User-manual for COOLFluiD Aerothermodynamics (version 2013.05)

Andrea Lani, lani@vki.ac.be
Von Karman Institute, Aeronautics & Aerospace Dept.

Introduction

The COOLFluiD platform (`http://coolfluidsrv.vki.ac.be/trac/coolfluid`) [2, 3, 4, 9, 10] is an object-oriented framework for high performance computing on unstructured grids and is the main in-house computational tool for CFD applications at the Von Karman Institute. COOLFluiD consists of a set of plug-in libraries that can be linked at run-time to a kernel where the basic parallel data structure and interface functionalities are defined. The platform is currently able to handle complex multi-physics simulations with a wide range of spatial discretization algorithms and time marching methods, both explicit and implicit. Linear systems arising from Newton linearizations of the space/time residual are efficiently solved in parallel with dedicated software packages that have been interfaced within COOLFluiD as plug-in libraries. Thanks to an extremely modular and scalable design, different functionalities (and corresponding library components) can be easily combined together to get more complex capabilities which will, in most cases, automatically work both serial and in parallel. Some of the available features offered by the COOLFluiD software environment are:

- **multiple space discretizations:** Cell Centered Finite Volume (FV), Residual Distribution (RD) Finite Element (FE), Spectral Finite Volume (SV), Spectral Finite Difference (SD), Discontinuous Galerkin (DG);
- **multiple time integration schemes:** Runge Kutta, 1- and 3-point Backward Euler, Crank-Nicholson, Time limited schemes;
- **multiple parallel linear system solvers:** PETsc, Trilinos, SAMG, Pardiso, jacobian free methods;
- **parallel infrastructure:** parallel I/O capabilities and domain decomposition;
- **multiple physical models:** Ideal Magneto Hydro Dynamics (MHD), RANS ($k-\omega$, SST, BSL, Spalart-Allmaras models), Linear Elasticity, Heat Transfer, Compressible and Incompressible high-enthalpy flows in thermo-chemical nonequilibrium (TCNEQ) or in Local Thermodynamic Equilibrium (LTE) (with fixed or variable elemental fractions), Aeroacustics (LEE), LES;
- **algorithms for loosely coupled multi-domain multi-physics simulations**: different numerical methods applied to different models on full-non matching unstructured meshes for Aeroelasticity, Aerothermoelasticity, Conjugate heat transfer, etc.
- **ALE formulation for unsteady simulations on moving meshes**: high quality parallel mesh movement and deformation algorithms.

The COOLFluiD platform has been **designed to grow according to the needs** and has demonstrated its flexibility by incorporating progressively more and more complex algorithms and physical models. Some key strong points are:

- **Flexible data-structure** that allows to implement complex numerical methodologies and equations and get them automatically working in parallel (all the described functionalities work in parallel!) to take full profit of computational power. The extreme flexibility is proved by the fact that is hard to find another software package, commercial or not, that can offer such a variety of space discretization algorithms (each one with its own data-structure) working in parallel *within the same framework*.
- **Scalable design** that allows researchers to integrate new functionalities by fully reusing existing ones and take immediate profit of others' work. As an example, physicists can work in the Mutation library to refine the transport and thermodynamic modeling and numericists can improve the numerical algorithms independently one from the other, or with extremely limited interaction. At the end, the functionalities work both separately, as new independent framework components, and together.
- **Multi-physics** simulations can be customized at will, reusing the available components or incrementally implement new ones.
- **State-of-the-art numerical algorithms** and **extremely advanced physical modeling** for high-enthalpy flow properties (rigorously derived from kinetic theory or statistical and quantum mechanics) are combined together.
- Possibility of accurately simulating high-enthalpy flows **from incompressible regime to hyper-velocity** (up to Mach 40 or more).

Aerothermodynamic solvers

As far as the simulation of aerothermodynamics is concerned (see [4, 7] for technical details), COOLFluiD offers:

- 2D / axisymmetric / 3D FV solver for thermo-chemical equilibrium (LTE with or without demixing effect) and NEQ (different multi-temperature models) viscous flows on unstructured hybrid grids with various schemes (AUSM family, HUS, Roe, modified Steger-Warming, etc.).
- The same solver can also handle incompressible inductively coupled plasmas in LTE (extension to thermo-chemical NEQ is underway) where the Navier-Stokes equations are weakly coupled with the electro-magnetic induction equations.
- COOLFluiD uses the Mutation library (version 2.0) for the accurate computation of thermodynamic, transport and chemical kinetics properties in all temperature regimes, with different LTE and thermo-chemical NEQ models, including pioneering collision-radiative models with > 100 chemical species [6].
- Simulations on neutral or ionized mixtures of argon, air, CO_2, nitrogen is available.
- A new generation Residual Distribution solver for improving accuracy of thermo-chemical NEQ flows simulations on unstructured simplex-element meshes (with triangles or tetrahedra) is currently under development [4].
- Possibility of reusing all the available coupling algorithms to get arbitrarily complex multi-physics steady or unsteady simulations on deforming meshes [10].

1 COOLFluiD Aerothermodynamics (Finite Volume)

1.1 Installation instructions

Detailed installation instructions are available online at

`http://coolfluidsrv.vki.ac.be/trac/coolfluid`

Interested users must first contact the project administrator (`lani@vki.ac.be`) to get a username and therefore access the source code. The latter is currently undergoing a major restructuring and will be released fully open source in the future.

1.1.1 Typical installation instructions

A typical installation procedure consists in the following steps:

1. Downloading the installation script with

   ```
   svn co https://coolfluidsrv.vki.ac.be/svn/coolfluid/Scripts/install-coolfluid-deps.pl
   ```

2. Running the script to install dependencies:

   ```
   ./install-coolfluid-deps.pl --tmp-dir=TDIR --install-dir=LDIR --install-mpi-dir=MDIR
   ```

   where `TDIR` is the full path to the directory where dependencies files will be unpacked, `LDIR` is the installation directory and `MDIR` is the directory where the Message Passing Interface (MPI) libraries will be installed. PETSc and ParMetis libraries will be installed inside the MPI directory. Different MPI installations can coexist: the user-defined *coolfluid.conf* file, the `PATH` and `LD_LIBRARY_PATH` environmental variables will decide which actual installation to use.

