# Coupling LAMMPS and OpenFOAM for Multi-Scale Models

**M. Magnini[1], E. R. Smith[2], G. J. Pringle[3], G. Gennari[4]**

[1]*Dept. of Mechanical Engineering, University of Nottingham, Nottingham.* E-mail: *mirco.magnini@nottingham.ac.uk*

[2]*Mechanical and Aerospace Engineering, Brunel University London: edward.smith@brunel.ac.uk*

[3]*EPCC, University of Edinburgh: g.pringle@epcc.ed.ac.uk*

[4]*Dept. of Mechanical Engineering, University of Nottingham, Nottingham*: *gabriele.gennari@nottingham.ac.uk*

**eCSE06-01: "Hybrid Atomistic-Continuum Simulations of Boiling Across Scales"**

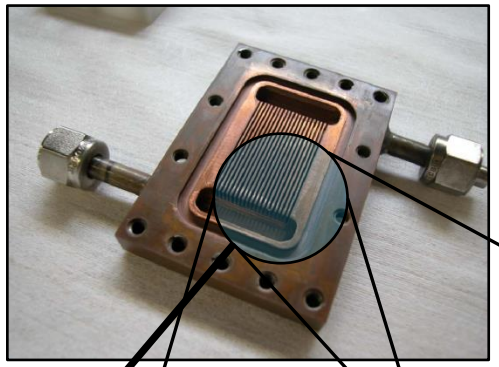
# Outline

- Hybrid Atomistic-Continuum Simulations of Boiling (Mirco)
- The CPL Library (Ed)
- Running CPL library in ARCHER2, installation overview, using the modules (Gavin)
- Coupled OpenFOAM’s solvers (Mirco)
- LAMMPS and coupling with CPL (Ed)
- The coupled Couette flow case (Ed)
- The coupled boiling case (Gabriele)
- Demo on ARCHER2 (Gabriele)

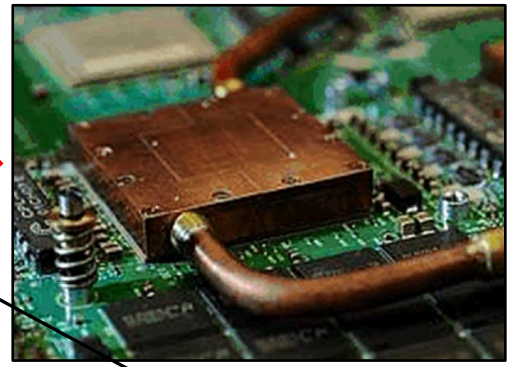
# Hybrid Atomistic-Continuum Simulations of Boiling: Why?

## Cooling using boiling flows

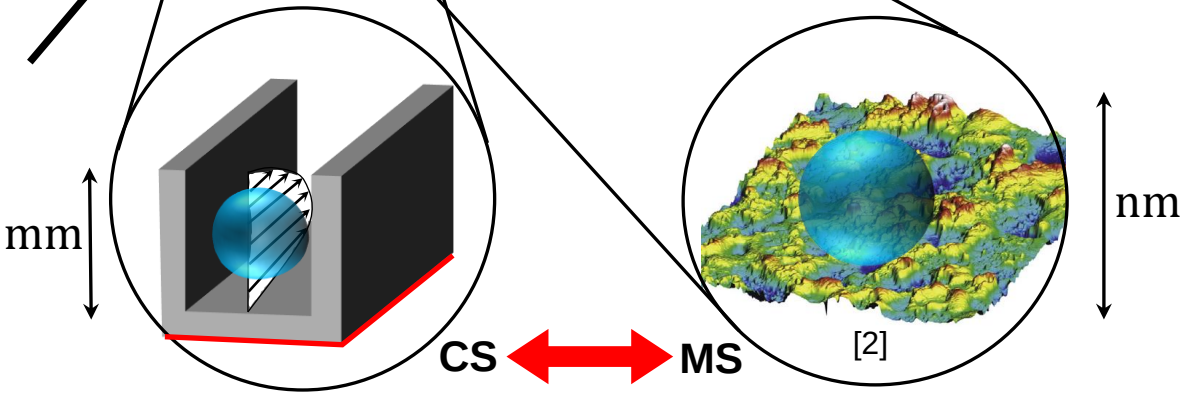
- Very high heat transfer rates O(MW/m$^2$)
- Uniform surface temperatures
- Microevaporators: high surface-to-volume ratio

