One- and Two-Phase Coupling of OpenFOAM with the Thermal-Hydraulic Code ATHLET for Nuclear Safety Analyses

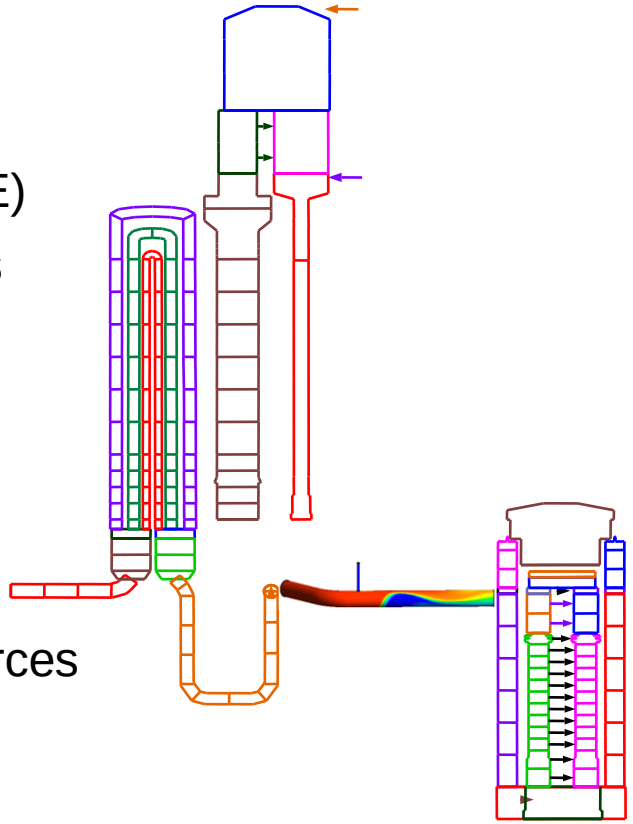
Joachim Herb & Flavius Chiriac, GRS

2016-06-29

11th OpenFOAM® Workshop, Guimarães, Portugal

Motivation to use OpenFOAM for simulations of NPP

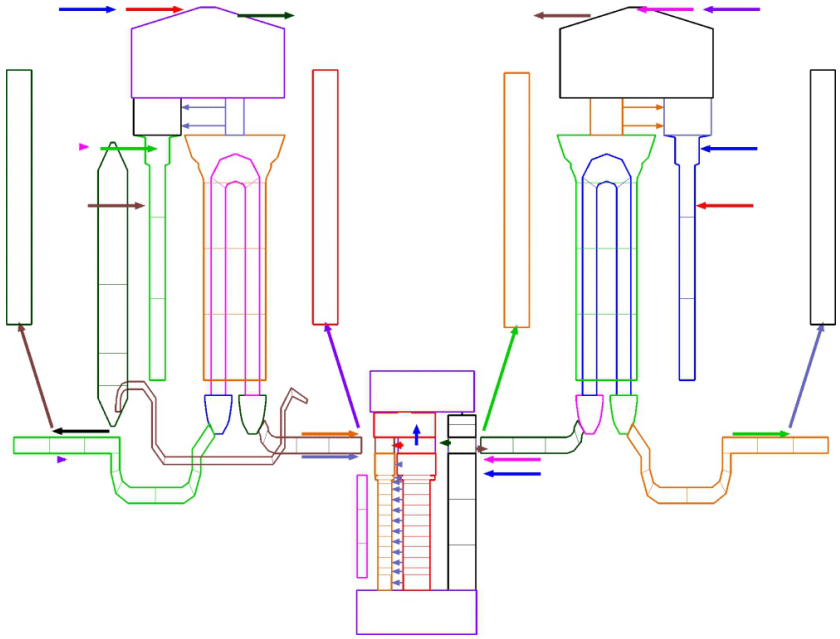
- Simulate accidents in nuclear power plants
- Traditionally, 1D “thermo-hydraulic” system codes are used (e.g. ATHLET, CATHARE, RELAP, TRACE)
- However, different effects in reactor safety analyses are 3D effects and require adequate tools, e.g.
  - High-cycle thermal fatigue by turbulent mixing
  - Stratification versus turbulent mixing (e.g. thermal, boric acid $\Rightarrow$ reactivity feedback)
- It is not possible to simulate the whole primary circuit by CFD due to required computational resources



