



**ISP RAS**

# Implementation of the solver for coupled heat transfer in gas and solid

Instructors:

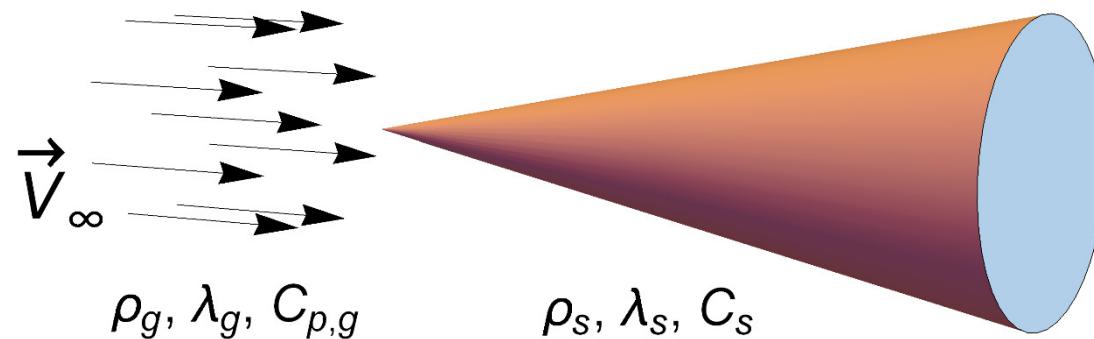
- 1) I. Marchevsky, Ph.D., Assoc.prof.
- 2) M. Kravoshin, Ph.D., Senior Researcher

# Contents

- Problem description
- Choice of the numerical method
- Implementation
- Test problems

# Problem description

- Mach number –  $0 < \text{Ma} < 5$
- EoS – perfect gas
- Viscous flow of Newtonian media
- Transient heat transfer in solid (Fourier law)



# Governing equations

## ■ Perfect gas

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\vec{U} \rho) = 0, \quad \frac{\partial(\rho \vec{U})}{\partial t} + \nabla \cdot (\vec{U} \rho \vec{U}) = \nabla \cdot \hat{\Pi}, \quad \frac{\partial(\rho e)}{\partial t} + \nabla \cdot (\vec{U} \rho e) = -\nabla \cdot \vec{q} + \nabla \cdot (\hat{\Pi} \cdot \vec{U})$$

$$\hat{\Pi} = \mu (\nabla \vec{U} + \nabla \vec{U}^T) - \frac{2}{3} \hat{I} \mu \nabla \cdot \vec{U} - \hat{I} p, \quad \vec{q} = -\lambda \nabla T, \quad p = \rho \frac{R}{M} T.$$

## ■ Solid

$$\rho C \frac{\partial T}{\partial t} - \nabla \cdot (\lambda \nabla T) = 0.$$

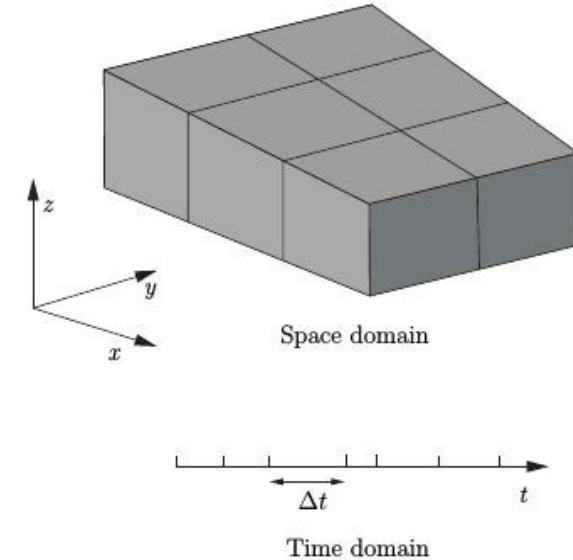
## ■ Ideal heat contact

$$-(\lambda \nabla T)_{solid} \cdot \vec{n} = -(\lambda \nabla T)_{fluid} \cdot \vec{n},$$

$$T_{solid} = T_{fluid}.$$

# Numerical model

- Space & time discretization
  - Finite Volume Method – space and time divided on non-intersecting cells (intervals)



- Numerical schemes
  - Mean theorem + Gauss theorem

$$\int_V \nabla \cdot \phi = \oint_{\partial V} \phi \cdot d\vec{S}$$

- Time integration algorithm
  - Operator-splitting methods like PISO or PIMPLE

$$p = p^* + p'$$

$$\vec{U} = \vec{U}^* + \vec{U}'$$

$$\vec{U}' \sim -\nabla p'$$

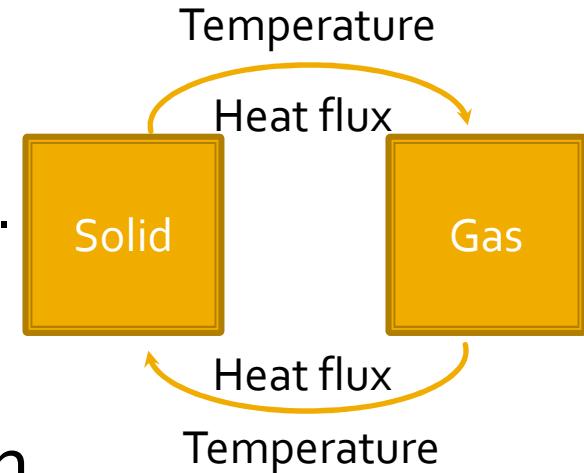
# Models coupling

- Single matrix
  - Single equation of energy for both gas & solid.
  - Stable and robust, but hard to implement and use.
- Multiple domains, iterative coupling
  - Easy to work with complex geometries, performance degradation<sup>\*).</sup>
  - Needs data interpolation on the interface.

<sup>\*</sup>) Not very important when using operator splitting methods like PISO

# Implementation in OpenFOAM

- Current implementation in OpenFOAM –  
**chtMultiRegionFoam**:
  - Iterative solution of energy balance between several solid and fluid regions.
  - Uses outer **PIMPLE** loop and special B.C. for coupling between domains.
- List of domains managed through the **regionProperties** dictionary.
- Connection between regions established via boundary conditions for temperature.
- Main limitation – subsonic speeds of fluid ( $\text{Ma} < 1$ ).