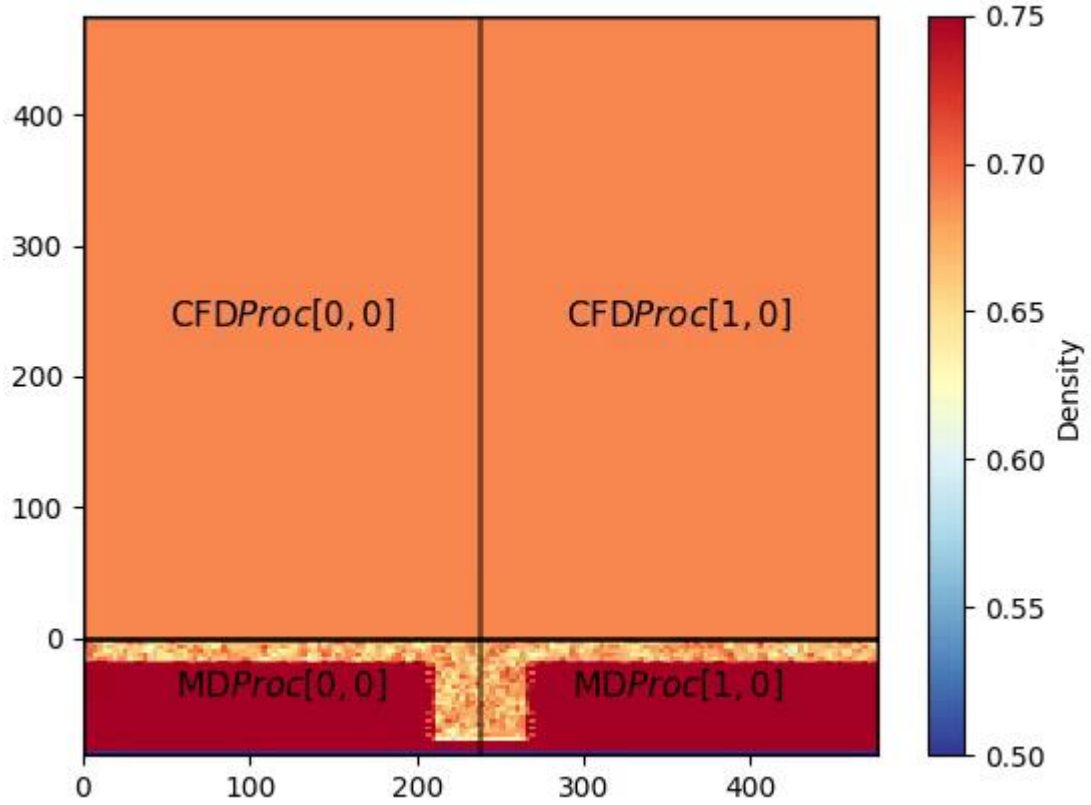


Multi-microchannel evaporator with 20 channels, 0.45x4.0 mm [1]