3. Updating environmental variables to include the appropriate paths to the dependency libraries in the `.bashrc`:

   ```
   export PATH=LDIR/bin:MDIR/bin:$PATH
   export LD_LIBRARY_PATH=LDIR/lib:MDIR/lib:MDIR/petsc/lib:$LD_LIBRARY_PATH
   ```

4. Checking out the COOLFluiD kernel source files:

   ```
   svn co https://coolfluidsrv.vki.ac.be/svn/coolfluid/Sources/Kernel/trunk coolfluid
   ```

5. Setting up the file *coolfluid.conf* with the appropriate `coolfluid_dir`, `basebuild_dir`, `install_dir`, paths to all dependencies (check example files in `coolfluid/tools/conf`) and modules to download.

6. Checking out the selected modules with

   ```
   ./prepare.pl --config-file=coolfluid.conf --mods-update
   ```

7. Generating the build (make) files in `basebuild_dir` in *debug* (full debugging options, very slow), *optim* (some debug, some optimization, **recommended**) or *release* (no debugging, full optimization) mode:

   ```
   ./prepare.pl --config-file=coolfluid.conf --build=optim
   ```

8. Compiling (typically on multiple cores, 4 in our example) and creating all COOLFluiD libraries:

   ```
   cd basebuild_dir ; make -j4 ; make install
   ```

9. Setting the appropriate paths to the COOLFluiD libraries in the `.bashrc`:

   ```
   export PATH=install_dir/bin:$PATH
   export LD_LIBRARY_PATH=install_dir/lib:$LD_LIBRARY_PATH
   ```

WATCH OUT: *Steps 1 and 2 are only needed if dependency libraries are not installed in your system yet. Internal users at the VKI do not need those steps, since public installations are available. In particular, lammpi, openmpi and mpich2 are all supported at the VKI.*

WATCH OUT: *Using "make install" in step 8 is not necessary if you choose to use soft links to the coolfluid-solver executable located in* `basebuild_dir/src/Solver` *directory.*

1.2 How to run

The command line to run COOLFluiD is

```
mpirun -np N ./coolfluid-solver --scase ./myfile.CFcase
```

from inside the testcase directory. The parameter N must be replaced by the number of processors. The format of the input file (called `myfile.CFcase` in our example) is described here after. For a serial run (N=1) the user can also use:

```
./coolfluid-solver --scase ./myfile.CFcase
```

WATCH OUT: *In order to be able to run successfully, a soft link to (or copy of) the* `coolfluid-solver` *executable and file* `coolfluid-solver.xml` *(providing useful info on the path to the COOLFluiD shared libraries) must be present in the working directory.*

1.3 Configuration file description

The format of the input file (with extension .CFcase) consists of lines in the form `KEY = VALUE`:

```
Simulator.OptionA = Value1               # use Value1 as value for OptionA
Simulator.Value1.OptionB = Value2        # use Value2 as value for OptionB
Simulator.Value1.Value2.OptionC = Value3 # use Value3 as value for OptionC
Simulator.Value1.Value2.OptionD = Value4 # use Value4 as value for OptionD
```

where, in each line, the whole LHS is the keyword and the RHS is the value. The latter, depending of the actual types defined in the code for each configurable parameter, can be:

- an alpha-numerical string
- an integer
- a boolean (`true` or `false`)
- a floating point number
- an arbitrarily complex analytical function
- an array of all the previous.

The keyword is composed of literal strings separated by ".", corresponding to different *entities* (configurable objects or parameters) defined inside the actual code. The configuration is hierarchical and recursive from top to lowest level. **The order in which the options are declared in the file are irrelevant**. If needed, the value can be broken into different lines by using the continuation character (back slash) at the end of each line (note that the number of spaces at the end or before the line is irrelevant):

```
Simulator.Example.arrays = 4 4 10 \
                           10 4 \
                           4 3
```

Comments start with "#": they can occupy full lines or be placed at the end of the line.

1.4 Environment

```
CFEnv.ErrorOnUnusedConfig = true
```

If activated this option makes the simulation crash if there are spelling mistakes in the given options. This option must always be inactivated when starting from a Gambit file (see below).

```
Simulator.Modules.Libs = libCFmeshFileReader libNavierStokes libFiniteVolume ...
```

List of the COOLFluiD dynamic libraries needed for the present simulation. In the following description, each section will indicate the required libraries whenever applicable.

```
Simulator.Paths.WorkingDir = ./
Simulator.Paths.ResultsDir = ./RESULTS
```

Paths to the working directory and to the directory where output files (convergence history, Tecplot files, CFmesh files) should be written.

1.5 Interactive file

Some parameters can be changed interactively during the simulation by editing a separate file where the full option setting (key and value) has to be present.

```
Simulator.SubSystem.InteractiveParamReader.FileName = ./out.inter
```

tells the path to the interactive file and

```
Simulator.SubSystem.InteractiveParamReader.readRate = 10
```

specifies how often the file should be read by the solver in order to update the corresponding interactive parameters.
WATCH OUT: *In a parallel run, this rate must be defined with a safe margin (depending of the speed of the iterative process), allowing the user to quickly edit, modify and close the file before the solver tries to read the file as well.*

1.6 Physical Model

Required libs: libNavierStokes, libNEQ.

```
Simulator.SubSystem.Default.PhysicalModelType = NavierStokes2DNEQ
```

defines a generic 2D thermo-chemical nonequilibrium model.

```
Simulator.SubSystem.NavierStokes2DNEQ.refValues = \
  1e-12 1e-6 1e-6 0.00002854 1e-6 0.00000866 1e-6 1e-6 \
  1e-6 1e-6 1e-6 11360. 1000. 195. 195.
```

provides values of the order of the free stream quantities for all stored variables (see `.updateVar` below), one per equation.
WATCH OUT: *None of those values can be zero, since they are actually used as denominator in scaling for numerical finite difference while computing numerical jacobians.*

```
Simulator.SubSystem.NavierStokes2DNEQ.nbSpecies = 11
Simulator.SubSystem.NavierStokes2DNEQ.nbEulerEqs = 3
Simulator.SubSystem.NavierStokes2DNEQ.nbVibEnergyEqs = 1
```

specify the number of chemical species, the total number of equations excluding species continuity and vibrational/electronic equations (it must be 3 for 2D cases, 4 for 3D cases), the number of vibrational energy equations.