- OpenFOAM applications in simulations for NPP: currently > 50 papers in peer-reviewed journals applying OpenFOAM https://openfoamwiki.net/index.php/SIG_Nuclear_/_Publications

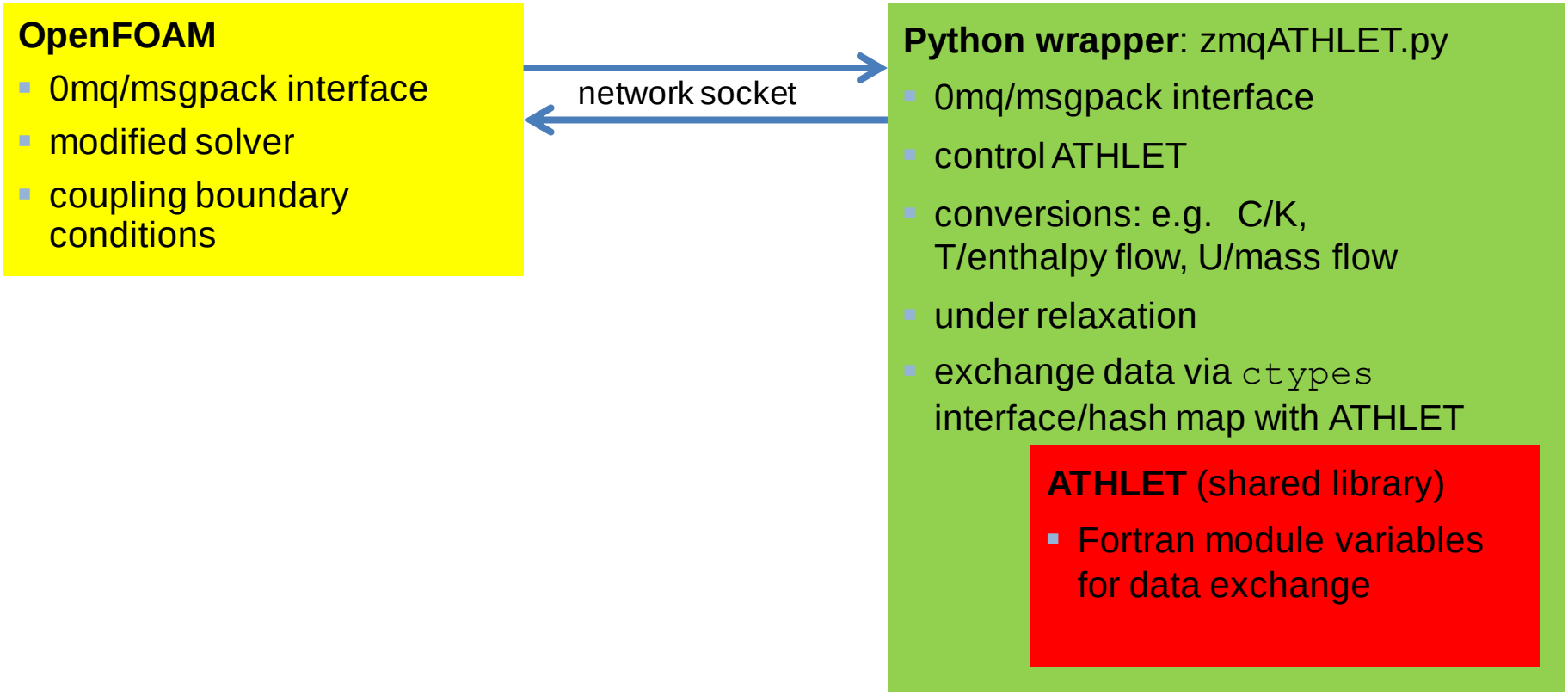
Thermo-hydraulic code ATHLET

- **A**nalysis of **TH**ermo-hydraulics of **LE**aks and **T**ransients
- 1D lumped parameter finite volume code for fluid dynamics
- Also modules for
  - heat transfer
  - nuclear heat generation
  - component behavior, I&C systems
- Solves 6 equations for:
  - Liquid/vapor mass
  - Liquid/vapor energy
  - Momentum



- Working fluids: Light water, heavy water, liquid metals (sodium, lead, lead-bismuth eutectic), helium
- FORTRAN!

