5)
    (0 0.5 -0.5)
```

```
    (-0.5 0 -0.5)
    (-2.5 0 -0.5)
    (-5 0 -0.5)
    (-5 1.7677 -0.5)
    (-1.7677 1.7677 -0.5)
    (-0.353553 0.353553 -0.5)
    (-5 5 -0.5)
    (-1.7677 5 -0.5)
    (0.5 0 0.5)
    (2.5 0 0.5)
    (8 0 0.5)
    (8 1.7677 0.5)
    (1.7677 1.7677 0.5)
    (0.353553 0.353553 0.5)
    (8 5 0.5)
    (1.7677 5 0.5)
    (0 5 0.5)
    (0 2.5 0.5)
    (0 0.5 0.5)
    (-0.5 0 0.5)
    (-2.5 0 0.5)
    (-5 0 0.5)
    (-5 1.7677 0.5)
    (-1.7677 1.7677 0.5)
    (-0.353553 0.353553 0.5)
    (-5 5 0.5)
    (-1.7677 5 0.5)
);

blocks
(
    hex (5 4 9 10 24 23 28 29) (30 10 1) simpleGrading (4 1 1)
    hex (0 1 4 5 19 20 23 24) (30 10 1) simpleGrading (4 1 1)
    hex (1 2 3 4 20 21 22 23) (20 10 1) simpleGrading (1 1 1)
    hex (4 3 6 7 23 22 25 26) (20 20 1) simpleGrading (1 1 1)
    hex (9 4 7 8 28 23 26 27) (10 20 1) simpleGrading (1 1 1)
    hex (15 16 10 9 34 35 29 28) (30 10 1) simpleGrading (0.25 1 1)
    hex (12 11 16 15 31 30 35 34) (30 10 1) simpleGrading (0.25 1 1)
    hex (13 12 15 14 32 31 34 33) (20 10 1) simpleGrading (1 1 1)
    hex (14 15 18 17 33 34 37 36) (20 20 1) simpleGrading (1 1 1)
    hex (15 9 8 18 34 28 27 37) (10 20 1) simpleGrading (1 1 1)
);

edges
(
    arc 0 5 (0.469846 0.17101 -0.5)
    arc 5 10 (0.17101 0.469846 -0.5)
    arc 1 4 (2.30970 0.956709 -0.5)
    arc 4 9 (0.956709 2.30970 -0.5)
    arc 19 24 (0.469846 0.17101 0.5)
    arc 24 29 (0.17101 0.469846 0.5)
    arc 20 23 (2.30970 0.956709 0.5)
    arc 23 28 (0.956709 2.30970 0.5)
    arc 11 16 (-0.469846 0.17101 -0.5)
```

```
    arc 16 10 (-0.17101 0.469846 -0.5)
    arc 12 15 (-2.30970 0.956709 -0.5)
    arc 15 9 (-0.956709 2.30970 -0.5)
    arc 30 35 (-0.469846 0.17101 0.5)
    arc 35 29 (-0.17101 0.469846 0.5)
    arc 31 34 (-2.30970 0.956709 0.5)
    arc 34 28 (-0.956709 2.30970 0.5)
);

patches
(
    patch down
    (
      //down
        (0 1 20 19)
        (1 2 21 20)
        (12 11 30 31)
        (13 12 31 32)
    )
    wall up
    (
         //up
        (7 8 27 26)
        (6 7 26 25)
        (8 18 37 27)
        (18 17 36 37)
    )
    patch right
    (
        (2 3 22 21)
        (3 6 25 22)
    )

    patch left
    (
        (14 13 32 33)
        (17 14 33 36)
    )
    wall cylinder
    (
        (10 5 24 29)
        (5 0 19 24)
        (16 10 29 35)
        (11 16 35 30)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

The cylinder is meshed in without any grading and consists of 4 blocks. It is found in the `heatedCylinder` folder.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.5-dev                               |
|   \\  /    A nd           | Revision: 1831                                  |
|    \\/     M anipulation  | Web:      http://www.OpenFOAM.org               |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.01;

vertices
(
    (0.20 0.20 -0.5)
    (-0.20 0.20 -0.5)
    (-0.20 0 -0.5)
    (0.20 0 -0.5)