Evaporator sealed on the chip die




**CS: mesh-based ↔ coupling ↔ MS: particle-based**

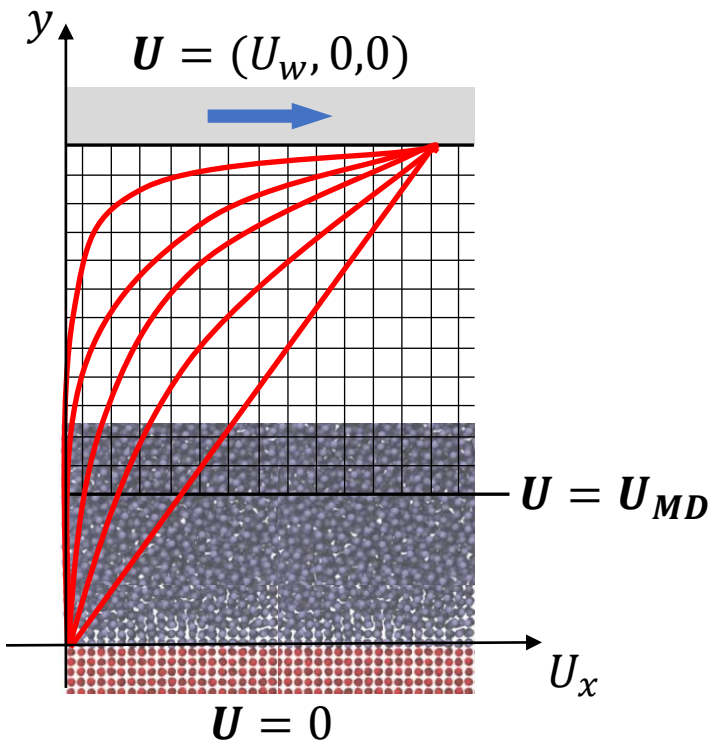

[1] J. Park, PhD Thesis, EPFL, 2008.
[2] Paz et al., Exp Therm Fluid Sci 64 (2015) 114.

# CPLIcoFoam

Transient solver for incompressible, laminar flow of Newtonian fluids

$$\nabla \cdot \boldsymbol{U} = 0$$

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot (\boldsymbol{U}\boldsymbol{U}) - \nabla \cdot (\nu \nabla \boldsymbol{U}) = -\nabla p$$




CPLIcoFoam.C

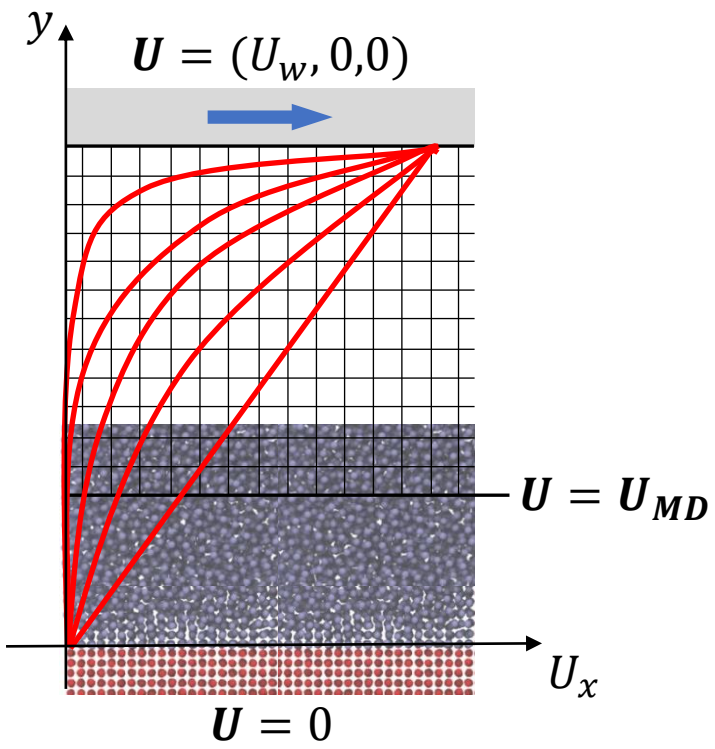
```
32  #include "fvCFD.H"
33  #include "pisoControl.H"
34  #include "CPLSocketFOAM.H"
35
36  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
37
38  int main(int argc, char *argv[])
39  {
40
41      //Check if coupled based on cpl/COUPLER.in input file
42      bool coupled;
43      if (file_exists("./cpl/COUPLER.in")) {
44          Info<< "Assuming coupled run as cpl/COUPLER.in exists\n" << endl;
45          coupled=true;
46      } else {
47          Info<< "Assuming uncoupled run as cpl/COUPLER.in does not exist\n" << endl;
48          coupled=false;
49      }
50
51
52      // Create a CPL object (not used if uncoupled) and intialises MPI
53      CPLSocketFOAM CPL;
54      if (coupled)
55          CPL.initComms(argc, argv);
```

# CPLIcoFoam

Transient solver for incompressible, laminar flow of Newtonian fluids

$$\nabla \cdot \boldsymbol{U} = 0$$

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot (\boldsymbol{U}\boldsymbol{U}) - \nabla \cdot (\nu \nabla \boldsymbol{U}) = -\nabla p$$