Architecture of coupling



Master: Transient OpenFOAM Solver (buoyantPimpleFoam, twoPhaseEulerFoam)

Slave: Python wrapper around ATHLET

Network interface

Commands to control ATHLET by OpenFOAM

```
dataFromOpenFOAM, dataToOpenFOAM
initCoupledSolver
initTransientRun
innerStep, finalInnerStep
stopCoupledSolver
setTime
loopCoupledSolver
```

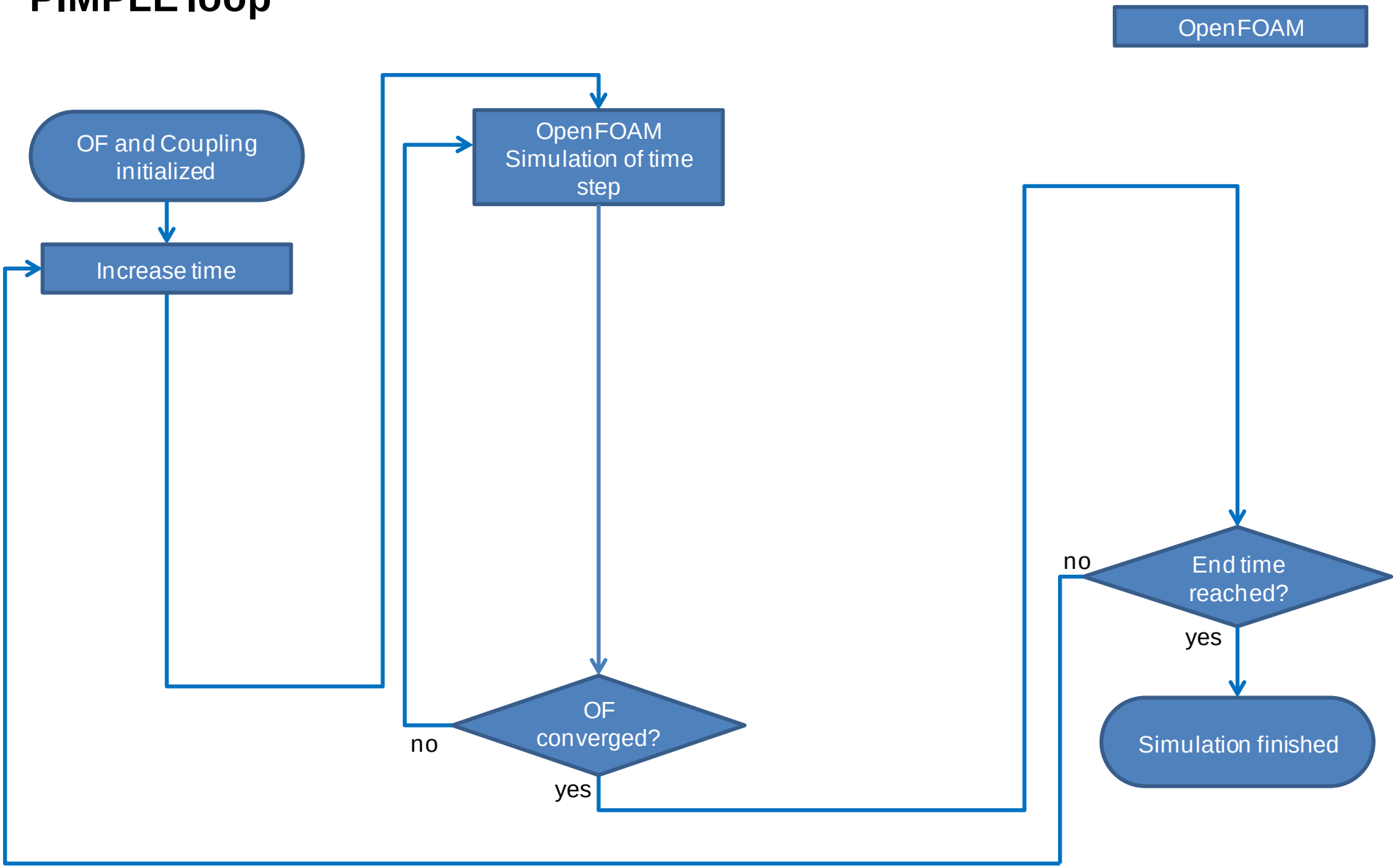
Interface code on Python side

```
message = self._socket.recv()
(cmd, dData, iData, sData) = msgpack.unpackb(message, use_list=False)

print("\n\n--Solver command:", cmd)

result_data = self._commands[cmd](dData, iData, sData) # call ATHLET

if not self._solver.is_running() and result_data[0] != 'stopOK':
    result_data[0] = "ERROR"

self._socket.send(msgpack.packb(result_data))
```

