UMB

Norwegian University of Life Sciences

CFD with OpenSource software, project assignment

myIcoFsiFoam and myInterFsiFoam

Constructing solvers for weakly coupled FSI problems using OpenFOAM-1.5-dev

Peer reviewed, updated version.

Author:
Karl Jacob Maus

Peer reviewed by:
Junfeng Yang
Håkan Nilsson

Contents

Chapter 1

1.1 Introduction

This report briefly documents the *icoFsiFoam* solver provided with the development version of OpenFOAM, **OpenFOAM-1.5-dev** [2], which is maintained by Hrvoje Jasak. *icoFsiFoam* is a fluid–structure interaction (FSI) solver for weakly coupled systems with small structural deformations and laminar fluid flow.

The *icoFsiFoam* solver is then modified and extended to create new FSI solvers constructed in a similar manner as *icoFsiFoam*. First, the *myIcoFsiFoam* solver is built almost from scratch using the *icoFoam* and *stressedFoam* solvers, such that the newly developed solver incorporates both stress component computations and the possibility for calculating thermal stresses. Then, another solver –*myInterFsiFoam*– is created using the *interFoam* solver for the fluid domain and utilizing the same approach as when creating *myIcoFsiFoam*, reusing some of the source files.

Finally, these newly created solvers will be used on a couple of test–tutorials; one is based on an already existing *interFoam* tutorial, and the other is based on a tutorial for *icoFsiFoam* not present in the standard OpenFOAM-1.5-dev installation, but which may be found here:
`http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/run/flappingConsoleSmall_HJ_21Mar2008.tgz`

All the case files used in this report are available here:
`http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/KarlJacobMaus/myFSITutorials.tar.gz`

And the solver source files can be found here:
`http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/KarlJacobMaus/myFSISolvers.tar.gz`

Some familiarity with both OpenFOAM and the operating system (OS) command shell of use are required on behalf of the reader. Some Linux bash–commands are provided, but in general it is expected that the reader knows how to e.g. open a file for viewing, change the working directory, or start an editor of choice and modify files. This report is NOT meant to be a bash–tutorial.

This report intends to be as system–independent as possible, so the author tries to use general concepts instead of system–specific commands when describing operations in the text. By the same philosophy, line numbers are rarely given when referring to files that has to be edited. The reason for this is that the comment header is of arbitrary length and often related to a previous version of OpenFOAM, and since the development version is under constant revision the line numbering might possibly change[1]. Therefore, the author finds it more useful to specify lines of code by using a uniquely identifying code snippet from the relevant line and recommends the reader to use the "search" capability of the editor of choice to find the respective lines.

[1]e.g. due to homogenization of the file headers or clean–up of the code commenting

All filenames and keywords in files are typeset in **`boldface typewriter`** style, and shell commands in either `verbatim` font or with a shell environment, i.e.

```
[dir]$ shell-command
```

This shell environment also specifies the current working directory; in the previous example this is the directory `dir`. Solver names are *emphasized*, except when used in headers. Quote marks " " and ' ' are reserved for textual emphasis.

1.2 Fluid–Structure Interaction

In a fluid flow system there is always some kind of FSI, though the interaction might not be noticeable on the domain of interest. For instance, flow around a building is usually modeled as a one–phase system since the building normally is regarded as a rigid body and does not influence the flow other than by its static obstruction of the flow. On the microscopic scale, however, there might be some wear on the walls of the building due to the flow which again might slightly alter the boundary layer behavior and thus the fluid flow. This interaction is usually deemed negligible.

When a solid body deforms, e.g. the blades of a wind turbine or an airplane wing, the domain over which the fluid flows alters and the flow is affected by this deformation. The new flow pattern will again influence the flexible body and there is non-negligible interaction between fluid and solid.

If this coupling is weak, that is if the deformations are small and the momentum variations are small between each time step, then one might solve each domain separately and transfer information between domains. This will result in each domain being solved in a slightly different time. If the boundary movement is small then the errors resulting from the use of this approach are acceptable, and the method is viable.

The spatial (mesh) and the temporal discretizations are vital in fluid–structure interaction problems. Since a finite volume (FV) discretization in OpenFOAM is cell-centered, while the boundary is usually defined at the vertices and edges this poses a problem for correct calculation of the mesh movement. This again will influence the convergence and propagate errors in the solid and fluid solvers. Hence, a vertex–based method is preferable. This is only feasible in OpenFOAM by using the finite element classes from the development version, if one is to avoid creating a completely new class of cells.

Time steps are preferably small. Reducing the time step could possibly also make more problems 'weak' with respect to the FSI coupling, while too small time steps have been known to bring errors into the solution [4]. This happens when the structure is flexible or if the mass density of the fluid is high compared to the density of the solid, due to destabilization of the partitioned coupling routine [1].

1.3 icoFsiFoam

The *icoFsiFoam* solver is a solver for weakly coupled FSI problems. The structural part is based on *stressedFoam* and as such limited to linear stress–strain relationships and small deformations. The fluid part of the solver is based on the *icoFoam* solver and is limited to Newtonian fluids in incompressible, laminar flow. *icoFsiFoam* does not use the PISO algorithm used in *icoFoam*, but a SIMPLE-based algorithm with corrector loops specified by PISO parameters in the **`system/fvSolution`** dictionary of the case of interest.

```
/*---------------------------------------------------------------------------*\
Solver: icoFsiFoam
/*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "dynamicFvMesh.H"
#include "patchToPatchInterpolation.H"
#include "tractionDisplacement/tractionDisplacementFvPatchVectorField.H"
#include "tetFemMatrices.H"
#include "tetPointFields.H"
#include "faceTetPolyPatch.H"
#include "tetPolyPatchInterpolation.H"
#include "fixedValueTetPolyPatchFields.H"
#include "pointMesh.H"
#include "pointFields.H"
#include "volPointInterpolation.H"
```

Figure 1.1: Preloaded headers in the `icoFsiFoam.C` file.

The source code for the *icoFsiFoam* solver is located in the `$FOAM_SOLVERS/stressAnalysis/icoFsiFoam` directory of OpenFOAM. The main file is `$FOAM_SOLVERS/stressAnalysis/icoFsiFoam/icoFsiFoam.C`, and the first part of this file is shown in figure 1.1.

For an overview of the *icoFsiFoam* solver and to get an idea on how to construct similar solvers, a list of all the externally loaded files in `icoFsiFoam.C` is presented, together with a short explanation of each file and location of the respective source file container directory. The lines which call files marked with an asterisk (*) are redundant `#include`-lines and can be omitted without changing the solver properties.

fvCFD.H Standard header file for finite volume method (FVM) in OpenFOAM (OF)
- location: `$FOAM_SRC/finiteVolume/cfdTools/general/include/`

dynamicFvMesh.H Header for class `dynamicFvMesh`
- location: `$FOAM_SRC/dynamicFvMesh/dynamicFvMesh/`

patchToPatchInterpolation.H Interpolation functions; imports `pointField.H`
- location: `$FOAM_SRC/OpenFOAM/interpolations/patchToPatchInterpolation/`

tetFemMatrices.H calls `tetFemScalarMatrix.H` which calls `tetFemMatrix.H` which again calls `tetPointFields.H`. Defines tetrahedral finite element (FE) matrix
- location: `$FOAM_SRC/tetDecompositionFiniteElement/tetFemMatrix/`

(*)tetPointFields.H Point field for finite element method (FEM) discretization. Has already been loaded by the file `tetFemMatrices.H`.

faceTetPolyPatch.H FE `tetPolyPatch` based on `PolyPatch`
- location: `$FOAM_SRC/tetDecompositionFiniteElement/tetPolyMesh/`

tetPolyPatchInterpolation.H Interpolates fields defined on faces into points on a `tetPolyPatch`
- location: `$FOAM_SRC/tetDecompositionFiniteElement/tetPolyPatchInterpolation/`

fixedValueTetPolyPatchFields.H Fixed value `tetPolyPatch` fields
- location: `$FOAM_SRC/tetDecompositionFiniteElement/tetPolyPatchFields/basic/fixedValue/`

(*)pointMesh.H Mesh for points from `polyMesh`. Is called by the next file.

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
    -I$(LIB_SRC)/dynamicMesh/lnInclude \
    $(WM_DECOMP_INC) \
    -I$(LIB_SRC)/tetDecompositionFiniteElement/lnInclude \
    -I$(LIB_SRC)/tetDecompositionMotionSolver/lnInclude \


EXE_LIBS = \
    -lfiniteVolume \
    -ldynamicFvMesh \
    -ldynamicMesh \
    $(WM_DECOMP_LIBS) \
    -llduSolvers
```

Figure 1.2: `$FOAM_SOLVERS/stressAnalysis/icoFsiFoam/Make/options`

pointFields.H Calls `pointMesh.H` - and thus makes the preceding `#include` statement redundant.
- location: `$FOAM_SRC/OpenFOAM/fields/GeometricFields/pointFields/`

volPointInterpolation.H Interpolation from volume field to point field
- location: `$FOAM_SRC/finiteVolume/interpolation/volPointInterpolation/`

The `Make/options` file, shown in figure 1.2, includes lines referring the compiler to the following list of library source directories[2] to ensure that the formerly mentioned header files can be found and loaded:

finiteVolume Contains the FV relevant libraries.

dynamicFvMesh Contains the dynamic FV mesh.

(*)dynamicMesh Contains dynamic mesh routines and is required (and accessed) by the dynamic finite volume mesh source file. Redundant `#include`–line.

tetDecompositionFiniteElement Contains the methods and routines for creation and interpolation of the FE mesh.

(*)tetDecompositionMotionSolver Is included, but not explicitly called by any of the related routines and can be omitted.

Again, (*) indicates lines that are redundant.

Note that while the local `tractionDisplacement` directory is loaded at compile–time by the *stressedFoam* solver from the `Make/options` file, figure 1.3, this is not the case with *icoFsiFoam*. Instead, the header file is included directly in the `icoFsiFoam.C` file without specifying container directory, and the corresponding line from `Make/options` has been altered (the `lnInclude` suffix of the directory reference in `Make/options` is removed).

1.3.1 Main program

Execution of the *icoFsiFoam* solver begins with calling all necessary headers as listed in figure 1.1, to ensure all relevant classes and namespaces have been declared before starting the simulation. These

[2]and dynamic libraries

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -ItractionDisplacement/lnInclude

EXE_LIBS = \
    -lfiniteVolume \
    -llduSolvers
```

Figure 1.3: `$FOAM_SOLVERS/stressAnalysis/stressedFoam/Make/options`.

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createDynamicFvMesh.H"
#   include "createStressMesh.H"
#   include "createFields.H"
#   include "createStressFields.H"
#   include "readMechanicalProperties.H"
#   include "readCouplingProperties.H"
#   include "readTimeControls.H"
#   include "initContinuityErrs.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

Figure 1.4: Initial part of the `main` program routine of the *icoFsiFoam* solver.

headers are the externally loaded files mentioned above, together with the local[3] file `tractionDisplacement/tractionDisplacementFvPatchVectorField.H` which takes care of the traction at the displacement boundary.

The files listed below are called at startup of the `main` function in `icoFsiFoam.C`, before the timeloop is initialized; see figure 1.4. In the following list the purpose of each file is explained in brief. In addition, comparisons with files related to *icoFsiFoam* and *stressedFoam*, located at `$FOAM_SOLVERS/incompressible/icoFoam/` and `$FOAM_SOLVERS/stressAnalysis/stressedFoam/` respectively, are made. The reader is encouraged to have a terminal window ready to be able to view and compare the contents of the files mentioned; these will not be presented in this report.

setRootCase.H Called by both `stressedFoam.C` and `icoFoam.C`. Checks root case.

createTime.H Also called by both original solvers. Initiates time variables.

createDynamicFvMesh.H Creates the dynamic mesh for the fluid. This file is used instead of `createMesh.C` as was used in `icoFoam.C`, due to the need of a dynamic FV–mesh to handle displacements of the solid domain.

createStressMesh.H Used instead of `createMesh.H` included in `stressedFoam.C`. This file is a local variant of the file `$FOAM_SRC/OpenFOAM/include/createMesh.H` (used in `icoFoam.C` and `stressedFoam.C`) which also covers creation of the point mesh needed for the finite element implementation of the fluid–structure boundary. The domain, fields, and various control variables are read from, and written to, the relevant `solid` subdirectories of the case of interest instead of the default region (see section 1.5).