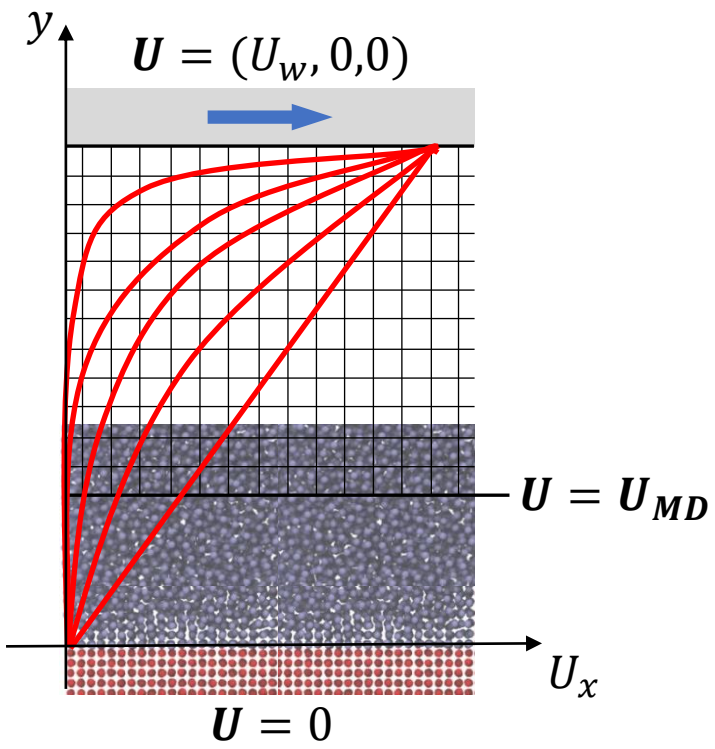

```
65          // MPI_Init is called somewhere in the PStream library
66      if (coupled)
67          CPL.initCFD(runTime, mesh);
68
69      // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
70
71
72          // Initial communication to initialize domains
73      if (coupled){
74          CPL.pack(U, p, nu, mesh, CPL.VEL);
75          CPL.send();
76          CPL.recvVelocity();
77          CPL.unpackVelocity(U, mesh);
78      //    CPL.recvVelocityPressure();
79      //    CPL.unpackVelocityPressure(U, p, mesh);
80      }
81
82      Info<< "\nStarting time loop\n" << endl;
83      while (runTime.loop())
```

# CPLIcoFoam

Transient solver for incompressible, laminar flow of Newtonian fluids

$$\nabla \cdot \boldsymbol{U} = 0$$

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot (\boldsymbol{U}\boldsymbol{U}) - \nabla \cdot (\nu \nabla \boldsymbol{U}) = -\nabla p$$




```
82          Info<< "\nStarting time loop\n" << endl;
83          while (runTime.loop())
84          {
85
86              if (coupled){
87                  std::cout << "CPL.VEL is on " << std::endl;
88                  CPL.pack(U, p, nu, mesh, CPL.VEL);
89                  //CPL.pack(U, p, nu, mesh, CPL.STRESS);
90                  CPL.send();
91                  CPL.recvVelocity();
92                  CPL.unpackVelocity(U, mesh);
93              //    CPL.recvVelocityPressure();
94              //    CPL.unpackVelocityPressure(U, p, mesh);
95              }
96
97              Info<< "Time = " << runTime.timeName() << nl << endl;
98
99              #include "CourantNo.H"
100
101             // Momentum predictor
102
103             fvVectorMatrix UEqn
104             (
105                 fvm::ddt(U)
```