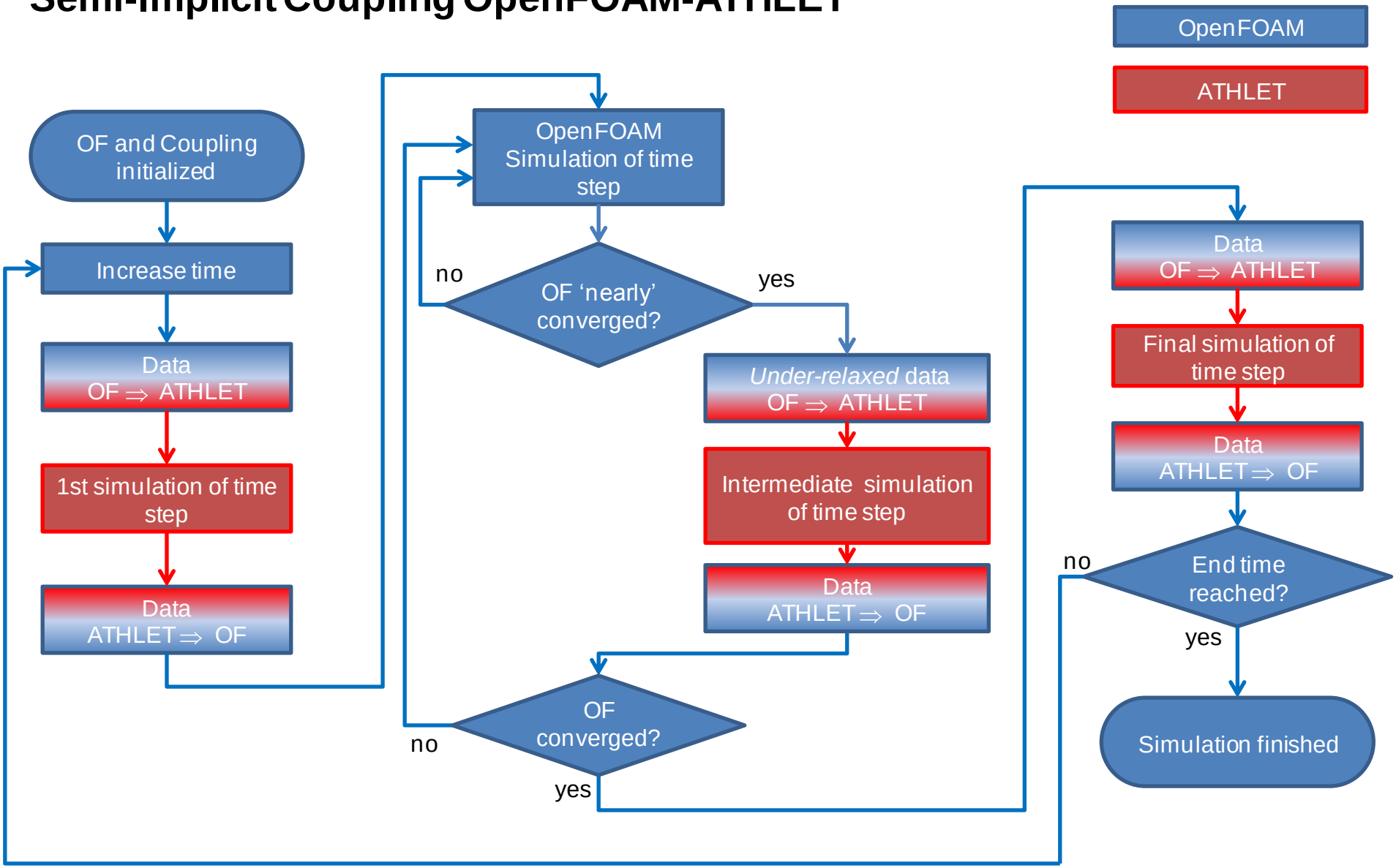
Semi-implicit coupling

- Use under-relaxation to increase stability, especially important if OpenFOAM domain is part of a ATHLET closed loop system
- Boundary conditions for ATHLET are updated with weighted average of current OpenFOAM values and previous results
- Repeat ATHLET calculation within outer corrector loop until OpenFOAM converges with updated ATHLET results

