UMB

Norwegian University of Life Sciences

CFD with OpenSource software, project assignment

---

icoFsiFoam and interFsiFoam

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

---

Not yet peer reviewed

Author:
Karl Jacob Maus

December 12, 2009

Contents

Chapter 1

1.1 Introduction

This report briefly documents the *icoFsiFoam* solver from the most recent user-contributed development version of OpenFOAM, version **OpenFOAM-1.5-dev**, maintained by Hrvoje Jasak. The *icoFsiFoam* is a fluid–structure interaction (FSI) solver for weakly coupled systems with small structural deformations and laminar fluid flow.

The solver is then used as a template solver for the creation of similar FSI solvers. First, the *icoFsiFoam* solver is built almost from scratch such that our newly developed solver incorporates both stress component computations and the possibility for calculating thermal stresses. Then, another solver is created by using the *interFoam* solver instead of the *icoFoam* solver but using the same approach and reusing some of the source files.

Finally, we will use these newly created solvers 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* which is not present in the installed files.

Sadly, this report did not get finished in time. Hopefully, some of this is possible to follow, while some parts are mere notes. With some imagination and adventurous hearts, it should still be possible to (re-) create the solvers and tutorials submitted.

1.2 Fluid–Structure Interaction
a brief introduction

In a fluid flow system there is always some kind of FSI, but the effect might not be noticable on the domain we wish to study. For instance, flow around a building is usually modelled as a one–phase system since the building usually is rigid and does not influence the flow other than statically. On the microscale, however, there might be some wear on the walls of the building which again might slightly alter the boundary layer behavior. This is usually deemed negligable.

When a body deforms, as the blades of a wind turbine or an airplane wing for instance, 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 we have non-negligable 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, but if the boundary movement is small then this is ok.

The mesh and the temporal discretization is vital in FSI problems. Since a finite volume (FV) discretization in OpenFOAM is cell-based, 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 quickly influence the convergence and propagate errors in the solid and fluid solvers. Hence, a vertex–based method is preferred. This is only feasable in OpenFOAM by using the finite element classes from the development version. Time steps are preferably small, this would possibly make more problems 'weak', but too small time steps also brings errors to the solution and should be discouraged.

1.3 icoFsiFoam

The *icoFsiFoam* is a solver for weakly coupled FSI problems. The structural part is based on *stressedFoam* and is 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 by Newtonian fluids in incompressible, laminar flow. The *icoFsiFoam* does not use the PISO algorithm, but instead a SIMPLE-based algorithm with corrector loops specified by PISO parameters in the `system/fvSolution` dictionary.

First we present a quick list of all the externally loaded files in `icoFsiFoam.C`, with a short explanation and location of relevant source file relative `$FOAM_SRC`:

fvCFD.H Standard header file for finite volume method (FVM) in OpenFOAM (OF)
- from `finiteVolume/cfdTools`

dynamicFvMesh.H Header for class `dynamicFvMesh`
- from `dynamicFvMesh`

patchToPatchInterpolation.H Interpolation functions; imports `pointField.H`
- from `OpenFOAM/interpolations`

tetFemMatrices.H calls `tetFemScalarMatrix.H` which calls `tetFemMatrix.H` which again includes `tetPointFields.H`. Defines tetrahedral finite element (FE) matrix
- from `tetDecompositionFiniteElement`

tetPointFields.H* Point field for finite element method (FEM) discretization. Reduntant to include due to previously included file.

faceTetPolyPatch.H FE `tetPolyPatch` based on `PolyPatch`
- from `tetDecompositionFiniteElement`

tetPolyPatchInterpolation.H Interpolates fields defined on faces into points on a `tetPolyPatch`
- from `tetDecompositionFiniteElement`

fixedValueTetPolyPatchFields.H Fixed value `tetPolyPatch` fields
- from `tetDecompositionFiniteElement`

pointMesh.H* Mesh for points from `polyMesh`. Is called by the next file.
- from `OpenFOAM/mesh`

pointFields.H Calls `pointMesh.H` - makes the above include statement redundant.
- from `OpenFOAM/mesh`

volPointInterpolation Interpolation from volume to point fields
- from `finiteVolume/interpolation`

The `Make/options` file includes the following lines to ensure that the formerly mentioned header files can be found:

finiteVolume Contains the FV relevant libraries.

dynamicFvMesh Contains the dynamic FV mesh.

dynamicMesh* Contains dynamic meshes and is required (and called) 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.

Note that while the local `tractionDisplacement` directory is loaded at compile–time by the *stressedFoam* solver from the relevant `Make` files, this will not work with the present recipe. Instead, the headerfile is included directly in the `myIcoFsiFoam.C` file, and the corresponding line from the `Make` files has been omitted.

1.3.1 Main program

First the needed headers are included, to ensure all relevant classes and namespaces have been declared. They are the externally loaded headers mentioned above together with the local `tractionDisplacement/tractionDisplacementFvPatchVectorField.H`.