[3]i.e., relative to `$FOAM_SOLVERS/stressAnalysis/icoFsiFoam/`

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    Info<< "\nStarting time loop\n" << endl;
    while (runTime.run())
    {
#       include "readPISOControls.H"
#       include "readTimeControls.H"
#       include "CourantNo.H"
#       include "setDeltaT.H"
        runTime++;
        Info<< "Time = " << runTime.timeName() << nl << endl;
#       include "setPressure.H"
#       include "solveSolid.H"
#       include "setMotion.H"
#       include "solveFluid.H"
        runTime.write();
        Info<< "ExecutionTime = "
            << runTime.elapsedCpuTime()
            << " s\n\n" << endl;
    }
    Info<< "End\n" << endl;
    return(0);
}
// ********************************************************************* //
```

Figure 1.5: Solution loop in the `main` program routine of the *icoFsiFoam* solver.

createFields.H This is the local copy of `$FOAM_SOLVERS/incompressible/icoFoam/createFields.H`, identical but with the addition of a routine reading the fluid density `"rho"` from the case file `constant/transportProperties`.

createStressFields.H Local copy of the file `$FOAM_SOLVERS/stressAnalysis/stressedFoam/createFields.H`. Velocity of the solid domain has been renamed to separate it from the fluid velocity. A new mesh object, `stressMesh`, is defined for the solid domain, and all references to thermal calculations have been stripped from this file (compare with the *stressedFoam* `createFields.H` file).

readMechanicalProperties.H Local file; reads density `rho` and elasticity parameters `nu` and `E` from solid dictionary `solid/constant/mechanicalProperties` of the relevant case and computes necessary coefficients. Also checks if the case problem is a "plane stress" problem or not and alters coefficients `lambda` and `threeK` accordingly.

readTimeControls.H Reads the time controls from the `system/controlDict` case dictionary (see below).

initContinuityErrs.H Initializes the continuity error variable.

While the solver is running, it loops over a set of `#include` instructions, in addition to taking the time of the main solver and outputting this time–information (figure 1.5). First, three operations are conducted: Reading time and solver parameters, checking Courant number, and setting a variable time step if the proper dictionary parameter is set (see below). The fact that all these `#include` lines are in the main time loop shows the run-time modifiability of the *icoFsiFoam* solver.

readPISOControls.H Reads the PISO control entries from the case file `system/fvSolution`.

readTimeControls.H Reads the entries `adjustTimeStep`, `maxDeltaT` and `maxCo` from the case `system/controlDict` file.

CourantNo.H Calculates and outputs the mean and maximum Courant numbers.

setDeltaT.H Reduces or increases the time step size according to the calculations of `CourantNo.H` in order to maintain a constant maximum Courant number. This only applies if the `adjustTimeStep` dictionary entry in `system/controlDict` is set to "yes".

Then the solver part starts by calling the following local files in this order:

setPressure.H Sets pressure on the solid patch by interpolating the fluid pressure on the fluid side of the boundary and alters the traction forces on the solid boundary accordingly.

solveSolid.H Solves the deformation of the solid according to the external pressure set previously. This is a stripped down version of the inner loop of `$FOAM_SOLVERS/stressAnalysis/stressedFoam/stressedFoam.C`, in which all thermal and stress component computations have been removed.
Calls two additional local files:

- **readStressedFoamControls.H** Reads two parameters, `nCorrectors` and `U`, from the `fvSolution` dictionary. These parameters control the number of corrector loops and tolerance on solid domain velocity calculation errors.
- **calculateStress.H** Calculates stress components.

setMotion.H Moves the mesh in accordance with the deformations of the solid domain calculated in the previous step. Calls `volContinuity.H` to calculate volume continuity errors.

solveFluid.H Solves the fluid motion on the updated geometry and mesh. Calls `movingMeshContinuityErrs.H` for the calculation of errors on a moving mesh.
Instead of the PISO loop used in the *icoFoam* solver, a SIMPLE loop with an added pressure correction loop is used. This pressure correction loop is controlled by the PISO controls in the `fvSolution` dictionary of the case of interest.

1.4 Building weakly coupled FSI solvers

1.4.1 The myIcoFsiFoam solver

The *icoFsiFoam* solver source code will be used as a template for the creation of a new FSI solver, the *myIcoFsiFoam* solver. The approach will be to combine code from *icoFoam* and *stressedFoam* to produce the new fluid–structure interaction solver. Complete transcripts of the resulting source files can be found in appendix B.1, and a complete walkthrough with terminal commands and detailed edit–notes are available in appendix A.1.1.

First, an appropriate directory must be created where the solver source codes are placed. The location is arbitrary, but a consistent directory address could be `$FOAM_RUN/applications/solvers/myIcoFsiFoam`. Then this directory is set as the working directory. Any references to directory `localdir`, or local directory, will in this section refer to the working directory just created.

The file `$FOAM_SOLVERS/incompressible/icoFoam/icoFoam.C` is used as it is since it contains the headers and solver code for the fluid part of the problem, and is thus copied twice to the local directory; once to the file `solveFluid.H` and once to the file `myIcoFsiFoam.C`.

Next, edit the file `solveFluid.H` by stripping it of any header information, thus retaining only the part inside the time loop, i.e. the part which will be removed from `myIcoFsiFoam.C` in the next step. Thus, edit `solveFluid.H` to remove the lines before and including the line beginning with
`#include CourantNo.H`
and also remove all lines including and after the line beginning with
`runTime.write()`
The entire code of `solveFluid.H` should be enclosed in braces ({ }) to avoid namespace confusion.

Then edit the `myIcoFsiFoam.C`-file to retain only the header- and loop information and remove everything between the `#include CourantNo.H`-line of the loop and the `runTime.write()`-line (i.e., retain all the information removed from the `solveFluid.H` and remove the lines kept in `solveFluid.H` that was performed in the previous step). The `include`–lines in the header of `myIcoFsiFoam.C` must be complete. All lines missing from `myIcoFsiFoam.C` that are present in the headers of either `stressedFoam.C` or `icoFsiFoam.C` must be included in `myIcoFsiFoam.C`. This is to ensure that *myIcoFsiFoam* can use the FEM routines of OF-1.5-dev and that the moving mesh and stress analysis tools work. And, since the use of a dynamic mesh, the entry `createMesh` must change to `createDynamicFvMesh` for the fluid domain.

A mesh for the structural domain must also be created; this file is `createStressMesh.H` and is created based on the `createMesh.H` file located in `$FOAM_SRC/OpenFOAM/include/` by copying this and renaming it. In addition, the traction displacement boundary code from *stressedFoam* is necessary to include. Copy the directory `$FOAM_SOLVERS/stressAnalysis/stressedFoam/tractionDisplacement/`, with contents, to the local directory and add a line `#include "tractionDisplacementFvPatchVectorField.H"` in the header of the solver `myIcoFsiFoam.C`.

The rest of the contents of `$FOAM_SOLVERS/stressAnalysis/stressedFoam/` is also necessary in order to solve the solid domain. Copy all the files, but be careful not to overwrite any existing files with the same name (e.g., `createFields.H` exists in both *icoFoam* and *stressedFoam* solver directories, so `stressedFoam/createFields.H` is copied to the local `createStressFields.H` to let the local `createFields.H` deal with the fluid domain fields). In addition, rename `stressedFoam.C` to `solveSolid.H` and strip everything away but retain the main solver loop (as was done with `solveFluid.H`), finally enclosing the code in braces ({ }).

The fluid field is created by the local `createFields.H`, so this file is left as it is. The stress field is created by the local file `createStressFields.H` which is a copy of the `createFields.H` file from *stressedFoam*, with the modifications that `U` and `T` is replaced by `Usolid` and `Tsolid`, and `mesh` replaced by `solidMesh`.

Create the `Make` subdirectory in the `myIcoFsiFoam` solver directory, and copy the files `files` and `options` from the *stressedFoam* `Make` subdirectory. The `Make/options` file must also change, by deleting the tail `/lnInclude` from the line specifying the `tractionDisplacement` directory. Then edit the `Make/files` file by changing all instances of `stressedFoam` to `myIcoFsiFoam` and adding `_USER_` to the `FOAM_APPBIN` environment variable. This ensures consistency in naming and stores the final executable together with the privately built solvers in `$FOAM_USER_APPBIN`.

The `Make/options` file must also be augmented to include all the needed libraries and directories to use. Add the following lines to the `EXE_INC = \` entry:

- `-ItractionDisplacement \`
  this is the local directory with the traction displacement code.
- `-I\$(LIB_SRC)/dynamicFvMesh/lnInclude \`
  contains the includes for the FV mesh.
- `$(WM_DECOMP_INC) \`
  is needed since it will be used to find the development version's FE routines. It points to the `faceDecomp` directories of the finite element routines.
- `tetDecompositionFiniteElement/lnInclude`
  contains the includes for the FE mesh.

The `EXE_LIBS = \` also needs some additional lines; adding `-ldynamicFvMesh \` specifies the dynamic finite volume mesh shared library and adding `$(WM_DECOMP_LIBS) \` are the FE face decomposition libraries.

Below is a summary of the steps needed to create the *myIcoFsiFoam* fluid–structure interaction solver except for the `Make` files manipulation described above (see appendix A.1.1 for a complete walkthrough of the procedure and B.1 for a transcript of the resulting source code files). The following instructions all assume that the current working directory is the source code directory for the new solver, which should be named `myIcoFsiFoam` to comply with the naming convention of OpenFOAM and could be located anywhere[4]. This is what is referred to in the following as the "local directory", and all references to a "local" file means that the file is present in the "local directory" or some specified subdirectory thereof.

File: `myIcoFsiFoam.C`

Action: Copy `$FOAM_SOLVERS/incompressible/icoFoam/icoFoam.C` to local file `myIcoFsiFoam.C`.

Action: Remove the interior of the `main` timeloop, retaining only the loop itself and any `Info`–statements. The main solver part will be moved to a seperate file; `solveFluid.H`.

File: `solveFluid.H`

Action: Copy file `$FOAM_SOLVERS/incompressible/icoFoam/icoFoam.C` to local file `solveFluid.H`.

Action: Strip everything away but the interior of the time loop, i.e., keep only the main fluid solver routine present between the following lines:

```
#include "CourantNo.H"
⋮
runTime.write();
```

Remember to enclose code in braces { }.

Action: In order to be able to calculate the errors on a moving mesh the file `movingMeshContinuityErrs.H` needs to be called instead of `continuityErrs.H`.

readStressedFoamControls.H

Action: Copy file `$FOAM_SOLVERS/stressAnalysis/stressedFoam/readStressedFoamControls.H` to local file `readStressedFoamControls.H`

Action: Change entry `mesh` to `stressMesh` to read from correct mesh (solid mesh).

solveSolid.H

Action: Copy file `$FOAM_SOLVERS/stressAnalysis/stressedFoam/stressedFoam.C` to local file `solveSolid.H`

Action: As with `solveFluid.H` only the core of the time loop is kept, and no `Info` lines are retained since they are produced by the main `myIcoFsiFoam.C`. Thus, remove all lines before `#include "readStressedFoamControls.H"` and all lines after `#include "calculateStress.H"`. Keep the commented information header and update this if you like, and remember to enclose code in braces { }.

Action: Replace all instances of `U` with `Usolid`, and `T` with `Tsolid`, since this is the solid domain.

[4]although `$FOAM_RUN` or some relevant subdirectory thereof would be a consistent location

File: `readMechanicalProperties.H`

Action: Copy file `$FOAM_SOLVERS/stressAnalysis/stressedFoam/readMechanicalProperties.H` to the local directory.

Action: Replace `mesh` with `stressMesh` to specify the solid mesh.

Action: Replace every instance of `nu` with `nuS` to separate it from the fluid `nu`.

File: `readThermalProperties.H`

Action: Copy file `$FOAM_SOLVERS/stressAnalysis/stressedFoam/readThermalProperties.H` to the local directory.

Action: On line 9, replace `mesh` with `stressMesh`

File: `calculateStress.H`

Action: Copy the file `$FOAM_SOLVERS/stressAnalysis/stressedFoam/calculateStress.H` to the local directory.

Action: Replace all instances of `U` and `T` with `Usolid` and `Tsolid`, respectively.

Action: Replace `mesh` with `stressMesh` in the stress meshes since this is in the solid domain.

File: `createStressMesh.H`

Action: Copy `$FOAM_SRC/OpenFOAM/include/createMesh.H` to local file `createStressMesh.H`.

Action: Replace `mesh` with `stressMesh` on line 4 to separate the solid mesh from the fluid mesh.

Action: Replace `Foam::fvMesh::defaultRegion,` with `"solid",` to specify a different region than the default, since the default region will be used for the fluid domain.

Action: Append the following lines to create an interpolator between the meshes:

```
Foam::pointMesh pStressMesh(stressMesh);

Foam::volPointInterpolation cpi
(
    stressMesh,
    pStressMesh
);
```

File: `createFields.H`

Action: Copy `$FOAM_SOLVERS/incompressible/icoFoam/createFields.H` to the local directory.

Action: Add the following lines to be read after the initialization of `nu`:

```
dimensionedScalar rhoFluid
(
    transportProperties.lookup("rho")
);
```

File: `createStressFields.H`

Action: Copy and rename the file
`$FOAM_SOLVERS/stressAnalysis/stressedFoam/createFields.H` to local file
`createStressFields.H`

```
[myIcoFsiFoam]$ cp $FOAM_SOLVERS/stressAnalysis/stressedFoam/createFields.H createStressFields.H
```

Action: Replace every instance of `mesh` with `stressMesh`

```
[myIcoFsiFoam]$ sed -i s/mesh/stressMesh/g createStressFields.H
```

Action: Define the volume vector field `Usolid` instead of `U`

```
[myIcoFsiFoam]$ sed -i s/'volVectorField U'/'volVectorField Usolid'/g createStressFields.H
```

Action: As opposed to the *icoFsiFoam* solver the thermal stress code is retained, and for consistency define the volume scalar field `Tsolid` instead of `T`

```
[myIcoFsiFoam]$ sed -i s/"T "/"Tsolid "/g createStressFields.H
```

The following files are copied from the `$FOAM_SOLVERS/stressAnalysis/icoFsiFoam` directory to the *myIcoFsiFoam* working directory:

```
[myIcoFsiFoam]$ cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/readCouplingProperties.H .
[myIcoFsiFoam]$ cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setMotion.H .
[myIcoFsiFoam]$ cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setPressure.H .
```

Compile using `wmake`, and if the OpenFOAM environment variables are correctly set this should give a working FSI solver.

```
[myIcoFsiFoam]$ wmake
```

The *myIcoFsiFoam* solver uses the PISO corrector instead of the SIMPLE–loop used by *icoFsiFoam*. It can solve for thermal stresses in the solid, but the fluid solver is not equipped with a thermal component and as such the solver cannot solve thermal fluid–structure interaction. However, if the thermal stresses cause the solid domain to deform then this will influence the fluid flow.

1.4.2 myInterFsiFoam

The previous approach will now be used to couple the *stressedFoam* solid solver with a different solver for the fluid domain, namely *interFoam*, in order to create the *myInterFsiFoam* FSI solver. The source code for each file is presented in subsections of appendix B.2, and a complete walkthrough of the creation process can be found in appendix A.1.2.

First, copy the entire `interFoam` source code directory to a new working directory for the new solver. It is assumed that the previously created `$FOAM_RUN` subdirectory structure exists.

```
[home]$ cp -r $FOAM_SOLVERS/multiphase/interFoam $FOAM_RUN/applications/solvers/myInterFsiFoam
```

Extract the main loop solver algorithm into a seperate file called `solveFluids.H` and rename main solver file:

```
[myInterFsiFoam]$ cp interFoam.C solveFluids.H

[myInterFsiFoam]$ mv interFoam.C myInterFsiFoam.C
```

The files need to be edited in a similar manner as was done for *myIcoFsiFoam*; by removing the solver part of the timeloop from `myInterFsiFoam.C` and stripping everything but the solver part from `solveFluids.H`. Next, some routines for the pressure transfer and mesh motion are necessary. Copy the following files from the `$FOAM_SOLVERS/stressAnalysis/icoFsiFoam/` directory

```
[myInterFsiFoam]$ cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setPressure.H .

[myInterFsiFoam]$ cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setMotion.H .

[myInterFsiFoam]$ cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/readCouplingProperties.H .

[myInterFsiFoam]$ cp -r $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/tractionDisplacement .
```

For the solver to compile, all headers must be loaded. This means the FEM routines, the dynamic mesh and the traction at the boundary. Thus, import all headers missing from `icoFsiFoam.C`, `interFoam.C` and `stressedFoam.C` into the header of `myInterFsiFoam.C`. The `main` function must also contain calls to the solid solver, the fluid solver, pressure transfer and mesh motion routines, contained in `setPressure.H,solveSolid.H, setMotion.H` and `solveFluids.H`, respectively. The order is copied from `myIcoFsiFoam.C` created previously.

The *stressedFoam* routines are also necessary. As with *myIcoFsiFoam*, section 1.4.1, the solver needs two files to create the stress–fields of the solid domain, `createStressFields.H` and `createStressMesh.H`. Reuse the ones from *myIcoFsiFoam* or build them from *stressedFoam* as before.

The solid density `rho` must be renamed to `rhoSolid`, and the fluid density `rhoFluid` must be renamed to `rho` due to *interFoam* design [5]. `rho` is present in `readThermalProperties.H`, `readMechanicalProperties.H`,
`tractionDisplacementFvPatchVectorField.C`, and `calculateStress.H` and must be altered to `rhoSolid` in these files. And, since `rho` is a field (not scalar) in *interFoam*, the `.value()` part from file `setPressure.H` must be removed.

Copy the `Make` subdirectory from *interFoam* and edit `Make/files`: Rename solver from `interFoam` to `myInterFsiFoam`, update application-directory environment variable to `FOAM_USER_APPBIN`, and include a `tractionDisplacement` line (as was done with *myIcoFsiFoam*).

`Make/options` must also be augmented. Add traction displacement subdirectory, `dynamicFvMesh` and all FE decomposition references from `myIcoFsiFoam/Make/options` (see section 1.4.1).

1.5 OpenFOAM FSI cases

Two test–cases are set up to test the solvers created. They are available and ready to run with their respective solvers at

[5]unless, of course, one wants to go through the *interFoam* code in `solveFluids.H` and related files...

`http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/KarlJacobMaus/myFSITutorials.tar.gz`. The creation of the *myInterFsiFoam* tutorial 'softDamBreak' will also be presented in greater detail. Download the zipped file and unpack into the `$FOAM_RUN` directory. The cases should now be accessible by

```
[$FOAM_RUN]$ cd flappingConsoleSmall
```

and

```
[$FOAM_RUN]$ cd softDamBreak
```

1.5.1 A flapping console

This is a modified version of a 2D test–case made for the *icoFsiFoam* solver which can be downloaded from
`http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/run/flappingConsoleSmall_HJ_21Mar2008.tgz`
The case by Dr. Jasak was created for the 1.4-dev version of OF.

The case is set up such that all files related to the solid domain is in the `solid` subdirectory of the case directory (`$FOAM_RUN/flappingConsoleSmall`), and all files governing the fluid domain are located in the `fluid` subdirectory. Each of these domains are set up as usual OpenFOAM test cases, with three subdirectories; `0`, which contains the initial field values, `constant`, which contains files describing the physical properties and the mesh, and `system` which contains control dictionaries for the domain solver. Thus, in, e.g., the case subdirectory `solid/constant/` one can find the `mechanicalProperties` dictionary for the solid domain, whereas in, e.g., the `fluid/0` subdirectory are the inital fields `U` and `p` for the fluid part of the problem. The fluid mesh is specified in `fluid/constant/polymesh/blockMeshDict` and the solid mesh in `solid/constant/polymesh/blockMeshDict`.

The solver is constructed in such a way that all input and output related to the solid domain are read from/to a `solid` subdirectory. This means that when solving equations related to `stressMesh` objects (part of the solid domain) the solver attempts to read and write to the `solid` subdirectory of whatever directory it would normally read from and write to. When run from the `fluid` directory, which is the intended run–directory for cases to the solvers that was made in this report, initial values are read from `fluid/0/` (fluid domain) and `fluid/0/solid/` (solid domain), solver parameters are read from the files within `fluid/system/` and `fluid/system/solid/`, respectively, and so on. To make the current set-up work, in which the domains are seperated into parallel subdirectories, symbolic links (or softlinks) are present were needed[6]. They are

- `fluid/0/solid --> ../../solid/0`
- `fluid/constant/solid --> ../../solid/constant`
- `fluid/system/solid --> ../../solid/system`
- `solid/system/controlDict --> ../../fluid/system/controlDict`

The solver to test is *myIcoFsiFoam* and will be run from the test case `fluid` directory. In addition to the files supplied by Dr. Jasak, the *myIcoFsiFoam* solver needs an additional file; `thermalProperties` in the `solid/constant` directory. This can be copied from any *stressedFoam* tutorial, e.g., `$FOAM_TUTORIALS/stressedFoam/plateHole/constant/thermalProperties`.

[6]The softlinks are set up in linux with the command `[dir]$ ln -s TARGET LINKNAME`

Also, beware of invalid solvers[7] in `fvSolution` and `tetFemSolution` of both the 'solid' domain and the 'fluid' domain. This will be evident from the program output. The casefiles provided with this report, [3], are already prepared, and all files are listed in appendix C

The mesh must be created for both domains. From the case main directory, call *blockMesh* twice, once per domain:

```
[flappingConsoleSmall]$ blockMesh -case solid
```

```
[flappingConsoleSmall]$ blockMesh -case fluid
```

The simulation is run by calling the *myIcoFsiFoam* solver from within the `fluid` subdirectory, or by calling

```
[flappingConsoleSmall]$ myIcoFsiFoam -case fluid
```

After the simulation has run, the result files are stored in their respective domain subdirectories. However, the fields for the solid domain are stored in `solid` subdirectories of the fluid domain for each timestep; i.e., the stresses and velocities of the solid domain for time = 1 second are stored in `flappingConsoleSmall/fluid/1/solid/`, and not in `flappingConsoleSmall/solid/1/`. So, if *paraFoam* is run from the `fluid` directory, only the fluid domain is visible (deformations are of course still visible). If *paraFoam* is run from within the `solid` subdirectory, there are no result files to view! To remedy this, a small `bash`-script, `linkSolidSolutions`, is provided with the case files. This script finds all the solution directories of the `fluid` domain and sets up a 'softlink' from the `solid` directory to point at the `fluid/[time]/solid/` subdirectory for each timestep written to disk. Run the `linkSolidSolutions` file from the case directory to link.