PIMPLE loop

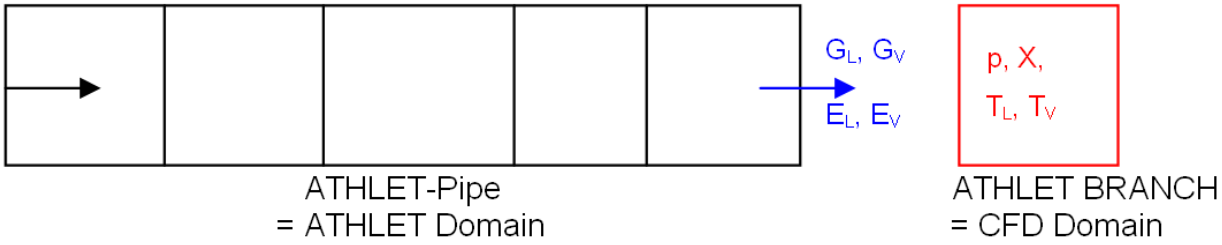

OpenFOAM
OF and Coupling initialized
Increase time
OpenFOAM Simulation of time step
OF converged?
no
yes
End time reached?
no
yes
Simulation finished

Semi-Implicit Coupling OpenFOAM-ATHLET



Boundary Conditions for Data Exchange (ATHLET)

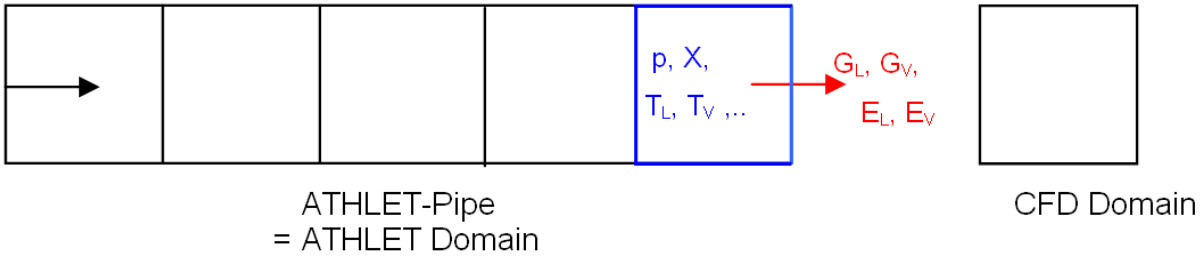
- ATHLET calculates flow quantities, EXT calculates scalar values (ATHLET p-h boundary)