# Designing new solver

- **PISO/SIMPLE/PIMPLE** are known to be oscillatory at high speeds
  - we need different method for flows with  $Ma > 1$ .
- **rhoCentralFoam** was designed for flows with  $Ma > 1$ .
  - it is explicit , not efficient when  $Ma < 1$ .
- Solution – hybrid approach
  - Combine pimpleCentralFoam and chtMultiRegionFoam
- For the most general implementation, see  
<https://github.com/TonkomoLLC/hybridCentralSolvers>

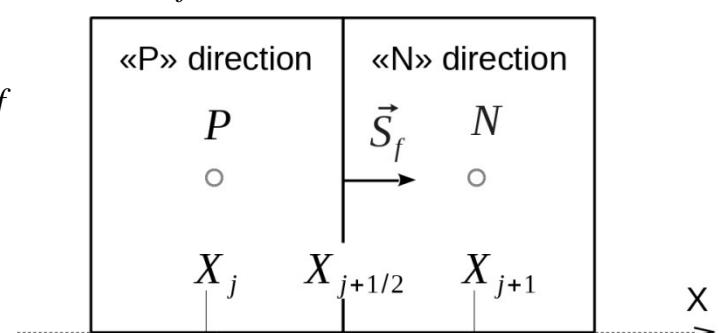
# pimpleCentralFoam

- Implicit KT/KNP fluxes for advection
- PIMPLE algorithm for  $p - \vec{U} - T$  coupling

$$\frac{\partial \rho \beta}{\partial t} + \nabla \cdot (\vec{U} \rho \beta) = \nabla \cdot D_\beta \nabla \beta$$
$$\frac{\rho^n \beta^n - \rho^o \beta^o}{\Delta t} + \underbrace{\frac{1}{V} \sum_f (\varphi_f^P + (1 - \kappa_f) \varphi_f^N) \beta_f^P + \frac{1}{V} \sum_f \kappa_f \varphi_f^N \beta_f^N}_{= \frac{1}{V} \sum_f D_{\beta,f} \frac{\partial \beta}{\partial \vec{n}_f} | \vec{S}_f} =$$

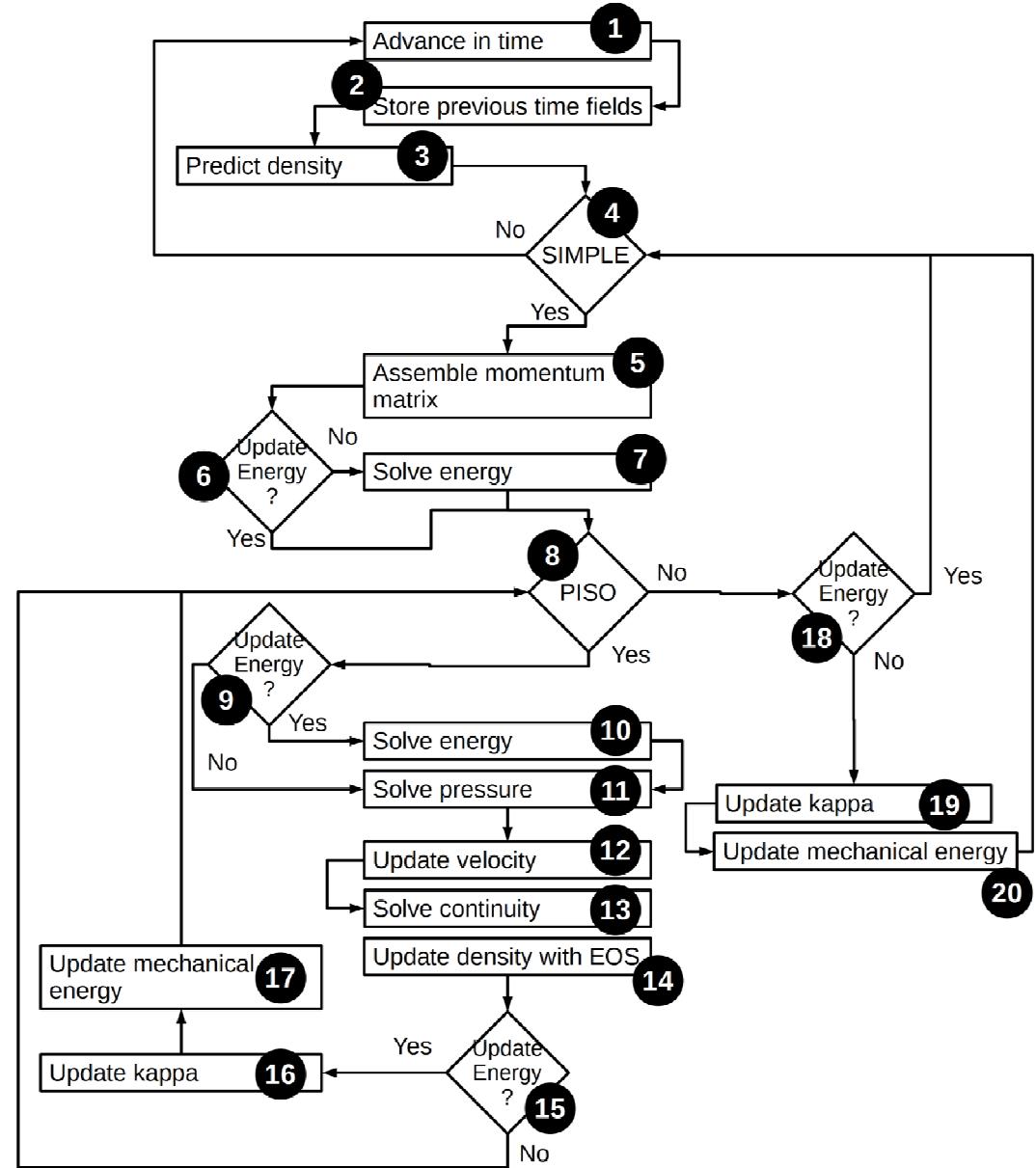
incompressible  $\kappa_f = 0$

compressible  $\kappa_f = 1$



# Hybrid PIMPLE/KT algorithm

- Increase current time – **1**
- Solve explicit equation for density – **3**
- Start SIMPLE loop – **4**
- Assemble momentum matrix – **5**
- If only PISO iterations, solve for energy – **7**
- Start PISO loop – **8**
- If more than 1 SIMPLE iterations, solve for energy – **9**
- Solve continuity equation formulated for pressure, update velocity and density with EOS – **10-14**
- Update blending field “kappa” – **16**
- Update mechanical energy transport – **17**