1.6.1 Mutation 2.0

Required libs: libMutation2OLD, libMutation2OLDI.

```
Simulator.SubSystem.NavierStokes2DNEQ.PropertyLibrary = Mutation2OLD
```

specifies Mutation 2.0.0 (slower but stable version of Mutation 2.0) [5, 7] as the physico-chemical library for computing thermodynamic, transport, chemical kinetics properties. This version of Mutation supports arbitrary chemical mixtures (neutral and ionized), chemical equilibrium models with fixed and variable elemental fractions, thermal and chemical nonequilibrium multi-temperature models, including full and reduced Collisional Radiative (CR) models for air [8].

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.mixtureName = air11
```

specifies the name of the mixture corresponding to a file `air11.mix` defined inside `PATH_TO_MUTATION/Mutation2.0.0I/data/mixture`.

WATCH OUT: *The mixture file specifies the order of the chemical species as they are used and stored by the flow solver.*

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.reactionName = parkair93
```

specifies the name of the chemical reactions model corresponding to a file `parkair93` defined inside `PATH_TO_MUTATION/Mutation2.0.0I/data/chemistry/gasreact`.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.transfName = air11
```

specifies the name of the energy transfer model corresponding to a file `air11` defined inside `PATH_TO_MUTATION/Mutation2.0.0I/data/chemistry/transfer`.

WATCH OUT: *The detailed description of the format for Mutation data files is out of the scope of this tutorial. The user is referred to the Mutation manual (contact* `magin@vki.ac.be`*) instead.*

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.TminFix = 100.
```

defines minimum temperature allowed inside Mutation routines.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.dynViscAlgo = CG
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.thermCondAlgo = Direct
```

specifies the transport algorithms to use for the computation of the dynamic viscosity and the thermal conductivity. Those settings are the most stable and should not be changed.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.includeElectronicEnergy = true
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.electrEnergyID = 0
```

Those options should always be activated when running ionized cases and deactivated otherwise.

```
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.path = \
   /data1/andrea/COOLFLUID/plugins/Mutation2.0.0I/
```

provides the full path (no environmental variables are allowed here!) to the Mutation 2.0 installation.

1.7 Output Format

Required libs: libCFmeshFileWriter, libTecplotWriter.

```
Simulator.SubSystem.OutputFormat    = Tecplot CFmesh
```

defines list of requested output files (only `Tecplot` and `CFmesh` are supported for nonequilibrium flows).

```
# in parallel runs, a file out-P0.CFmesh is written
Simulator.SubSystem.CFmesh.FileName = out.CFmesh

# every how many iterations the file is saved
Simulator.SubSystem.CFmesh.SaveRate = 1000

# append iteration number to the file name
Simulator.SubSystem.CFmesh.AppendIter = true
```

specifies the settings for the CFmesh file format (the internal format of COOLFluiD, including both mesh and solution). Only one CFmesh file is written even in a parallel simulation.

```
# in parallel runs, one file per processor out-P*.plt is written
Simulator.SubSystem.Tecplot.FileName       = out.plt

# output variables name (RhoivtTv correspond to [rho_i v T T_v])
Simulator.SubSystem.Tecplot.Data.outputVar = RhoivtTv

# write also density, total enthalpy, Mach number and pressure
Simulator.SubSystem.Tecplot.Data.printExtraValues = true

# name of the boundary patch for which a file out-P*-surf.plt will be saved
Simulator.SubSystem.Tecplot.Data.SurfaceTRS = Wall

# every how many iterations the file is saved
Simulator.SubSystem.Tecplot.SaveRate = 1000

# append iteration number to the file name
Simulator.SubSystem.Tecplot.AppendIter = false
```

specifies the settings for the Tecplot file format. One file per processor will be written.

1.8 Stop Condition

The simulation can be stopped by prescribing a maximum number of steps:

```
Simulator.SubSystem.StopCondition = MaxNumberSteps
Simulator.SubSystem.MaxNumberSteps.nbSteps = 2
```

of by looking at the norm of the residual

```
Simulator.SubSystem.StopCondition       = Norm
Simulator.SubSystem.Norm.valueNorm      = -3.0
```

A threshold of $<= -3.$ is reasonably good for most cases, if the temperature is used as variable to monitor (see MonitoredVarID below).

1.9 Mesh Reader

Required libs: libCFmeshFileReader.

```
Simulator.SubSystem.Default.listTRS = InnerFaces Wall Symmetry Inlet Outlet
```

specifies the list of all Topological Region Sets (TRS), i.e. the boundary patches as defined in the mesh file. `InnerFaces` dose not need to be included in the list.

```
Simulator.SubSystem.MeshCreator = CFmeshFileReader
Simulator.SubSystem.CFmeshFileReader.Data.FileName = ./input.CFmesh
```

specify the reading from a file called `Restart.CFmesh` in CFmesh format.

```
Simulator.SubSystem.CFmeshFileReader.Data.ScalingFactor = 1000.
```

specifies a factor for which the input mesh must be *divided* (the name "Scaling" is misleading here).
WATCH OUT: *the scaling factor must be used only when starting from scratch and not from a CFmesh file containing the solution.*

```
Simulator.SubSystem.CFmeshFileReader.ParReadCFmesh.ParCFmeshFileReader.NbOverlapLayers = 2
```

This option is obsolete, but kept for compatibility with older versions of the code. It specifies the number of overlap layers in parallel computations. This value should be 2 for second order calculations, but it is now automatically calculated.

1.9.1 Reading from Gambit files

Required libs: libGambit2CFmesh.

