Conjugate Simulations and Fluid-Structure Interaction In OpenFOAM

Hrvoje Jasak

`h.jasak@wikki.co.uk`

Wikki Ltd, United Kingdom and

FSB, University of Zagreb, Croatia

7-9th June 2007

Conjugate Simulations

Objective

- Review the machinery for Fluid-Structure Interaction (FSI) and coupled multi-domain solvers in OpenFOAM

Topics

- Multiple meshes and solver algorithms in a single application
- Explicit coupling: separate solvers and solution algorithm
- Surface data mapping tools
- Close coupling at matrix-level: conjugate heat transfer
- Future work

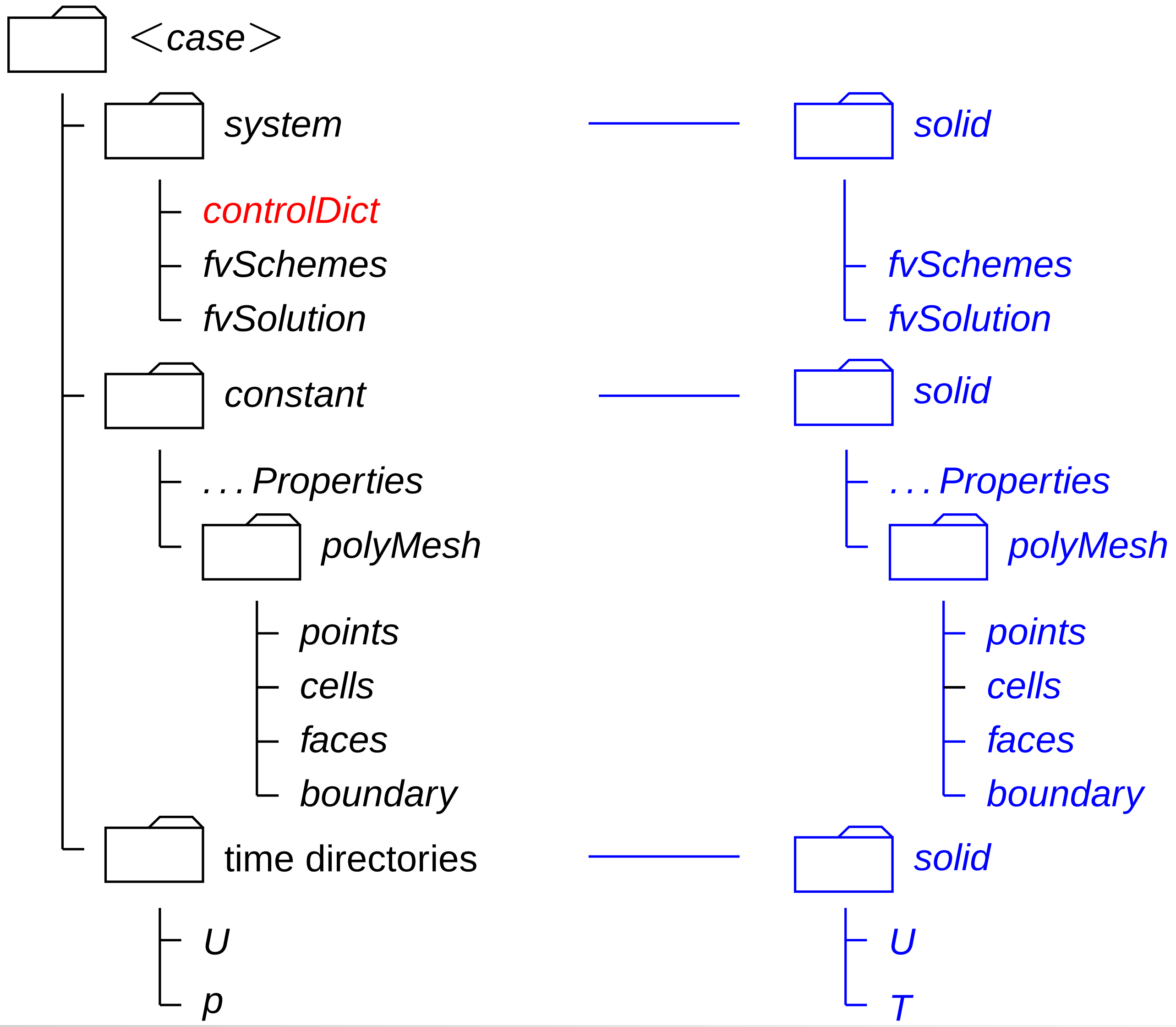
Multiple Domain Support

Multiple Domains in a Single Simulation

- Original class-based design allows for multiple object of the same type in a single simulation, e.g. meshes and fields
- Before OpenFOAM version 1.3, issues with object registry (name clashes) and database organisation
- Solution: **hierarchical object registry**
  - Multiple named mesh databases within a single simulation:
    1 mesh = 1 domain, with separate fields and physics
  - Fields, material properties and solution controls separate for each mesh
- “Main” mesh controls time advancement (with possible sub-cycling)
- Note: Every individual mesh represents a **single addressing space**, with its own internal faces and boundaries. Operations on various face types are consistent: consequences for conjugate heat transfer type of coupling