```
[flappingConsoleSmall]$ ./linkSolidSolutions
```

Then, run *paraFoam* from the `fluid` directory (or with `-case fluid` argument), and from within *paraview*, open the file `flappingConsoleSmall/solid/solid.OpenFOAM`. Then both the fluid and the solid domains are visible, and, e.g., stresses can be shown in the solid domain.

Figure 1.6 shows the deformation of the solid domain due to the fluid flow. The flow has not yet developed properly, but it is clear that the solid has deformed.

Flapping console with thermals

By slightly modifying the previous case, thermal stress calculations can be included in the flapping console. *myIcoFsiFoam* can handle thermal stresses in the solid domain, but does not calculate thermals in the fluid domain, so there is no true thermal FSI. However, if some (possibly unphysical) boundary conditions are set, for demonstration purposes, on the FSI boundary then the solver should calculate deformations in the solid domain due to thermal stresses. A few minor modifications to the 'flapping console' test case are sufficient[8]:

- Change `solid/constant/thermalProperties` dictionary to compute thermal stresses by setting parameter `thermalStress` to `yes;`.
- `solid/system/fvSolution` needs an input for the solution of `T`. For now it is possible to just copy the line for `U`.

[7]the dictionaries may need an update due to case was set up for an older version of OF

[8]except for setting `thermalStress` to `yes`, these changes are already present in the provided case

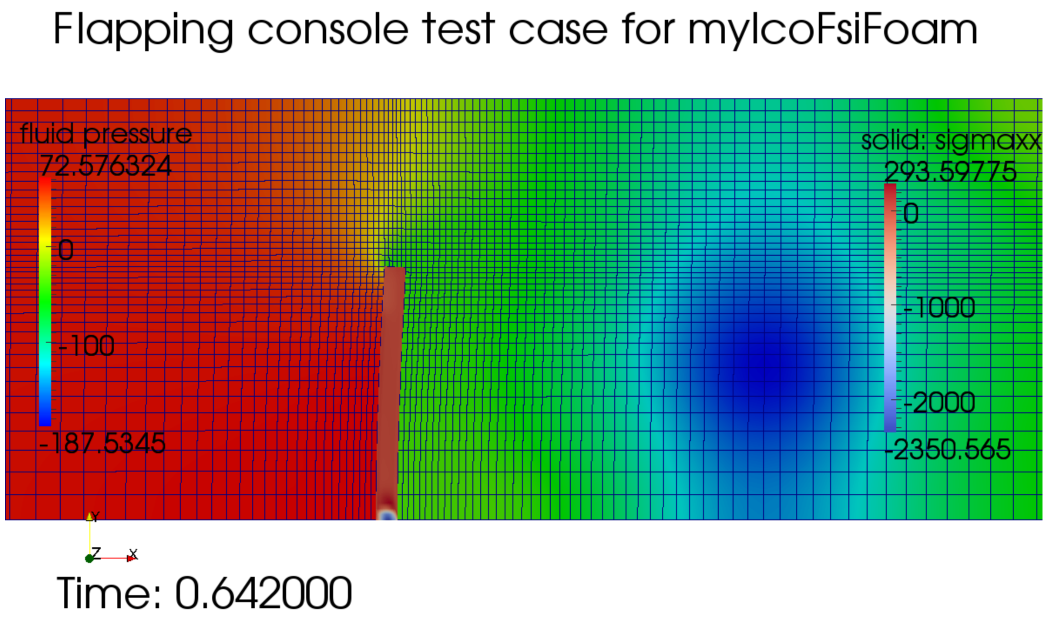

Figure 1.6: Flapping console: Soft cantilever beam deformed in fluid flow.

- `solid/system/fvSchemes` needs an input for `ddtSchemes`. For now just copy the `d2dt2Schemes` lines and rename it to `ddtSchemes`, using the same `default:Euler`

- In addition, an initial `T` file is needed in the `0` directory. Copy the one from `$FOAM_TUTORIALS/stressedFoam/platehole` tutorial and edit to comply with the patches of the solid domain. Set appropriate boundary conditions (`zeroGradient, fixedValue`, etc...)

In figure 1.7 the effect of an oscillating temperature boundary condition on the FSI interface of the solid domain is seen; the structure has an extra bend which comes from thermal expansion and contraction of the solid boundary.

1.5.2 A soft dam break

The purpose of this test–case is twofold; run a dam break simulation with a 'soft' dam, and learn how to set up an FSI tutorial when using a coupled solver like *icoFsiFoam*. The basis for this tutorial is the 'damBreak' tutorial (`$FOAM_TUTORIALS/interFoam/damBreak`). The dam, modelled as a part of the ground missing from the domain, will now be filled by a solid domain mesh just like the cantilever in the previous case.

First, the directory structure must be created; the 'flapping console' case will be used as template so the case directory and two subdirectories must be present. Then, copy the `solid` subdirectory from the 'flapping console' case to the local `solid` directory and copy the entire 'dam break' case into the local `fluid` directory.

```
[$FOAM_RUN]$ mkdir softDamBreak

[$FOAM_RUN]$ cd softDamBreak

[softDamBreak]$ cp -r ../flappingConsoleSmall/solid solid
```

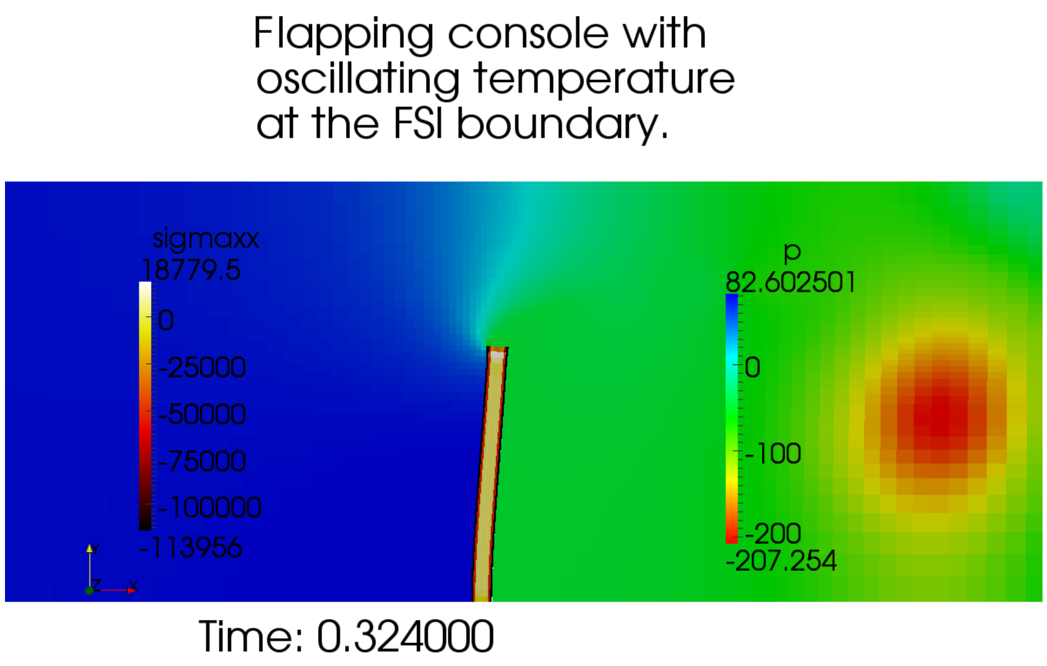

Figure 1.7: A flapping console with thermal stresses.

```
[softDamBreak]$ cp -r $FOAM_TUTORIALS/interFoam/damBreak fluid
```

It is assumed that the cases copied are empty except for the three subdirectories `0`, `constant` and `system`. Otherwise, delete any result–directories.

In addition, the need to make the softlinks mentioned in the 'flapping console' case is present. Create the following list of links, replacing any existing files with identical names in the same directory:

- `[fluid/0]$ ln -s ../../solid/0/ solid`
- `[fluid/constant]$ ln -s ../../solid/constant/ solid`
- `[fluid/system]$ ln -s ../../solid/system/ solid`
- `[solid/system]$ ln -s ../../fluid/system/controlDict controlDict`

mesh

To run there must be some patches defined for the fluid–structure interface. By performing the following, these are added: In the `fluid/constant/blockMeshDict`, move the following patches from `lowerWall` into a new patch called `consoleFluid`

```
        patch consoleFluid
    (
        (1 5 17 13)
        (5 6 18 17)
        (2 14 18 6)
    )
```

In the `solid/constant/polyMesh` directory, edit `blockMeshDict` by replacing vertices $0, 1, 2, 3, 4, 5, 6, 7$ with vertices $1, 2, 6, 5, 13, 14, 18, 17$ from the `fluid/constant/polyMesh` directory `blockMeshDict` file. Remember to specify the correct `ConvertToMeter` parameter value of 0.146. The boundary conditions must also be set in the file `constant/polyMesh/boundary` of both `solid` and `fluid` subdirectories: add patch

```
    consoleFluid
{
     type  patch;
}
```

and alter number of patches from 5 to 6 in the `boundary` file of both directories. Please feel free to refine the mesh so it fits better at the boundary between the solid and the fluid domains.

fluid/constant

The `transportProperties` dictionaries from `damBreak` are reused. Copy the `couplingProperties` and `dynamicMeshDict` dictionaries from the `flappingConsoleSmall/fluid/constant` directory to local directory `fluid/constant/` and reuse.

solid/constant

The `materialProperties` dictionary must be altered; since the fluid density of one of the phases is so large, this problem is actually not so 'weak' as the solver would like. So, to avoid a program crash when run, the `rho` will be set to a typical steel value of 7850 kg/m^3 and the stiffness on the order of 10^{11}. This means that there will be little deformation of the solid, but stress resulting from the oncoming fluid should be evident from the simulation.

system

The control dictionary must be augmented a bit. Use `controlDict` from `damBreak`, with the following modifications:

- `endTime = 2;`
- `deltaT = 0.0003;`
- `writeInterval = 0.005;`
- `writeCompression = compressed;`
- `maxDeltaT = 0.1;`

0

Add the following `boundaryField` to the field `fluid/0/U` to specify the FSI boundary as a moving wall:

```
    consoleFluid
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
```

Copy the initial field `flappingConsole/fluid/0/motionU`, describing the mesh motion, to `fluid/0/` and correct all patch names. They should comply with the patch names that already exist.

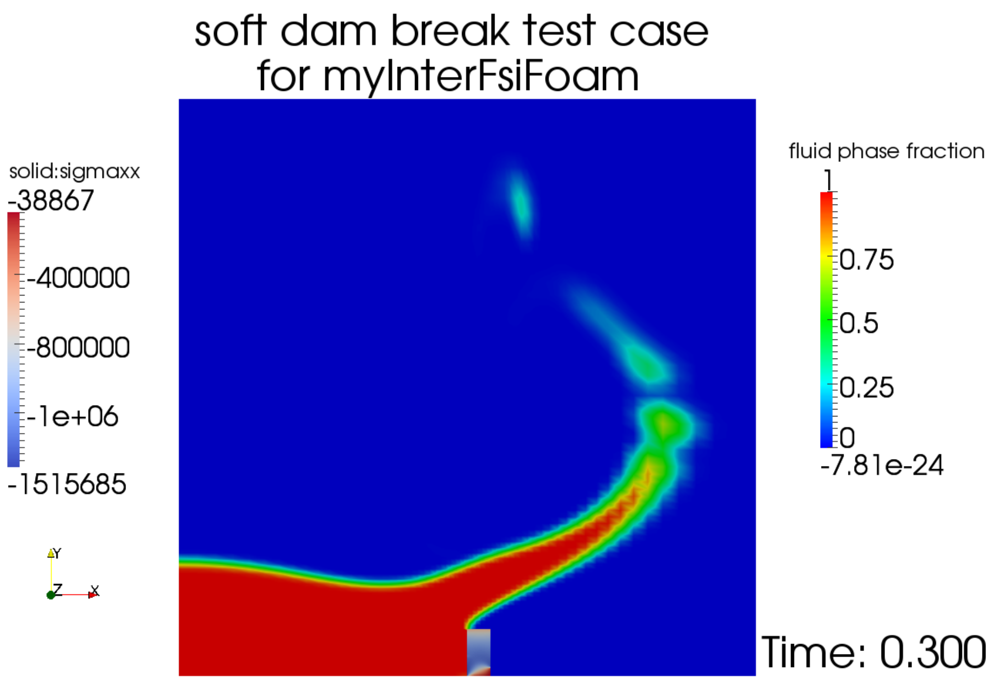

Figure 1.8: Soft dam break with stress calculations.

Open the fieldsgamma and pd from fluid/0 for edit and add boundaryField consoleFluid as type zeroGradient to both files. In addition, copy gamma to gamma.org to keep an original after running *setFields*.

As with 'flapping console', *blockMesh* must be run for each domain, and in this case the *setFields* utility must be run from the fluid directory to initiate the alpha phase distribution of the *interFoam* part of the solver.

```
[myInterFsiFoam]$ blockMesh -case solid

[myInterFsiFoam]$ blockMesh -case fluid

[myInterFsiFoam]$ setFields -case fluid

[myInterFsiFoam]$ myInterFsiFoam -case fluid
```

Results

One timestep of the 'soft dam break' case, just after the fluid hits the dam, is shown in figure 1.8. The dam is not soft (i.e., no deformation takes place) due to the high modulus of rigidity in the steel dam, but stress components are calculated.

1.6 Final remarks

The partitioned approach for creating solvers for weak FSI problems in OpenFOAM by combining existing structure and fluid solvers was demonstrated, resulting in *myIcoFsiFoam* and *myInterFsiFoam*. These were tested and proven functioning, although neither solver was benchmarked and compared to real physical measurement data. Thus, the solvers still need verification of results but the method of constructing the solvers has been shown effective. A consistent and intuitive way of creating test cases is also shown.

Investigations into the stability of temporal discretization and the boundaries of the valid domains for weak coupling are still necessary to determine the applicability of such partitioned solvers, for instance with respect to near identical material densities of fluid and solid.

Bibliography

[1] J. Degroote, P. Bruggeman, R. Haelterman, and J. Vierendeels. "stability of a coupling technique for partitioned solvers in FSI applications". *Computers & Structures*, 86:2224 – 2234, 2008.

[2] H. Jasak. OpenFOAM-1.5-dev. URL: http://openfoam.-extend.svn.sourceforge.net/.

[3] K.J. Maus. OpenFOAM-1.5-dev FSI casefiles for myIcoFsiFoam and myInterFsiFoam fluid–structure interaction solvers. URL: `http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/KarlJacobMaus/myFSITutorials.tar.gz`.

[4] M. Olivier and J. Degroote. "icoFsiFoam simulation crashes when using a smaller timestep". Forum discussion on *CFD Online*, January 2009. URL: http://www.cfd-online.com/Forums/openfoam-solving/58152-icofsifoam-simulation-crashes-when-using-smaller-timestep.html.

Acronyms

FE	finite element
FEM	finite element method
FSI	fluid–structure interaction
FV	finite volume
FVM	finite volume method
OF	OpenFOAM
OS	operating system

Appendix A

Walkthrough: Creation of the solvers

These instructions assume **OpenFOAM-1.5-dev** has been installed correctly on a linux distribution and all environment variables function properly in the shell environment.

A.1 Solvers

A.1.1 myIcoFsiFoam

Create directories

```
[home]$  mkdir --parents $FOAM_RUN/applications/solvers/
[home]$  cd $FOAM_RUN/applications/solvers/
```

Copy files

```
[solvers]$  cp -r $FOAM_SOLVERS/stressAnalysis/stressedFoam myIcoFsiFoam
[solvers]$  wclean myIcoFsiFoam
[solvers]$  mv myIcoFsiFoam/stressedFoam.C myIcoFsiFoam/solveSolid.H
[solvers]$  mv myIcoFsiFoam/createFields.H myIcoFsiFoam/createStressFields.H
[solvers]$  cp $FOAM_SOLVERS/incompressible/icoFoam/icoFoam.C myIcoFsiFoam/myIcoFsiFoam.C
[solvers]$  cp $FOAM_SOLVERS/incompressible/icoFoam/icoFoam.C myIcoFsiFoam/solveFluid.H
[solvers]$  cp $FOAM_SOLVERS/incompressible/icoFoam/createFields.H myIcoFsiFoam/.
[solvers]$  cp $FOAM_SRC/OpenFOAM/include/createMesh.H myIcoFsiFoam/createStressMesh.H
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setPressure.H myIcoFsiFoam/.
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setMotion.H myIcoFsiFoam/.
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/readCouplingProperties.H myIcoFsiFoam/.
[solvers]$  cd myIcoFsiFoam
```

File: `myIcoFsiFoam.C`

- Open `myIcoFsiFoam.C` in an editor:
  - Remove the entire following section (only the beginning and the end of section are shown):

```
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
            .
            .
            .
            U -= rUA*fvc::grad(p);
            U.correctBoundaryConditions();
         }
```

- Add the following lines to the header, before the definition of the `main` program together with the call for `fvCFD.H`:

```
#include "dynamicFvMesh.H"
#include "tractionDisplacementFvPatchVectorField.H"
#include "patchToPatchInterpolation.H"
#include "tetFemMatrices.H"
#include "faceTetPolyPatch.H"
#include "tetPolyPatchInterpolation.H"
#include "fixedValueTetPolyPatchFields.H"
#include "pointFields.H"
#include "volPointInterpolation.H"
```

- Alter the line
  `# include "createMesh.H"`
  so it reads
  `# include "createDynamicFvMesh.H"`
- Insert the line
  `# include "createStressMesh.H"`
  between the lines
  `# include "createDynamicFvMesh.H"`
  and
  `# include "createFields.H"`
- Add the following lines directly after
  `# include "createFields.H"`:

```
#   include "readMechanicalProperties.H"   //From stressedFoam solver
#   include "readThermalProperties.H"      //From stressedFoam solver
#   include "createStressFields.H"         //Was stressedFoam "createFields.H"
#   include "readCouplingProperties.H"     //From icoFsiFoam solver
#   include "readTimeControls.H"           //Adjustable timestep
```

- Add the following lines inside the main loop, right after
  `# include "CourantNo.H"`:

```
#       include "readTimeControls.H"
#       include "setDeltaT.H"
//   Main solver code:
#       include "setPressure.H"
#       include "solveSolid.H"
#       include "setMotion.H"
#       include "solveFluid.H"
```

File: `solveFluid.H`

- Open `solveFluid.H` in an editor:

- Remove all lines before and including
  `# include "CourantNo.H"`
- Remove all files including and after
  `runTime.write();`
- Enclose the entire file contents in braces { }
- Change line
  `# include "continuityErrs.H"`
  to
  `# include "movingMeshContinuityErrs.H"`

File: `solveSolid.H`

- Open `solveSolid.H` in an editor:
  - Remove all lines before and including
    `Info<< "Iteration:  " << runTime.timeName() << nl << endl;`
  - Remove all lines including and after
    `Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"`
  - Enclose the entire file contents in braces { }
  - Replace all occurances of the keyword `U` with `Usolid` and `T` with `Tsolid`[1]

File: `createFields.H`

- Open the file `createFields.H` in an editor and add the following part, e.g., directly after initialization of `nu`, i.e., directly following the line
  `dimensionedScalar nu(transportProperties.lookup("nu"));`:

```
    dimensionedScalar rhoFluid
    (
        transportProperties.lookup("rho")
    );
```

File: `createStressFields.H`

```
[myIcoFsiFoam]$  sed -i s/mesh/stressMesh/g createStressFields.H
[myIcoFsiFoam]$  sed -i s/'volVectorField U'/'volVectorField Usolid'/g createStressFields.H
[myIcoFsiFoam]$  sed -i s/"T "/"Tsolid "/g createStressFields.H
```

File: `createStressMesh.H`

```
[myIcoFsiFoam]$  sed -i s/mesh/stressMesh/g createStressMesh.H
[myIcoFsiFoam]$  sed -i s/'Foam::fvMesh::defaultRegion'/'"solid"'/g createStressMesh.H
```

---

[1]Note: Be aware, when using `sed` or similar command line editors, that there is a reference to `"laplacian(DU,U)"` which should stay untouched, and should NOT read `"laplacian(DUsolid,Usolid)"` after editing!

File: `createStressMesh.H`

- Append the following lines to the end of the file (in the editor of choice) to create an interpolator between the meshes:

```
Foam::pointMesh pStressMesh(stressMesh);

Foam::volPointInterpolation cpi
(
    stressMesh,
    pStressMesh
);
```

File: `calculateStress.H`

```
[myIcoFsiFoam]$  sed -i s/mesh/stressMesh/g calculateStress.H
[myIcoFsiFoam]$  sed -i 's/T)/Tsolid)/g' calculateStress.H
[myIcoFsiFoam]$  sed -i s/"(U)"/"(Usolid)"/g calculateStress.H
```

File: `readThermalProperties.H`

```
[myIcoFsiFoam]$  sed -i s/mesh/stressMesh/g readThermalProperties.H
```

File: `readMechanicalProperties.H`

```
[myIcoFsiFoam]$  sed -i s/mesh/stressMesh/g readMechanicalProperties.H
[myIcoFsiFoam]$  sed -i s/nu/nuS/g readMechanicalProperties.H
[myIcoFsiFoam]$  sed -i s/'"nuS"'/'"nu"'/g readMechanicalProperties.H
```

File: `readStressedFoamControls.H`

```
[myIcoFsiFoam]$  sed -i s/mesh/stressMesh/g readStressedFoamControls.H
```

File: `Make/files`

```
[myIcoFsiFoam]$  sed -i s/stressedFoam/myIcoFsiFoam/g Make/files
[myIcoFsiFoam]$  sed -i s/AM_APP/AM_USER_APP/g Make/files
```

File: `Make/options`

- Open `Make/options` in a file editor:
  - Substitute `/lnInclude` with \ on the line `-ItractionDisplacement/lnInclude`
  - Append the following to the `EXE_INC =` \ entries directly after previously edited line:

    ```
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
    $(WM_DECOMP_INC) \
    -I$(LIB_SRC)/tetDecompositionFiniteElement/lnInclude
    ```

  - Insert the lines

    ```
     -ldynamicFvMesh \
    $(W_DECOMP_LIBS) \
    ```

    between the two existing lines of the `EXE_LIBS =` \ section of `Make/options`.

Compile with `wake`

```
[myIcoFsiFoam]$  wmake
```

A.1.2 myInterFsiFoam

Change directory

```
[home]$  cd $FOAM_RUN/applications/solvers/
```

Copy files

```
[solvers]$  cp -r $FOAM_SOLVERS/multiphase/interFoam myInterFsiFoam
[solvers]$  wclean myInterFsiFoam
[solvers]$  mv myInterFsiFoam/interFoam.C myInterFsiFoam/myInterFsiFoam.C
[solvers]$  cp myInterFsiFoam/myInterFsiFoam.C myInterFsiFoam/solveFluids.H
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/stressedFoam/stressedFoam.C myInterFsiFoam/solveSolid.H
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/stressedFoam/createFields.H myInterFsiFoam/createStressFields.H
[solvers]$  cp $FOAM_SRC/OpenFOAM/include/createMesh.H myInterFsiFoam/createStressMesh.H
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/stressedFoam/read* myInterFsiFoam/.
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/stressedFoam/calculateStress.H myInterFsiFoam/.
[solvers]$  cp -r $FOAM_SOLVERS/stressAnalysis/stressedFoam/tractionDisplacement myInterFsiFoam/.
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setPressure.H myInterFsiFoam/.
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/setMotion.H myInterFsiFoam/.
[solvers]$  cp $FOAM_SOLVERS/stressAnalysis/icoFsiFoam/readCouplingProperties.H myInterFsiFoam/.
[solvers]$  cd myInterFsiFoam
```

File: `myInterFsiFoam.C`

- Open `myInterFsiFoam.C` in an editor:
  - Remove the main solver routine, i.e., the entire following section (only the beginning and the end of section are shown):

    ```
    twoPhaseProperties.correct();
    #include "gammaEqnSubCycle.H"
    #include "UEqn.H"
          .
          .
          .
          pRefValue - getRefCellValue(p, pdRefCell)
      );
     }
    ```

  - Add the following lines to the header, before the definition of the `main` program after `#include "twoPhaseMixture.H"`:

    ```
    #include "dynamicFvMesh.H"
    #include "tractionDisplacementFvPatchVectorField.H"
    #include "patchToPatchInterpolation.H"
    #include "tetFemMatrices.H"
    #include "faceTetPolyPatch.H"
    #include "tetPolyPatchInterpolation.H"
    #include "fixedValueTetPolyPatchFields.H"
    #include "pointFields.H"
    #include "volPointInterpolation.H"
    ```

- After initialization of `main`, alter the line
  `# include "createMesh.H"`
  so it reads
  `# include "createDynamicFvMesh.H"`
- Insert the line
  `# include "createStressMesh.H"`
  between the lines
  `# include "createDynamicFvMesh.H"`
  and
  `# include "readEnvironmentalProperties.H"`
- Add the following lines between
  `# include "createFields.H"` and
  `# include "readTimeControls.H"`:

```
#   include "readMechanicalProperties.H"   //From stressedFoam solver
#   include "readThermalProperties.H"      //From stressedFoam solver
#   include "createStressFields.H"         //Was  stressedFoam "createFields.H"
#   include "readCouplingProperties.H"     //From icoFsiFoam solver
```

- Add the following lines inside the main loop, on the line after
  `Info<< "Time = " << runTime.timeName() << nl << endl;`:

```
//   Main solver code:
#       include "setPressure.H"
#       include "solveSolid.H"
#       include "setMotion.H"
#       include "solveFluids.H"
```

File: `solveFluids.H`

- Open `solveFluids.H` in an editor:
  - Remove all lines before and including
    `Info<< "Time = " << runTime.timeName() << nl << endl;`
  - Remove all lines including and after
    `Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"`
  - Enclose the entire file contents in braces { }
  - Change line
    `# include "continuityErrs.H"`
    to
    `# include "movingMeshContinuityErrs.H"`

File: `solveSolid.H`

- Open `solveSolid.H` in an editor:
  - Remove all lines before and including
    `Info<< "Iteration:  " << runTime.timeName() << nl << endl;`
  - Remove all files including and after
    `Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"`
  - Enclose the entire file contents in braces { }.
  - Replace all occurances of the keyword `U` with `Usolid` and `T` with `Tsolid`.

File: `createStressFields.H`

```
[myInterFsiFoam]$  sed -i s/mesh/stressMesh/g createStressFields.H
[myInterFsiFoam]$  sed -i s/'volVectorField U'/'volVectorField Usolid'/g createStressFields.H
[myInterFsiFoam]$  sed -i s/"T "/"Tsolid "/g createStressFields.H
```

File: `createStressMesh.H`

```
[myInterFsiFoam]$  sed -i s/mesh/stressMesh/g createStressMesh.H
[myInterFsiFoam]$  sed -i s/'Foam::fvMesh::defaultRegion'/'"solid"'/g createStressMesh.H
```

File: `createStressMesh.H`

- Append the following lines to the end of the file (in the editor of choice) to create an interpolator between the meshes:

```
Foam::pointMesh pStressMesh(stressMesh);

Foam::volPointInterpolation cpi
(
    stressMesh,
    pStressMesh
);
```

File: `calculateStress.H`

```
[myInterFsiFoam]$  sed -i s/mesh/stressMesh/g calculateStress.H
[myInterFsiFoam]$  sed -i 's/T)/Tsolid)/g' calculateStress.H
[myInterFsiFoam]$  sed -i s/"(U)"/"(Usolid)"/g calculateStress.H
[myInterFsiFoam]$  sed -i s/rho/rhoSolid/g calculateStress.H
```

File: `readThermalProperties.H`

- `[myInterFsiFoam]$ sed -i s/mesh/stressMesh/g readThermalProperties.H`
- Open file for editing and change all instances of keyword `rho` with keyword `rhoSolid`

File: `readMechanicalProperties.H`

- `[myInterFsiFoam]$ sed -i s/mesh/stressMesh/g readMechanicalProperties.H`
- `[myInterFsiFoam]$ sed -i s/nu/nuS/g readMechanicalProperties.H`
- `[myInterFsiFoam]$ sed -i s/'"nuS"'/'"nu"'/g readMechanicalProperties.H`
- Open file for editing and change all instances of keyword `rho` to keyword `rhoSolid`.

File: `readStressedFoamControls.H`

```
[myInterFsiFoam]$  sed -i s/mesh/stressMesh/g readStressedFoamControls.H
```

File: `tractionDisplacement/tractionDisplacementFvPatchVectorField.C`

- Open file for editing and change all instances of keyword `rho` with keyword `rhoSolid`

File: `setPressure.H`

- `[myInterFsiFoam]$ sed -i s/'rhoFluid.value()'/rho/g setPressure.H`

File: `Make/files`

- `[myInterFsiFoam]$ sed -i s/interFoam/myInterFsiFoam/g Make/files`
- `[myInterFsiFoam]$ sed -i s/AM_APP/AM_USER_APP/g Make/files`
- Open the file `Make/files` for editing and add the line: `tractionDisplacement/tractionDisplacementFvPatchVectorField.C` as the first line of the file.

File: `Make/options`

- Open `Make/options` in a file editor:
  - Add a \ to the end of line `-I$(LIB_SRC)/finiteVolume/lnInclude`
  - Append the following to the `EXE_INC = \` entries directly after previously edited line:

    ```
    -ItractionDisplacement \
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
    $(WM_DECOMP_INC) \
    -I$(LIB_SRC)/tetDecompositionFiniteElement/lnInclude
    ```

  - Insert the lines

    ```
     -ldynamicFvMesh \
    $(W_DECOMP_LIBS) \
    ```

    between the two last lines of the `EXE_LIBS = \` section of `Make/options`, i.e.,
    `-lfiniteVolume \`
    and
    `-llduSolvers`

Compile with `wake`

```
[myInterFsiFoam]$  wmake
```

Appendix B

Complete sourcecodes

This appendix lists all the source code produced in this report. The information header of each file is omitted from the file content output.

B.1 myIcoFsiFoam

B.1.1 myIcoFsiFoam.C

```
/*---------------------------------------------------------------------------*\
Application
    myIcoFsiFoam
Description
    Solver for weakly coupled fluid-structure interaction problems.
    Based on icoFoam and stressedFoam. Uses FE-moving mesh.
\*---------------------------------------------------------------------------*/
#include "fvCFD.H"
#include "dynamicFvMesh.H"
#include "tractionDisplacementFvPatchVectorField.H"
#include "patchToPatchInterpolation.H"
#include "tetFemMatrices.H"
#include "faceTetPolyPatch.H"
#include "tetPolyPatchInterpolation.H"
#include "fixedValueTetPolyPatchFields.H"
#include "pointFields.H"
#include "volPointInterpolation.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
int main(int argc, char *argv[])
{
#   include "setRootCase.H"
#   include "createTime.H"
#   include "createDynamicFvMesh.H"
#   include "createStressMesh.H"
#   include "createFields.H"
#   include "readMechanicalProperties.H"
#   include "readThermalProperties.H"
#   include "createStressFields.H"
#   include "readCouplingProperties.H"  //
#   include "readTimeControls.H"        //
#   include "initContinuityErrs.H"      //
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    Info<< "\nStarting time loop\n" << endl;
    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;
#       include "readPISOControls.H"
#       include "CourantNo.H"
#       include "readTimeControls.H"
#       include "setDeltaT.H"
//   Main solver code:
#       include "setPressure.H"
#       include "solveSolid.H"
#       include "setMotion.H"
#       include "solveFluid.H"
////////////////////////
        runTime.write();
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
    Info<< "End\n" << endl;
    return(0);
}
// ************************************************************************* //
```

B.1.2 solveFluid.H

```
/*---------------------------------------------------------------------------*\
Application
    icoFoam part of myIcoFsiFoam
Description
    Transient solver for incompressible, laminar flow of Newtonian fluids.
\*---------------------------------------------------------------------------*/
/* removed headers etc. which are all part of myIcoFsiFoam.C. Here's the
   main icoFoam solver code:                                           */
// Note: This is NOT identical with the icoFsiFoam solveFluid.H !!
{
        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );
        solve(UEqn == -fvc::grad(p));
////////// --- PISO loop begins here:
        for (int corr=0; corr<nCorr; corr++)
        {
            volScalarField rUA = 1.0/UEqn.A();
            U = rUA*UEqn.H();
            phi = (fvc::interpolate(U) & mesh.Sf())
                + fvc::ddtPhiCorr(rUA, U, phi);
            adjustPhi(phi, U, p);
            for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
            {
                fvScalarMatrix pEqn
                (
                    fvm::laplacian(rUA, p) == fvc::div(phi)
                );
                pEqn.setReference(pRefCell, pRefValue);
                pEqn.solve();
                if (nonOrth == nNonOrthCorr)
                {
                    phi -= pEqn.flux();
                }
            }
#           include "movingMeshContinuityErrs.H"
            U -= rUA*fvc::grad(p);
            U.correctBoundaryConditions();
        }
}
// ************************************************************************* //
```

B.1.3 solveSolid.H

```
/*---------------------------------------------------------------------------*\
Application
    stressedFoam part of myIcoFsiFoam.
Description
    Transient/steady-state segregated finite-volume solver of linear-elastic,
    small-strain deformation of a solid body, with optional thermal
    diffusion and thermal stresses.
    Simple linear elasticity structural analysis code.
    Solves for the displacement vector field U, also generating the
    stress tensor field sigma.
\*---------------------------------------------------------------------------*/
{
#       include "readStressedFoamControls.H"
        int iCorr = 0;
        scalar initialResidual = 0;
        do
        {
            volTensorField gradU = fvc::grad(Usolid);
            if (thermalStress)
            {
                solve
                (
                    fvm::ddt(Tsolid) == fvm::laplacian(DT, Tsolid)
                );
            }
            fvVectorMatrix UEqn
            (
                fvm::d2dt2(Usolid)
             ==
                fvm::laplacian(2*mu + lambda, Usolid, "laplacian(DU,U)")
              + fvc::div
                (
                    mu*gradU.T() + lambda*(I*tr(gradU)) - (mu + lambda)*gradU,
                    "div(sigma)"
                )
            );
            if (thermalStress)
            {
                UEqn += threeKalpha*fvc::grad(Tsolid);
            }
            initialResidual = UEqn.solve().initialResidual();
        } while (initialResidual > convergenceTolerance && ++iCorr < nCorr);
#       include "calculateStress.H"
}
// ************************************************************************* //
```

B.1.4 readStressedFoamControls.H