The following files are called at startup of the main program before the timeloop is initialized:

setRootCase.H Present in both `stressedFoam.C` and `icoFoam.C`. Checks root case.

createTime.H Also present in 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`, since we need a dynamic FV–mesh to handle solid displacements.

createStressMesh.H Used instead of `createMesh.H` from `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 includes creation of the point mesh needed for the finite element implementation of the fluid–structure boundary. The domain and various control variables are read from the relevant `solid` subdirectories instead of the default region.

createFields.H This is the local copy of `$FOAM_SOLVERS/incompressible/icoFoam/createFields.H` with the addition of reading the fluid density `"rho"` from the file `transportProperties`.

createStressFields.H Local copy of the file `$FOAM_SOLVERS/stressAnalysis/stressedFoam/createFields.H`. Solid velocity 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.

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

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

initContinuityErrors.H Initializes the contintuity error variable.

While the solver is running it loops over a set of includes, in addition to taking the time of the main solver and outputting this. First, time and solver parameters are read, Courant number is checked, and variable time step is set. The fact that all these are in the loop shows the run-time modifiability of the *icoFsiFoam* solver.

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

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

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

setDeltaT.H Reduces or increases the timestep according to the calculations of `CourantNo.H` to try and maintain a constant maximum Courant number.

Then the solver part starts, in the following 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_SRC/applications/solvers/stressAnalysis/stressedFoam.C`, with all thermal and stress component computations removed.

setMotion.H Moves the mesh in accordance with the previously calculated solid deformations.

solveFluid.H Solves the fluid motion on the updated geometry and mesh. Does not use the PISO loop of the *icoFoam* solver, but uses a SIMPLE loop with an added loop for pressure correction. The pressure correction loop is controlled by the PISO controls of the `fvSolution` dictionary.

1.3.2 Building the icoFsiFoam solver

We need most of this file as it is since it contains the solver code for the fluid part of the problem, and thus copy it twice; once to the file `solveFluid.H` and once to the file `icoFsiFoam.C`. We then edit the file `solveFluid.H` by stripping it of any header information and retain only the part inside the time loop. Then we edit the `icoFsiFoam.C`-file to retain only the header- and loop information and strip everything between the `#include CourantNo.H`-line of the loop and the `runTime.write()`-line. Remember to check the `stressedFoam.C`-file to make sure the header- and loop information is consistent with both original solvers *icoFoam* and *stressedFoam*. Since we use a dynamic mesh, we must change `createMesh` to `createDynamicFvMesh` for the fluid. We must also create a mesh for the structural domain, we call this file `createStressMesh.H` and will create this in section 1.3.1 based on the `createMesh.H` file. In addition, we need the traction displacement boundary code found in the local file `tractionDisplacementFvPatchVectorField.H` in the `tractionDisplacement` subdirectory. So, we must add a line `#include "tractionDisplacementFvPatchVectorField.H"` in the header, and we must add the corresponding directory to the `Make/options` file by deleting the tail `/lnInclude` from the line specifying the `tractionDisplacement` directory.

The fluid field is created by the local `createFields.H`, so we leave this line 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` is replaced by `Usolid` and `mesh` by `solidMesh`.

Create the `Make` subdirectory in the `myIcoFsiFoam` solver directory, and copy the files `files` and `options` from the *stressedFoam* `Make` subdirectory. Then edit the `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 `options` file must also be augmented to include all the needed libraries and directories we use. Add the following lines to the `EXE_INC = \` entry:

- `-ItractionDisplacement \` is the local directory with the traction displacement code.
- `-I\$(LIB_SRC)/dynamicFvMesh/lnInclude \` contains the includes for the FV mesh.
- `$(W_DECOMP_INC) \` is needed since we will use 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; `-ldynamicFvMesh \` specifies the dynamic finite volume mesh shared library and `$(W_DECOMP_LIBS) \` are the FE face decomposition libraries.

myIcoFsiFoam.C

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

---

solveFluid.H

Action: Copy file `.` to local `solveFluid.H`.

Action: Strip everything away but the interior of the time loop and must remember to update the `#include continuityErrs.H` to be able to calculate the errors on a moving mesh we need instead to call the file `movingMeshContinuityErrs.H`

Action:

---

readStressedFoamControls.H

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

Action: As with `solveFluid.H` we only want to keep the core of the time loop, and no *Info* lines are retained since they are produced by our 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.

Action: Remember to replace all instances of `U` with `Usolid` since we are in the solid domain.