# pimpleCentralFoam source code

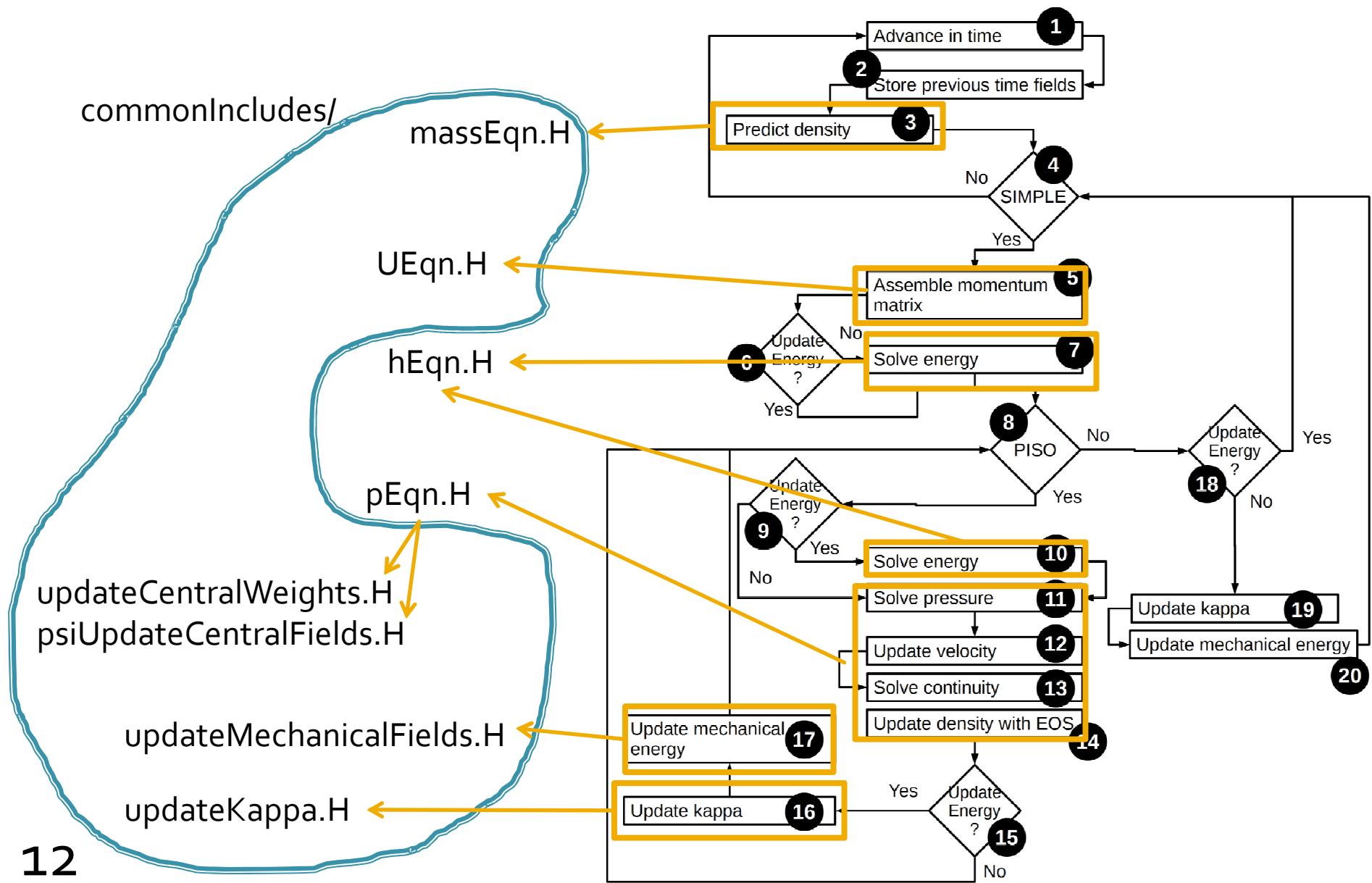
Library  
**libpisoCentral.so**  
with common  
operations used  
**pimpleCentralFoam**

Source code for  
**pimpleCentralFoam**  
solver

hybridCentralSolvers/OpenFOAM-4.1/

- pisoCentral/
  - commonIncludes/
    - Eqns
    - createCommonCentralFields.H
    - updateKappa.H
  - kappaFunctions/
- pimpleCentralFoam/
  - Make/
  - createFields.H
  - hEqn.H
  - pEqn.H
  - pimpleCentralFoam.C

# Hybrid PIMPLE/KT algorithm sources

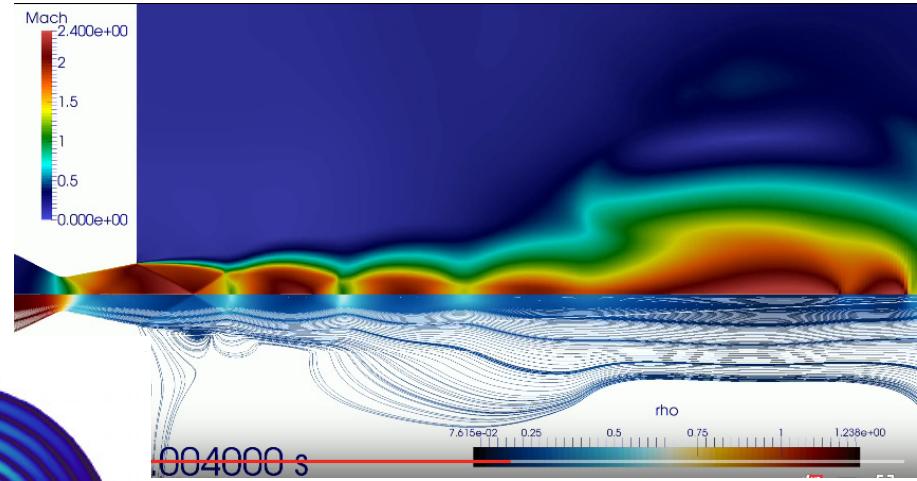


# Other hybrid PIMPLE/KT solvers

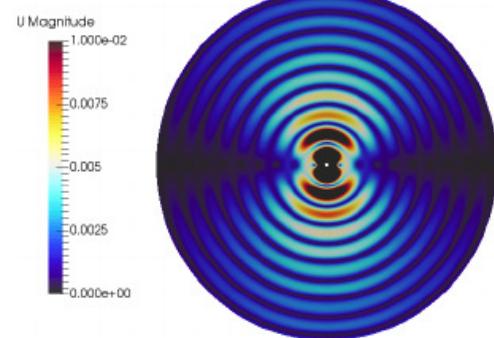
- **pimpleCentralDyMFoam** – solver with mesh motion
- **rhoPimpleCentralFoam** – solver for gases with equations of state, different to perfect gas
- **reactingPimpleCentralFoam** – solver for multicomponent mixtures flow with chemical reactions
- **twoPhaseMixingCentralFoam** – solver for two phase homogeneous liquids flows (water & air, for example)
- **chtMultiRegionCentralFoam** – this Track solver

