Adding electric conduction and Joule heating to chtMultiRegionFoam

Niklas Järvstråt

Table of contents

1 Physical background and equations solved

1.1 Welding

The application we want to simulate is electric arc welding, 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.

1.1.1 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. What complicate matters is that the heat is generated by the electric field, 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 in the plasma.

1.2 Physical equations

1.2.1 Flow and momentum

For the fluid flow, the simple momentum equation used in chtMultiRegionFoam will be retained unchanged as more detailed plasma models will be added later. For the solid, the velocity field will be given prescribed constant values in each region. I.e. the wire-feed speed in the wire and zero in the nozzle and plate.

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

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

$$\rho \cdot C_p \cdot \frac{dT}{dt} = \nabla(\mathrm{K}\nabla\mathrm{T})$$

is solved. For our application, we need to add the Joule heating term:

$$\rho \cdot C_p \cdot \frac{dT}{dt} = \nabla(\mathrm{K}\nabla\mathrm{T}) + \sigma|\nabla\mathrm{V}|^2$$

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:

$$\rho \cdot C_p \cdot \frac{dT}{dt} = \nabla(\mathrm{K}\nabla\mathrm{T}) + \sigma|\nabla\mathrm{V}|^2 - \rho \cdot C_p \cdot U \cdot \nabla T$$

1.2.3 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

$$-\nabla(\sigma\,\nabla V) = 0\,,$$

where σ is the conductivity, V is the electric potential field and $\sigma\,\nabla V$ is the electric current density vector.

Note that this equation only is 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

2.1 Overview and aim

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

1. Create a simple test model with appropriate boundary conditions for testing the changes.
2. Add field variables, electric potential scalar as a variable and include the velocity field also in the solid regions, although the velocity will not be solved for there. Add the material property conductivity as a field variable.
3. Add solving for electric potential in both solid and fluid solvers.
4. Add Joule heating term to both solid and fluid heat conduction solvers.
5. Add passive transport term to solid heat conduction solver.
6. Adapt internal boundary patches to also cover the electric field.

2.2 Solid part solver

The solid part solver in chtMultiRegionFoam is simply the heat conduction equation (this is the entire unmodified solveSolids.H)

```
{
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 fluid solver, although one of the simpler, boyancy 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 independently or recursively. These equations are here left unchanged, however, as a research project on plasma simulation at HV will provide an advanced solver for the fluid, to be integrated later.

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 rather than the temperature, as conventions differ between solid mechanics and fluid mechanics.

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)
);
```

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 Change geometry

3.1.1 blockMeshDict

This blockMeshDict file defines an entire block, bounding the computational domain, both solid and fluid regions. Because of the forced flow of protection gas inside the nozzle, the boundary had to be divided in two atches in the redial direction, one inside the nozzle and one outside the nozzle.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  1.6                                   |
|   \\  /    A nd           | Web:      http://www.OpenFOAM.org               |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
```

```
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
(     0 -.020       0  )
(     0  .030       0  )
(  .020 -.020    .002  )
(  .020  .030    .002  )
(  .020 -.020   -.002  )
(  .020  .030   -.002  )
(  .005 -.020    .0005 )
(  .005  .030    .0005 )
(  .005 -.020   -.0005 )
(  .005  .030   -.0005 )
);

blocks
(
hex (6 7 9 8 0 1 1 0) (50 1 10) simpleGrading (1 1 1)
hex (2 3 5 4 6 7 9 8) (50 1 30) simpleGrading (1 1 1)
);

edges
(
arc 4 2 (.020099751  -.020  0)
arc 5 3 (.020099751  .030  0)
arc 8 6 (.005024938  -.020  0)
arc 9 7 (.005024938  .030  0));

patches
(
patch maxX
(
(3 2 4 5)
)
patch minYInner
(
(0 0 8 6)
)
patch minYOuter
(
(6 8 4 2)
)
patch maxYInner
(
(1 1 7 9)
)
patch maxYOuter
(
(7 3 5 9)
)
cyclic frontAndBack
(
(0 6 7 1)
(6 2 3 7)
(0 1 9 8)
(8 9 5 4)
)
symmetryPlane axis
(
(0 1 1 0)
)
);

mergePatchPairs
(
);

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

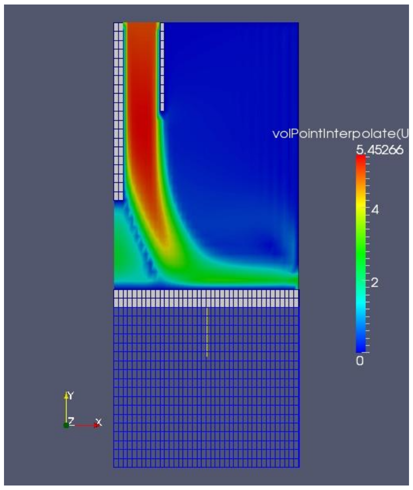

Figure 1: Mesh with velocity field using prescribed flow at the nozzle inlet, outlet boundary condition at the oputer radius of the model and zero velocity at all other boundaries.

