



CFD with open source software, assignment 3:

Adding electric conduction and Joule heating to chtMultiRegionFoam

Niklas Järnstråt

Reviewed by Arash Eslamdoost

Table of contents

<u>1.Physical background and equations solved.....</u>	<u>3</u>
<u>1.1.Welding.....</u>	<u>3</u>
<u>1.2.Electrical conduction and heat transfer.</u>	<u>3</u>
<u>1.3.Physical equations.....</u>	<u>4</u>
a) <u>Flow and momentum.....</u>	<u>4</u>
b) <u>Thermal conduction (with Joule heating and passive transport included).....</u>	<u>4</u>
c) <u>Electrical conduction (added).....</u>	<u>4</u>
<u>2.Numerical implementation and solvers used.....</u>	<u>4</u>
<u>2.1.Overview and aim.....</u>	<u>5</u>
<u>2.2.Solid part solver.....</u>	<u>5</u>
<u>2.3.Fluid part solver.....</u>	<u>5</u>
<u>3.Modifications implemented.....</u>	<u>6</u>
<u>3.1.Defining the geometry.....</u>	<u>6</u>
a) <u>Solid regions.....</u>	<u>7</u>
b) <u>Fluid regions.....</u>	<u>8</u>
<u>3.2.blockMeshDict</u>	<u>8</u>
<u>3.3.A problem with the wedge boundary condition.....</u>	<u>8</u>
<u>3.4.Adding fields to solver.....</u>	<u>9</u>
<u>4.Wirefeed.....</u>	<u>12</u>
<u>4.1.Changes in the solver.....</u>	<u>12</u>
<u>4.2.Changes in input.....</u>	<u>12</u>
<u>5.Electric conduction.....</u>	<u>13</u>
<u>5.1.Changes in the solver.....</u>	<u>13</u>
<u>5.2.Changes in the interface boundary patch.....</u>	<u>14</u>
<u>5.3.Changes in input.....</u>	<u>15</u>
<u>6. Joule heating.....</u>	<u>15</u>
<u>7.“Lazydog” for using the new solver.....</u>	<u>16</u>
<u>7.1.Mesh generation.....</u>	<u>16</u>
<u>7.2.Assign region properties and solution</u>	<u>16</u>
a) <u>Allrun.....</u>	<u>16</u>
b) <u>Fluid property folders.....</u>	<u>17</u>
<u>7.3.Fields and boundary conditions.....</u>	<u>17</u>
<u>7.4.Solution control.....</u>	<u>19</u>
<u>8.Summary and future work.....</u>	<u>20</u>
<u>9.References.....</u>	<u>20</u>

1. Physical background and equations solved

1.1. Welding

The application we want to simulate is electric arc welding (c.f. Fig. 1), where a current flows through a metal electrode, through a plasma created by elevating temperatures in a gas, into the base metal plates that are to be joined by the process. The energy needed to melt the metal is supplied through electrical resistive heating, also called Joule heating. The electric heating will affect both metal parts and the plasma in the arc. Incidentally, the welding gas (usually a mixture of argon and carbon dioxide) has poor conductivity until sufficient energy has been supplied to heat the gas into a plasma state.

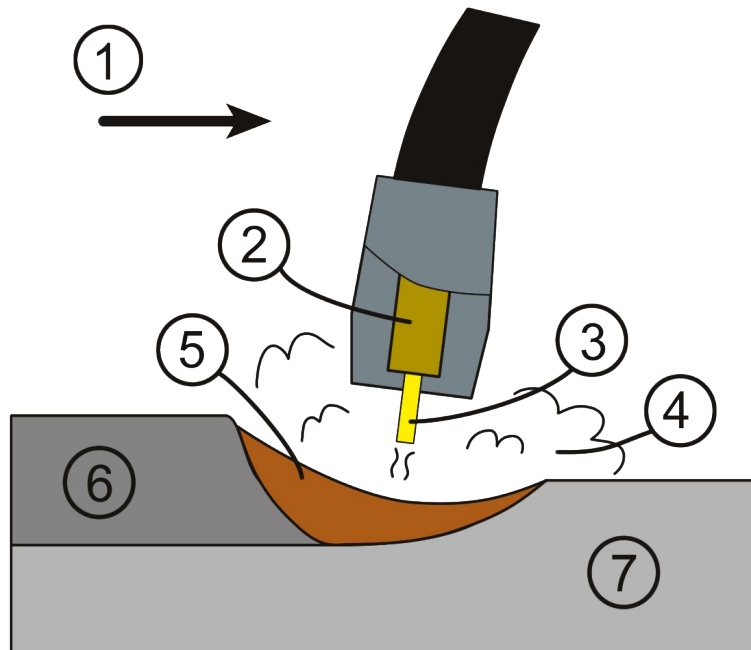


Figure 1: Schematic picture of arc welding (from wikipedia commons). (1) Direction of travel, (2) Contact tube/electrode wire guide, (3) Electrode, (4) Shielding gas, (5) Molten weld metal, (6) Solidified weld metal, (7) Workpiece.