# Examples of pimpleCentralFoam solver

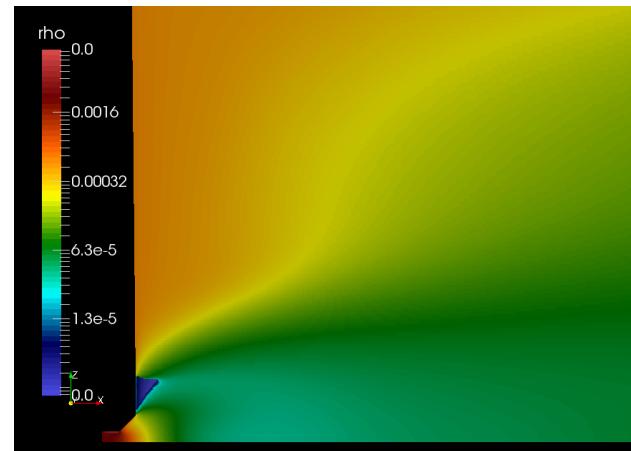
- Supersonic jet flows



- Acoustics



- Plasma flows



# Boundary conditions

- Inlet (for subsonic and supersonic flows)
- Outlet – mixed subsonic/supersonic  
(**subsonicSupersonicPressureOutlet**,  
**libcompressibleTools**, can be downloaded  
from [github.com](https://github.com))
- Walls – adiabatic, coupled to solids

# Steps to develop supersonic coupled with heat transfer solver

- Step 1. Create new solver and link with **regionProperties** to allow multiple domains in single case.
- Step 2. Move fluid domain from default location to user-specified.
- Step 3. Create source code to solve for heat transfer in several (solid) bodies.
- Step 4. Link created code with new solver. Compile the solver.

**Resulting source codes of each step stored in separate folders of training track materials in the folder src/**

<https://github.com/unicfdlab/TrainingTracks/tree/master/OpenFOAM/gasThermoCoupled-OF4.1>

# Simplifications

Single domain for fluid, multiple domains for solids



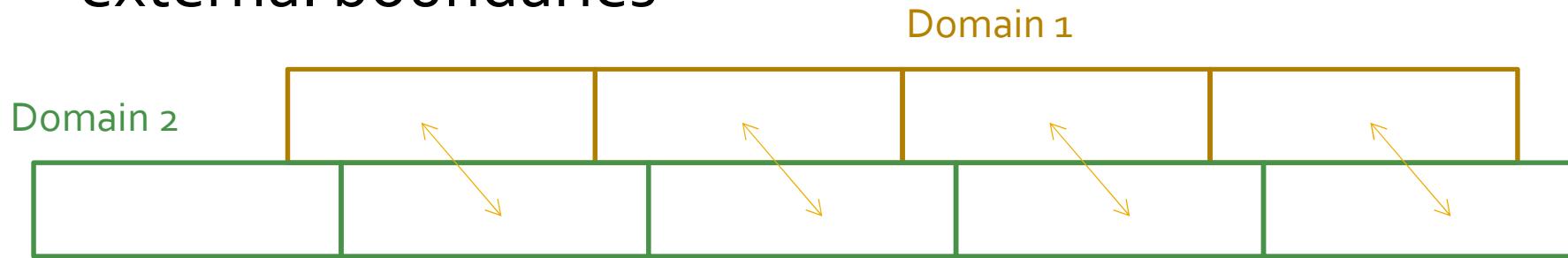
Application: flow over several bodies, or structure composed of several materials

Benefits: No need to rewrite most of the code for gas dynamics

More general case – TonkomoLLC github page  
<https://github.com/TonkomoLLC/hybridCentralSolvers>

## Simplifications - 2

- Nearest cell interpolation — for meshes with relatively uniform and equal grid distribution on external boundaries



- **turbulentTemperatureCoupledBaffleMixed** uses approximated expressions for heat fluxes to satisfy ideal heat contact condition

# Initialization steps

1. Download hybridCentralSolvers  
<https://github.com/unicfdlab/hybridCentralSolvers>
2. Build library and solvers: ./Allwmake
3. Download libcompressibleTools  
<https://github.com/unicfdlab/libcompressibleTools>
4. Build library ./makeLib.sh

# Step 1

- Make a copy of pimpleCentralFoam solver, rename it to myChtPimpleCentralFoam and update its wmake settings:

```
mv pimpleCentralFoam.C myChtPimpleCentralFoam.C
```

- Update Make/files

```
myChtPimpleCentralFoam.C
```

```
EXE = $(FOAM_USER_APPBIN)/myChtPimpleCentralFoam
```

- Update Make/options

```
hybridCentralSolvers=/path/to/library/source
```

```
EXE_INC = \
```

```
  -I$(hybridCentralSolvers)/pisoCentral/lnInclude \
```

```
...
```

```
-I$(LIB_SRC)/regionModels/regionModel/lnInclude
```

```
...
```

```
EXE_LIBS = \
```

```
...
```

```
-lregionModels
```

Include path and library of classes  
for handling multi-domain cases

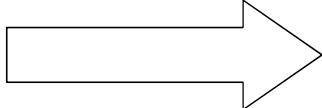
## Step 1 results

- After this step we have new solver “**myChtPimpleCentralFoam**” which can work potentially with multiple domains (meshes) simultaneously.

## Step 2

- Change space discretization storage for gas from «**defaultRegion**» to «**fluid**» region

0/  
**constant/**  
**polyMesh/**  
**system/**



0/  
**constant/**  
**fluid/**  
**polyMesh/**  
**system/**  
**fluid/**

- Goal of this step is to separate mesh storage for fluid (gas) and solids. Each mesh is stored in the unique folder on HDD (and object in memory)

**regionProperties rp(runTime);**

To manage several domains  
**regionProperties** object must be created in the program before executing all other mesh operations

# General settings

- Create file **createMeshes.H**:

```
regionProperties rp(runTime);  
#include "createFluidMeshes.H"
```

Operations to create  
mesh and fields for fluid  
are now stored in new  
files

- Update **createFields.H**:

```
#include "createFluidFields.H"
```

- Update **Make/options**:

```
EXE_INC = \
```

```
...
```

```
-I ./fluid
```

Fluid model files are stored in folder  
«fluid»  
which is known to compiler to  
lookup for includes

# Remove LTS

- Update **main** procedure in **myChtPimpleCentralFoam.C** —  
remove LTS \*)
- Start of the time step will look like:

```
....  
while (pimple.loop())  
{  
    #include "readAdditionalPimpleControl.H"  
    #include "acousticCourantNo.H"  
    #include "centralCompressibleCourantNo.H"  
    #include "readTimeControls.H"  
    #include "setDeltaT.H"
```

~~if (LTS)  
{  
...  
}~~

- Add include for **regionProperties** class:

```
#include <>regionProperties.H>
```

\*) LTS – Local Time Stepping

# Create fluid mesh & fields

- Put fluid-associated operations in separate files:

```
mkdir fluid; cd fluid/  
mv ../hEqn.H ./  
mv ../pEqn.H ./  
mv ../createFields.H createFluidFields.H  
touch createFluidMeshes.H
```