3.1.2 makeCellSets

A problem was encountered when attempting to make an axisymmetric model by using wedge boundary conditions, as the utility splitMeshRegions crashed mysteriously on encountering the fourth region. It was particularly puzzling, as both solid and fluid regions had been processed without problems. The problem occurred for the “wire" region, top left corner of the figure. Thus, a series of trials with different settings for the inner block and for the wire dimension was performed, giving the following results:

blockMesh divide radius	makeCellSets inner wire radius	makeCellSets outer wire radius	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 gaving an error in splitMeshRegions, regardless of where the divide radii were placed with respect to the wire and nozzle regions.

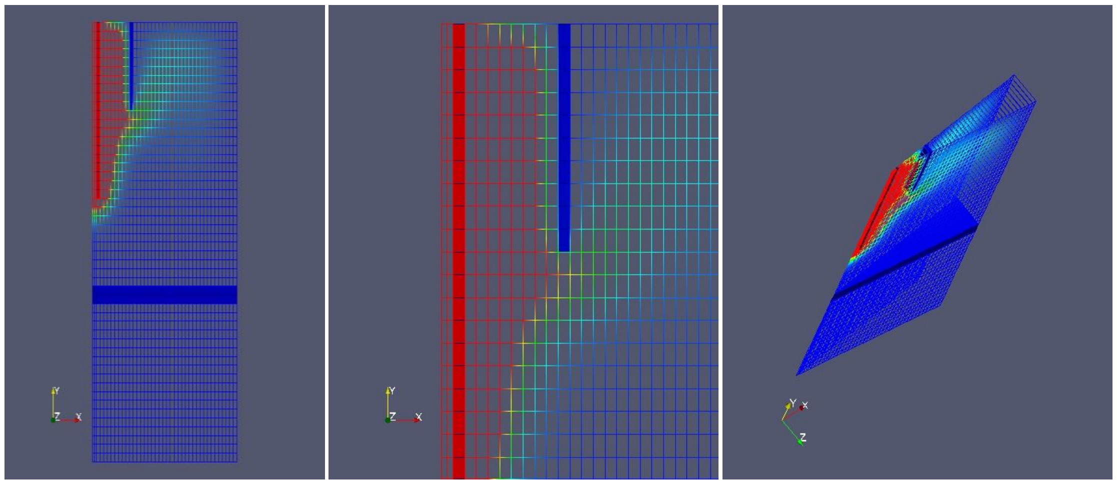

Figure 2: Successful test of wedge patch (first row of the table). Simulation with hot "wire" with center hole, wedge boundary conditions, thermal and fluid flow simulation only. Temperature contours after some simulation time. Flow only through convection.

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.

3.2 *Adding fields to solver*

The new field electric potential (V) as well as the electrical conductivity (sigma) 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 that 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*

Add passive transport term to solid heat equation in solveSolid.H (Not fully tested yet, but 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 must be defined in solid regions too. Note that since it is not actually solved for, it is not saved in time-history folders. Would be good to have it, but I haven't found how that could be done.

5 Electric conduction

5.1 *Changes in the solver*

Add solving electric conduction as an equation in both solid(solveSolid.H) and fluid(solveFluid.H) regions. Implemented as a separate file VEqn.H,