1.2. Electrical conduction and heat transfer.

The main physical processes to simulate in welding arc simulation are electric conduction and heat transfer, both processes described accurately by simple Laplacian equations. There are, however, many factors complicating the calculation of temperature; the main source of heat is electric resistance, convection in the plasma is strongly influenced by both temperature and the electric field, the heat generated in the plasma and the potential drop in the plasma influences both temperature and electric fields in the solids, and all material properties are temperature dependent and varies dramatically over the very large temperature range experienced - from room temperature to maybe 30000 degrees centigrade in the plasma. For this report, several simplifications will be made, as described below.

1.3. Physical equations

There are three separate physical equations to solve: the mechanical equations of flow and equilibrium, the heat equation including conduction and convection and the electrical conduction equation. These are described below.

a) Flow and momentum

For the fluid flow, the simple momentum equation used in chtMultiRegionFoam (see e.g. [Moradnia, 2007]) will be retained unchanged as more detailed plasma models will be added later. For the solid, the velocity field will be set equal to the wire-feed speed in the wire and zero in the stationary nozzle and plate.

b) Thermal conduction (with Joule heating and passive transport included)

Thermal conduction is treated in chtMultiRegionFoam, as the standard heat equation

$$\rho \cdot C_p \cdot dT/dt = \nabla(K \nabla T) \quad [1]$$

is solved, where ρ is the density, C_p the specific heat, K the thermal conductivity and T the temperature. We will also need the electrical conductivity σ , the electric potential V and the velocity vector \mathbf{U} . For our application, we need to add the Joule heating term:

$$\rho \cdot C_p \cdot dT/dt = \nabla(K \nabla T) + \sigma |\nabla V|^2 \quad [2]$$

Convective heat transfer is already included as a major part of the fluid equations, but must be added also in the solid regions, if weld material is going to be added as wire feed in the process:

$$\rho \cdot C_p \cdot dT/dt = \nabla(K \nabla T) + \sigma |\nabla V|^2 - \rho \cdot C_p \cdot \mathbf{U} \cdot \nabla T \quad [3]$$

c) Electrical conduction (added)

The electrical conduction can be considered a quasi-stationary process in the time scale of heat transfer and convection. Thus, it is described by the simple Lagrangian equation

$$- \nabla(\sigma \cdot \nabla V) = 0 \quad [4]$$

where σ is the conductivity, V is the electric potential field.

Note that the the electric current density vector $\sigma \cdot \nabla V$ does not have to be explicitly stored as it can easily be calculated during post-processing. Further, equation [4] is only valid for stationary DC current. In the case of AC or pulsed current, it is necessary to also include the magnetic field and the full set of Maxwell equations.

2. Numerical implementation and solvers used

The solver used as a base for the project is chtMultiRegionFoam, that is a combination of heatConductionFoam and buoyantFoam for conjugate heat transfer between a solid region and fluid region. The solver includes as a main feature "interface patches" to handle the coupling between regions with different physical models, specifically solid and fluid regions.

2.1. Overview and aim

The following changes to chtMultiRegionFoam were identified as necessary steps for the intended application:

General and mechanical:

1. Create a simple test model with appropriate boundary conditions for testing the changes.
2. Add the velocity field also in the solid regions, because it is needed for thermal transport due to wire feed.

Thermal conduction:

3. Add a Joule heating term to both solid and fluid heat conduction solvers.
4. Add passive transport term to solid heat conduction solver.

Electrical conduction:

5. Add the electric potential as a scalar field variable .
6. Add the material property electrical conductivity as a field variable.
7. Add solving for electric potential in both solid and fluid solvers.
8. Adapt internal boundary patches to also cover the electric field.

2.2. Solid part solver

In chtMultiRegionFoam, the only equation solved in the solid regions is simple heat conduction (equation [1]). The entire unmodified solveSolids.H is listed here:

```
{
for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
tmp<fvScalarMatrix> TEqn
(
fvm::ddt(rho*cp, T)
- fvm::laplacian(K, T)
);
TEqn().relax();
TEqn().solve();
}

Info<< "Min/max T:" << min(T) << ' ' << max(T) << endl;
}
```

2.3. Fluid part solver

The mechanical part of the fluid solver, although one of the simpler, buoyancy driven flow, is significantly more complex than the thermal equation. The flow equations are solved by separating flow parameters in density, pressure and velocity, solving these iteratively. These equations are here left unchanged, however, as a research project on plasma simulation at University West (see http://www.ptc.hv.se/extra/pod/?action=pod_show&id=1089&module_instance=1) will provide an advanced solver for the fluid, to be integrated later.

Momentum equation in uEqn.H:

```
// Solve the Momentum equation
tmp<fvVectorMatrix> UEqn
(
  fvm::ddt(rho, U)
+ fvm::div(phi, U)
+ turb.divDevRhoReff(U)
);
```

Momentum equation in pEqn.H:

```
fvScalarMatrix pEqn
(
  fvm::ddt(psi, p)
+ fvc::div(phi)
- fvm::laplacian(rhorUAf, p)
);
```

The thermal solver for the fluid region is essentially the same as for the solid part, although the variable solved for is the thermal energy $h = C_p \cdot dT$ rather than the temperature, as conventions differ between solid mechanics and fluid mechanics. When the heat capacity is temperature dependent, the conversion between thermal energy and temperature becomes somewhat complex, but all we need to know for now is that in effect, equation [1] is solved, albeit using different notation.