```
    const dictionary& stressControl =
        stressMesh.solutionDict().subDict("stressedFoam");
    int nCorr(readInt(stressControl.lookup("nCorrectors")));
    scalar convergenceTolerance(readScalar(stressControl.lookup("U")));
```

B.1.5 readMechanicalProperties.H

```
    Info<< "Reading mechanical properties\n" << endl;
    IOdictionary mechanicalProperties
    (
        IOobject
        (
            "mechanicalProperties",
            runTime.constant(),
            stressMesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    dimensionedScalar rho(mechanicalProperties.lookup("rho"));
    dimensionedScalar rhoE(mechanicalProperties.lookup("E"));
    dimensionedScalar nuS(mechanicalProperties.lookup("nu"));
    Info<< "Normalising E : E/rho\n" << endl;
    dimensionedScalar E = rhoE/rho;
    Info<< "Calculating Lame's coefficients\n" << endl;
    dimensionedScalar mu = E/(2.0*(1.0 + nuS));
    dimensionedScalar lambda = nuS*E/((1.0 + nuS)*(1.0 - 2.0*nuS));
    dimensionedScalar threeK = E/(1.0 - 2.0*nuS);
    Switch planeStress(mechanicalProperties.lookup("planeStress"));
    if (planeStress)
    {
        Info<< "Plane Stress\n" << endl;
        //- change lambda and threeK for plane stress
        lambda = nuS*E/((1.0 + nuS)*(1.0 - nuS));
        threeK = E/(1.0 - nuS);
    }
    else
    {
        Info<< "Plane Strain\n" << endl;
    }
    Info<< "mu = " << mu.value() << " Pa/rho\n";
    Info<< "lambda = " << lambda.value() << " Pa/rho\n";
    Info<< "threeK = " << threeK.value() << " Pa/rho\n";
```

B.1.6 readThermalProperties.H

```
    Info<< "Reading thermal properties\n" << endl;
    IOdictionary thermalProperties
    (
        IOobject
        (
            "thermalProperties",
            runTime.constant(),
            stressMesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    Switch thermalStress(thermalProperties.lookup("thermalStress"));
    dimensionedScalar threeKalpha
    (
        "threeKalpha",
        dimensionSet(0, 2, -2 , -1, 0),
        0
    );
    dimensionedScalar DT
    (
        "DT",
        dimensionSet(0, 2, -1 , 0, 0),
        0
    );
    if (thermalStress)
    {
        dimensionedScalar C(thermalProperties.lookup("C"));
        dimensionedScalar rhoK(thermalProperties.lookup("k"));
        dimensionedScalar alpha(thermalProperties.lookup("alpha"));
        Info<< "Normalising k : k/rho\n" << endl;
        dimensionedScalar k = rhoK/rho;
        Info<< "Calculating thermal coefficients\n" << endl;
        threeKalpha = threeK*alpha;
        DT.value() = (k/C).value();
        Info<< "threeKalpha = " << threeKalpha.value() << " Pa/rho\n";
```

```
}
```

B.1.7 calculateStress.H

```
if (runTime.outputTime())
{
    volTensorField gradU = fvc::grad(Usolid);
    volSymmTensorField sigma =
        rho*(2.0*mu*symm(gradU) + lambda*I*tr(gradU));
    if (thermalStress)
    {
        sigma = sigma - I*(rho*threeKalpha*Tsolid);
    }
    volScalarField sigmaEq
    (
        IOobject
        (
            "sigmaEq",
            runTime.timeName(),
            stressMesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        sqrt((3.0/2.0)*magSqr(dev(sigma)))
    );
    Info<< "Max sigmaEq = " << max(sigmaEq).value()
        << endl;
    volScalarField sigmaxx
    (
        IOobject
        (
            "sigmaxx",
            runTime.timeName(),
            stressMesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        sigma.component(symmTensor::XX)
    );
    volScalarField sigmayy
    (
        IOobject
        (
            "sigmayy",
            runTime.timeName(),
            stressMesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        sigma.component(symmTensor::YY)
    );
    volScalarField sigmazz
    (
        IOobject
        (
            "sigmazz",
            runTime.timeName(),
            stressMesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        sigma.component(symmTensor::ZZ)
    );
    Info<< "Max sigmazz = " << max(sigmazz).value()
        << endl;
    volScalarField sigmaxy
    (
        IOobject
        (
            "sigmaxy",
            runTime.timeName(),
            stressMesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        sigma.component(symmTensor::XY)
    );
    volScalarField sigmaxz
    (
        IOobject
        (
            "sigmaxz",
            runTime.timeName(),
            stressMesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        sigma.component(symmTensor::XZ)
    );
    volScalarField sigmayz
    (
        IOobject
        (
            "sigmayz",
            runTime.timeName(),
            stressMesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        sigma.component(symmTensor::YZ)
    );
    runTime.write();
}
```

B.1.8 createStressMesh.H

```
Foam::Info<< "Create mesh for time = "
<< runTime.timeName() << Foam::nl << Foam::endl;
Foam::fvMesh stressMesh
(
    Foam::IOobject
    (
        "solid",
        runTime.timeName(),
        runTime,
        Foam::IOobject::MUST_READ
    )
);
Foam::pointMesh pStressMesh(stressMesh);
Foam::volPointInterpolation cpi
(
    stressMesh,
    pStressMesh
);
```

B.1.9 createStressFields.H

```
Info<< "Reading field U\n" << endl;
volVectorField Usolid
(
    IOobject
    (
        "U",
        runTime.timeName(),
        stressMesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    stressMesh
);
volScalarField* Tptr = NULL;
if (thermalStress)
{
    Info<< "Reading field T\n" << endl;
    Tptr = new volScalarField
    (
        IOobject
        (
            "T",
            runTime.timeName(),
            stressMesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        stressMesh
    );
}
volScalarField& Tsolid = *Tptr;
```

B.1.10 createFields.H

```
    Info<< "Reading transportProperties\n" << endl;
    IOdictionary transportProperties
    (
        IOobject
        (
            "transportProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    dimensionedScalar nu
    (
        transportProperties.lookup("nu")
    );
    dimensionedScalar rhoFluid
    (
        transportProperties.lookup("rho")
    );
    Info<< "Reading field p\n" << endl;
    volScalarField p
    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
    Info<< "Reading field U\n" << endl;
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
#   include "createPhi.H"
    label pRefCell = 0;
    scalar pRefValue = 0.0;
    setRefCell(p, mesh.solutionDict().subDict("PISO"), pRefCell, pRefValue);
```

B.1.11 setPressure.H

```
{
    // Setting pressure on solid patch
    Info << "Setting pressure" << endl;
    scalarField solidPatchPressure =
        interpolatorFluidSolid.faceInterpolate
        (
            p.boundaryField()[fluidPatchID]
        );
    solidPatchPressure *= rhoFluid.value();
    tForce.pressure() = solidPatchPressure;
    vector totalPressureForce =
        sum
        (
            p.boundaryField()[fluidPatchID]*
            mesh.Sf().boundaryField()[fluidPatchID]
        );
    Info << "Total pressure force = " << totalPressureForce << endl;
}
```

B.1.12 setMotion.H

```
{
    // Setting mesh motion
    pointVectorField solidPointsDispl =
        cpi.interpolate(Usolid - Usolid.oldTime());
    vectorField newPoints =
        stressMesh.points()
      + solidPointsDispl.internalField();
    stressMesh.movePoints(newPoints);
    vectorField fluidPatchPointsDispl =
        interpolatorSolidFluid.pointInterpolate
        (
            solidPointsDispl.boundaryField()[solidPatchID].
            patchInternalField()
        );
    motionUFluidPatch ==
        tppi.pointToPointInterpolate
        (
            fluidPatchPointsDispl/runTime.deltaT().value()
        );
    mesh.update();
#   include "volContinuity.H"
    Info << "Motion magnitude: mean = "
        << average(mag(Usolid.boundaryField()[solidPatchID]))
        << " max = "
        << max(mag(Usolid.boundaryField()[solidPatchID])) << endl;
}
```

B.1.13 readCouplingProperties.H

```
Info << "Reading coupling properties" << endl;
IOdictionary couplingProperties
(
    IOobject
    (
        "couplingProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ,
        IOobject::NO_WRITE
    )
);
// Read solid patch data
word solidPatchName(couplingProperties.lookup("solidPatch"));
label solidPatchID =
    stressMesh.boundaryMesh().findPatchID(solidPatchName);
// Read fluid patch data
word fluidPatchName(couplingProperties.lookup("fluidPatch"));
label fluidPatchID =
    mesh.boundaryMesh().findPatchID(fluidPatchName);
if (solidPatchID < 0 || fluidPatchID < 0)
{
    FatalErrorIn(args.executable())
        << "Problem with patch interpolation definition"
        << abort(FatalError);
}
// Create interpolators
patchToPatchInterpolation interpolatorFluidSolid
(
    mesh.boundaryMesh()[fluidPatchID],
    stressMesh.boundaryMesh()[solidPatchID]
);
patchToPatchInterpolation interpolatorSolidFluid
(
    stressMesh.boundaryMesh()[solidPatchID],
    mesh.boundaryMesh()[fluidPatchID]
);
// Grab solid patch field
tractionDisplacementFvPatchVectorField& tForce =
    refCast<tractionDisplacementFvPatchVectorField>
    (
        Usolid.boundaryField()[solidPatchID]
    );
// Grab motion field
// Read fluid patch data
word movingRegionName(couplingProperties.lookup("movingRegion"));
const fvMesh& motionMesh =
    runTime.objectRegistry::lookupObject<fvMesh>(movingRegionName);
tetPointVectorField& motionU =
    const_cast<tetPointVectorField&>
    (
        motionMesh.objectRegistry::lookupObject<tetPointVectorField>
        (
            "motionU"
        )
    );
fixedValueTetPolyPatchVectorField& motionUFluidPatch =
    refCast<fixedValueTetPolyPatchVectorField>
    (
        motionU.boundaryField()[fluidPatchID]
    );
tetPolyPatchInterpolation tppi
(
    refCast<const faceTetPolyPatch>(motionUFluidPatch.patch())
);
```

B.1.14 tractionDisplacementFvPatchVectorField.C

```
\*---------------------------------------------------------------------------*/
#include "tractionDisplacementFvPatchVectorField.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_(p.size(), vector::zero),
    pressure_(p.size(), 0.0)
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    fixedGradientFvPatchVectorField(tdpvf, p, iF, mapper),
    traction_(tdpvf.traction_, mapper),
    pressure_(tdpvf.pressure_, mapper)
{}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_("traction", dict, p.size()),
    pressure_("pressure", dict, p.size())
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf
)
:
    fixedGradientFvPatchVectorField(tdpvf),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const DimensionedField<vector, volMesh>& iF
)
```

```
:
    fixedGradientFvPatchVectorField(tdpvf, iF),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
void tractionDisplacementFvPatchVectorField::autoMap
(
    const fvPatchFieldMapper& m
)
{
    fixedGradientFvPatchVectorField::autoMap(m);
    traction_.autoMap(m);
    pressure_.autoMap(m);
}
// Reverse-map the given fvPatchField onto this fvPatchField
void tractionDisplacementFvPatchVectorField::rmap
(
    const fvPatchVectorField& ptf,
    const labelList& addr
)
{
    fixedGradientFvPatchVectorField::rmap(ptf, addr);
    const tractionDisplacementFvPatchVectorField& dmptf =
        refCast<const tractionDisplacementFvPatchVectorField>(ptf);
    traction_.rmap(dmptf.traction_, addr);
    pressure_.rmap(dmptf.pressure_, addr);
}
// Update the coefficients associated with the patch field
void tractionDisplacementFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }
    const dictionary& mechanicalProperties =
        db().lookupObject<IOdictionary>("mechanicalProperties");
    const dictionary& thermalProperties =
        db().lookupObject<IOdictionary>("thermalProperties");
    dimensionedScalar rho(mechanicalProperties.lookup("rho"));
    dimensionedScalar rhoE(mechanicalProperties.lookup("E"));
    dimensionedScalar nu(mechanicalProperties.lookup("nu"));
    dimensionedScalar E = rhoE/rho;
    dimensionedScalar mu = E/(2.0*(1.0 + nu));
    dimensionedScalar lambda = nu*E/((1.0 + nu)*(1.0 - 2.0*nu));
    dimensionedScalar threeK = E/(1.0 - 2.0*nu);
    Switch planeStress(mechanicalProperties.lookup("planeStress"));
    if (planeStress)
    {
        lambda = nu*E/((1.0 + nu)*(1.0 - nu));
        threeK = E/(1.0 - nu);
    }
    vectorField n = patch().nf();
    const fvPatchField<tensor>& gradU =
        patch().lookupPatchField<volTensorField, tensor>("grad(U)");
    gradient() =
    (
        (traction_ - pressure_*n)/rho.value()
      - (n & (mu.value()*gradU.T() - (mu + lambda).value()*gradU))
      - n*tr(gradU)*lambda.value()
    )/(2.0*mu + lambda).value();
    Switch thermalStress(thermalProperties.lookup("thermalStress"));
    if (thermalStress)
    {
        dimensionedScalar alpha(thermalProperties.lookup("alpha"));
        dimensionedScalar threeKalpha = threeK*alpha;
        const fvPatchField<scalar>& T =
            patch().lookupPatchField<volScalarField, scalar>("T");
        gradient() += n*threeKalpha.value()*T/(2.0*mu + lambda).value();
    }
    fixedGradientFvPatchVectorField::updateCoeffs();
}
// Write
void tractionDisplacementFvPatchVectorField::write(Ostream& os) const
{
    fvPatchVectorField::write(os);
    traction_.writeEntry("traction", os);
    pressure_.writeEntry("pressure", os);
    writeEntry("value", os);
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
makePatchTypeField(fvPatchVectorField, tractionDisplacementFvPatchVectorField);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
} // End namespace Foam
// ************************************************************************* //
```

B.1.15 tractionDisplacementFvPatchVectorField.H

```
\*---------------------------------------------------------------------------*/
#include "tractionDisplacementFvPatchVectorField.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_(p.size(), vector::zero),
    pressure_(p.size(), 0.0)
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    fixedGradientFvPatchVectorField(tdpvf, p, iF, mapper),
    traction_(tdpvf.traction_, mapper),
    pressure_(tdpvf.pressure_, mapper)
{}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_("traction", dict, p.size()),
    pressure_("pressure", dict, p.size())
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf
)
:
    fixedGradientFvPatchVectorField(tdpvf),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedGradientFvPatchVectorField(tdpvf, iF),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
void tractionDisplacementFvPatchVectorField::autoMap
(
    const fvPatchFieldMapper& m
)
{
    fixedGradientFvPatchVectorField::autoMap(m);
    traction_.autoMap(m);
    pressure_.autoMap(m);
}
// Reverse-map the given fvPatchField onto this fvPatchField
void tractionDisplacementFvPatchVectorField::rmap
(
    const fvPatchVectorField& ptf,
    const labelList& addr
)
{
    fixedGradientFvPatchVectorField::rmap(ptf, addr);
    const tractionDisplacementFvPatchVectorField& dmptf =
        refCast<const tractionDisplacementFvPatchVectorField>(ptf);
    traction_.rmap(dmptf.traction_, addr);
    pressure_.rmap(dmptf.pressure_, addr);
}
// Update the coefficients associated with the patch field
void tractionDisplacementFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }
    const dictionary& mechanicalProperties =
        db().lookupObject<IOdictionary>("mechanicalProperties");
    const dictionary& thermalProperties =
        db().lookupObject<IOdictionary>("thermalProperties");
    dimensionedScalar rho(mechanicalProperties.lookup("rho"));
    dimensionedScalar rhoE(mechanicalProperties.lookup("E"));
    dimensionedScalar nu(mechanicalProperties.lookup("nu"));
    dimensionedScalar E = rhoE/rho;
    dimensionedScalar mu = E/(2.0*(1.0 + nu));
    dimensionedScalar lambda = nu*E/((1.0 + nu)*(1.0 - 2.0*nu));
    dimensionedScalar threeK = E/(1.0 - 2.0*nu);
    Switch planeStress(mechanicalProperties.lookup("planeStress"));
    if (planeStress)
    {
        lambda = nu*E/((1.0 + nu)*(1.0 - nu));
        threeK = E/(1.0 - nu);
    }
    vectorField n = patch().nf();
    const fvPatchField<tensor>& gradU =
        patch().lookupPatchField<volTensorField, tensor>("grad(U)");
    gradient() =
    (
        (traction_ - pressure_*n)/rho.value()
      - (n & (mu.value()*gradU.T() - (mu + lambda).value()*gradU))
      - n*tr(gradU)*lambda.value()
    )/(2.0*mu + lambda).value();
    Switch thermalStress(thermalProperties.lookup("thermalStress"));
    if (thermalStress)
    {
        dimensionedScalar alpha(thermalProperties.lookup("alpha"));
        dimensionedScalar threeKalpha = threeK*alpha;
        const fvPatchField<scalar>& T =
            patch().lookupPatchField<volScalarField, scalar>("T");
        gradient() += n*threeKalpha.value()*T/(2.0*mu + lambda).value();
    }
    fixedGradientFvPatchVectorField::updateCoeffs();
}
// Write
void tractionDisplacementFvPatchVectorField::write(Ostream& os) const
{
    fvPatchVectorField::write(os);
    traction_.writeEntry("traction", os);
    pressure_.writeEntry("pressure", os);
    writeEntry("value", os);
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
makePatchTypeField(fvPatchVectorField, tractionDisplacementFvPatchVectorField);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
} // End namespace Foam
// ************************************************************************* //
```

B.1.16 Make/options

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -ItractionDisplacement \
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
    $(WM_DECOMP_INC) \
    -I$(LIB_SRC)/tetDecompositionFiniteElement/lnInclude


EXE_LIBS = \
    -lfiniteVolume \
    -ldynamicFvMesh \
    $(W_DECOMP_LIBS) \
    -llduSolvers
```