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

}
```

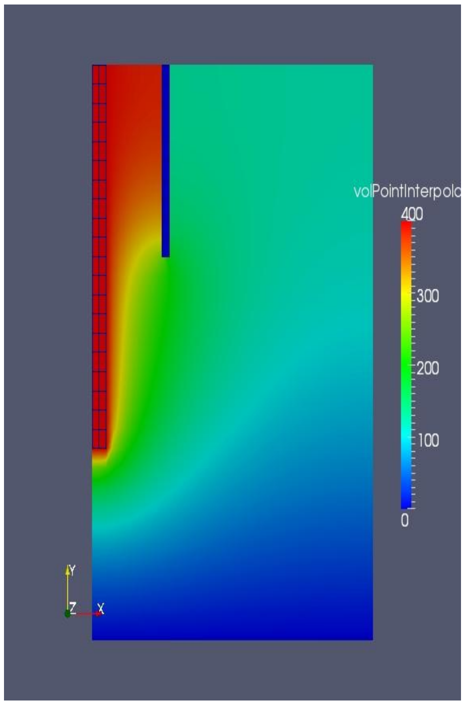

Figure 3: Electric field, wire uniformly at 400 Volts, plate (bottom of picture) at 0 volts, all other boundaries are set to zeroGradient.

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, however, gave unreasonable and unstable results, 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 that solution diverged when using this settings.

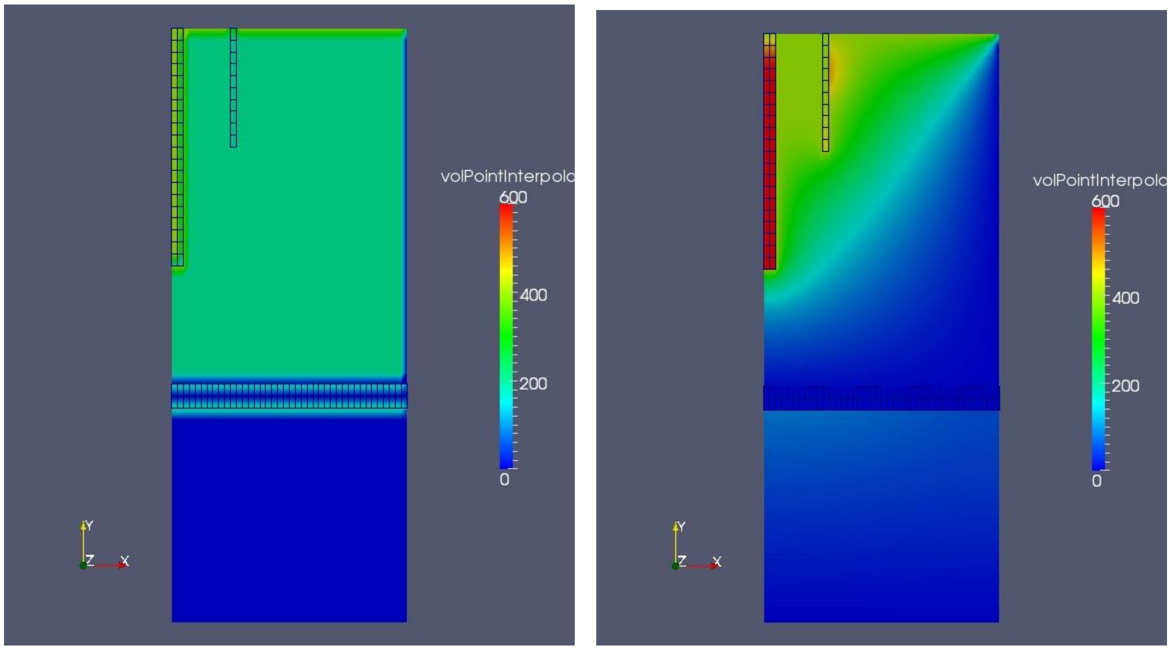

Figure 4: Non-continuous solution after applying the solidWallHeatFluxElectricFvPatchScalarField patch for the electrical field Vel. The left frame shows initial and boundary conditions, while the second frame shows calculated results after one short time-step. In this case, the field of the topAir fluid 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

The electric field initial and boundary conditions must be defined, and the electrical conductivity must also be defined as a field by initial and boundary conditions.

6 Joule heating

Add Joule heating to thermal equations. (add a term in solveSolids.H and hEqn.H)

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

7 How-to “lazydog” for using the new solver

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 makeCellSets, as defined in the file makeCellSets.setSet. 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, once you have created Sets and use the appropriate utilities as in Allrun, it is necessary to make sure that calculations are performed on each region and to define bondary conditions and interface conditions between regions. Regions are specified in Allrun because simulations are run in one region at the time, but also in constant/regionProperties, where the sets “fluidRegionNames” and “solidRegionNames” are defined.

7.2.1 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,alphat and an extra p is removed for solids. Presumably, these variables are used for turbulence, but they can be removed when running a simulation without turbulence.

chtMultiRegionFoam:

```
# 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:

```
# 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
```

7.2.2 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.

T:
The temperature is one of the main independent variables 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.

U:
The velocity is one of the main independent variables in the fluid regions, and gan be given all the usual boundary conditions, zeroGradient, fixedValue, InletOutlet. In JouleMultiRegionFoam, U must be defined also in solid regions, to allow wirefeed to be accounted for.

p:
The pressure is one of the main independent variables in the fluid regions, and gan be given all the usual boundary conditions, zeroGradient, fixedValue, InletOutlet

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.

Vel:
Electric potential field, solved for in the electric potential equation, and thus required in both solid and fluid Regions for JouleMultiRegionFoam, but not for chtMultiRegionFoam.

sigma:
Electric conductivity field, needed to solve the electric potential equation, and thus required in both solid and fluid Regions for JouleMultiRegionFoam, but not for 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 Next steps

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

1. Improve the simple test model and establish a set of test cases with appropriate boundary conditions for testing the performance of the model.
2. The velocity vector does not appear in output files for solid regions, although it does influence calculation of the transport term in the heat equation.
3. The electric potential is solved in both solid and fluid regions, though the performance has not been tested and integration and convergence criteria should be properly defined.
4. 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, ad.
5. 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.
6. The internal boundary patch does not seem to work properly. Investigate whether this is due to incorrect usage, that it is not general enough to be used or that it is not possible to use it for two different fields at the same time.