# CPLinterFoamHardtPhaseChange

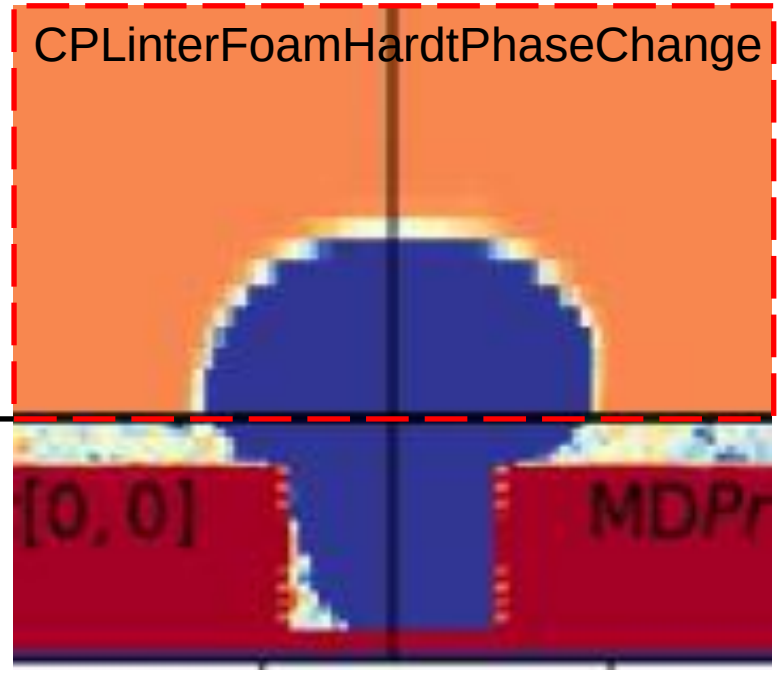
Solver for two incompressible, immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, and Hardt and Wondra model [1,2] for phase change

$$\nabla \cdot \boldsymbol{U} = \frac{\dot{\rho}}{\rho}$$

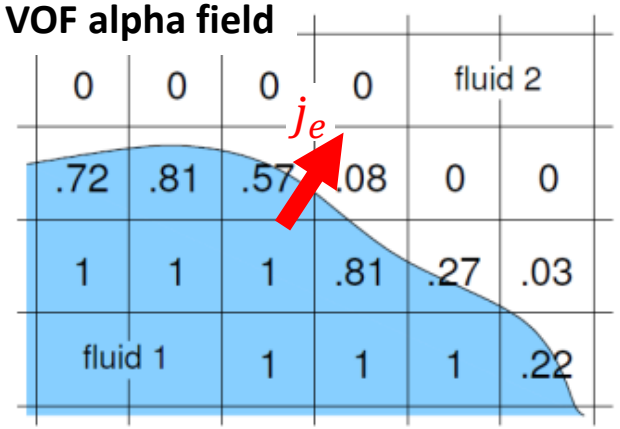
$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \boldsymbol{U}) = \frac{\dot{\rho}}{\rho}\alpha$$

$$\frac{\partial (\rho \boldsymbol{U})}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \boldsymbol{F_\sigma}$$

$$\frac{\partial (\rho c_p T)}{\partial t} + \nabla \cdot (\rho c_p \boldsymbol{U} T) = \nabla \cdot (\lambda \nabla T) - \dot{\rho} h_{lv}$$




**VOF alpha field**




$$\dot{\rho} = j_e |\nabla \alpha|$$

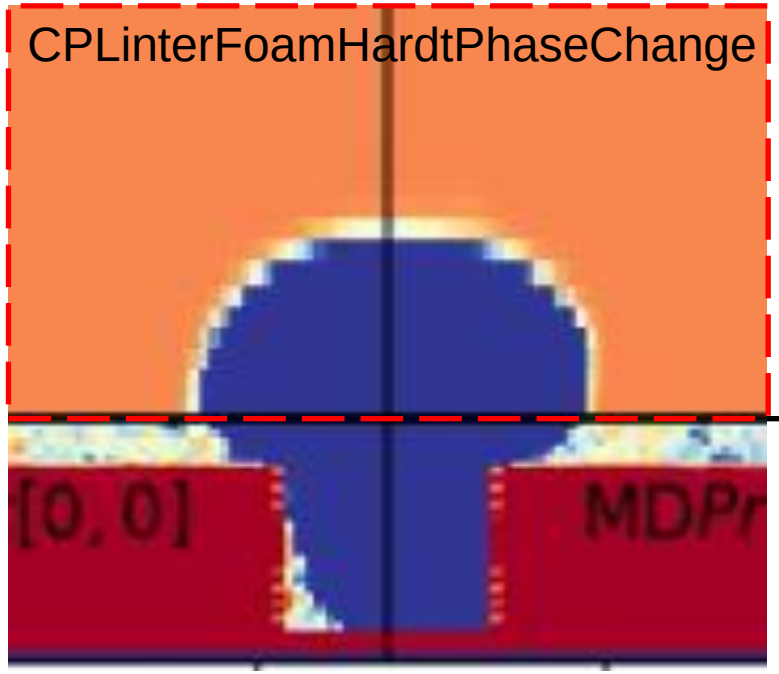
$$j_e = \frac{h_i}{h_{lv}}(T_i - T_{sat})$$

$$h_i = \frac{2\gamma}{2-\gamma}\left(\frac{\bar{M}}{2\pi \bar{R}}\right)^{1/2} \frac{\rho_v h_{lv}^2}{T_{sat}^{3/2}}$$