If the mesh file is not yet in CFmesh format and it's coming from the ANSYS Gambit mesh generator, the following settings must e defined:

```
Simulator.SubSystem.CFmeshFileReader.convertFrom = Gambit2CFmesh
Simulator.SubSystem.CFmeshFileReader.Gambit2CFmesh.Discontinuous = true
Simulator.SubSystem.CFmeshFileReader.Gambit2CFmesh.SolutionOrder = P0
```

In this case the solver expects a file called `input.neu` placed inside the working directory.
WATCH OUT: *all Gambit settings must be commented out when restarting from a previous CFmesh solution (see Restart option).*

1.10 Convergence Method

Required libs: libNewtonMethod.

We consider here only the steady implicit time stepping case, corresponding to a first-order accurate Backward Euler integration.

```
Simulator.SubSystem.ConvergenceMethod = NewtonIterator

# this value must be > 1 only for unsteady simulations
Simulator.SubSystem.NewtonIterator.Data.MaxSteps = 1
```

The CFL parameter which controls the stability of the calculation can be specified in two ways: interactively or with a user-defined function.

```
Simulator.SubSystem.NewtonIterator.StdUpdateSol.Relaxation = 1.0
```

provides a relaxation parameter (<= 1.0): a single value for all equations or an array of values (with size equal to the number of equations). This typically can be kept equal to 1.

```
Simulator.SubSystem.NewtonIterator.Data.L2.MonitoredVarID = 13
```

indicates the ID of the variable to be monitored for convergence (see StopCondition). In chemically reacting flows, the ID corresponding to the temperature (total energy equation) is recommended.

```
Simulator.SubSystem.NewtonIterator.Data.L2.ComputedVarID = 13
```

indicates the ID of the variable whose norm will be computed and written to screen. If this line is commented out, residuals for all variables will be computed.

```
Simulator.SubSystem.NewtonIterator.Data.FilterState = Max

# flags (0 or 1) to tell which variables must be clipped
Simulator.SubSystem.NewtonIterator.Data.Max.maskIDs = \
 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1

# real values to tell the minimum values to be imposed for each variable
# only values given for flagged variables (maskID = 1) will be clipped
Simulator.SubSystem.NewtonIterator.Data.Max.minValues = \
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

provide some filtering of the solution values before the solution update.

1.10.1 Interactive CFL

```
Simulator.SubSystem.NewtonIterator.Data.CFL.ComputeCFL = Interactive
```

declares the CFL interactive and its value will be read from the interactive file (`out.inter` in our example). In this case the line

```
Simulator.SubSystem.NewtonIterator.Data.CFL.Interactive.CFL = 10.0
```

must be present and, if needed, modified in the interactive file.

1.10.2 Function CFL

```
Simulator.SubSystem.NewtonIterator.Data.CFL.Value = 1.0
Simulator.SubSystem.NewtonIterator.Data.CFL.ComputeCFL = Function
Simulator.SubSystem.NewtonIterator.Data.CFL.Function.Def = \
  if(i<1000,1.0,if(i<2000.,1.01*cfl, min(1200.,1.05*cfl)))
```

In order to automatize the iterative process, a function specifying an arbitrarily complex CFL law can be provided. The variable that can appear in this expression are: `i` (iteration number), `cfl` (previous CFL value), `r` (current residual), `ri` (initial residual), `rl` (last residual), `rmax` (maximum residual).

WATCH OUT: *no spaces are allowed within the expression and the supported operators and mathematical functions are indicated in [1].*

1.11 Linear System Solver

Required libs: libPetsc.

We include here only the settings corresponding to the PETSC linear system solver library, even though Trilinos is also interfaced within COOLFluiD.

```
Simulator.SubSystem.LinearSystemSolver = PETSC
Simulator.SubSystem.LSSNames = NewtonIteratorLSS
Simulator.SubSystem.NewtonIteratorLSS.Data.PCType = PCASM
Simulator.SubSystem.NewtonIteratorLSS.Data.KSPType = KSPGMRES
Simulator.SubSystem.NewtonIteratorLSS.Data.MatOrderingType = MATORDERING_RCM
```

specify the basic settings for a GMRES solver combined with a parallel Additive Schwartz preconditioner.

```
Simulator.SubSystem.NewtonIteratorLSS.Data.MaxIter = 1000
```

defines the maximum allowed number of GMRES iterations: should be typically kept $<= 1000$.

```
Simulator.SubSystem.NewtonIteratorLSS.Data.RelativeTolerance = 1e-4
```

defines the relative tolerance for the GMRES solver: $1e-4$ or $1e-3$ are recommended values, since with higher values convergence can be very slow and requiring many more GMRES iterations per time step.

1.12 Space Method

Required libs: libFiniteVolume, libNavierStokes, libFiniteVolumeNavierStokes, libNEQ, libFiniteVolumeNEQ.

The following standard settings for the implicit Finite Volume solver should not be changed:

```
Simulator.SubSystem.SpaceMethod = CellCenterFVM
Simulator.SubSystem.CellCenterFVM.ComputeRHS = NumJacobFast
Simulator.SubSystem.CellCenterFVM.NumJacobFast.FreezeDiffCoeff = true
Simulator.SubSystem.CellCenterFVM.ComputeTimeRHS = PseudoSteadyTimeRhs
```

In order to restart from a previous CFmesh file (with saved solution), the following option must be set to true, otherwise must be commented out.

```
Simulator.SubSystem.CellCenterFVM.Restart = true
```

1.13 Numerical schemes for computing convective fluxes in NEQ

The user must choose one of the following (AUSM+ is recommended for most cases):

```
# AUSM+ is the most stable and works for all equilibrium and NEQ models
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = AUSMPlusMS2D
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusMS2D.choiceA12 = 5

# AUSM+up flux works for all equilibrium and nonequilibrium models
# it includes some built-in preconditioning to handle low Mach flows
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = AUSMPlusUpMS2D
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusUpMS2D.choiceA12 = 5
# the free stream Mach number must be specified
Simulator.SubSystem.CellCenterFVM.Data.AUSMPlusUpMS2D.machInf = 30.