    (0.20 0.20 0.5)
    (-0.20 0.20 0.5)
    (-0.20 0 0.5)
    (0.20 0 0.5)

    (0.353553 0.353553 -0.5)
    (-0.353553 0.353553 -0.5)
    (-0.5 0 -0.5)
    (0.5 0 -0.5)

    (0.353553 0.353553 0.5)
    (-0.353553 0.353553 0.5)
    (-0.5 0 0.5)
    (0.5 0 0.5)

);
blocks
(
    // center block 1
    hex (2 3 0 1 6 7 4 5) (10 10 1) simpleGrading (1 1 1)
    // right 1
    hex (3 11 8 0 7 15 12 4) (10 10 1) simpleGrading (1 1 1)
    // top 1
    hex (1 0 8 9 5 4 12 13) (10 10 1) simpleGrading (1 1 1)
    // left 1
    hex (10 2 1 9 14 6 5 13) (10 10 1) simpleGrading (1 1 1)
);
```

```
edges
(
arc 8 11 (0.4619 0.19134 -0.5)
arc 12 15 (0.4619 0.19134 0.5)
arc 8 9 (0 0.5 -0.5)
arc 12 13 (0 0.5 0.5)
arc 9 10 (-0.4619 0.19134 -0.5)
     arc 13 14 (-0.4619 0.19134 0.5)
);

patches
(
    patch cylinder
    (
        (11 15 12 8)
        (8 12 13 9)
        (9 13 14 10)
    )
    patch symmetry
    (
        (4 7 15 12)
        (5 4 12 13)
        (14 6 5 13)
(6 7 4 5)
        (2 3 0 1)
        (10 2 1 9)
        (1 0 8 9)
        (0 3 11 8)
    )
    patch bottom
    (
(10 2 6 14)
(2 3 7 6)
(3 11 15 7)
     )
);

mergePatchPairs
(
);

// ************************************************************************* //
```

s Create the meshes for the two domains by:

```
blockMesh -case conjugateDomain
blockMesh -case heatedCylinder
```

Next step is to specify which type each boundary should be. This is done according to the problem specification. Open file `conjugateDomain/constant/polyMesh/boundary` and set the conjugateDomain case boundary conditions. Since the solution is symmetric accoring to the `down` surface a standard symmetric OpenFOAM boundary condition is used, `symmetryPlane`. This boundary condition mirrors the plane solution at the specified boundary:

Boundary types on fluid domain:		
Surface:	**Boundary type:**	
down	symmetryPlane	
top	wall	
left	patch	
right	patch	
cylinder	regionCouple	
	shadowRegion	solid
	shadowPatch	cylinder
	attached	on
defaultFaces	empty	

Continue by set the heatedCylinder boundaries:

Boundary types on cylinder domain:		
Surface:	**Boundary type:**	
cylinder	regionCouple	
	shadowRegion	region0
	shadowPatch	cylinder
	attached	on
symmetry	empty	
bottom	patch	

It is important to specify the regionCouple boundary condition correctly. This is described above in the report in section 2.3 which tells that the shadowRegion tells the solver on which region it should get the neighbour information. The shadowPatch tells the solver on which patch in this region the intersection occurs. At last a specification if the surfaces should be attached or not can be made.

5.3 Boundary conditions and initial fields

The boundary conditions on the `conjugateDomain` case is set to the following.
U:
Set internal field to: `internalField   uniform (0 0 0);`

Boundary condition U:		
Surface:		**Boundary type:**
up		fixedValue
	value	uniform (0 0 0)
down		symmetryPlane
right		zeroGradient
left		fixedValue
	value	uniform (10 0 0)
cylinder		fixedValue
	value	uniform (0 0 0)
defaultFaces		empty

p:
Set internal field to: `internalField   uniform 0;`

Boundary condition p:		
Surface:		**Boundary type:**
up		zeroGradient
down		symmetryPlane
right		fixedValue
	value	uniform 0
left		zeroGradient
cylinder		zeroGradient
defaultFaces		empty

T:
Set internal field to: `internalField uniform 300;` and the film temperature at the regionCoupling zone to 450 K. This is a proper assumptoin since the fluid close to the wall is in the mean at much higher temperature than the free stream temperature.

Boundary condition T:		
Surface:		**Boundary type:**
up		zeroGradient
down		symmetryPlane
right		zeroGradient
left		zeroGradient
cylinder		regionCoupling
	value	uniform 450
	remoteField	T
defaultFaces		empty

DT:
Set internal field to the previously calculated thermal diffusivity at 300 K:
`internalField uniform 0.000022488;`. The film temperature is set at the regionCoupling zone and the thermal diffusivity in the same way.

Boundary condition DT:		
Surface:		**Boundary type:**
up		zeroGradient
down		symmetryPlane
right		zeroGradient
left		zeroGradient
cylinder		regionCoupling
	value	uniform 0.000040987
	remoteField	DT
defaultFaces		empty

Continue by setting up the boundary conditions on the `heatedCylinder` case. The thermal diffusivity in on the cylinder is assumed to be 10. This causes the cylinder to be 605 K constantly through the iteration process. T:
Set internal field to: `internalField uniform 605;`

Boundary condition T:		
Surface:		**Boundary type:**
cylinder		regionCoupling
	value	uniform 1
	remoteField	T
bottom		fixedValue
	value	uniform 605
symmetry		empty

DT:
Set internal field to: `internalField uniform 10;`

Boundary condition DT:		
Surface:		**Boundary type:**
cylinder		regionCoupling
	value	uniform 10
	remoteField	DT
bottom		zeroGradient
symmetry		empty

The last thing to specify now is the `transportProperties` which is found in the constant directory. The `nu` should be set to 0.00002092 and the `DT` to 0.000022488.

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          transportProperties;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

nu              nu [0 2 -1 0 0 0 0] 0.00002092;

DT              DT [0 2 -1 0 0 0 0] 0.000022488;


// ************************************************************************* //
```

5.4 Case control

The conjugateHeatFoam solver is very dependent on the mesh and the timestep. Small cells requires a small time step and vice versa. In this case set the `deltaT` to 0.00001 and `writeInterval` to 1000 and the `endTime` to 0.06. Its enough to set the PISO algorithm to `nCorrectors 2;` in the `fvSolution`.

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.3                                   |
|   \\  /    A nd           | Web:      http://www.openfoam.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          controlDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         0.06;

deltaT          0.00001;

writeControl    timeStep;

writeInterval   1000;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;
// ************************************************************************* //
```

5.5 Run the case

Run the case by:

```
conjugateHeatFoam -case conjugateDomain
```

Wait and the evaluate the result.

Figure 7: Convective conductive heat transfer around a hot cylinder

6 References

[1] OpenFOAM Extend homepage
http://openfoam-extend.svn.sourceforge.net/viewvc/openfoam-extend/trunk/Core/OpenFOAM-1.5-dev/appli

[2] The PISO algorithm in OpenFOAM
http://openfoamwiki.net/index.php/the_PISO_algorithm_in_OpenFOAM

[3] OpenFOAM Standard solvers
http://www.openfoam.com/features/standard-solvers.php

[4] Abolfazl Shiri, 2008, OpenFOAM Course Final Assignment: Tutorial for Natural Convection Boundary Layer
http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/AbolfazlShiri/NC_Tutorial_Shiri.pdf

[5] basicThermo Class Reference
http://foam.sourceforge.net/doc/Doxygen/html/classFoam_1_1basicThermo.html

[6] wall Heat Flux at Doxygen
http://foam.sourceforge.net/doc/Doxygen/html/dir_4dab3e162f81298be7a4b0af77ce58dd.html

[7] Solution and algorithm control
http://www.openfoam.com/docs/user/fvSolution.php