[1] S. Hardt and F. Wondra, J. Computational Physics 227 (2008) 5871.
[2] F. Municchi et al., Int J Heat Mass Transf 195 (2022) 123166.

# CPLinterFoamHardtPhaseChange

Solver for two incompressible, immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, and Hardt and Wondra model [1,2] for phase change




$$(\boldsymbol{U}, \rho, \alpha, T) = (\boldsymbol{U}, \rho, \alpha, T)_{MD}$$

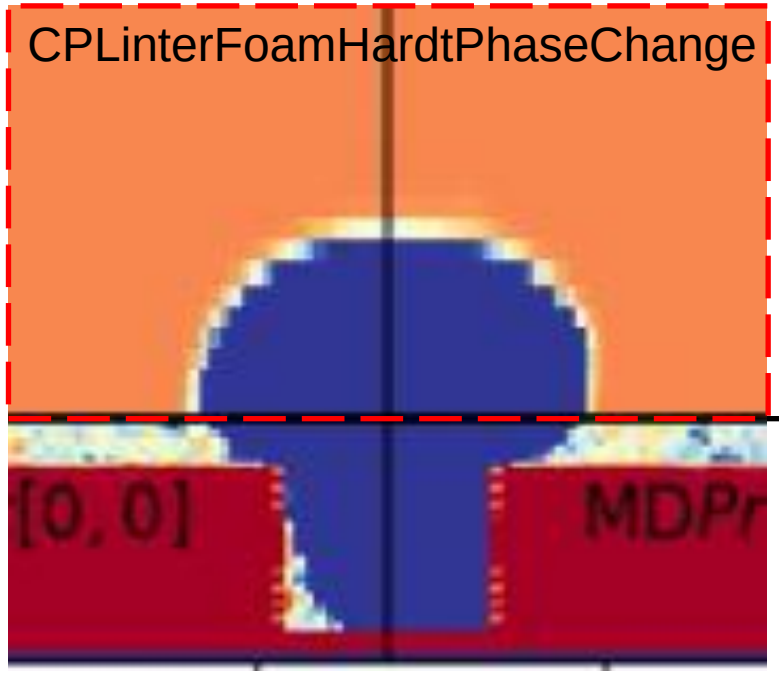
**CPLinterFoamHardtPhaseChange.C**

```
51  #include "CPLSocketFOAM.H"
52
53  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
54
55  int main(int argc, char *argv[])
56  {
57    Info << "Hello, this is CPLinterFoamHardtPhaseChange from ~/demo/CPL..." << endl;
58      argList::addNote
59      (
60          "Solver for two incompressible, isothermal immiscible fluids"
61          " using VOF phase-fraction based interface capturing.\n"
62          "With optional mesh motion and mesh topology changes including"
63          " adaptive re-meshing."
64      );
65
66      //Check if coupled based on cpl/COUPLER.in input file
67      bool coupled;
68      if (file_exists("./cpl/COUPLER.in")) {
69          Info<< "Assuming coupled run as cpl/COUPLER.in exists\n" << endl;
70          coupled=true;
71      } else {
72          Info<< "Assuming uncoupled run as cpl/COUPLER.in does not exist\n" << endl;
73          coupled=false;
74      }
75
76      // Create a CPL object (not used if uncoupled) and intialises MPI
77      CPLSocketFOAM CPL;
78
79      if (coupled)
80          CPL.initComms(argc, argv);
```


[1] S. Hardt and F. Wondra, J. Computational Physics 227 (2008) 5871.
[2] F. Municchi et al., Int J Heat Mass Transf 195 (2022) 123166.