# HUS flux works for all NEQ models but is generally less stable and carbuncle prone
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = HUSMS2D
Simulator.SubSystem.CellCenterFVM.Data.HUSMS2D.isNatural = true

# Roe scheme (Sanders' carbuncle fix) works only for neutral mixtures
# in thermo-chemical NEQ
Simulator.SubSystem.CellCenterFVM.Data.FluxSplitter = RoeTCNEQ2DSA
Simulator.SubSystem.CellCenterFVM.Data.RoeTCNEQ2DSA.entropyFixID = 1  #2 or 3
Simulator.SubSystem.NavierStokes2DNEQ.Mutation2OLD.noElectronicEnergy = true
```

1.13.1 Variable sets for NEQ

The following settings define some variable that are needed in different phases of the simulation.

```
# variables in which the solution is stored and updated
# (use Rhoivt for thermal equilibrium)
Simulator.SubSystem.CellCenterFVM.Data.UpdateVar = RhoivtTv

# variables in which the equations are formulated must ALWAYS be Cons
Simulator.SubSystem.CellCenterFVM.Data.SolutionVar = Cons

# variables in which the diffusive fluxes are computed
# (use Rhoivt for thermal equilibrium)
Simulator.SubSystem.CellCenterFVM.Data.DiffusiveVar = RhoivtTv

# diffusive flux must be NavierStokes for viscous computations
Simulator.SubSystem.CellCenterFVM.Data.DiffusiveFlux = NavierStokes
```

1.13.2 2D and axisymmetric settings for NEQ

```
Simulator.SubSystem.CellCenterFVM.Data.isAxisymm = true
```

should always be present in 2D axisymmetric computations.

The following options must be activated only for axisymmetric calculations in thermo-chemical NEQ:

```
# source terms for the axisymmetric case
Simulator.SubSystem.CellCenterFVM.Data.SourceTerm = \
  NavierStokes2DTCNEQAxiST Euler2DCTNEQST

# IDs corresponding to the velocity variables (momentum equations)
Simulator.SubSystem.CellCenterFVM.Data.NavierStokes2DTCNEQAxiST.uvIDs = 11 12
```

The following options must be activated only for axisymmetric calculations in chemical NEQ (thermal equilibrium):

```
# source terms for the axisymmetric case
Simulator.SubSystem.CellCenterFVM.Data.SourceTerm = \
  NavierStokes2DNEQAxiST Euler2DCNEQST

# IDs corresponding to the velocity variables (momentum equations)
Simulator.SubSystem.CellCenterFVM.Data.NavierStokes2DNEQAxiST.uvIDs = 11 12
```

WATCH OUT: *In 2D non-axisymmetric cases, only* `Euler2DCTNEQST` *or* `Euler2DCNEQST` *should be declared as source term.*

1.13.3 Polynomial reconstruction

The following options should be kept frozen:

```
Simulator.SubSystem.CellCenterFVM.SetupCom = LeastSquareP1Setup
Simulator.SubSystem.CellCenterFVM.SetupNames = Setup1
Simulator.SubSystem.CellCenterFVM.Setup1.stencil = FaceVertexPlusGhost
Simulator.SubSystem.CellCenterFVM.UnSetupCom = LeastSquareP1UnSetup
Simulator.SubSystem.CellCenterFVM.UnSetupNames = UnSetup1
Simulator.SubSystem.CellCenterFVM.Data.PolyRec = LinearLS2D
Simulator.SubSystem.CellCenterFVM.Data.Limiter = Venktn2D
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.coeffEps = 1.0
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.useFullStencil = true
# set true the following for backward compatibility, but false should behave better
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.useNodalExtrapolationStencil = false
Simulator.SubSystem.CellCenterFVM.Data.Venktn2D.length = 1.0
```

The following factor determines if the simulation is of first, second or in-between order:

```
# 0 <= gradientFactor <= 1, with  0. (first order), 1. (second order)
Simulator.SubSystem.CellCenterFVM.Data.LinearLS2D.gradientFactor = 0.
```

This is an interactive parameter that can be placed into the interactive file. Another interactive parameter, important for second order computations, is

```
Simulator.SubSystem.CellCenterFVM.Data.LinearLS2D.limitRes = -4.0
```

This corresponds to the minimum residual at which the freezing of the flux limiter should be applied for flows exhibiting discontinuities or steep gradients (e.g., in temperature). In practice, `limitRes` can be kept at -4 till when the simulation reaches a limit cycle and then can be increased to 8. in order to exit the cycle. This cure is not always effective and it often depends on the moment when `limitRes` is increased.
WATCH OUT: *Before restarting a simulation from a second order solution, if the limiter has not been explicitly saved in the CFmesh file (see below),* `limitRes` *has to be set back to* -4.

In order to save the limiter in second order calculations (when `gradientFactor = 1.`), the following options must be added to the CFcase **before** starting the computation:

```
Simulator.SubSystem.CFmesh.Data.ExtraStateVarNames = limiter

# the following must be the total number of equations
Simulator.SubSystem.CFmesh.Data.ExtraStateVarStrides = 15
```

Finally, in order to restart from a file in which the limiter **has been already saved**, the following line should be included in the CFcase file:

```
Simulator.SubSystem.CFmeshFileReader.Data.ExtraStateVarNames = InitLimiter
```

1.13.4 Initial Field

The initial field conditions are typically prescribed on the full domain with a set of user-defined analytical functions depending on the position vector (x,y,z) [1], one for each of the update variables (`RhoivtTv` in our current example).

The following settings are always required:

```
# "InitState" is the name of the object implementing an initial state
Simulator.SubSystem.CellCenterFVM.InitComds = InitState

# "InField" is a user-defined alias that will be used to configure InitState
Simulator.SubSystem.CellCenterFVM.InitNames = InField

# from now on, only "InField" is used for the initialization settings
# "InnerFaces" is the boundary patch (TRS) on which InField is active
Simulator.SubSystem.CellCenterFVM.InField.applyTRS = InnerFaces
```

The following settings define the analytical functions.