Thermal equation in hEqn.H:

```
fvScalarMatrix hEqn
(
  fvm::ddt(rho, h)
+ fvm::div(phi, h)
- fvm::laplacian(turb.alphaEff(), h)
==
DpDt
);
```

3. Modifications implemented

3.1. Defining the geometry

We use a stylized weld nozzle modeled axisymmetrically. The model consists of 5 separate regions as shown in fig. 2 and described below.

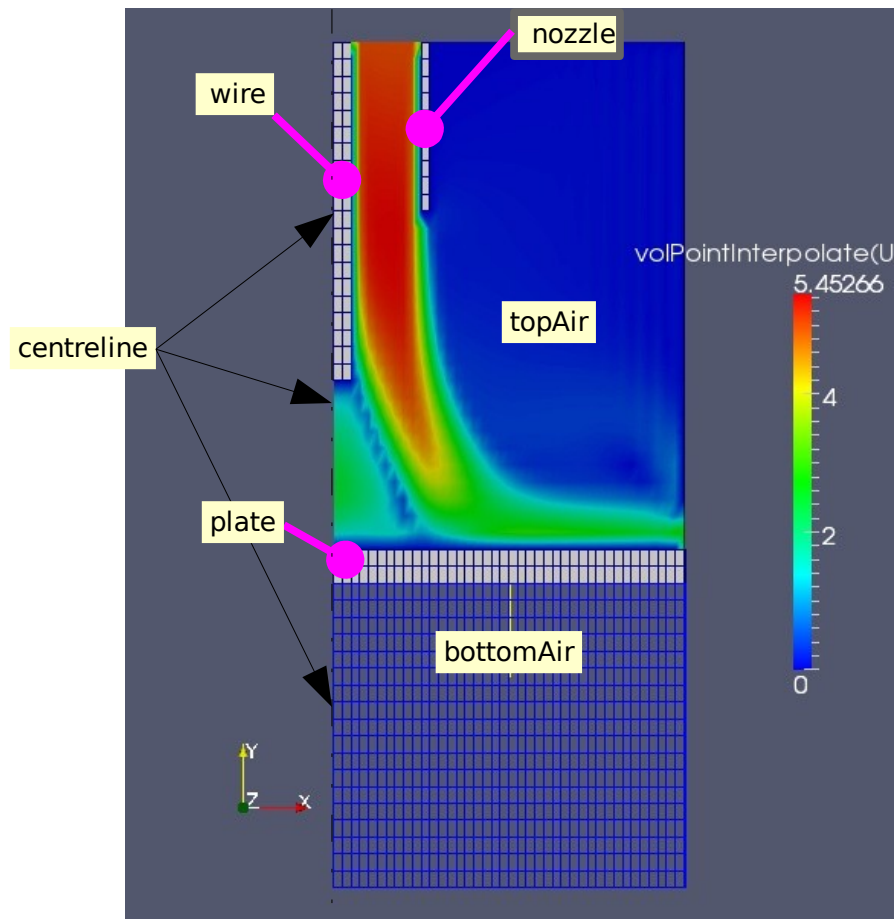


Figure 2: Computational model. Solid regions (wire, nozzle, plate) shown in grey with mesh, for the fluid region bottomAir the mesh is shown in blue and in the fluid region topAir the velocity field is shown, calculated using prescribed flow at the nozzle inlet, outlet boundary condition at the outer radius of the model and zero velocity at all other boundaries.

a) Solid regions

“wire”: the top left corner in figure, this is the filler material also acting as an electrode. The wirefeed from the top and melting off at the bottom is modelled by prescribing an internal velocity value corresponding to the wirefeed rate in the vertical direction.

“nozzle”: a cover containing the protective welding gas and guiding it around the wire to protect molten metal from oxidation. The nozzle is tube shaped and coaxial with the wire. It is the second gray line, one element thick in figure .

“plate”: In real welding, this would be the two pieces of metal being joined. Here, however, as a simplification we model one solid sheet, omitting also the ridge that would be formed by filler material being added. The plate is electrically grounded, acting as the other electrode. The grounding cable is attached far from the weld, which is simulated by prescribing zero electric potential along the circumference of the plate.

b) Fluid regions

“topAir”: the upper fluid region, in contact with all three solid bodies. topAir is the coloured region in figure. The boundary conditions are constant flow rate and constant temperature at the inlet inside the nozzle, and free outlet at the right hand boundary (outer circumference).

“bottomAir”: the lower fluid region. this region is not very important for the simulation, but serves to give mild cooling to the bottom of the plate.