ATHLET

EXT Program

- ATHLET calculates scalar values, EXT calculates flow quantities (ATHLET Fill)



Boundary Conditions for Data Exchange (OpenFOAM)

Calculate input for ATHLET

Area weighted averaged values over OpenFOAM boundary

```
// Total area of your boundary Patch
scalar area = gSum(this->patch().magSf());
// Average pressure on patch
scalar pC = gSum(this->patch().magSf() * (*this)) / area;
return pC;
```

Mass flow weighted averaged values over OpenFOAM boundary

```
scalar flux = gSum(phip);
const fvPatchField<scalar>& rhop =
      patch().lookupPatchField<volScalarField, scalar>(this->rhoName_);
scalar rhom = gSum(phip * rhop) / (flux + VSMALL);
return flux / (rhom * area + VSMALL);
```

Boundary Conditions for Data Exchange (OpenFOAM)

Calculate input for OpenFOAM based on new boundary condition types

- `couplingInletOutlet` based on `inletOutlet` (for temperature)
- `couplingNormalInletOutletVelocity` based on `normalInletOutletVelocity` and `fixedMean`
- `couplingBuoyantPressure` based on `buoyantPressure`

```
OUTLET
   {
       type            couplingBuoyantPressure;
       couplingID      2; // ATHLET interface number
       coupledField    p; // ATHLET variable name
       value           uniform 447622; // value for initialization
   }
OUTLET
   {
       type            couplingInletOutlet;
       couplingID      2;  // ATHLET interface number
       coupledField    Tl; // ATHLET variable name
       value           uniform 512.65; // value for initialization
   }
```

Support for Parallel Processing

- Communicate with coupled code/ATHLET only in `Pstream::master()` process
- Share data and synchronize between OpenFOAM processes by `Pstream::scatter(xxx, Pstream::blocking)`
- Use OpenFOAM methods/macros (`gSum`, `reduce(xxx, sumOp<scalar>())`, ...) for multiprocessing support

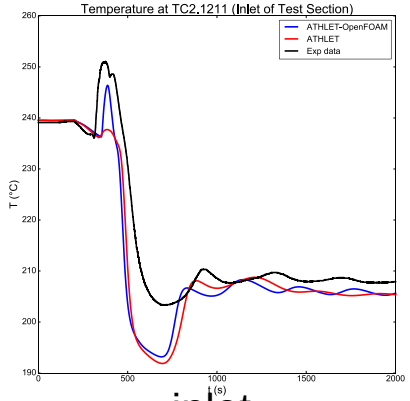
OpenFOAM Solver Modifications (e.g. in buoyantPimpleFoam)

```
 [...]
#include "fixedFluxPressureFvPatchScalarField.H"
+#include "couplingSolver.H"

int main(int argc, char *argv[]) {
     #include "setRootCase.H"
[...]
     #include "readGravitationalAcceleration.H"
+    #include "createCoupling.H"
     #include "createFields.H"
[...]
     #include "setInitialDeltaT.H"
     pimpleControl pimple(mesh);
+    cA.startupExternalSolver(runTime);
     Info<< "\nStarting time loop\n" << endl;

-    while (runTime.run())
+    while (runTime.run() && cA.run())
     {
[...]
         runTime++;
         Info<< "Time = " << runTime.timeName() << nl
             << endl;

+        cA.prepareNextTimeStep(runTime);
         #include "rhoEqn.H"
```