```
# independent variables (use "x y z" in 3D)
Simulator.SubSystem.CellCenterFVM.InField.Vars = x y

# arbitrarily complex function definitions (one per update variable, 15 in this case)
# no space allowed within a single function
# NOTE: this is just an illustrative example with no physical sense!
Simulator.SubSystem.CellCenterFVM.InField.Def = \
 0. 0. 0. if(sqrt(x^2+y^2)<1.,0.00002854,0.00002854/2.) \
 0. if(sqrt(x^2+y^2)<1.,0.00000866,0.00000866/2.) 0. 0. 0. 0. 0.
 if(x<1.0,11360.,100.*(sqrt(x^2+y^2)-1.)) 0. 195. 195.
```

If analytical expressions are particularly complex, the user can use a more advance 2-step initializer. The previous example can be simplified as:

```
# "InitStateAddVar" is used instead of "InitState"
Simulator.SubSystem.CellCenterFVM.InitComds = InitStateAddVar
Simulator.SubSystem.CellCenterFVM.InitNames = InField
Simulator.SubSystem.CellCenterFVM.InField.applyTRS = InnerFaces

Simulator.SubSystem.CellCenterFVM.InField.InitVars = x y
Simulator.SubSystem.CellCenterFVM.InField.InitDef = sqrt(x^2+y^2)

# here "rad" is a new user-defined variable that can be used
# to simplify the final expressions
Simulator.SubSystem.CellCenterFVM.InField.Vars = x y rad
Simulator.SubSystem.CellCenterFVM.InField.Def = \
 0. 0. 0. if(rad<1.,0.00002854,0.00002854/2.) \
 0. if(rad<1.,0.00000866,0.00000866/2.) 0. 0. 0. 0. 0.
 if(x<1.0,11360.,100.*(rad-1.)) 0. 195. 195.
```

1.13.5 Boundary Conditions

Boundary conditions fields will be applied also during initialization on the corresponding boundary TRS, in such a way that *ghost states* (dummy cell centers that lie outside the computational domain) are set consistently before starting computing numerical fluxes.
The following example shows how to specify a full set of boundary conditions (four in this case, but real settings will obviously depend on the mesh in use).

```
# list of the names of the objects defining each boundary condition
Simulator.SubSystem.CellCenterFVM.BcComds = \
    NoSlipWallIsothermalNSrvtMultiFVMCC \
    MirrorVelocityFVMCC \
    SuperInletFVMCC \
    SuperOutletFVMCC

# list of aliases that the use must define for configuring each BC
Simulator.SubSystem.CellCenterFVM.BcNames = NSWall Mirror SInlet SOutlet
```

Noslip wall (isothermal, adiabatic or with radiative equilibrium)

```
# apply NoSlipWallIsothermalNSrvtMultiFVMCC to the Wall
# (TRS name coming from the initial mesh file)
Simulator.SubSystem.CellCenterFVM.NSWall.applyTRS = Wall

# if an adiabatic condition is needed set this flag to true
Simulator.SubSystem.CellCenterFVM.NSWall.Adiabatic = false

# imposed wall temperature
Simulator.SubSystem.CellCenterFVM.NSWall.TWall = 615.0

In order to impose a radiative equilibrium condition, additional options are needed:
\begin{verbatim}
Simulator.SubSystem.CellCenterFVM.NSWall.RadEquilibrium = true

# emissivity
Simulator.SubSystem.CellCenterFVM.NSWall.Emissivity = 0.9

# maximum allowable change in temperature between two consecutive time step
Simulator.SubSystem.CellCenterFVM.NSWall.MaxRadEqDTwall = 100.

# temperature of the distant body (typically 0 or free stream value)
Simulator.SubSystem.CellCenterFVM.NSWall.DistantBodyTemp = 0.
```

A super-catalytic wall condition, imposing LTE at the wall, can be imposed by replacing the BC object name `NoSlipWallIsothermalNSrvtMultiFVMCC` with `NoSlipWallIsothermalNSrvtLTEMultiFVMCC`.

Symmetry plane or slip wall

```
# apply MirrorVelocityFVMCC to the Symmetry TRS
Simulator.SubSystem.CellCenterFVM.Mirror.applyTRS = Symmetry

# IDs corresponding to the velocity components
# (they dependent on the chemical model in use)
Simulator.SubSystem.CellCenterFVM.Mirror.VelocityIDs = 11 12

# array of flags where "1" correspond to variables for which
# a zero gradient has to be imposed
Simulator.SubSystem.CellCenterFVM.Mirror.ZeroGradientFlags = \
  1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
```

Super inlet

```
# apply SuperInletFVMCC to the Inlet TRS
Simulator.SubSystem.CellCenterFVM.SInlet.applyTRS = Inlet

# analytical functions can be defined here as for InitState
# (2-step option is not available though)
Simulator.SubSystem.CellCenterFVM.SInlet.Vars = x y
Simulator.SubSystem.CellCenterFVM.SInlet.Def = \
 0. 0. 0. 0.00002854 0. 0.00000866 0. 0. 0. 0. 0. 11360. 0. 195. 195.
```

specify the settings for a supersonic inlet (all RhoivtTv variables have to be prescribed). Additional interactive parameters (to be put in the interactive file) can be used for running stiff cases (typically 3D), for which, for instance, it may be impossible to start with the full velocity:

```
# the following settings tell the solver to multiply the variable with
# ID = 11 (x-velocity) by a factor (must <= 1.0)
Simulator.SubSystem.CellCenterFVM.SInlet.InteractiveVarIDs = 11
Simulator.SubSystem.CellCenterFVM.SInlet.InteractiveFactor = 1.0
```

Super outlet

```
Simulator.SubSystem.CellCenterFVM.SOutlet.applyTRS = Outlet

# array of flags where "1" correspond to variables for which
# a zero gradient has to be imposed
Simulator.SubSystem.CellCenterFVM.SOutlet.ZeroGradientFlags = \
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