# CPLinterFoamHardtPhaseChange

Solver for two incompressible, immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, and Hardt and Wondra model [1,2] for phase change



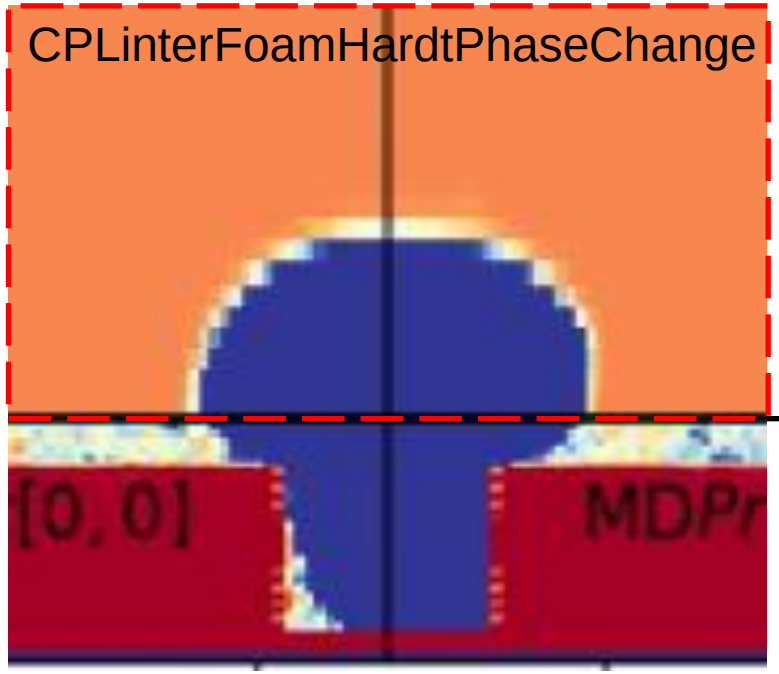

$$(\boldsymbol{U}, \rho, \alpha, T) = (\boldsymbol{U}, \rho, \alpha, T)_{MD}$$

```
97          // MPI_Init is called somewhere in the PStream library
98          if (coupled)
99            CPL.initCFD(runTime, mesh);
100
101         if (!LTS)
102         {
103             #include "CourantNo.H"
104             #include "setInitialDeltaT.H"
105         }
106
107         // * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
108
109         // Initial communication to initialize domains
110         if (coupled){
111           CPL.pack(U, p, nu, mesh, CPL.VEL);
112           CPL.send();
113           CPL.recvVelocityPressure();
114           CPL.unpackVelocityVOF(U, alpha1, alpha2,
115                                 rho1, rho2,
116                                 T, mesh);
117         }
118
119         Info<< "\nStarting time loop\n" << endl;
120
121         while (runTime.run())
122         {
```


[1] S. Hardt and F. Wondra, J. Computational Physics 227 (2008) 5871.
[2] F. Municchi et al., Int J Heat Mass Transf 195 (2022) 123166.

# CPLinterFoamHardtPhaseChange

Solver for two incompressible, immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, and Hardt and Wondra model [1,2] for phase change




$$(\boldsymbol{U}, \rho, \alpha, T) = (\boldsymbol{U}, \rho, \alpha, T)_{MD}$$

```
141         while (pimple.loop())
142         {
143             if (pimple.firstIter() || moveMeshOuterCorrectors)
144             {
145                 mesh.update();
146
147                 if (mesh.changing())
148                 {
149                     // Do not apply previous time-step mesh compression flux
150                     // if the mesh topology changed
151                     if (mesh.topoChanging())
152                     {
153                         talphaPhi1Corr0.clear();
154                     }

183             if (coupled){
184               CPL.pack(U, p, nu, mesh, CPL.VEL);
185               //CPL.pack(U, p, nu, mesh, CPL.STRESS);
186               CPL.send();
187               CPL.recvVelocityPressure();
188               CPL.unpackVelocityVOF(U, alpha1, alpha2,
189                                     rho1, rho2,
190                                     T, mesh);
191             }
192
193             #include "alphaControls.H"
194             #include "alphaEqnSubCycle.H"
195
196             mixture.correct();
```

[1] S. Hardt and F. Wondra, J. Computational Physics 227 (2008) 5871.
[2] F. Municchi et al., Int J Heat Mass Transf 195 (2022) 123166.