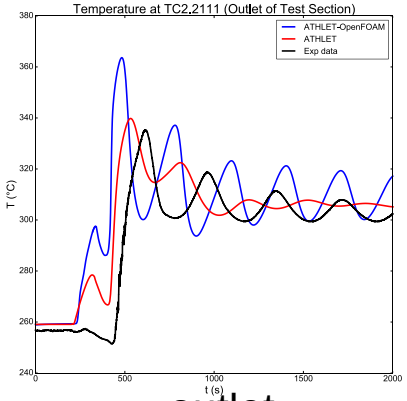
```
// --- Pressure-velocity PIMPLE corrector loop
-        while (pimple.loop())
+        while (cA.loop(pimple))
+        // at least one of both solvers has not
+        // converged yet
         {
             #include "UEqn.H"
             #include "EEqn.H"
             // --- Pressure corrector loop
             while (pimple.correct())
             {
                 #include "pEqn.H"
             }
             if (pimple.turbCorr())
             {
                 turbulence->correct();
             }
+            cA.callExternalSolver(pimple, runTime);
         }
         rho = thermo.rho();
         runTime.write();

+        cA.finalStep(runTime);
         Info<< "ExecutionTime = "
[...]
     }
+    cA.stop(false);
     Info<< "End\n" << endl;
     return 0;
```

Coupled Simulation of TALL Test Facility

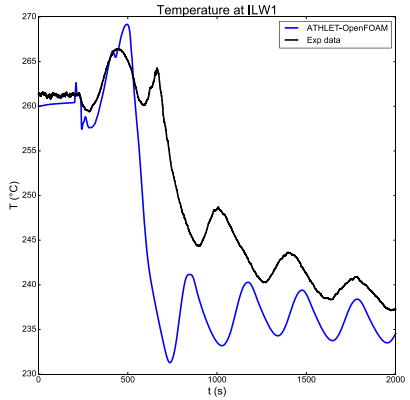
- Test facility for lead bismuth eutectic fluid
- Simulation of transition from forced to natural convection after pump trip