1.13.6 Nodal Extrapolation

Since flow solution is computed in the cell centers, nodal values must be extrapolated from cell centers to the mesh vertices for visualization purposes or for computing viscous gradients. This

is accomplished by `NodalExtrapolation` objects. In viscous NEQ cases, where a slip condition and, possibly, a temperature are imposed at the wall, the following settings must be added in order to strongly impose the desired values.

```
# this specifies the name of the nodal extrapolator object
Simulator.SubSystem.CellCenterFVM.Data.NodalExtrapolation = DistanceBasedGMoveRhoivt

# the name(s) of the boundary TRS(s) on which imposing values strongly
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.TRSName = Wall

# IDs of the variables to be imposed strongly on the boundaries listed in TRSName
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.ValuesIdx = \
  11 12 13 14

# values of the selected variables to be imposed strongly
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.Values = \
  0. 0. 615. 615.

# list determines the priority of one TRS (and BC) over another in corner nodes
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.TrsPriorityList = \
   Wall Symmetry Inlet Outlet
```

WATCH OUT: *Nodal extrapolation settings must be consistent with the boundary conditions (same velocity IDs, same temperature at the wall, etc.).*

Radiative equilibrium case When radiative equilibrium is imposed at the wall, besides specifying appropriate boundary condition settings (see above), the nodal extrapolation must also be adapted consistently, as follows.

```
# only velocity IDs and values must be prescribed at the wall nodes, since
# temperature will be computed on-the-fly by an iterative procedure
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.ValuesIdx = 11 12
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.Values = 0. 0.

# this flag must be activated
Simulator.SubSystem.CellCenterFVM.Data.DistanceBasedGMoveRhoivt.RadEquilibrium = true
```

Super-catalytic case (LTE at the wall) A super-catalytic BC requires setting `DistanceBasedGMoveRhoivtLTE` instead of `DistanceBasedGMoveRhoivt`.

1.14 Post-processing: radiation coupling and surface quantities computation

We declare two post-processing numerical commands.

```
# name of the post-processing objects
Simulator.SubSystem.DataPostProcessing = DataProcessing DataProcessing
```

```
# user-defined configuration names for the post-processing object
Simulator.SubSystem.DataPostProcessingNames = DataProcessing2 DataProcessing3
```

1.14.1 Radiation coupling

Required libs: libparade, libradiation, libParadeI, libRadiativeTransfer.

The following options control the postprocessing:

```
# how often the post-processing is applied (this is an interactive option,
# it can go to the interactive file)
Simulator.SubSystem.DataProcessing2.ProcessRate = 100

# flag telling if to skip the coupling for the first iteration (advised)
Simulator.SubSystem.DataProcessing2.SkipFirstIteration = true
```

The following options allow for solving the radiation transport on the stagnation line (works only in serial mode) using PARADE or to solve optically thin in the whole domain.

```
# name of the command object implementing the radiation coupling
# Slab1DFVMCC corresponds to 1D infinite tangent slab method
Simulator.SubSystem.DataProcessing2.Comds = Slab1DFVMCC

# user-defined configuration name for "Slab1DFVMCC"
Simulator.SubSystem.DataProcessing2.Names = Radiation1D

# wall TRS boundary required by the algorithm
Simulator.SubSystem.DataProcessing2.Radiation1D.applyTRS = Wall

# if this option is present, providing the x,y coordinates of the stagnation point,
# the RTE will be solved ONLY on the stagnation line
# this setting should be commented out in case all mesh lines are needed (not supported)
Simulator.SubSystem.DataProcessing2.Radiation1D.StagnationPoint = 0. 0.

# radiation library (only Parade is available at the moment)
Simulator.SubSystem.DataProcessing2.Radiation1D.RadiationLibrary = Parade

# flag telling if the flow is emission dominated (optically thin assumption)
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.EmisDom = true

# temperature to impose on the wall boundary
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.Tb1 = 615.0

# free stream temperature to impose on the inlet boundary
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.Tb2 = 195.0

# under-relaxation factor (must be <= 1)
Simulator.SubSystem.DataProcessing2.Radiation1D.Parade.UndRel = 1.0
```

The following options allow for solving the radiation transport with the **Monte-Carlo** algorithm and to couple back the computed $\nabla \cdot \mathbf{q}_{rad}$ to the flow energy conservation equation.

```
# name of the command object implementing the radiation transport via Monte-Carlo
Simulator.SubSystem.DataProcessing2.Comds = RadiativeTransferMonteCarloFVMCC

# user-defined configuration name for "RadiativeTransferMonteCarloFVMCC"
Simulator.SubSystem.DataProcessing2.Names = RT

# Name of the topological region where to apply the algorithm
Simulator.SubSystem.DataProcessing2.RT.applyTRS = InnerFaces

# Names of the topological regions corresponding to solid walls
Simulator.SubSystem.DataProcessing2.RT.wallTrsNames = Wall1 Wall2

# Names of the topological regions corresponding to symmetry planes
Simulator.SubSystem.DataProcessing2.RT.symmetryTrsNames = Symmetry

# Names of the topological regions corresponding to other boundaries
Simulator.SubSystem.DataProcessing2.RT.boundaryTrsNames = Inlet Outlet

# User-defined number of rays (photons) shot by each computational cell
Simulator.SubSystem.DataProcessing2.RT.numberOfRays = 20

# Maximum number of visited cell during ray tracing
Simulator.SubSystem.DataProcessing2.RT.MaxNbVisitedCells = 200

# Number of wavelengths to be considered to obtain a reduced spectra
Simulator.SubSystem.DataProcessing2.RT.ReducedSpectralSize = 500

# Wall emissivity
Simulator.SubSystem.DataProcessing2.RT.WallEmissivity = 0.

# Wall absorption coefficient
Simulator.SubSystem.DataProcessing2.RT.WallAbsorption = 0.644

# ID of the temperature in the state vector
# (typically = number of species + spatial dimension)
Simulator.SubSystem.DataProcessing2.RT.TemperatureID = 13

# Free stream temperature (if>0, it will impose q_rad=0 for the cell where active)
#Simulator.SubSystem.DataProcessing2.RT.FreeStreamTemperature = 640.

# Flag to tell to use the Planck function at the wall
# (alternative to specifying WallEmissivity)
#Simulator.SubSystem.DataProcessing2.RT.PlanckFunction = true
```