B.1.17 Make/files

```
tractionDisplacement/tractionDisplacementFvPatchVectorField.C
myIcoFsiFoam.C

EXE = $(FOAM_USER_APPBIN)/myIcoFsiFoam
```

B.2 myInterFsiFoam

B.2.1 myInterFsiFoam.C

```
/*---------------------------------------------------------------------------*\
Application
    myInterFsiFoam
Description
    FSI solver for weakly coupled interaction.
    Combines interFoam and stressedFoam.
\*---------------------------------------------------------------------------*/
#include "fvCFD.H"
#include "MULES.H"
#include "subCycle.H"
#include "interfaceProperties.H"
#include "twoPhaseMixture.H"
//From icoFsiFoam:
//moving mesh
#include "dynamicFvMesh.H"
//local traction
#include "tractionDisplacementFvPatchVectorField.H"
//FE decomposition
#include "patchToPatchInterpolation.H"
#include "tetFemMatrices.H"
#include "faceTetPolyPatch.H"
#include "tetPolyPatchInterpolation.H"
#include "fixedValueTetPolyPatchFields.H"
#include "pointFields.H"
#include "volPointInterpolation.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
#   include "createDynamicFvMesh.H"
#   include "createStressMesh.H"
    #include "readEnvironmentalProperties.H"
    #include "readPISOControls.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
//From icoFsiFoam
#   include "readMechanicalProperties.H"
#   include "readThermalProperties.H"
#   include "createStressFields.H"
#   include "readCouplingProperties.H"  //
    #include "readTimeControls.H"
    #include "correctPhi.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    Info<< "\nStarting time loop\n" << endl;
    while (runTime.run())
    {
        #include "readPISOControls.H"
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"
        runTime++;
        Info<< "Time = " << runTime.timeName() << nl << endl;
//Main solid and mesh solvers - icoFsiFoam setup
#       include "setPressure.H"
#       include "solveSolid.H"
#       include "setMotion.H"
        //Internal interFoam loop
#       include "solveFluids.H"
//////////////////////////////////
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
    Info<< "End\n" << endl;
    return(0);
}
// ************************************************************************* //
```

B.2.2 solveFluid.H

```
/*---------------------------------------------------------------------------*\
Application
    interFoam main solver
Description
    Solver for 2 incompressible, isothermal immiscible fluids using a VOF
    (volume of fluid) phase-fraction based interface capturing approach.
    The momentum and other fluid properties are of the "mixture" and a single
    momentum equation is solved.
    For a two-fluid approach see twoPhaseEulerFoam.
\*---------------------------------------------------------------------------*/
{
        twoPhaseProperties.correct();
        #include "gammaEqnSubCycle.H"
        #include "UEqn.H"
        // --- PISO loop
        for (int corr=0; corr<nCorr; corr++)
        {
            #include "pEqn.H"
        }
        #include "movingMeshContinuityErrs.H"
        p = pd + rho*gh;
        if (pd.needReference())
        {
            p += dimensionedScalar
            (
                "p",
                p.dimensions(),
                pRefValue - getRefCellValue(p, pdRefCell)
            );
        }
        runTime.write();
}
// ************************************************************************* //
```

B.2.3 solveSolid.H

```
/*---------------------------------------------------------------------------*\
Application
    stressedFoam part of myIcoFsiFoam.
Description
    Transient/steady-state segregated finite-volume solver of linear-elastic,
    small-strain deformation of a solid body, with optional thermal
    diffusion and thermal stresses.
    Simple linear elasticity structural analysis code.
    Solves for the displacement vector field U, also generating the
    stress tensor field sigma.
\*---------------------------------------------------------------------------*/
{
#       include "readStressedFoamControls.H"
        int iCorr = 0;
        scalar initialResidual = 0;
        do
        {
            volTensorField gradU = fvc::grad(Usolid);
            if (thermalStress)
            {
                solve
                (
                    fvm::ddt(Tsolid) == fvm::laplacian(DT, Tsolid)
                );
            }
            fvVectorMatrix UEqn
            (
                fvm::d2dt2(Usolid)
             ==
                fvm::laplacian(2*mu + lambda, Usolid, "laplacian(DU,U)")
              + fvc::div
                (
                    mu*gradU.T() + lambda*(I*tr(gradU)) - (mu + lambda)*gradU,
                    "div(sigma)"
                )
            );
            if (thermalStress)
            {
                UEqn += threeKalpha*fvc::grad(Tsolid);
            }
            initialResidual = UEqn.solve().initialResidual();
        } while (initialResidual > convergenceTolerance && ++iCorr < nCorr);
#       include "calculateStress.H"
}
// ************************************************************************* //
```

B.2.4 readStressedFoamControls.H

```
    const dictionary& stressControl =
        stressMesh.solutionDict().subDict("stressedFoam");
    int nCorr(readInt(stressControl.lookup("nCorrectors")));
    scalar convergenceTolerance(readScalar(stressControl.lookup("U")));
```

B.2.5 readMechanicalProperties.H

```
    Info<< "Reading mechanical properties\n" << endl;
    IOdictionary mechanicalProperties
    (
        IOobject
        (
            "mechanicalProperties",
            runTime.constant(),
            stressMesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    dimensionedScalar rhoSolid(mechanicalProperties.lookup("rho"));
    dimensionedScalar rhoE(mechanicalProperties.lookup("E"));
    dimensionedScalar nuS(mechanicalProperties.lookup("nu"));
    Info<< "Normalising E : E/rho\n" << endl;
    dimensionedScalar E = rhoE/rhoSolid;
    Info<< "Calculating Lame’s coefficients\n" << endl;
    dimensionedScalar mu = E/(2.0*(1.0 + nuS));
    dimensionedScalar lambda = nuS*E/((1.0 + nuS)*(1.0 - 2.0*nuS));
    dimensionedScalar threeK = E/(1.0 - 2.0*nuS);
    Switch planeStress(mechanicalProperties.lookup("planeStress"));
    if (planeStress)
    {
        Info<< "Plane Stress\n" << endl;
        //- change lambda and threeK for plane stress
        lambda = nuS*E/((1.0 + nuS)*(1.0 - nuS));
        threeK = E/(1.0 - nuS);
    }
    else
    {
        Info<< "Plane Strain\n" << endl;
    }
    Info<< "mu = " << mu.value() << " Pa/rho\n";
    Info<< "lambda = " << lambda.value() << " Pa/rho\n";
    Info<< "threeK = " << threeK.value() << " Pa/rho\n";
```

B.2.6 readThermalProperties.H

```
    Info<< "Reading thermal properties\n" << endl;
    IOdictionary thermalProperties
    (
        IOobject
        (
            "thermalProperties",
            runTime.constant(),
            stressMesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    Switch thermalStress(thermalProperties.lookup("thermalStress"));
    dimensionedScalar threeKalpha
    (
        "threeKalpha",
        dimensionSet(0, 2, -2 , -1, 0),
        0
    );
    dimensionedScalar DT
    (
        "DT",
        dimensionSet(0, 2, -1 , 0, 0),
        0
    );
    if (thermalStress)
    {
        dimensionedScalar C(thermalProperties.lookup("C"));
        dimensionedScalar rhoK(thermalProperties.lookup("k"));
        dimensionedScalar alpha(thermalProperties.lookup("alpha"));
        Info<< "Normalising k : k/rho\n" << endl;
        dimensionedScalar k = rhoK/rhoSolid;
        Info<< "Calculating thermal coefficients\n" << endl;
        threeKalpha = threeK*alpha;
        DT.value() = (k/C).value();
        Info<< "threeKalpha = " << threeKalpha.value() << " Pa/rho\n";
    }
```

B.2.7 calculateStress.H

```
    if (runTime.outputTime())
    {
        volTensorField gradU = fvc::grad(Usolid);
        volSymmTensorField sigma =
            rhoSolid*(2.0*mu*symm(gradU) + lambda*I*tr(gradU));
        if (thermalStress)
        {
            sigma = sigma - I*(rhoSolid*threeKalpha*Tsolid);
        }
        volScalarField sigmaEq
        (
            IOobject
            (
                "sigmaEq",
                runTime.timeName(),
                stressMesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            sqrt((3.0/2.0)*magSqr(dev(sigma)))
        );
        Info<< "Max sigmaEq = " << max(sigmaEq).value()
            << endl;
        volScalarField sigmaxx
        (
            IOobject
            (
                "sigmaxx",
                runTime.timeName(),
                stressMesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            sigma.component(symmTensor::XX)
        );
        volScalarField sigmayy
        (
            IOobject
            (
                "sigmayy",
                runTime.timeName(),
                stressMesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            sigma.component(symmTensor::YY)
        );
        volScalarField sigmazz
        (
            IOobject
            (
                "sigmazz",
                runTime.timeName(),
                stressMesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            sigma.component(symmTensor::ZZ)
        );
        Info<< "Max sigmazz = " << max(sigmazz).value()
            << endl;
        volScalarField sigmaxy
        (
            IOobject
            (
                "sigmaxy",
                runTime.timeName(),
                stressMesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            sigma.component(symmTensor::XY)
        );
        volScalarField sigmaxz
        (
            IOobject
            (
                "sigmaxz",
                runTime.timeName(),
                stressMesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            sigma.component(symmTensor::XZ)
        );
        volScalarField sigmayz
        (
            IOobject
            (
                "sigmayz",
                runTime.timeName(),
                stressMesh,
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            sigma.component(symmTensor::YZ)
        );
        runTime.write();
    }
```

B.2.8 createStressMesh.H

```
    Foam::Info<< "Create mesh for time = "
    << runTime.timeName() << Foam::nl << Foam::endl;
        Foam::fvMesh stressMesh
    (
        Foam::IOobject
        (
            "solid",
            runTime.timeName(),
            runTime,
            Foam::IOobject::MUST_READ
        )
    );
    Foam::pointMesh pStressMesh(stressMesh);
    Foam::volPointInterpolation cpi
    (
        stressMesh,
        pStressMesh
    );
```

B.2.9 createStressFields.H

```
    Info<< "Reading field U\n" << endl;
    volVectorField Usolid
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            stressMesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        stressMesh
    );
    volScalarField* Tptr = NULL;
    if (thermalStress)
    {
        Info<< "Reading field T\n" << endl;
        Tptr = new volScalarField
        (
            IOobject
            (
                "T",
                runTime.timeName(),
                stressMesh,
                IOobject::MUST_READ,
                IOobject::AUTO_WRITE
            ),
            stressMesh
        );
    }
    volScalarField& Tsolid = *Tptr;
```

B.2.10 createFields.H

```
    Info<< "Reading field pd\n" << endl;
    volScalarField pd
    (
        IOobject
        (
            "pd",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
    Info<< "Reading field gamma\n" << endl;
    volScalarField gamma
    (
        IOobject
        (
            "gamma",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
    Info<< "Reading field U\n" << endl;
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
#   include "createPhi.H"
    Info<< "Reading transportProperties\n" << endl;
    twoPhaseMixture twoPhaseProperties(U, phi, "gamma");
    const dimensionedScalar& rho1 = twoPhaseProperties.rho1();
    const dimensionedScalar& rho2 = twoPhaseProperties.rho2();
// Need to store rho for ddt(rho, U)
    volScalarField rho
    (
        IOobject
        (
            "rho",
            runTime.timeName(),
            mesh,
            IOobject::READ_IF_PRESENT
        ),
        gamma*rho1 + (scalar(1) - gamma)*rho2,
        gamma.boundaryField().types()
    );
    rho.oldTime();
// Mass flux
// Initialisation does not matter because rhoPhi is reset after the
// gamma solution before it is used in the U equation.
    surfaceScalarField rhoPhi
    (
        IOobject
        (
            "rho*phi",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        rho1*phi
    );
    Info<< "Calculating field g.h\n" << endl;
    volScalarField gh("gh", g & mesh.C());
    surfaceScalarField ghf("gh", g & mesh.Cf());
    volScalarField p
    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        pd + rho*gh
    );
    label pdRefCell = 0;
    scalar pdRefValue = 0.0;
    setRefCell(pd, mesh.solutionDict().subDict("PISO"), pdRefCell, pdRefValue);
    scalar pRefValue = 0.0;
    if (pd.needReference())
    {
        pRefValue = readScalar
        (
            mesh.solutionDict().subDict("PISO").lookup("pRefValue")
        );
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pdRefCell)
        );
    }
// Construct interface from gamma distribution
    interfaceProperties interface(gamma, U, twoPhaseProperties);
```

B.2.11 setPressure.H

```
{
    // Setting pressure on solid patch
    Info << "Setting pressure" << endl;
    scalarField solidPatchPressure =
        interpolatorFluidSolid.faceInterpolate
        (
            p.boundaryField()[fluidPatchID]
        );
    solidPatchPressure *= rho;
       tForce.pressure() = solidPatchPressure;
    vector totalPressureForce =
        sum
        (
            p.boundaryField()[fluidPatchID]*
            mesh.Sf().boundaryField()[fluidPatchID]
        );
    Info << "Total pressure force = " << totalPressureForce << endl;
}
```

B.2.12 setMotion.H

```
{
    // Setting mesh motion
    pointVectorField solidPointsDispl =
        cpi.interpolate(Usolid - Usolid.oldTime());
    vectorField newPoints =
        stressMesh.points()
      + solidPointsDispl.internalField();
    stressMesh.movePoints(newPoints);
    vectorField fluidPatchPointsDispl =
        interpolatorSolidFluid.pointInterpolate
        (
            solidPointsDispl.boundaryField()[solidPatchID].
            patchInternalField()
        );
    motionUFluidPatch ==
        tppi.pointToPointInterpolate
        (
            fluidPatchPointsDispl/runTime.deltaT().value()
        );
    mesh.update();
#   include "volContinuity.H"
    Info << "Motion magnitude: mean = "
        << average(mag(Usolid.boundaryField()[solidPatchID]))
        << " max = "
        << max(mag(Usolid.boundaryField()[solidPatchID])) << endl;
}
```

B.2.13 readCouplingProperties.H

```
    Info << "Reading coupling properties" << endl;
    IOdictionary couplingProperties
    (
        IOobject
        (
            "couplingProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );
    // Read solid patch data
    word solidPatchName(couplingProperties.lookup("solidPatch"));
    label solidPatchID =
        stressMesh.boundaryMesh().findPatchID(solidPatchName);
    // Read fluid patch data
    word fluidPatchName(couplingProperties.lookup("fluidPatch"));
    label fluidPatchID =
        mesh.boundaryMesh().findPatchID(fluidPatchName);
    if (solidPatchID < 0 || fluidPatchID < 0)
    {
        FatalErrorIn(args.executable())
            << "Problem with patch interpolation definition"
            << abort(FatalError);
    }
    // Create interpolators
    patchToPatchInterpolation interpolatorFluidSolid
    (
        mesh.boundaryMesh()[fluidPatchID],
        stressMesh.boundaryMesh()[solidPatchID]
    );
    patchToPatchInterpolation interpolatorSolidFluid
    (
        stressMesh.boundaryMesh()[solidPatchID],
        mesh.boundaryMesh()[fluidPatchID]
    );
    // Grab solid patch field
    tractionDisplacementFvPatchVectorField& tForce =
        refCast<tractionDisplacementFvPatchVectorField>
        (
            Usolid.boundaryField()[solidPatchID]
        );
    // Grab motion field
    // Read fluid patch data
    word movingRegionName(couplingProperties.lookup("movingRegion"));
    const fvMesh& motionMesh =
        runTime.objectRegistry::lookupObject<fvMesh>(movingRegionName);
    tetPointVectorField& motionU =
        const_cast<tetPointVectorField&>
        (
            motionMesh.objectRegistry::lookupObject<tetPointVectorField>
            (
                "motionU"
            )
        );
    fixedValueTetPolyPatchVectorField& motionUFluidPatch =
        refCast<fixedValueTetPolyPatchVectorField>
        (
            motionU.boundaryField()[fluidPatchID]
        );
    tetPolyPatchInterpolation tppi
    (
        refCast<const faceTetPolyPatch>(motionUFluidPatch.patch())
    );
```