3.2. blockMeshDict

The blockMeshDict file defines as blocks, bounding the entire computational domain, that will later be divided into solid and fluid regions. In the tutorial, only one set of boundary conditions per direction is applied in each of solid and fluid regions. However, because of the forced flow of protection gas inside the nozzle, we need to apply different boundary conditions inside the nozzle and outside it from the top edge. Thus, we need the boundary to be divided in two patches in the radial direction and to achieve this, two blocks also must be defined in blockMeshDict. (see the file at the course homepage, http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/NiklasJarvstrat/JouleMultiRegionFoam-solverAndCases.tgz)

3.3. A problem with the wedge boundary condition

A problem was encountered by the utility splitMeshRegions when attempting to make an axisymmetric model by using the wedge boundary condition. SplitMeshRegions is used to define regions after the cells have been created. In the multiRegionHeater tutorial, setSets was first used to define one square bounding box containing all the geometry. Then square regions were defined and assigned sets and properties. The crucial difference in our example was that it was desired to assign different boundary conditions to parts of the top boundary, with 5m/s gas velocity prescribed inside the nozzle and stationary wall boundary outside the nozzle. For this purpose, it was necessary to have two different boundary patches and thus two different interior patches were created, as defined in figure 3. For this geometry, splitMeshRegions correctly meshed the first three regions, including two fluid and one solid region, but crashed on encountering the fourth region. The problem occurred for the “wire” region, top left corner of the model, see fig. 2. Thus, a series of trials with different settings for the block divide radius and wire dimensions was performed, giving the results shown in table 1. Simulation results using the data in the first row of table 1 is shown in figure 4.

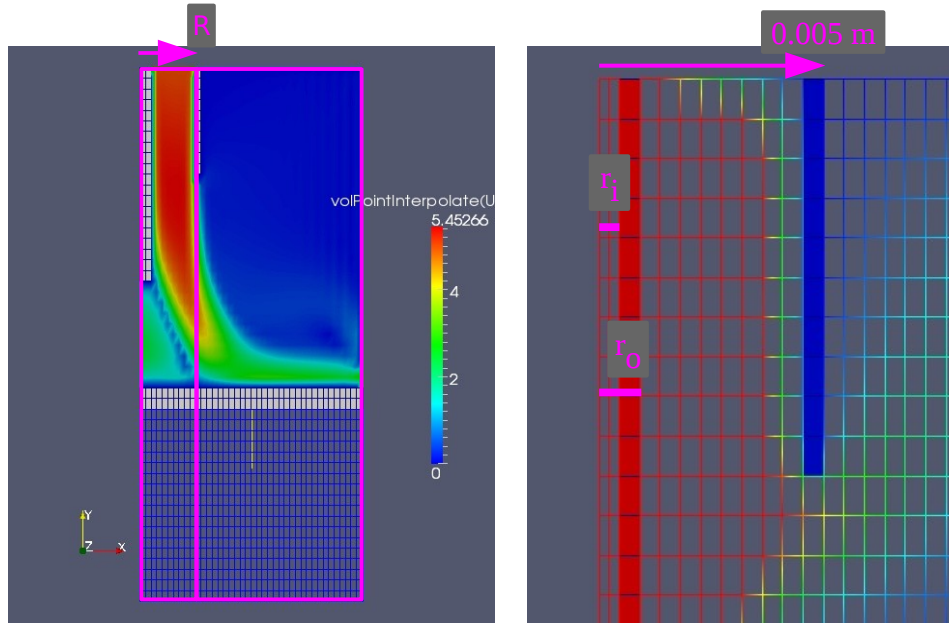


Figure 3a,b: Geometry for wedge boundary error tests. a(left): Definition of patches and blockMesh divide radius (r). b(right): Definition of wire inner (r_i) and outer (r_o) radii.

Table 1: Problem inventory tests

blockMesh divide radius (R)	makeCellSets inner wire radius (r_i)	makeCellSets outer wire radius (r_o)	Error in splitMeshRegions
0.0005	0.0005	0.001	No
0.0005	0	0.001	No
0.001	0	0.001	Yes
0.001	0	0.0015	No
0.005 (desired geometry)	0	0.001	Yes

Trials with three separate blocks in blockMeshDict, were also performed, all giving an error in splitMeshRegions, regardless of where the divide radii were placed with respect to the wire and nozzle regions.