In axisymmetric cases, the following must be specified:

```
# Specify if to use the axisymmetric ray tracing
Simulator.SubSystem.DataProcessing2.RT.Axi = true

# Specify the number of cells in the streamwise direction
Simulator.SubSystem.DataProcessing2.RT.nCx = 80

# Specify the number of cells normal to the wall
Simulator.SubSystem.DataProcessing2.RT.nCy = 150
```

PARADE settings to be used in combination with Monte-Carlo:

```
# radiation library (only Parade is available at the moment)
Simulator.SubSystem.DataProcessing2.RT.RadiationLibrary = Parade

# flag telling if the flow is emission dominated (optically thin assumption)
Simulator.SubSystem.DataProcessing2.RT.Parade.EmisDom = false

# Minumum wavelength (must be consistent with "wavlo" declared in parade.con)
Simulator.SubSystem.DataProcessing2.RT.Parade.WavelengthMin = 2000.

# Maximum wavelength (must be consistent with "wavhi" declared in parade.con)
Simulator.SubSystem.DataProcessing2.RT.Parade.WavelengthMax = 40000.

# number of wavelengths for which radiative properties are computed at a time
# if < "npoints" declared in parade.con, Monte-Carlo will compute radiative properties
# for this number of wavelengths at once
Simulator.SubSystem.DataProcessing2.RT.Parade.WavelengthStride = 10000

# Path where PARADE is installed with all data files
Simulator.SubSystem.DataProcessing2.RT.Parade.path = /home/myuser/PARADEv3.1/parade31

# when set to 1, this allows for restarting from previously computed radiative properties
# as long as the number of processors remains the same as in the previous run
Simulator.SubSystem.DataProcessing1.RT.Parade.ReuseProperties = 0
```

The flow radiation cupoling is controlled by one interactive parameter (can be changed interactively inside the .inter file)

```
# RadRelaxationFactor = 0       uncoupled case
# 0 < RadRelaxationFactor <= 1  coupled case, this is an under-relaxation factor
Simulator.SubSystem.CellCenterFVM.Data.Euler2DCTNEQST.RadRelaxationFactor = 1.0
```

WATCH OUT: *In order to run the flow-radiation coupling the executable* `parade` *together with all required PARADE input files (parade.con) must be present in the working directory.*

1.15 Surface quantities

Required libs: libAeroCoefFVM, libAeroCoefFVMNEQ.

Surface quantities such as surface pressure, temperature, heat flux and skin friction can be computed and saved to **one single Tecplot file** with the following settings.

```
# how often the post-processing is applied (this is an interactive option,
# it can go to the interactive file)
Simulator.SubSystem.DataProcessing3.ProcessRate = 10

# name of the command object implementing the post-processing
Simulator.SubSystem.DataProcessing3.Comds = NavierStokesSkinFrictionHeatFluxCCNEQ

# user-defined configuration name for "NavierStokesSkinFrictionHeatFluxCCNEQ"
Simulator.SubSystem.DataProcessing3.Names = SkinFriction

# boundary TRS on which applying the post-process
Simulator.SubSystem.DataProcessing3.SkinFriction.applyTRS = Wall

# output Tecplot data file on which surface quantities will be written
Simulator.SubSystem.DataProcessing3.SkinFriction.OutputFileWall = walldata.plt

# ALL the following free stream values and update variable IDs MUST be specified
Simulator.SubSystem.DataProcessing3.SkinFriction.rhoInf = 0.0000372 # density
Simulator.SubSystem.DataProcessing3.SkinFriction.pInf = 2.1         # pressure
Simulator.SubSystem.DataProcessing3.SkinFriction.uInf = 11360.      # x-velocity
Simulator.SubSystem.DataProcessing3.SkinFriction.TInf = 195.        # temperature
Simulator.SubSystem.DataProcessing3.SkinFriction.UID = 11           # x-velocity ID
Simulator.SubSystem.DataProcessing3.SkinFriction.VID = 12           # y-velocity ID
Simulator.SubSystem.DataProcessing3.SkinFriction.TID = 13           # temperature ID
```

References

[1] J. Nieminen, J. Yliluoma. *Function Parser for C++*, http://warp.povusers.org/ FunctionParser/, 2009.

[2] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, *The COOLFluiD Framework - Design Solutions for High-Performance Object Oriented Scientific Computing Software*, International Conference Computational Science 2005, Atlanta (GA), LNCS 3514, Vol.1, pp. 281-286, Springer-Verlag, 2005.

[3] A. Lani, T. Quintino, D. Kimpe, H. Deconinck, S. Vandewalle and S. Poedts, *Reusable Object-Oriented Solutions for Numerical Simulation of PDEs in a High Performance Environment*, Scientific Programming. ISSN 1058-9244, Vol. 14, N. 2, pp. 111-139, IOS Press, 2006.

[4] A. Lani, *An Object Oriented and High Performance Platform for Aerothermodynamics Simulation*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2008.

[5] T. E. Magin, *A model for Inductive Plasma Wind Tunnels*, Ph.D. thesis, von Karman Institute, Rhode-Saint-Genèse, Belgium, 2004.

[6] A. Munafo, M. Panesi, R. Jaffe, A. Lani, T. Magin, *Vibrational State to State Kinetics in Expanding and Compressing Nitrogen Flows*, AIAA-2010-4335, 10th AIAA/ASME Joint Thermophysics and Heat Transfer Conference, Chicago, Illinois, June 28-July 1, 2010.

[7] M. Panesi, *Physical Models for Nonequilibrium Plasma Flow Simulations at High Speed Re-entry Conditions*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2009.

[8] M. Panesi, A. Lani and O. Chazot, *Reduced Kinetic Mechanism for CFD Applications*, AIAA-2009-3920, 41st AIAA Thermophysics Conference, San Antonio, Texas, June 22-25, 2009.

[9] T. L. Quintino. *A Component Environment for High-Performance Scientific Computing. Design and Implementation*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2008.

[10] Thomas Wuilbaut, *Algorithmic Developments for a Multiphysics Framework*, Ph.D. thesis,von Karman Institute, Rhode-Saint-Genèse, Belgium, 2008.