Fluid fields are now created in **createFluidFields.H**

- Create mesh for gas domain - **createFluidMeshes.H**:

```
const wordList fluidNames(rp["fluid"]);  
autoPtr<fvMesh> fluidMeshPtr;  
fluidMeshPtr.reset(  
    new fvMesh(  
        IOobject(  
            fluidNames[0], ←  
            runTime.timeName(), ←  
            runTime,  
            IOobject::MUST_READ  
        )  
    );  
)
```

Pointer to gas mesh

Specify location for gas mesh

Create reference to fluid (gas) mesh

```
fvMesh& fluidMesh =  
fluidMeshPtr();  
fvMesh& mesh = fluidMesh;
```

# Update `createFluidFields.H`

- Move initialization operations related to fluid (gas) domain from `myChtPimpleCentralFoam.C` to `createFluidFields.H`:

```
#include "readAdditionalPimpleControl.H"
#include "createCommonCentralFields.H",
autoPtr<compressible::turbulenceModel> turbulence
(compressible::turbulenceModel::New(rho,U,phi,thermo));
#include "createFvOptions.H"
#include "createMRF.H"
#include "initContinuityErrs.H"
#include "readCourantType.H"
#include "markBadQualityCells.H"
#include "psiUpdateCentralFields.H"
#include "updateKappa.H"
#include "updateCentralWeights.H"
#include "createCentralCourantNo.H"
```

## Step 2 results

- At this step gas domain relocated from standard folder to location specified by user in the **regionProperties** dictionary.
- Source code of gas dynamics equations approximation is now stored in separate folder “**fluid/**”.

# Step 3

- Create source code to solve for heat transfer equations in solids and link necessary libraries to executable.

```
mkdir solid
```

← Folder to store sources for solid body heat transfer model

- Examples of the source code can be retrieved from chtMultiRegionPimpleFoam sources.
- Add “includes” to main procedure (**myChtPimpleCentralFoam.C**):

```
#include "coordinateSystem.H"  
#include "solidThermo.H"
```

- Update wmake settings (**Make/options**):

```
-I$(LIB_SRC)/thermophysicalModels/  
    solidThermo/lnInclude \
```

- Resulting source code structure will become:

```
myChtPimpleCentralFoam/  
    fluid/    solid/    Make/
```

# General solution procedure for solids

- Create meshes for solids.
- Create fields describing heat exchange process in solids.
- Create approximations of energy balance equations in solids (matrices).
- Solve matrices (equations) for energy balances for all solids.

# Create solid domains

- Unlike gas, for solids we have several computational domains and thus we have to store several meshes, several temperature fields, several thermal conductivity coefficients and so on.
- Create [createSolidMeshes.H](#):

```
const wordList solidsNames(rp["solid"]);
PtrList<fvMesh> solidRegions(solidsNames.size());
forAll(solidsNames, i)
{
    solidRegions.set
    (
        i,
        new fvMesh
        (
            IOobject
            (
                solidsNames[i],
                runTime.timeName(),
                runTime,
                IOobject::MUST_READ
            ...
        )
    );
}
```

Loop over all solid domains

Names for solid domains

Number of solid domains

For solid No. i, create mesh

For complete example see  
[createSolidMeshes.H](#) from training materials

# Create solid fields and objects

- Place initialization operations in **createSolidFields.H** file.
- For each solid we must create:
  - Coordinate system transformation tensor (for anisotropic conductivity)

```
PtrList<coordinateSystem> coordinates(solidRegions.size());
```

- Thermodynamic library

```
PtrList<solidThermo> thermos(solidRegions.size());
```

- Volumetric heat sources (fvOptions objects)

```
PtrList<fv::options> solidHeatSources(solidRegions.size());
```

- Porosity field

```
PtrList<volScalarField> betavSolid(solidRegions.size());
```

- Field of anisotropic thermal diffusivity coefficient

```
PtrList<volSymmTensorField> aniAlphas(solidRegions.size());
```

# Initialization of solid fields and objects

- After declaration, objects are initialized (see **createSolidFields.H** for example) in cycle:

```
forAll(solidRegions, i)
{
    ...
    thermos.set(i, solidThermo::New(solidRegions[i]));
    ...
    solidHeatSources.set
    (i, new fv::options(solidRegions[i]));
    ...
    coordinates.set
    (i, coordinateSystem::New(solidRegions[i], thermos[i]));
    ...
    aniAlphas.set(i, new volSymmTensorField...
    ...
    betavSolid.set(i, new volScalarField...
    ...
}
```

# Solution process for heat transfer in multiple solid bodies

- Loop over all solids (**solveSolidRegions.H**)

```
forAll (solidRegions, i)
{
```

- Create references for fields, coefficients and mesh for region No. i

```
#include "setRegionSolidFields.H"
```

- Solve for energy balance for solid No. i

```
#include "solveSolid.H"
```

# Energy equation in solid

- In case of isotropic material properties, energy balance in solid (in terms of enthalpy) reads:

$$\frac{\partial \beta_v \rho h}{\partial t} - \nabla \cdot \left( \beta_v \frac{\lambda}{C} \nabla h \right) = S$$

- Change of energy in volume

```
fvScalarMatrix hEqn
(
    fvm::ddt(solidBetav, solidRho, solidH)
    -
```

- Heat flux

```
    fvm::laplacian
    (
        solidBetav*solidThermo.alpha(),
        solidH,
        "laplacian(alpha,h)"
    )
```

- Volumetric energy sources

```
    ==
    solidFvOptions(solidRho, solidH)
);
```

## Step 3 results

- Source code for simulation of heat transfer in solids using single solver was created
- Simulation process includes:
  - Initialization of meshes and fields for each solid:  
`createSolidMeshes.H` and `createSolidFields.H`
  - Energy balance equation discretization and solution: files  
`solveSolidRegions.H`, `setRegionSolidFields.H` and  
`solveSolid.H`
- Source code of heat transfer approximation for solid bodies stored in separate folder “`solid/`”
- Though sources created, they are not linked to the `main` program

# Step 4

- Put solid equations solution block in PIMPLE algorithm for coupled simulation:

- Update **createMeshes.H**:

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

- Update **createFields.H**:

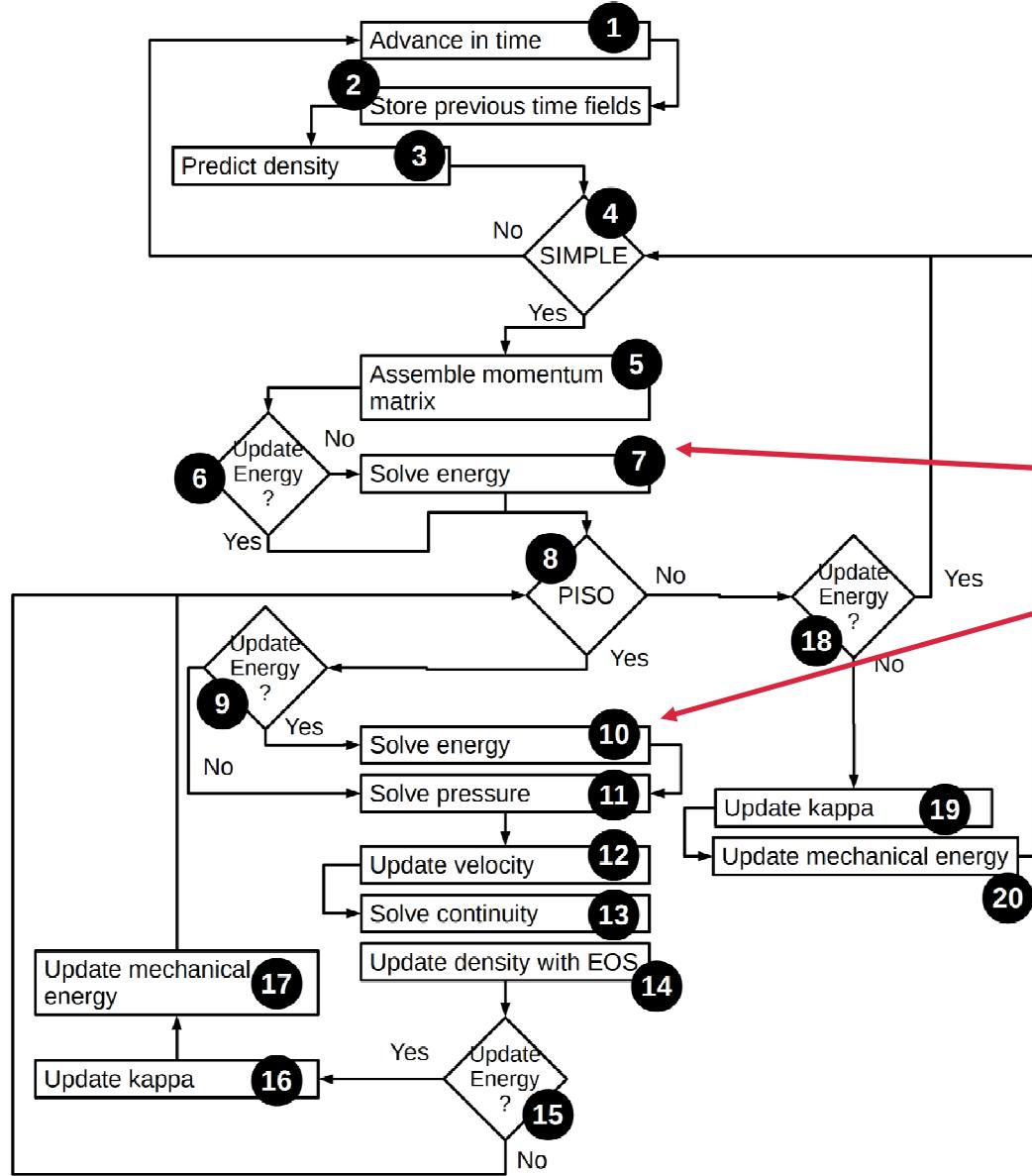
```
#include "createFluidFields.H"  
#include "createSolidFields.H"
```

- Update **Make/options**:

```
EXE_INC = \  
... \  
-I ./solid
```

- Update **myChtPimpleCentralFoam.C** — insert solution for heat transfer in solids in the PIMPLE algorithm

# Where to place solid body heat transfer block

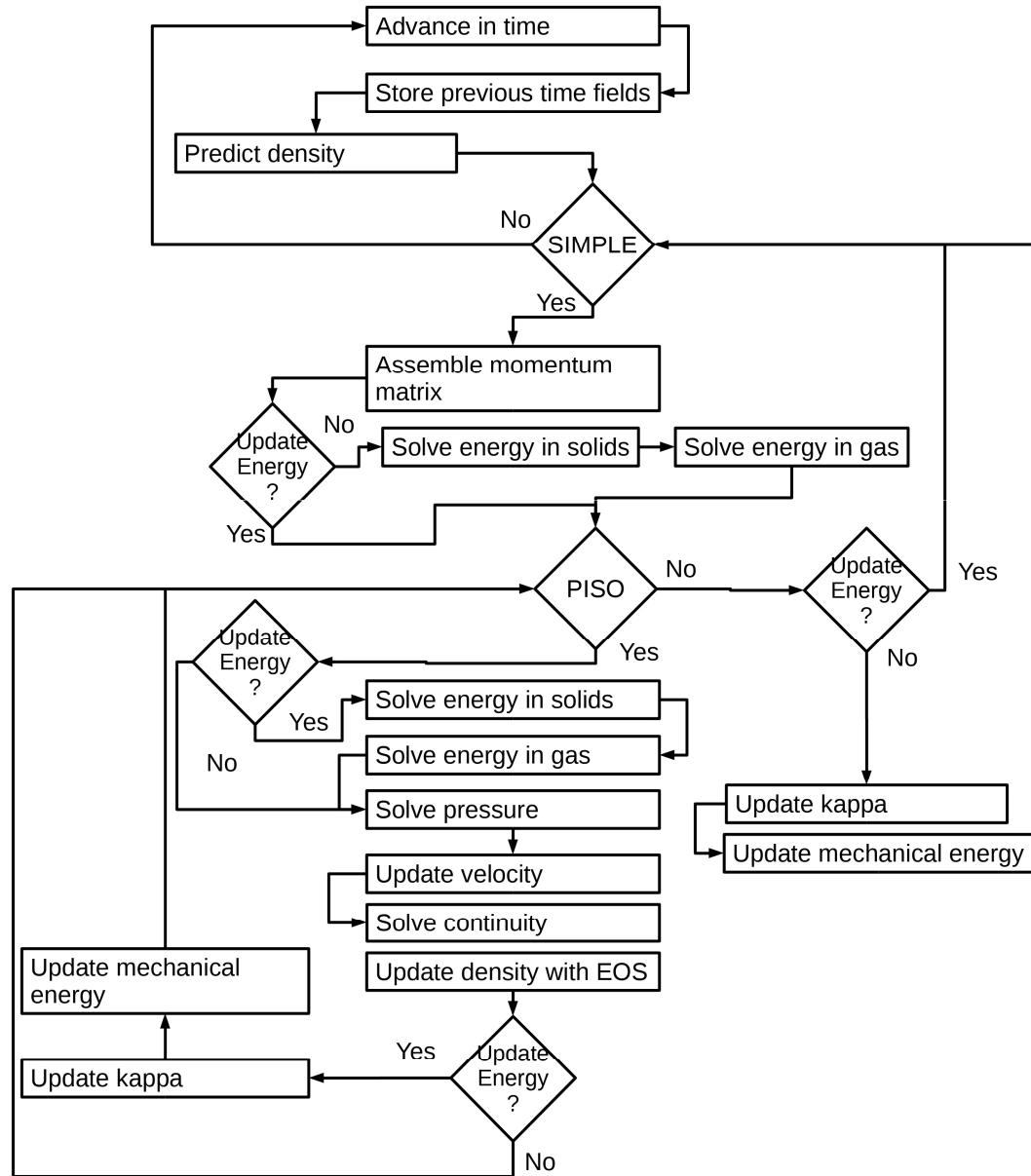


Add steps to solve for energy balance in solids before solving energy balance in fluid