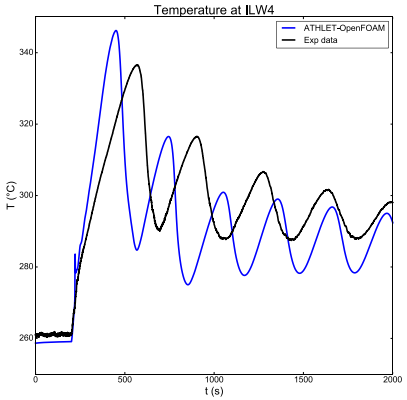
inlet



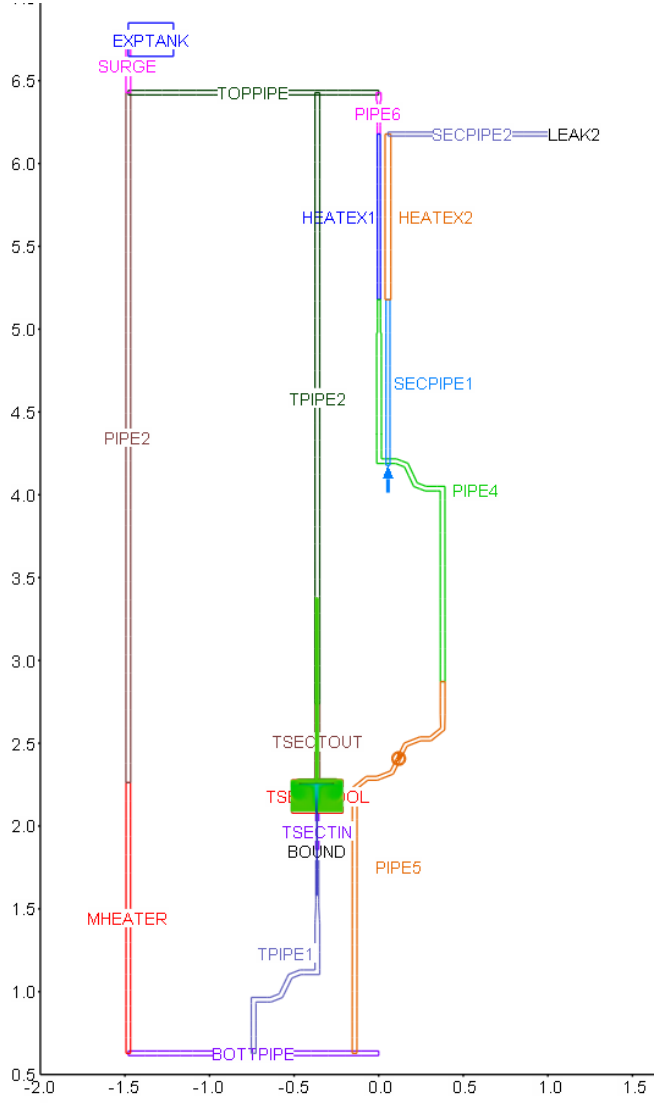
outlet



side wall



side wall

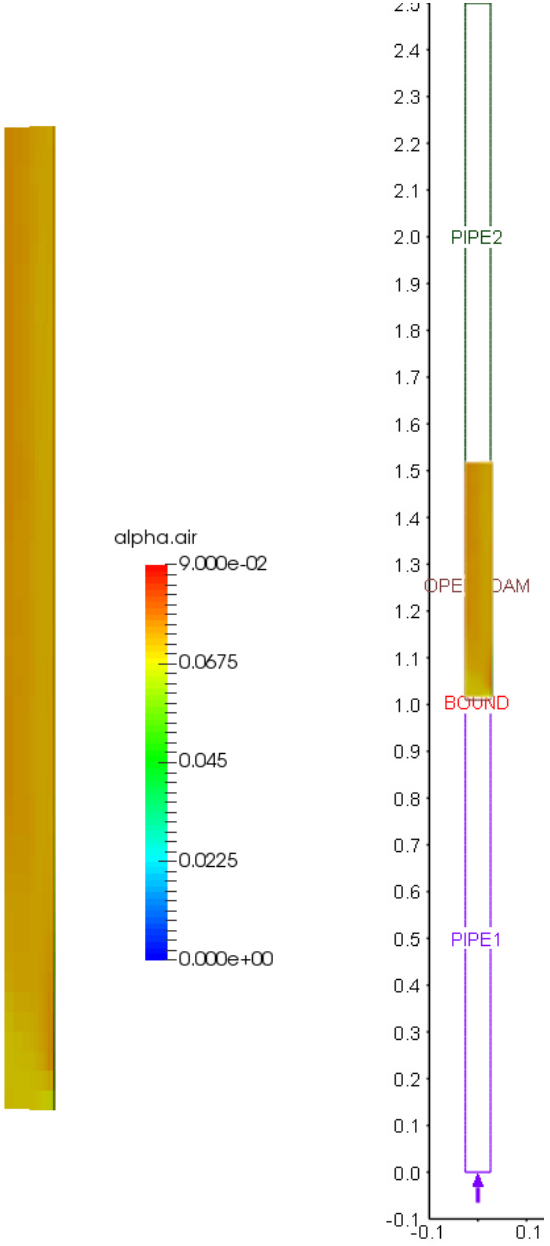


/PAP 15/

Two-Phase Coupling

Coupled fields

- $T_{liquid/vapor}$
- $U_{liquid/vapor}$
- pressure
- χ/α (steam quality/volume fraction)



Summary and Outlook

Current state

- Coupling between OpenFOAM and thermo-hydraulic code ATHLET:
  Implemented via network socket communication
- Single phase flows with buoyancy effects:
  First validation with experiments
- Two-phase flows:
  water/vapor

Next steps

- One-phase flows: Further validation with experiments
- Two-phase flows:
  - water/non-condensable gases
  - validation with experiments
- Release OpenFOAM part of coupling as Open Source

References

/PAP 15/ A. Papukchiev et al., "Multiscale Analysis of Forced and Natural Convection including Heat Transfer Phenomena in the TALL-3D Experimental Facility", NURETH-16, International Topical Meeting on Nuclear Reactor Thermal Hydraulics, August 30 - September 4, Chicago, 2015

/RIV 14/ J. Rivero Pérez-Aradros, "Test and Validation of the coupling between ATHLET and OpenFOAM using the example of Liquid Metal Cooling Systems", Master Thesis, Technische Universität München, February 2014

/CHI 16/ F. Chiriac, Extension of the ATHLET-OpenFOAM Coupling Capabilities to Two-Phase Flows, Master Thesis, Technische Universität München, February 2016