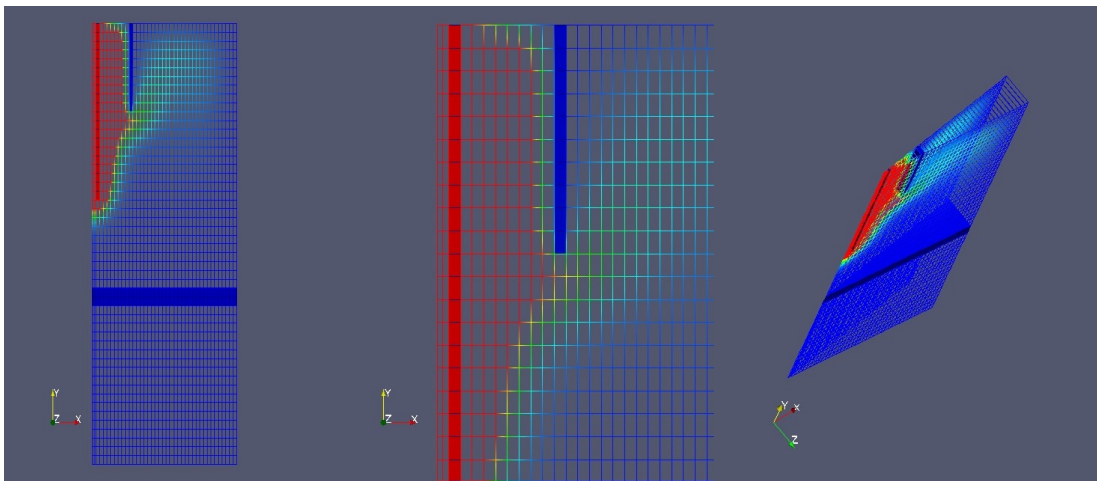


Figure 4: Successful test of wedge patch (first row of the table). Simulation with hot "wire" with center hole, wedge boundary conditions, thermal and fluid flow simulation only. Temperature contours after some simulation time. Flow only through convection.

Although the cause of this problem is not precisely identified, it seems to relate to using several wedge patches in the radial direction, in one or more of the utilities `setSet`, `setsToZones` or `splitMeshRegions`. The problem can be bypassed nicely by instead using the cyclic patch as boundary condition in the circumferential direction.

3.4. Adding fields to solver

The new field electric potential (V) as well as the electrical conductivity (σ) needs to be added to both solid and fluid regions, and the velocity field U has to be included also in solid regions. The two files `createSolidFields.H` and `setRegionSolidFields.H` for the solid case and `createFluidFields.H` and `setRegionFluidFields.H` for the fluid case must thus be changed.

Additions to `createSolidFields.H`:

```

// Initialise solid field pointer lists
...
// Added pointers for JouleMultiRegionFoam
PtrList<volVectorField> USolid(solidRegions.size());
PtrList<volScalarField> VSolid(solidRegions.size());
PtrList<volScalarField> sigmaSolid(solidRegions.size());

// Populate solid field pointer lists
...
// Added populations for JouleMultiRegionFoam
Info<< "    Adding to USolid\n" << endl;
USolid.set
(
i,
new volVectorField
(
IOobject
(
"U",
runTime.timeName(),
solidRegions[i],
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
solidRegions[i]
)
);

Info<< "    Adding to VSolid\n" << endl;
VSolid.set
(
i,
new volScalarField
(
IOobject
(
"Vel",
runTime.timeName(),
solidRegions[i],
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
solidRegions[i]
)
);

Info<< "    Adding to sigmaSolid\n" << endl;
sigmaSolid.set
(
i,
new volScalarField
(
IOobject
(
"sigma",
runTime.timeName(),
solidRegions[i],
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
solidRegions[i]
)
);

```

Additions to setRegionSolidFields.H:

```
...
// Added for JouleMultiRegionFoam
volVectorField& U = USolid[i];
volScalarField& Vel = VSolid[i];
volScalarField& sigma = sigmaSolid[i];
```

Additions to createFluidFields.H:

```
// Initialise solid field pointer lists
...
// Added pointer for JouleMultiRegionFoam
PtrList<volScalarField> VFluid(solidRegions.size());
PtrList<volScalarField> sigmaFluid(fluidRegions.size());

...
// Added populations for JouleMultiRegionFoam
Info<< "    Adding to VFluid\n" << endl;
VFluid.set
(
i,
new volScalarField
(
IOobject
(
"Vel",
runTime.timeName(),
fluidRegions[i],
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
fluidRegions[i]
);
Info<< "    Adding to sigmaFluid\n" << endl;
sigmaFluid.set
(
i,
new volScalarField
(
IOobject
(
"sigma",
runTime.timeName(),
fluidRegions[i],
IOobject::MUST_READ,
IOobject::AUTO_WRITE
),
fluidRegions[i]
);
```

Additions to setRegionFluidFields.H:

```
...
// Added for JouleMultiRegionFoam
volScalarField& Vel = VFluid[i];
volScalarField& sigma = sigmaFluid[i];
```

Note: It doesn't seem like the current field will need to be defined explicitly, as it can simply be calculated as the gradient of potential field divided by the resistivity.

4. Wirefeed

4.1. Changes in the solver

The passive transport term as defined in equation [3] is added to the solid heat equation in solveSolid.H to account for the thermal energy transported by the moving wire.:

Note: this has not fully tested yet, although it seems to work.

```
(
fvm::ddt(rho*cp, T)
- fvm::laplacian(K, T)
// Added for JouleMultiRegionFoam
==rho*cp* (U & fvc::grad(T))
);
```

4.2. Changes in input

The velocity field U must be defined in solid regions too. Here is an example from system/wire/cghangeDictionaryDict, applying a material feed rate of 1 m/s downwards in the wire.

```

U
{
    internalField    uniform ( 0 -1 0 );
    boundaryField
    {
        maxYInner
        {
            type      fixedValue;
            value      uniform ( 0 -1 0 );
        }
        axis
        {
            type      zeroGradient;
        }

        wire_to_topAir
        {
            type      zeroGradient;
        }
    }
}

```

Note: Since it is not actually solved for, the velocity field is not saved in time-history folders. It would have been good to have for postprocessing, but I haven't found how that could be done.