`solveSolidRegions.H`

`solveSolidRegions.H`

# Final PIMPLE algorithm

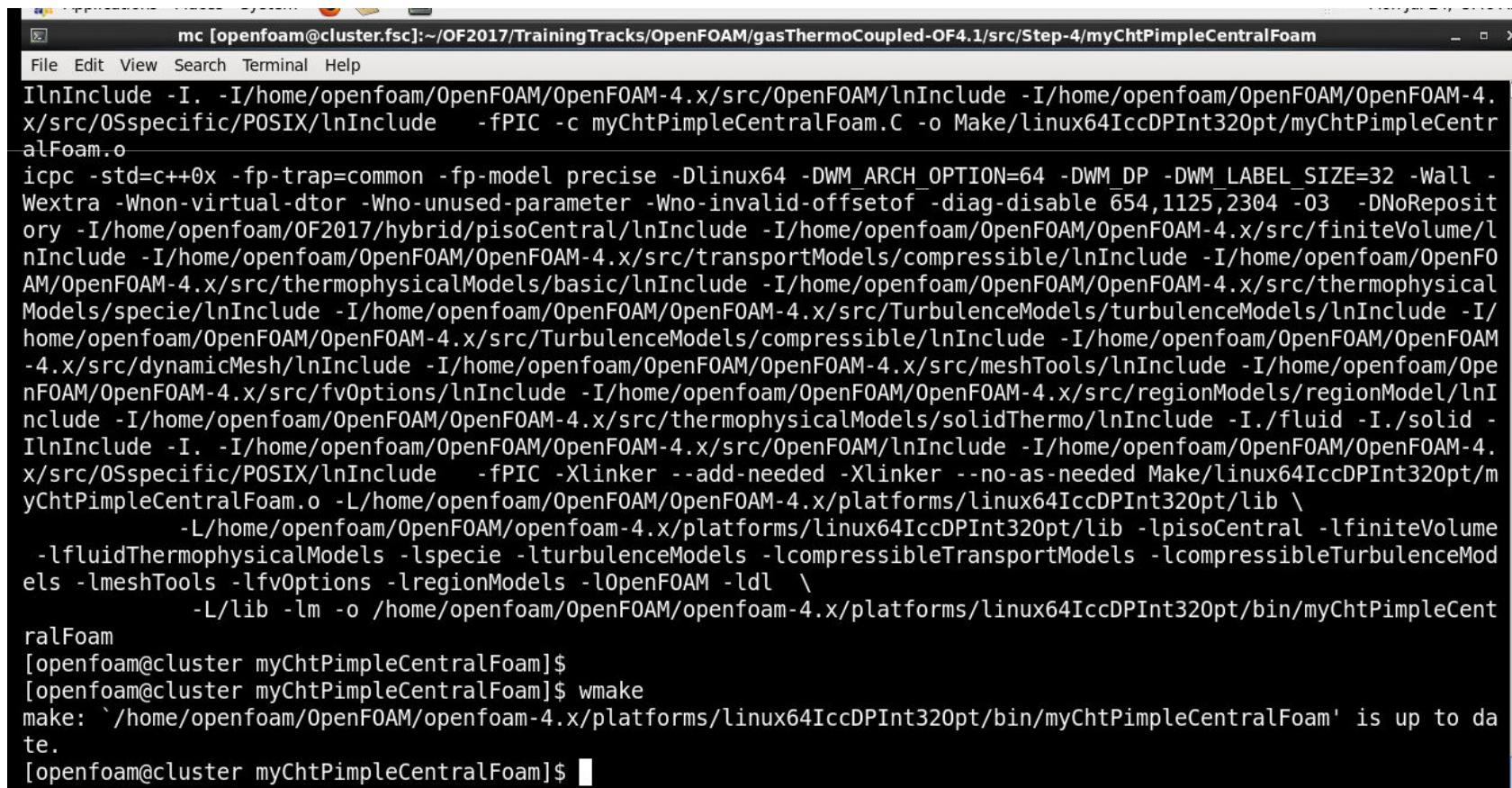


# Compile solver

- To compile solver, type

wmake

- Screenshot



The screenshot shows a terminal window titled "mc [openfoam@cluster.fsc]:~/OF2017/TrainingTracks/OpenFOAM/gasThermoCoupled-OF4.1/src/Step-4/myChtPimpleCentralFoam". The terminal displays the command "wmake" being run, which triggers a complex compilation process. The output shows numerous compiler flags (e.g., -I, -D, -fPIC, -c) and linker options (-L, -l, -o) for linking against various OpenFOAM libraries and models. The compilation is successful, as indicated by the message "make: `/home/openfoam/OpenFOAM/openfoam-4.x/platforms/linux64IccDPInt320pt/bin/myChtPimpleCentralFoam' is up to date." at the end.

```
mc [openfoam@cluster.fsc]:~/OF2017/TrainingTracks/OpenFOAM/gasThermoCoupled-OF4.1/src/Step-4/myChtPimpleCentralFoam
File Edit View Search Terminal Help
IlnInclude -I. -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/OpenFOAM/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/OSspecific/POSIX/lInclude -fPIC -c myChtPimpleCentralFoam.C -o Make/linux64IccDPInt320pt/myChtPimpleCentralFoam.o
icpc -std=c++0x -fp-trap=common -fp-model precise -Dlinux64 -DWM_ARCH_OPTION=64 -DWM_DP -DWM_LABEL_SIZE=32 -Wall -Wextra -Wnon-virtual-dtor -Wno-unused-parameter -Wno-invalid-offsetof -diag-disable 654,1125,2304 -O3 -DNoRepository -I/home/openfoam/OF2017/hybrid/pisoCentral/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/finiteVolume/lInclude -I/home/openfoam/OpenFOAM-4.x/src/transportModels/compressible/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/thermophysicalModels/basic/lInclude -I/home/openfoam/OpenFOAM-4.x/src/thermophysicalModels/specie/lInclude -I/home/openfoam/OpenFOAM-4.x/src/TurbulenceModels/turbulenceModels/lInclude -I/home/openfoam/OpenFOAM-4.x/src/TurbulenceModels/compressible/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/dynamicMesh/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/meshTools/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/fvOptions/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/regionModels/regionModel/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/thermophysicalModels/solidThermo/lInclude -I./fluid -I./solid -IlnInclude -I. -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/OpenFOAM/lInclude -I/home/openfoam/OpenFOAM/OpenFOAM-4.x/src/OSspecific/POSIX/lInclude -fPIC -Xlinker --add-needed -Xlinker --no-as-needed Make/linux64IccDPInt320pt/myChtPimpleCentralFoam.o -L/home/openfoam/OpenFOAM/OpenFOAM-4.x/platforms/linux64IccDPInt320pt/lib \
-L/home/openfoam/OpenFOAM/openfoam-4.x/platforms/linux64IccDPInt320pt/lib -lpisoCentral -lfiniteVolume \
-lfluidThermophysicalModels -lspecie -lturbulenceModels -lcompressibleTransportModels -lcompressibleTurbulenceModels -lmeshTools -lfvOptions -lregionModels -lOpenFOAM -ldl \
-L/lib -lm -o /home/openfoam/OpenFOAM/openfoam-4.x/platforms/linux64IccDPInt320pt/bin/myChtPimpleCentralFoam
[openfoam@cluster myChtPimpleCentralFoam]$
[openfoam@cluster myChtPimpleCentralFoam]$ wmake
make: `/home/openfoam/OpenFOAM/openfoam-4.x/platforms/linux64IccDPInt320pt/bin/myChtPimpleCentralFoam' is up to date.
[openfoam@cluster myChtPimpleCentralFoam]$
```