B.2.14 tractionDisplacementFvPatchVectorField.C

```
\*---------------------------------------------------------------------------*/
#include "tractionDisplacementFvPatchVectorField.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_(p.size(), vector::zero),
    pressure_(p.size(), 0.0)
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    fixedGradientFvPatchVectorField(tdpvf, p, iF, mapper),
    traction_(tdpvf.traction_, mapper),
    pressure_(tdpvf.pressure_, mapper)
{}
```

```
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_("traction", dict, p.size()),
    pressure_("pressure", dict, p.size())
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf
)
:
    fixedGradientFvPatchVectorField(tdpvf),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedGradientFvPatchVectorField(tdpvf, iF),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
// * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
void tractionDisplacementFvPatchVectorField::autoMap
(
    const fvPatchFieldMapper& m
)
{
    fixedGradientFvPatchVectorField::autoMap(m);
    traction_.autoMap(m);
    pressure_.autoMap(m);
}
// Reverse-map the given fvPatchField onto this fvPatchField
void tractionDisplacementFvPatchVectorField::rmap
(
    const fvPatchVectorField& ptf,
    const labelList& addr
)
{
    fixedGradientFvPatchVectorField::rmap(ptf, addr);
    const tractionDisplacementFvPatchVectorField& dmptf =
        refCast<const tractionDisplacementFvPatchVectorField>(ptf);
    traction_.rmap(dmptf.traction_, addr);
    pressure_.rmap(dmptf.pressure_, addr);
}
// Update the coefficients associated with the patch field
void tractionDisplacementFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }
    const dictionary& mechanicalProperties =
        db().lookupObject<IOdictionary>("mechanicalProperties");
    const dictionary& thermalProperties =
        db().lookupObject<IOdictionary>("thermalProperties");
    dimensionedScalar rhoSolid(mechanicalProperties.lookup("rho"));
    dimensionedScalar rhoE(mechanicalProperties.lookup("E"));
    dimensionedScalar nu(mechanicalProperties.lookup("nu"));
    dimensionedScalar E = rhoE/rhoSolid;
    dimensionedScalar mu = E/(2.0*(1.0 + nu));
    dimensionedScalar lambda = nu*E/((1.0 + nu)*(1.0 - 2.0*nu));
    dimensionedScalar threeK = E/(1.0 - 2.0*nu);
    Switch planeStress(mechanicalProperties.lookup("planeStress"));
    if (planeStress)
    {
        lambda = nu*E/((1.0 + nu)*(1.0 - nu));
        threeK = E/(1.0 - nu);
    }
    vectorField n = patch().nf();
    const fvPatchField<tensor>& gradU =
        patch().lookupPatchField<volTensorField, tensor>("grad(U)");
    gradient() =
    (
        (traction_ - pressure_*n)/rhoSolid.value()
      - (n & (mu.value()*gradU.T() - (mu + lambda).value()*gradU))
      - n*tr(gradU)*lambda.value()
    )/(2.0*mu + lambda).value();
    Switch thermalStress(thermalProperties.lookup("thermalStress"));
    if (thermalStress)
    {
        dimensionedScalar alpha(thermalProperties.lookup("alpha"));
        dimensionedScalar threeKalpha = threeK*alpha;
        const fvPatchField<scalar>& T =
            patch().lookupPatchField<volScalarField, scalar>("T");
        gradient() += n*threeKalpha.value()*T/(2.0*mu + lambda).value();
    }
    fixedGradientFvPatchVectorField::updateCoeffs();
}
// Write
void tractionDisplacementFvPatchVectorField::write(Ostream& os) const
{
    fvPatchVectorField::write(os);
    traction_.writeEntry("traction", os);
    pressure_.writeEntry("pressure", os);
    writeEntry("value", os);
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
makePatchTypeField(fvPatchVectorField, tractionDisplacementFvPatchVectorField);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
} // End namespace Foam
// ************************************************************************* //
```

B.2.15 tractionDisplacementFvPatchVectorField.H

```
\*---------------------------------------------------------------------------*/
#include "tractionDisplacementFvPatchVectorField.H"
#include "addToRunTimeSelectionTable.H"
#include "volFields.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
namespace Foam
{
// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_(p.size(), vector::zero),
    pressure_(p.size(), 0.0)
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const fvPatchFieldMapper& mapper
)
:
    fixedGradientFvPatchVectorField(tdpvf, p, iF, mapper),
    traction_(tdpvf.traction_, mapper),
    pressure_(tdpvf.pressure_, mapper)
{}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const fvPatch& p,
    const DimensionedField<vector, volMesh>& iF,
    const dictionary& dict
)
:
    fixedGradientFvPatchVectorField(p, iF),
    traction_("traction", dict, p.size()),
    pressure_("pressure", dict, p.size())
{
    fvPatchVectorField::operator=(patchInternalField());
    gradient() = vector::zero;
}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf
)
:
    fixedGradientFvPatchVectorField(tdpvf),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
tractionDisplacementFvPatchVectorField::
tractionDisplacementFvPatchVectorField
(
    const tractionDisplacementFvPatchVectorField& tdpvf,
    const DimensionedField<vector, volMesh>& iF
)
:
    fixedGradientFvPatchVectorField(tdpvf, iF),
    traction_(tdpvf.traction_),
    pressure_(tdpvf.pressure_)
{}
// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //
void tractionDisplacementFvPatchVectorField::autoMap
(
    const fvPatchFieldMapper& m
)
{
    fixedGradientFvPatchVectorField::autoMap(m);
    traction_.autoMap(m);
    pressure_.autoMap(m);
}
// Reverse-map the given fvPatchField onto this fvPatchField
void tractionDisplacementFvPatchVectorField::rmap
(
    const fvPatchVectorField& ptf,
    const labelList& addr
)
{
    fixedGradientFvPatchVectorField::rmap(ptf, addr);
    const tractionDisplacementFvPatchVectorField& dmptf =
        refCast<const tractionDisplacementFvPatchVectorField>(ptf);
    traction_.rmap(dmptf.traction_, addr);
    pressure_.rmap(dmptf.pressure_, addr);
}
// Update the coefficients associated with the patch field
void tractionDisplacementFvPatchVectorField::updateCoeffs()
{
    if (updated())
    {
        return;
    }
    const dictionary& mechanicalProperties =
        db().lookupObject<IOdictionary>("mechanicalProperties");
    const dictionary& thermalProperties =
        db().lookupObject<IOdictionary>("thermalProperties");
    dimensionedScalar rhoSolid(mechanicalProperties.lookup("rho"));
    dimensionedScalar rhoE(mechanicalProperties.lookup("E"));
    dimensionedScalar nu(mechanicalProperties.lookup("nu"));
    dimensionedScalar E = rhoE/rhoSolid;
    dimensionedScalar mu = E/(2.0*(1.0 + nu));
    dimensionedScalar lambda = nu*E/((1.0 + nu)*(1.0 - 2.0*nu));
    dimensionedScalar threeK = E/(1.0 - 2.0*nu);
    Switch planeStress(mechanicalProperties.lookup("planeStress"));
    if (planeStress)
    {
        lambda = nu*E/((1.0 + nu)*(1.0 - nu));
        threeK = E/(1.0 - nu);
    }
    vectorField n = patch().nf();
    const fvPatchField<tensor>& gradU =
        patch().lookupPatchField<volTensorField, tensor>("grad(U)");
    gradient() =
    (
        (traction_ - pressure_*n)/rhoSolid.value()
      - (n & (mu.value()*gradU.T() - (mu + lambda).value()*gradU))
      - n*tr(gradU)*lambda.value()
    )/(2.0*mu + lambda).value();
    Switch thermalStress(thermalProperties.lookup("thermalStress"));
    if (thermalStress)
    {
        dimensionedScalar alpha(thermalProperties.lookup("alpha"));
        dimensionedScalar threeKalpha = threeK*alpha;
        const fvPatchField<scalar>& T =
            patch().lookupPatchField<volScalarField, scalar>("T");
        gradient() += n*threeKalpha.value()*T/(2.0*mu + lambda).value();
    }
    fixedGradientFvPatchVectorField::updateCoeffs();
}
// Write
void tractionDisplacementFvPatchVectorField::write(Ostream& os) const
{
    fvPatchVectorField::write(os);
    traction_.writeEntry("traction", os);
    pressure_.writeEntry("pressure", os);
    writeEntry("value", os);
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
makePatchTypeField(fvPatchVectorField, tractionDisplacementFvPatchVectorField);
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
} // End namespace Foam
// ************************************************************************* //
```

B.2.16 Make/options

```
EXE_INC = \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/transportModels/incompressible/lnInclude \
    -I$(LIB_SRC)/transportModels/interfaceProperties/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -ItractionDisplacement \
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
    $(WM_DECOMP_INC) \
    -I$(LIB_SRC)/tetDecompositionFiniteElement/lnInclude


EXE_LIBS = \
    -linterfaceProperties \
    -lincompressibleTransportModels \
    -lfiniteVolume \
    -ldynamicFvMesh \
    $(W_DECOMP_LIBS) \
    -llduSolvers
```

B.2.17 Make/files

```
tractionDisplacement/tractionDisplacementFvPatchVectorField.C
myInterFsiFoam.C

EXE = $(FOAM_USER_APPBIN)/myInterFsiFoam
```

B.2.18 Unaltered files

The following files are from the *interFoam* source code directory `$FOAM_SOLVERS/multiphase/interFoam/` that are used unchanged in the *myInterFsiFoam* solver.

pEqn.H

```
{
    volScalarField rUA = 1.0/UEqn.A();
    surfaceScalarField rUAf = fvc::interpolate(rUA);
    U = rUA*UEqn.H();
    surfaceScalarField phiU
    (
        "phiU",
        (fvc::interpolate(U) & mesh.Sf()) + fvc::ddtPhiCorr(rUA, rho, U, phi)
    );
    phi = phiU +
        (
            fvc::interpolate(interface.sigmaK())*fvc::snGrad(gamma)
          - ghf*fvc::snGrad(rho)
        )*rUAf*mesh.magSf();
    adjustPhi(phi, U, pd);
    for(int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        fvScalarMatrix pdEqn
        (
            fvm::laplacian(rUAf, pd) == fvc::div(phi)
        );
        pdEqn.setReference(pdRefCell, pdRefValue);
        if (corr == nCorr-1 && nonOrth == nNonOrthCorr)
        {
            pdEqn.solve(mesh.solver(pd.name() + "Final"));
        }
        else
        {
            pdEqn.solve(mesh.solver(pd.name()));
        }
        if (nonOrth == nNonOrthCorr)
        {
            phi -= pdEqn.flux();
        }
    }
    U += rUA*fvc::reconstruct((phi - phiU)/rUAf);
    U.correctBoundaryConditions();
}
```

UEqn.H

```
    surfaceScalarField muf = twoPhaseProperties.muf();
    fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U)
      + fvm::div(rhoPhi, U)
      - fvm::laplacian(muf, U)
      - (fvc::grad(U) & fvc::grad(muf))
    //- fvc::div(muf*(fvc::interpolate(dev(fvc::grad(U))) & mesh.Sf()))
    );
    UEqn.relax();
    if (momentumPredictor)
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
                    fvc::interpolate(interface.sigmaK())*fvc::snGrad(gamma)
                  - ghf*fvc::snGrad(rho)
                  - fvc::snGrad(pd)
                ) * mesh.magSf()
            )
        );
    }
```

UBlendingFactor.H

```
    surfaceScalarField gammaf = fvc::interpolate(gamma);
    surfaceScalarField UBlendingFactor
    (
        "UBlendingFactor",
        sqrt(max(min(4*gammaf*(1.0 - gammaf), 1.0), 0.0))
    );
```

gammaEqnSubCycle.H

```
label nGammaCorr
(
    readLabel(piso.lookup("nGammaCorr"))
);
label nGammaSubCycles
(
    readLabel(piso.lookup("nGammaSubCycles"))
);
if (nGammaSubCycles > 1)
{
    dimensionedScalar totalDeltaT = runTime.deltaT();
    surfaceScalarField rhoPhiSum = 0.0*rhoPhi;
    for
    (
        subCycle<volScalarField> gammaSubCycle(gamma, nGammaSubCycles);
        !(++gammaSubCycle).end();
    )
    {
#       include "gammaEqn.H"
        rhoPhiSum += (runTime.deltaT()/totalDeltaT)*rhoPhi;
    }
    rhoPhi = rhoPhiSum;
}
else
{
#       include "gammaEqn.H"
}
interface.correct();
rho == gamma*rho1 + (scalar(1) - gamma)*rho2;
```

gammaEqn.H

```
{
    word gammaScheme("div(phi,gamma)");
    word gammarScheme("div(phirb,gamma)");
    surfaceScalarField phic = mag(phi/mesh.magSf());
    phic = min(interface.cGamma()*phic, max(phic));
    surfaceScalarField phir = phic*interface.nHatf();
    for (int gCorr=0; gCorr<nGammaCorr; gCorr++)
    {
        surfaceScalarField phiGamma =
            fvc::flux
            (
                phi,
                gamma,
                gammaScheme
            )
          + fvc::flux
            (
                -fvc::flux(-phir, scalar(1) - gamma, gammarScheme),
                gamma,
                gammarScheme
            );
        MULES::explicitSolve(gamma, phi, phiGamma, 1, 0);
        rhoPhi = phiGamma*(rho1 - rho2) + phi*rho2;
    }
    Info<< "Liquid phase volume fraction = "
        << gamma.weightedAverage(mesh.V()).value()
        << "  Min(gamma) = " << min(gamma).value()
        << "  Max(gamma) = " << max(gamma).value()
        << endl;
}
```

correctPhi.H

```
{
#   include "movingMeshContinuityErrs.H"
    wordList pcorrTypes(pd.boundaryField().types());
    for (label i=0; i<pd.boundaryField().size(); i++)
    {
        if (pd.boundaryField()[i].fixesValue())
        {
            pcorrTypes[i] = fixedValueFvPatchScalarField::typeName;
        }
    }
    volScalarField pcorr
    (
        IOobject
        (
            "pcorr",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        mesh,
        dimensionedScalar("pcorr", pd.dimensions(), 0.0),
        pcorrTypes
    );
    dimensionedScalar rUAf("(1|A(U))", dimTime/rho.dimensions(), 1.0);
    adjustPhi(phi, U, pcorr);
    for(int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
    {
        fvScalarMatrix pcorrEqn
        (
            fvm::laplacian(rUAf, pcorr) == fvc::div(phi)
        );
        pcorrEqn.setReference(pdRefCell, pdRefValue);
        pcorrEqn.solve();
        if (nonOrth == nNonOrthCorr)
        {
            phi -= pcorrEqn.flux();
        }
    }
#   include "movingMeshContinuityErrs.H"
}
```

Appendix C

Cases

The following files are the casefiles provided with this report. The sectioning of this appendix follows and denotes the directory containing the listed files. Only the functioning code remains in the listings due to report length considerations.

C.1 flappingConsoleSmall

C.1.1 fluid/0/

solid

This is just a symbolic link pointing to the directory `../../solid/0/`. It is made with

```
[fluid/0]$ ln -s ../../solid/0/ solid
```

U

```
FoamFile
{
    version 2.0;
    format ascii;
    class volVectorField;
    object U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    consoleFluid
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    topWall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    bottomWall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            zeroGradient;
    }
    inlet
    {
        type            fixedValue;
        value           uniform (4 0 0);
    }
    frontAndBackPlanes
    {
        type            empty;
    }
}
// ************************************************************************* //
```

motionU

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           tetPointVectorField;
    object          motionU;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
   consoleFluid
   {
      type fixedValue;
      value uniform (0 0 0);
   }
   topWall
   {
      type slip;
   }
   bottomWall
   {
      type slip;
   }
   outlet
   {
      type fixedValue;
      value uniform (0 0 0);
   }
   inlet
   {
    type fixedValue;
    value uniform (0 0 0);
   }
   frontAndBackPlanes
   {
      type empty;
   }
}
// ************************************************************************* //
```

p

```
FoamFile
{
    version 2.0;
    format ascii;
    class volScalarField;
    object p;
}
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 2 -2 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    consoleFluid
    {
        type            zeroGradient;
    }
    topWall
    {
        type            zeroGradient;
    }
    bottomWall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            totalPressure;
        p0              uniform 0;
        U               U;
        phi             phi;
        rho             none;
        psi             none;
        gamma           1;
        value           uniform 0;
    }
    inlet
    {
        type            zeroGradient;
    }
    frontAndBackPlanes
    {
        type            empty;
    }
}
// ************************************************************************* //
```

C.1.2 fluid/constant/

solid

This is just a symbolic link pointing to the directory `../../solid/constant/`. It is made with

```
[fluid/constant]$ ln -s ../../solid/constant/ solid
```

couplingProperties

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          couplingProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solidPatch      consoleSolid;
fluidPatch      consoleFluid;
movingRegion    region0;
// ************************************************************************* //
```

dynamicMeshDict

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          motionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dynamicFvMesh dynamicMotionSolverFvMesh;
twoDMotion      yes;
solver          laplaceFaceDecomposition;
diffusivity     quadratic;
frozenDiffusion on;
distancePatches
(
    consoleFluid
);
// ************************************************************************* //
```

transportProperties

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
nu              nu [0 2 -1 0 0 0 0] 0.001;
rho             rho [1 -3 0 0 0 0 0] 1;
// ************************************************************************* //
```

C.1.3 fluid/constant/polyMesh

blockMeshDict

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 1;
vertices
(
    (-2   0    -0.1)
    (0    0    -0.1)
    (0.05 0    -0.1)
    (4    0    -0.1)
    (-2   0.6  -0.1)
    (0    0.6  -0.1)
    (0.05 0.6  -0.1)
    (4    0.6  -0.1)
    (-2   1    -0.1)
    (0    1    -0.1)
    (0.05 1    -0.1)
    (4    1    -0.1)