5. Electric conduction

5.1. Changes in the solver

Solving for electric conduction was implemented by adding a separate equation in both solid(solveSolid.H) and fluid(solveFluid.H) regions, see fig. 5 for an example. Equation [4] is contained in a separate file VEqn.H, called by both region solvers:

```

{
for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
{
solve
(
fvm::laplacian(sigma, Vel)
);
}
Info<< "Min/max V:" << min(Vel) << ' '
<< max(Vel) << endl;
}

```

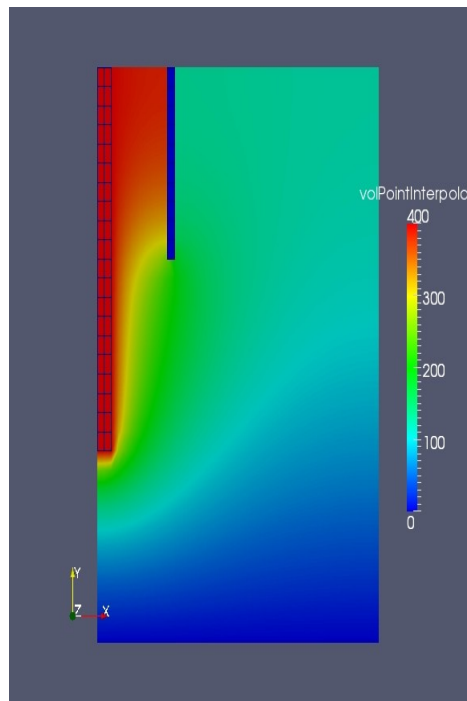


Figure 5: Electric field calculated in the topAir region, with the wire potential set uniformly at 400 Volts, and the plate potential (bottom of picture) set to 0 Volts, all other boundaries are set to zeroGradient.

5.2. Changes in the interface boundary patch

The interface boundary patch `solidWallMixedTemperatureCoupledFvPatchScalarField` works by assigning the temperature on one side of the patch as boundary condition for the other side, and the heat flux from the other side as boundary condition for the first side. At a first glance, the interface boundary patch `solidWallMixedTemperatureCoupledFvPatchScalarField` should be generic enough to use for other variables. First tests with the new equation gave unreasonable and unstable results (see fig. 6), and continuity in the potential was not observed across the interface patches. It seems that this is caused by the temperature being explicitly transferred to the patch, while the heat flux is collected implicitly from the set of thermal variables. Thus, it is probably necessary to either generalize the patch so that the same patch can be used for both fields, or to write a separate patch for the electric field, using the thermal patch as a template. Perhaps the problem is that it could actually be used for any one of the fields, but not for both?

A typical entry in `system/topAir/changeDictionaryDict` used was

```

topAir_to_wire
{
    type            solidWallMixedTemperatureCoupled;
    neighbourFieldName T;
    K               sigma;
    value          uniform 300;
}

```

Note: The solution diverged when using these settings, probably due to an error in application of the interface boundary patch to the electric potential.

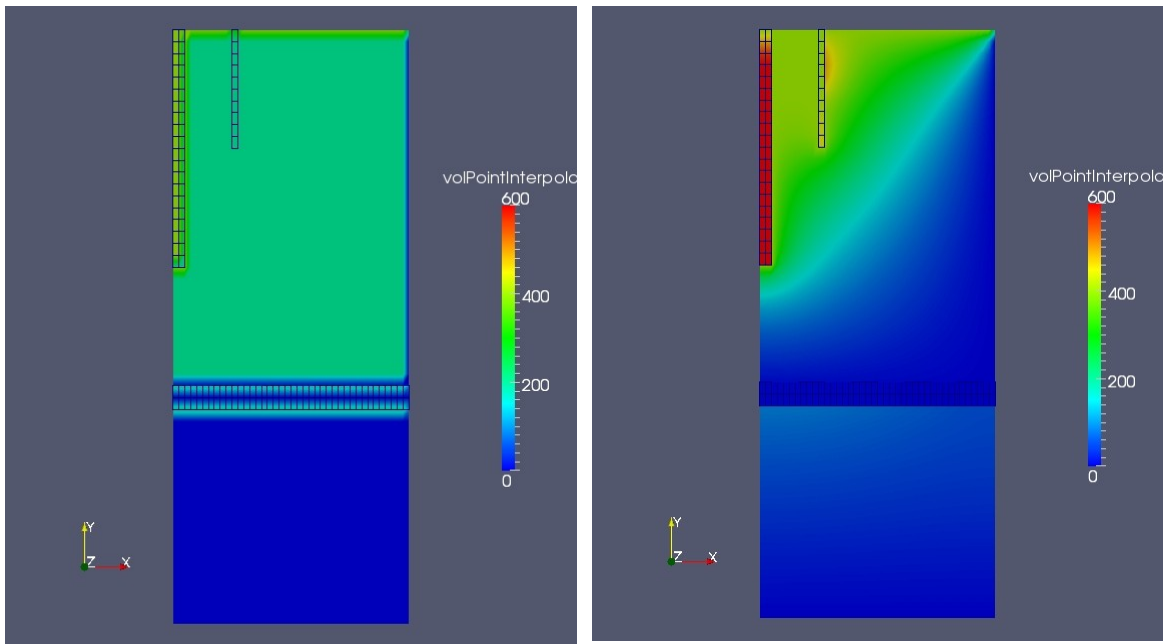


Figure 6: Non-continuous solution after applying the `solidWallHeatFluxElectricFvPatchScalarField` patch for the electric field `Vel`. The left frame shows initial and boundary conditions, while the second frame shows calculated results after one short time-step. In this case, the field of the `topAir` fluid is reasonably correct, while the solution in the wire is quite unstable, exceeding expected results by more than an order of magnitude.