## Step 4 results

- Source code for simulation of heat transfer in solid bodies added to `main` program.
- Energy balance equations for solids are solved in `PIMPLE` algorithm before solving for energy balance of gas.
- New solver `myChtPimpleCentralFoam` was compiled.

# Test cases

- Laminar flow
- Turbulent flow

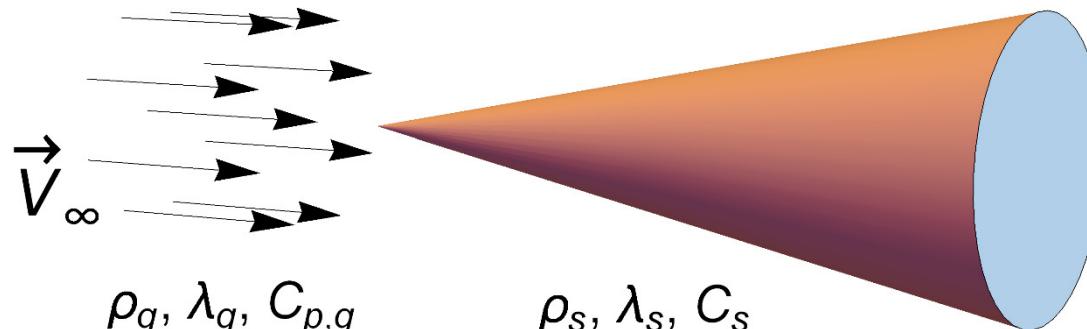
$$V_{g,\infty} = 750 \text{ m/s}$$

$$p_{g,\infty} = 26500 \text{ Pa}$$

$$T_{g,\infty} = 223 \text{ K}$$

$$C_{p,g} = 1005 \text{ J/(kg} \cdot \text{K)}$$

Viscosity model - Sutherland



$$\rho_g, \lambda_g, C_{p,g}$$

$$\rho_s, \lambda_s, C_s$$

$$\rho = 4500 \text{ kg/m}^3$$

$$\lambda = 20 \text{ W/(m} \cdot \text{K)}$$

$$T_{s,0} = 293 \text{ K}$$

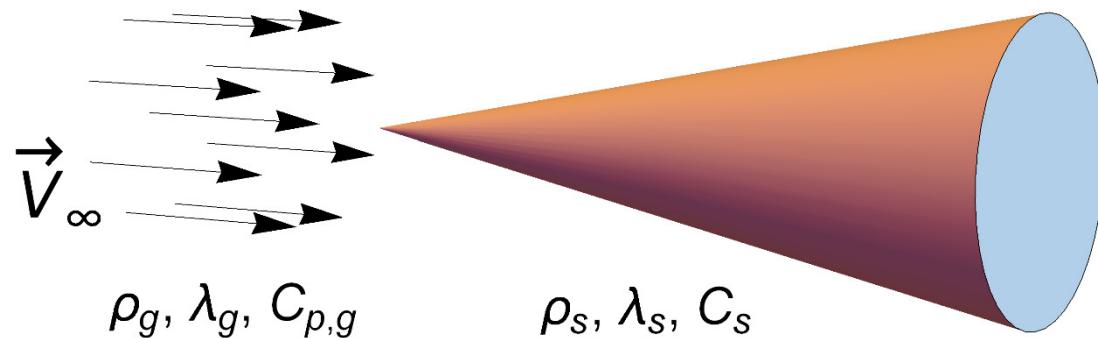
$$C_{p,g} = 600 \text{ J/(kg} \cdot \text{K)}$$

RAS simulation – k-w SST turbulence model

<https://github.com/unicfdlab/TrainingTracks/tree/master/OpenFOAM/gasThermoCoupled-OF4.1/cases>

# Run test cases

- Laminar flow
- Turbulent flow



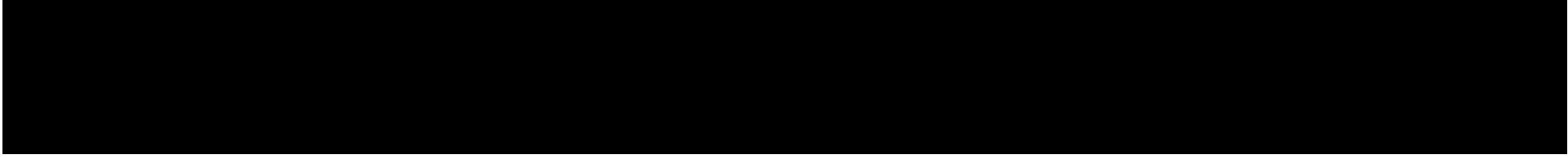
- Decompose regions
  - `decomposePar -region solid`
  - `decomposePar -region fluid`
- Run cases
  - `mpirun -np 4 myChtPimpleCentralFoam -parallel`
- Reconstruct solution (if necessary)
  - `reconstructPar`

# wallHeatFlux postprocessor

- Source code is copied to `/src`
- `/Make/options` is changed  
 $\text{\$FOAM_APPBIN} \longrightarrow \text{\$FOAM_USER_APPBIN}$
- To compile: `wmake libso`
- Run:  
`mpirun -np 4 wallHeatFlux -parallel -region solid`  
or  
`wallHeatFlux -region solid`

**Computed heat flux (wedge angle 5 deg.)**

- for laminar case: ~ 100 W
- for turbulent case: ~ 1150 W



# Thank you for your attention!

All the source codes and numerical examples  
are available online

<https://github.com/unicfdlab/>