    (-2   0     0.1)
    (0    0     0.1)
    (0.05 0     0.1)
    (4    0     0.1)
    (-2   0.6   0.1)
    (0    0.6   0.1)
    (0.05 0.6   0.1)
    (4    0.6   0.1)
    (-2   1     0.1)
    (0    1     0.1)
    (0.05 1     0.1)
    (4    1     0.1)
);
blocks
(
    hex (0 1 5 4 12 13 17 16) (40 20 1) simpleGrading (0.1 0.2 1)
    hex (2 3 7 6 14 15 19 18) (80 20 1) simpleGrading (10 0.2 1)
    hex (4 5 9 8 16 17 21 20) (40 20 1) simpleGrading (0.1 2 1)
    hex (5 6 10 9 17 18 22 21) (5 20 1) simpleGrading (1 2 1)
    hex (6 7 11 10 18 19 23 22) (80 20 1) simpleGrading (10 2 1)
);
edges
(
);
patches
(
    patch consoleFluid
    (
        (1 13 17 5)
        (5 17 18 6)
        (6 18 14 2)
    )
    wall topWall
    (
        (8 20 21 9)
        (9 21 22 10)
        (10 22 23 11)
    )
    wall bottomWall
    (
        (0 1 13 12)
        (2 3 15 14)
    )
    patch outlet
    (
        (3 7 19 15)
        (7 11 23 19)
    )
    wall inlet
    (
        (0 12 16 4)
        (4 16 20 8)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

boundary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
6
(
    consoleFluid
    {
        type            patch;
        nFaces          45;
        startFace       9615;
    }
    topWall
    {
        type            wall;
        nFaces          125;
        startFace       9660;
    }
    bottomWall
    {
        type            wall;
        nFaces          120;
        startFace       9785;
    }
    outlet
    {
        type            patch;
        nFaces          40;
        startFace       9905;
    }
    inlet
    {
        type            wall;
        nFaces          40;
        startFace       9945;
    }
    defaultFaces
    {
        type            empty;
        nFaces          9800;
        startFace       9985;
    }
)
// ************************************************************************* //
```

C.1.4 fluid/system

solid

This is just a symbolic link pointing to the directory `../../solid/system/`. It is made with

```
[fluid/system]$ ln -s ../../solid/system/ solid
```

controlDict

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application icoFoam;
startFrom       latestTime;
startTime       0;
stopAt          endTime;
endTime         50;
deltaT          0.0003;
```

```
writeControl    timeStep;
writeInterval   20;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression compressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
adjustTimeStep    no;
maxCo             0.5;
// ************************************************************************* //
```

fvSolution

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    p               ICCG 1e-06 0;
    U               BICCG 1e-05 0;
}
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 1;
    pRefCell        0;
    pRefValue       0;
}
// ************************************************************************* //
```

fvSchemes

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
    default         Euler;
}
d2dt2Schemes
{
    d2dt2(U)        Euler;
}
gradSchemes
{
    default         Gauss linear;
    grad(p)         Gauss linear;
}
divSchemes
{
    default         none;
    div(phiNet,U)      Gauss limitedLinearV 1;
    div(phi,U)      Gauss limitedLinearV 1;
}
```

```
laplacianSchemes
{
    default         none;
    laplacian(nu,U) Gauss linear corrected;
    laplacian((1|A(U)),p) Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
    interpolate(HbyA) linear;
}
snGradSchemes
{
    default         corrected;
}
fluxRequired
{
    default         no;
    p;
}
// ************************************************************************* //
```

tetFemSolution

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          tetFemSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
//    motionU         ICCG 1e-06 0;
    motionU         BICCG 1e-06 0 100;
}
// ************************************************************************* //
```

C.1.5 solid/0/

U

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           volVectorField;
    object          U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 0 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
consoleSolid
{
    type            tractionDisplacement;
    traction        uniform (0 0 0);
    pressure        uniform 0;
    value           uniform (0 0 0);
}
consoleFixed
{
    type    fixedValue;
    value uniform (0 0 0);
}
frontAndBackPlanes
{
    type empty;
}
}
// ************************************************************************* //
```

T

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           volScalarField;
    object          T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 1 0 0 0];
internalField   uniform 400;
boundaryField
{
    consoleSolid
    {
        type            oscillatingFixedValue;
        refValue        uniform 400;
        amplitude       100;
        frequency       2;
    }
    consoleFixed
    {
        type            fixedValue;
        value           uniform 400;
    }
    frontAndBackPlanes
    {
        type            empty;
    }
}
// ************************************************************************* //
```

C.1.6 solid/constant/

mechanicalProperties

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          mechanicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
rho             rho [1 -3 0 0 0 0 0] 1000;
nu              nu [0 0 0 0 0 0 0] 0.3;
E               E [1 -1 -2 0 0 0 0] 2e+6;
planeStress     yes;
// ************************************************************************* //
```

thermalProperties

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          thermalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
C               C [0 2 -2 -1 0 0 0] 434;
k               k [1 1 -3 -1 0 0 0] 60.5;
alpha           alpha [0 0 0 -1 0 0 0] 1.1e-05;
thermalStress   no;
// ************************************************************************* //
```

C.1.7 solid/constant/polyMesh

blockMeshDict

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 1;
vertices
(
    (0 0 -0.1)
    (0.05 0 -0.1)
    (0.05 0.6 -0.1)
    (0 0.6 -0.1)
    (0 0 0.1)
    (0.05 0 0.1)
    (0.05 0.6 0.1)
    (0 0.6 0.1)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (10 45 1) simpleGrading (1 1 1)
);
edges
(
);
patches
(
    patch consoleSolid
    (
        (3 7 6 2)
        (0 4 7 3)
        (2 6 5 1)
    )
    patch consoleFixed
    (
        (1 5 4 0)
    )
    empty frontAndBackPlanes
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

boundary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
3
(
    consoleSolid
    {
        type            patch;
        nFaces          100;
        startFace       845;
    }
    consoleFixed
    {
        type            patch;
        nFaces          10;
        startFace       945;
    }
    frontAndBackPlanes
    {
        type            empty;
        nFaces          900;
        startFace       955;
    }
)
// ************************************************************************* //
```

C.1.8 solid/system

controlDict

This is just a symbolic link pointing to the file `fluid/system/controlDict`. It is made with

```
[solid/system]$ ln -s ../../fluid/system/controlDict controlDict
```

fvSolution

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    T               ICCG 1e-09 0.01;
    U               ICCG 1e-09 0.01;
//    U               AMG 1e-09 0.01 100;
}
stressedFoam
{
    nCorrectors     50;
    U               1e-07;
}
// ************************************************************************* //
```

fvSchemes

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
    default     Euler;
}
d2dt2Schemes
{
    default     Euler;
}
gradSchemes
{
    default         Gauss linear;
    grad(U)         Gauss linear;
    grad(T)         Gauss linear;
}
divSchemes
{
    default         none;
    div(sigma)      Gauss linear;
}
laplacianSchemes
{
    default         none;
    laplacian(DU,U) Gauss linear corrected;
    laplacian(DT,T) Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         corrected;
}
fluxRequired
{
    default         no;
    U;
    T;
}
// ************************************************************************* //
```

C.2 softDamBreak

C.2.1 fluid/0/

solid

This is just a symbolic link pointing to the directory `../../solid/0/`. It is made with

```
[fluid/0]$ ln -s ../../solid/0/ solid
```

U

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    consoleFluid
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    leftWall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    rightWall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    lowerWall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        phi             phi;
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

motionU

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           tetPointVectorField;
    object          motionU;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    consoleFluid
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    atmosphere
    {
        type slip;
    }
    lowerWall
    {
        type slip;
    }
    leftWall
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    rightWall
    {
        type fixedValue;
        value uniform (0 0 0);
    }
    frontAndBackPlanes
    {
        type empty;
    }
}
// ************************************************************************* //
```

pd

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      pd;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [1 -1 -2 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    leftWall
    {
        type            zeroGradient;
    }
    rightWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            totalPressure;
        U               U;
        phi             phi;
        rho             rho;
        psi             none;
        gamma           1;
        p0              uniform 0;
        value           uniform 0;
    }
    consoleFluid
    {
        type            zeroGradient; //slip; // ??? NOT SURE !!
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

gamma

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      gamma;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 0 0 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    leftWall
    {
        type            zeroGradient;
    }
    rightWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    consoleFluid
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

C.2.2 fluid/constant/

solid

This is just a symbolic link pointing to the directory `../../solid/constant/`. It is made with

```
[fluid/constant]$ ln -s ../../solid/constant/ solid
```

couplingProperties

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          couplingProperties;
}
```

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solidPatch      consoleSolid;
fluidPatch      consoleFluid;
movingRegion    region0;
// ************************************************************************* //
```

dynamicMeshDict

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          motionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dynamicFvMesh dynamicMotionSolverFvMesh;
twoDMotion      yes;
solver          laplaceFaceDecomposition;
diffusivity     quadratic;
frozenDiffusion on;
distancePatches
(
    consoleFluid
);
// ************************************************************************* //
```

transportProperties

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
phase1
{
    transportModel  Newtonian;
    nu              nu [0 2 -1 0 0 0 0] 1e-06;
    rho             rho [1 -3 0 0 0 0 0] 1000;
}
phase2
{
    transportModel  Newtonian;
    nu              nu [0 2 -1 0 0 0 0] 1.48e-05;
    rho             rho [1 -3 0 0 0 0 0] 1;
}
sigma           sigma [1 0 -2 0 0 0 0] 0.07;
// ************************************************************************* //
```

environmentaProperties

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      environmentalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
g               g [0 1 -2 0 0 0 0] (0 -9.81 0);
// ************************************************************************* //
```

C.2.3 fluid/constant/polyMesh

blockMeshDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
convertToMeters 0.146;
vertices
(
    (0 0 0)
    (2 0 0)
    (2.16438 0 0)
    (4 0 0)
    (0 0.32876 0)
    (2 0.32876 0)
    (2.16438 0.32876 0)
    (4 0.32876 0)
    (0 4 0)
    (2 4 0)
    (2.16438 4 0)
    (4 4 0)
    (0 0 0.1)
    (2 0 0.1)
    (2.16438 0 0.1)
    (4 0 0.1)
    (0 0.32876 0.1)
    (2 0.32876 0.1)
    (2.16438 0.32876 0.1)
    (4 0.32876 0.1)
    (0 4 0.1)
    (2 4 0.1)
    (2.16438 4 0.1)
    (4 4 0.1)
);
blocks
(
    hex (0 1 5 4 12 13 17 16) (30 20 1) simpleGrading (0.2 1 1)
    hex (2 3 7 6 14 15 19 18) (25 20 1) simpleGrading (5 1 1)
    hex (4 5 9 8 16 17 21 20) (30 60 1) simpleGrading (0.2 5 1)
    hex (5 6 10 9 17 18 22 21) (5 60 1) simpleGrading (1 5 1)
    hex (6 7 11 10 18 19 23 22) (25 60 1) simpleGrading (5 5 1)
);
edges
(
);
patches
(
    wall leftWall
    (
        (0 12 16 4)
        (4 16 20 8)
    )
    wall rightWall
    (
        (7 19 15 3)
        (11 23 19 7)
    )
    wall lowerWall
    (
        (0 1 13 12)
        (2 3 15 14)
    )
    patch consoleFluid
    (
        (1 5 17 13)
        (5 6 18 17)
        (2 14 18 6)
    )
    patch atmosphere
    (
        (8 20 21 9)
        (9 21 22 10)
        (10 22 23 11)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

boundary

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       polyBoundaryMesh;
    location    "constant/polyMesh";
    object      boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
6
(
    leftWall
    {
        type            wall;
        nFaces          50;
        startFace       4432;
    }
    rightWall
    {
        type            wall;
        nFaces          50;
        startFace       4482;
    }
    lowerWall
    {
        type            wall;
        nFaces          42;
        startFace       4532;
    }
    consoleFluid
    {
        type            patch;
        nFaces          20;
        startFace       4574;
    }
    atmosphere
    {
        type            patch;
        nFaces          46;
        startFace       4594;
    }
    defaultFaces
    {
        type            empty;
        nFaces          4536;
        startFace       4640;
    }
)
// ************************************************************************* //
```

C.2.4 fluid/system

solid

This is just a symbolic link pointing to the directory `../../solid/system/`. It is made with

```
[fluid/system]$ ln -s ../../solid/system/ solid
```

controlDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application interFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         10;
deltaT          0.0003;
writeControl    adjustableRunTime;
writeInterval   0.005;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
adjustTimeStep  yes;
maxCo           0.5;
maxDeltaT       0.1;
// ************************************************************************* //
```

fvSolution

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    pcorr PCG
    {
        preconditioner   DIC;
        tolerance        1e-10;
        relTol           0;
    };
    pd PCG
    {
        preconditioner   DIC;
        tolerance        1e-7;
        relTol           0.05;
    };
    pdFinal PCG
    {
        preconditioner   DIC;
        tolerance        1e-7;
        relTol           0;
    };
    U PBiCG
    {
        preconditioner   DILU;
        tolerance        1e-06;
        relTol           0;
    };
}
PISO
{
    momentumPredictor no;
    nCorrectors     3;
    nNonOrthogonalCorrectors 0;
    nGammaCorr      1;
    nGammaSubCycles 2;
    cGamma          1;
}
// ************************************************************************* //
```

fvSchemes

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
    default Euler;
}
gradSchemes
{
    default         Gauss linear;
    grad(U)         Gauss linear;
    grad(gamma)     Gauss linear;
}
divSchemes
{
    div(rho*phi,U)  Gauss limitedLinearV 1;
    div(phi,gamma)  Gauss vanLeer;
    div(phirb,gamma) Gauss interfaceCompression;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         corrected;
}
fluxRequired
{
    default         no;
    pd;
    pcorr;
    gamma;
}
// ************************************************************************* //
```

tetFemSolution

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          tetFemSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
//    motionU         ICCG 1e-06 0;
    motionU         BICCG 1e-06 0 100;
}
// ************************************************************************* //
```

setFieldsDict

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
defaultFieldValues
(
    volScalarFieldValue gamma 0
);
regions
(
    boxToCell
    {
        box (0 0 -1) (0.1461 0.292 1);
        fieldValues
        (
            volScalarFieldValue gamma 1
        );
    }
);
// ************************************************************************* //
```

C.2.5 solid/0/

U

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           volVectorField;
    object          U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 0 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
   consoleSolid
   {
       type            tractionDisplacement;
       traction        uniform (0 0 0);
       pressure        uniform 0;
       value           uniform (0 0 0);
   }
   consoleFixed
   {
       type    fixedValue;
       value uniform (0 0 0);
   }
   frontAndBackPlanes
   {
       type empty;
   }
}
// ************************************************************************* //
```

C.2.6 solid/constant/

mechanicalProperties

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          mechanicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
rho             rho [1 -3 0 0 0 0 0] 7850;
nu              nu [0 0 0 0 0 0 0] 0.3;
E               E [1 -1 -2 0 0 0 0] 2e+11;
planeStress     yes;
// ************************************************************************* //
```

thermalProperties

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          thermalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
C               C [0 2 -2 -1 0 0 0] 434;
k               k [1 1 -3 -1 0 0 0] 60.5;
alpha           alpha [0 0 0 -1 0 0 0] 1.1e-05;
thermalStress   no;
// ************************************************************************* //
```

C.2.7 solid/constant/polyMesh

blockMeshDict

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

```
convertToMeters 0.146;
vertices
(
    (2 0 0)
    (2.16438 0 0)
    (2.16438 0.32876 0)
    (2 0.32876 0)
    (2 0 0.1)
    (2.16438 0 0.1)
    (2.16438 0.32876 0.1)
    (2 0.32876 0.1)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (10 45 1) simpleGrading (1 1 1)
);
edges
(
);
patches
(
    patch consoleSolid
    (
        (3 7 6 2)
        (0 4 7 3)
        (2 6 5 1)
    )
    patch consoleFixed
    (
        (1 5 4 0)
    )
    empty frontAndBackPlanes
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);
mergePatchPairs
(
);
// ************************************************************************* //
```

boundary

```
FoamFile
{
    version 2.0;
    format ascii;
    class polyBoundaryMesh;
    object boundary;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
3
(
consoleSolid
{
    type patch;
}
consoleFixed
{
    type patch;
}
frontAndBackPlanes
{
    type empty;
}
)
```

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

C.2.8 solid/system

controlDict

This is just a symbolic link pointing to the file `fluid/system/controlDict`. It is made with

```
[solid/system]$ ln -s ../../fluid/system/controlDict controlDict
```

fvSolution

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
solvers
{
    U               ICCG 1e-09 0.01;
//    U               AMG 1e-09 0.01 100;
}
stressedFoam
{
    nCorrectors     50;
    U               1e-07;
}
// ************************************************************************* //
```

fvSchemes

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
d2dt2Schemes
{
    default     Euler;
}
gradSchemes
{
    default         Gauss linear;
    grad(U)         Gauss linear;
    grad(T)         Gauss linear;
}
divSchemes
{
    default         none;
    div(sigma)      Gauss linear;
}
laplacianSchemes
{
    default         none;
    laplacian(DU,U) Gauss linear corrected;
    laplacian(DT,T) Gauss linear corrected;
}
interpolationSchemes
```

```
{
    default         linear;
}
snGradSchemes
{
    default         corrected;
}
fluxRequired
{
    default         no;
    U;
    T;
}
// ************************************************************************* //
```