5.3. Changes in input

For the electric field `Vel`, initial and boundary conditions must be defined, and the electrical conductivity `sigma` must also be defined as a field by initial and boundary conditions. See files `0/Vel` and `0/sigma` for initial conditions and `system/*/changeDictionaryDict` for boundary conditions.

6. Joule heating

Joule heating (also known as resistive heating) has been added to the thermal equation [4], by adding a term in `solveSolids.H`:

```
(
  fvm::ddt(rho*cp, T)
  - fvm::laplacian(K, T)
  == (U & fvc::grad(T))
  // Added for JouleMultiRegionFoam
  + sigma*(fvc::grad(Vel) ) & fvc::grad(Vel)
);
```

and for the fluid regions, in `hEqn.H`:


```

fvScalarMatrix hEqn
(
    fvm::ddt(rho, h)
  + fvm::div(phi, h)
  - fvm::laplacian(turb.alphaEff(), h)
  ==
    DpDt
  // Added Joule heating for JouleMultiRegionFoam
  - (fvc::grad(Vel) & fvc::grad(Vel))*sigma
);

```

7. “Lazydog” for using the new solver

A “lazydog” (from the swedish ”lathund”) is intended as a very short version of a manual, containing all that is needed, but in condensed form.

7.1. Mesh generation

Mesh generation is done as usual, except that cellSets must be defined. It would probably be appropriate to discuss here the difference between Sets, Zones and Regions, but we will only mention that there are utilities to create Zones and then Regions from a given set of Sets.

In the chtMultiRegionFoam tutorial, the overall calculation region (solid regions + fluid regions), is defined in the file constant/polyMesh/blockMeshDict. The sets are then defined using the utility setSet, as defined in the file makeCellSets.setSet.

Note: A drawback of this solution is that it is necessary to adjust element sizes so that boundaries between regions fall exactly on cell boundaries as otherwise the regions will not have the desired size. This is because makeCellSets selects cells with centres within the specified box and assigns them to the Set. Nevertheless, it is a simple and rather straightforward way of defining a not too complicated block mesh consisting of several regions.

7.2. Assign region properties and solution

Although assignment of Regions is done automatically from existing Sets using the utilities setsToZones and splitMeshRegions, it is necessary to make sure that calculations are performed on each region and to define boundary conditions and interface conditions between regions. Regions are specified in Allrun because simulations are run in one region at the time, but also in the file constant/regionProperties, where the sets “fluidRegionNames” and “solidRegionNames” are defined.

a) Allrun

In the file Allrun, the utilities and solvers called are defined, and for chtMultiRegionFoam, it is necessary to also define here which regions are to be solved as fluids and which shall be solved as solids. Replace “leftSolid Rightsolid” in the chtMultiRegionFoam tutorial with the names of your solid regions and “bottomAir topAir” with the names of your fluid regions. In the cleanup for postprocessing, the field “U” shall not be removed from solid regions when running JouleMultiRegionFoam. Also, in the tutorial, the variables mut, alphas and an extra p are removed for solids. Presumably, these variables are used in modeling turbulence, but they can be removed when running a simulation without turbulence.

ChtMultiRegionFoam (original file):

```
# remove fluid fields from solid regions (important for post-processing)
for i in heater leftSolid rightSolid
do
    rm -f 0*/$i/{epsilon,k,p,U}
done

# remove solid fields from fluid regions (important for post-processing)
for i in bottomAir topAir
do
    rm -f 0*/$i/{cp,K}
done
```

JouleMultiRegionFoam (modified for wireStickout example):

```
# remove fluid fields from solid regions (important for post-processing)
for i in wire nozzle plate
do
    rm -f 0*/$i/{epsilon,k,p}
done

# remove solid fields from fluid regions (important for post-processing)
for i in bottomAir topAir
do
    rm -f 0*/$i/{cp,K}
done
```

b) Fluid property folders

For each fluid region, a folder bearing the name of that region must be placed in the constant folder. Each of these folders must contain the files “g”, “RASProperties”, “thermophysicalproperties” and “turbulenceProperties”, including material properties for that fluid region. Such a folder is not required for solid regions as all material properties for solid regions are defined as fields.