---

readMechanicalProperties.H

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

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`.

readThermalProperties.H

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

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

calculateStress.H

Action: Copy main file from `stressedFoam`

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

Action: Replace `mesh` with `stressMesh` in the stress meshes since we are in the solid domain.

createStressMesh.H

Action: Copy `$FOAM_SRC/OpenFOAM/include/createMesh.H` to local `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
);
```

createFields.H

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

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

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

createStressFields.H

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

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

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

Action: We keep the thermal stress code, and for consistency we define the volume scalar field `Tsolid` instead of `T`.

The following files are copied as is from the `icoFsiFoam` directory:

- `readCouplingProperties.H`
- `setMotion.H`
- `setPressure.H`

Compile with wmake, and if the OpenFOAM environment variables are correctly set then we should have a working FSI solver. 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 our solver cannot solve thermal fluid–structure interaction. However, if the thermal stresses causes the solid domain to deform then this will influence the fluid flow.

1.4 Building a weakly coupled FSI solver

We will now attempt to use the previous method of coupling with a different solver. We keep the solid solver and try to couple the *interFoam* multiphase solver with the *stressedFoam* solver in the same fashion as we did with the *icoFoam* solver.

1.4.1 myInterFsiFoam

Import all necessary headers into main `.C`-file.
Copy all relevant files from myIcoFsiFoam.
rho must be renamed to rhoSolid, and rhoFluid must be renamed to rho due to interFoam design. This must be done properly throughout...
rho is a field (not scalar) in interFoam. Must delete `.value()` from file `setPressure.H`
`myIcoFsiFoam::rho` present in `readThermalProperties`, `readMechanicalProperties`, `tractionDisplacementFvPatchVectorField.C`, and `calculateStress.H` and must be altered to `rhoSolid`
Extract main loop algorithm into seperate file called `solveFluids.H`.
Call all files in similar order as in myIcoFsiFoam inside time loop.
Copy and edit interFoam Make/files: Rename solver, update app-dir env.variable, include tractionDisplacement.
`Make/options`: Add traction displacement, dynamicFvMesh and all FE decomposition references from `myIcoFsiFoam/Make/options`.
Cross fingers `;)`

1.5 FSI OpenFOAM cases

To test our newly built solvers we create a few test–cases.

1.5.1 A flapping console

Run from fluid dir.
Needed files: `thermalProperties` in constant dir. Also, check for valid solvers (may need update) in `fvSolution`, `tetFemSolution`, etc.
Run `linkSolidSolutions` file to link for paraFoam run.

Flapping console with thermals

By slightly modifying the previous case we can include thermal stresses in the flapping console. For now, the solver computes thermal stresses in the solid domain with a few minor modifications in the case dictionaries of the solid domain:
1. Change controlDict to compute thermal stresses.
2. fvSolution needs an input for the solution of T. For now we can just copy the line for U.
3. fvSchemes needs an input for ddtSchemes. For now: copy the d2dt2Schemes (def:Euler)
4. In addition we need an initial T file 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 apropriate boundary conditions (zeroGradient, fixedValue, etc...)

1.5.2 A soft dam break

Run dam break with "'soft"' dam.
—redefine dir.structure. Include all relevant dicts. Remember to alter solid mesh geometry to comply with the fluid mesh!
Redefine mesh for better adjacent sizes around solid block.

mesh

Copy `blockMesh` from `damBreak/constant/polyMesh` to `fluid/constant/polyMesh`
Change patches: Move the following patches from `lowerWall` into new patch `consoleFluid`

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

Copy `blockMesh` from `icoFsiFoam/solid/constant/polyMesh` to `solid/constant/polyMesh`
Edit `blockMesh`: replace vertices $0, 1, 2, 3, 4, 5, 6, 7$ with vertices $1, 2, 6, 5, 13, 14, 18, 17$ from the fluid directory `blockMesh` file. Remember to specify the correct `ConvertToMeter` parameter value.
`fluid/constant/polyMesh/boundary`: add patch

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

and alter number of patches from 5 to 6.

fluid/constant

Copy `transportProperties` from `damBreak` and reuse.
Copy `couplingProperties` from `flappingConsole` and `environmentalProperties` from `damBreak` and reuse.
Use `dynamicMeshDict` from `flappingConsole`. Create softlink `fluid/constant/solid` pointing to `solid/constant` dir.

solid/constant

Use dictionaries from `flappingConsole`.

system

Create softlink `solid/system/controlDict` to `fluid/system/controlDict`.
Make softlink `fluid/system/solid` to `solid/system` dir.
Reuse `flappingConsole/solid/fvSolution` and `fvSchemes` in `solid/system`.
Reuse `fvSchemes` and `fvSolution` from `damBreak/system` in `fluid/system`.
Use `controlDict` from `damBreak`, with the following modifications:
endTime=2;
deltaT=0.0003;
writeInterval=0.005;
writeCompression=compressed;
maxDeltaT=0.1;

0

Softlink `fluid/0/solid` to `solid/0`.
Copy `dambreak/0/U` to `fluid/0/` and add a `boundaryField`

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

to inital field.
Copy `flappingConsole/fluid/0/motionU` to `fluid/0/` and correct patch names. Test: let left and right walls be of `fixedValue` and let `atmosphere` and `lower wall` slip.
Copy `gamma` and `pd` from `damBreak/0` to `fluid/0` and add `boundaryField consoleFluid` as type `zeroGradient`.
In addition, copy `gamma` to `gamma.org` to keep an original after we run *setFields*.

Acronyms

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