7.3. Fields and boundary conditions

In JouleMultiRegionFoam, as well as in chtMultiRegionFoam, boundary conditions must be set (in the files system/*/changeDictionaryDict and 0/*) not only for external boundaries, but also for the interface boundaries between the regions. External boundaries are treated as usual, and the internal boundaries that have automatically been defined by the utility splitMeshRegions may also be treated as external boundaries. However, the idea of MixedBoundary is to use internal boundaries as interfaces between regions, preserving the continuity of the solution. For internal interface regions, the patch solidWallMixedTemperatureCoupled should be used. Note that "reverse" boundary conditions and sample region = the bounding region need to be specified too. For example, if the boundary from the region wire to topAir has been defined in system/wire/changeDictionaryDict as:

```

boundary
{
  wire_to_topAir
  {
    offset          ( 0 0 0 );
    sampleMode      nearestPatchFace;
    sampleRegion    topAir;
    samplePatch     topAir_to_wire;
  }
}

T
{
  boundaryField
  {
    wire_to_topAir
    {
      type          solidWallMixedTemperatureCoupled;
      neighbourFieldName T;
      K             K;
      value         uniform 300;
    }
  }
}

```

then it is necessary to define the same boundary in `system/wire/changeDictionaryDict` as:

```

boundary
{
  topAir_to_wire
  {
    offset          ( 0 0 0 );
    sampleMode      nearestPatchFace;
    sampleRegion    wire;
    samplePatch     wire_to_topAir;
  }
}

T
{
  boundaryField
  {
    topAir_to_wire
    {
      type          solidWallMixedTemperatureCoupled;
      neighbourFieldName T;
      K             K;
      value         uniform 300;
    }
  }
}

```

The following field variables are used and must be defined in the interior and on external and internal boundaries:

U, p:

The velocity and the pressure are considered two independent variables for the mechanical equations in the fluid regions, and can be given all the usual boundary conditions, zeroGradient, fixedValue, InletOutlet. In JouleMultiRegionFoam, p is not used in the solid regions, but U must be defined as a field also in the solid regions, to allow wirefeed to be accounted for.

T:

The temperature is the main independent variable for heat conduction in both solid and fluid regions, and can be given all the usual external boundary conditions: zeroGradient, fixedValue, InletOutlet. Between regions, solidWallMixedTemperatureCoupled or fixedValue can be used.

Vel:

The electric potential is the main independent variable for the potential equation, and thus required in both solid and fluid Regions for JouleMultiRegionFoam, but not for chtMultiRegionFoam.

epsilon, k:

These are not calculated, just material constants used in fluid regions. Boundary conditions are still required by OpenFOAM though, so set zeroGradient for simplicity.

cp, rho, K:

These are not calculated, just material constants for solid regions. Boundary conditions are still required by OpenFOAM though, so set zeroGradient for simplicity.

sigma:

The electric conductivity is the only additional material property that is needed to solve the electric potential equation. In JouleMultiRegionFoam, it must be input as a field in both solid and fluid regions. It is not used in chtMultiRegionFoam.

7.4. Solution control

startTime should be set to the same value as deltaT in system/controlDict, because the first timestep folder is used to setup boundary and initial conditions for the different regions. For both solid and fluid regions, the solution scheme for the electric field must be defined, as follows.

In the files system/regionName/fvSchemes, the basic solution scheme for the Laplacian is specified:

```
// added electrical continuity eqn for JouleMultiRegionFoam
laplacian(sigma, Vel) Gauss linear corrected;
```

And, in the files system/regionName/fvSolution, the solution scheme is further specified:

```
Vel
{
    solver          PCG;
    preconditioner  DIC;
    tolerance       1e-14;
    relTol          0;
}
```

8. Summary and future work

As described above, a lot remains to be done to make this implementation work, notably, reusing the numbering of the necessary steps identified in section 2.1 above:

General and mechanical:

1. A simple test case has been created and used in development. This model should be improved to establish a set of test cases with appropriate boundary conditions for testing the performance of the model. Later, a 3D model of a moving torch should be modeled, perhaps with different joint geometries.
2. The velocity field has been introduced for solid regions. Unfortunately, the velocity vector does not appear in the output files for solid regions, although it does influence calculation of the transport term in the heat equation.

Thermal conduction:

3. The Joule heating term, although implemented in both solid and fluid heat conduction solvers does not seem to yield correct results. Improving the test case will help in debugging this.
4. The passive transport term in the solid heat conduction solver seems to work correctly, but needs verification against exact values for a simple test case.

Electrical conduction:

5. The electric potential has been added as a scalar field variable.
6. Electrical conductivity has been added as a field variable.
7. Solving the electric potential has been introduced in both solid and fluid regions, though the performance has not been tested and integration and convergence criteria should be properly defined..
3. The internal boundary patch does not seem to work properly for electrical conduction. Further investigation is needed to determine the cause of this malfunction. Possible explanations include: incorrect usage; the patch not being general enough to be used like this; or that it is not possible to use the patch for two different field variables at the same time.

Further, most material properties are temperature dependent and this is considered to have a large impact on the physical behaviour on the system. Thus, adding temperature dependence to the material properties will also be needed for an accurate simulation.

9. References

Moradnia, Pirooz, 2008: "[A description of how to do Conjugate Heat Transfer in OpenFOAM](http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/chtFoam.pdf)", http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/chtFoam.pdf