Multiple Domain Support

Case Organisation for Multiple Meshes: “Main Mesh” and `solid`



Explicit Coupling

Multiple Solvers Side-by-Side

- With multiple mesh support, creating side-by-side solvers is trivial: multiple fields and equations in a single executable
- Each solver uses its own mesh, with access to its fields, material properties, solver controls etc.
- Coupling achieved through boundary condition update
- Example: Fluid-Structure Interaction
  - Move pressure data from fluid to structure; solve structural equations
  - Move displacement to fluid mesh; solve fluid flow with mesh motion
- Auxiliary operations
  - Surface solution data mapping
  - Automatic mesh motion: works!
- Example solver written for specific coupling physics: `icoFsiFoam`
- Summary: It works, but not very object-oriented. More on this later

Surface Mapping Tools

Surface Mapping: `patchToPatchInterpolation`

- Data transfer required between two non-matching surfaces: moving face-centred pressure or vertex-based displacement
- In some cases, data source may be external: FSI simulation with fluid flow solver in OpenFOAM and external structural analysis package
- In all cases, mapping operation is identical: already implemented
- In other cases (external data source), external data will be transferred to OpenFOAM for coupling. Data mapping operation is still the same
  - Make external mesh proxy in OpenFOAM
  - Use `patchToPatchInterpolation` to execute mapping
- Fortunately, `PatchToPatchInterpolation` is templated on patch type
- Also, `PrimitivePatch` is templated on a type of face (triangle, polygon) and a type of face container: assembling a stand-alone patch in OpenFOAM tools is easy!

Surface Mapping Tools

Surface Mapping: Using `patchToPatchInterpolation`

- Case 1: All data on OpenFOAM meshes

```
patchToPatchInterpolation interpolatorFluidSolid
(
    mesh.boundaryMesh()[fluidPatchID],
    stressMesh.boundaryMesh()[solidPatchID]
);

scalarField solidPatchPressure =
    interpolatorFluidSolid.faceInterpolate
    (
        p.boundaryField()[fluidPatchID]
    );

solidPatchPressure *= rhoFluid.value();

tForce.pressure() = solidPatchPressure;
```

Surface Mapping Tools

Surface Mapping: Using `patchToPatchInterpolation`

- Case 2: External mapping data source for structural mesh

```
typedef PrimitivePatch<face, List, const pointField&>
    standAlonePatch;

PatchToPatchInterpolation<polyPatch,standAlonePatch> airToStruct
(
    CFDPatch,
    StructPatch,
    intersection::FULL_RAY
);

PatchToPatchInterpolation<standAlonePatch,polyPatch> structToAir
(
    StructPatch,
    CFDPatch,
    intersection::VISIBLE
);
```

Close Coupling: Matrix Level

Matrix Level Coupling

- In many cases, Picard iterations (explicit coupling) simply does not work or it is too slow
- Discretisation machinery in OpenFOAM is satisfactory and needs to be preserved
- Multi-domain support must allow for some variables/equations to be coupled, while others remain separated
- Example: conjugate heat transfer
    - Fluid flow equations solved on fluid only
    - Energy equation discretised separately on the fluid and solid region but solved in a single linear solver call
- Historically, conjugate heat transfer in commercial CFD is “hacked” as a special case: we need a **general arbitrary matrix-to-matrix coupling**
- The problem is in insufficient flexibility of matrix support

Linear Algebra and Solver Support

Rewrite of Linear Algebra Classes

- Current implementation of linear algebra and matrix support is over 12 years old
- . . . yet it is used throughout the library
- Limitations and implementation issues
  - Matrix stored in arrow format with scalar coefficients
  - Handling of vector variables poor and messy, especially for coupled boundary conditions, e.g. symmetry plane for vectors
  - Equation segregation enforced by form of matrix: unacceptable!
  - Need flexibility in addressing assembly and faster linear solvers
- Implementation issues for complex linear systems: multi-region equations, block coupled matrices, saddle-point system, complex constraints
- Some new algorithms difficult to handle: pressure-based coupled solver

Objective: Complete Rewrite of Linear Algebra and Linear Solver Classes

- Conjugate heat transfer class of problems and solver methodology
- Block solution of vector/tensor variables with full coupling
- New fast linear solvers handling multi-matrix and block-matrix systems

Close Coupling: Matrix Level

Assembling a Matrix for Conjugate Heat Transfer Problems

- OpenFOAM supports multi-region simulations, with possibility of separate addressing and physics for each mesh: multiple meshes, with local fields
- Some equations present only locally, while others span multiple meshes

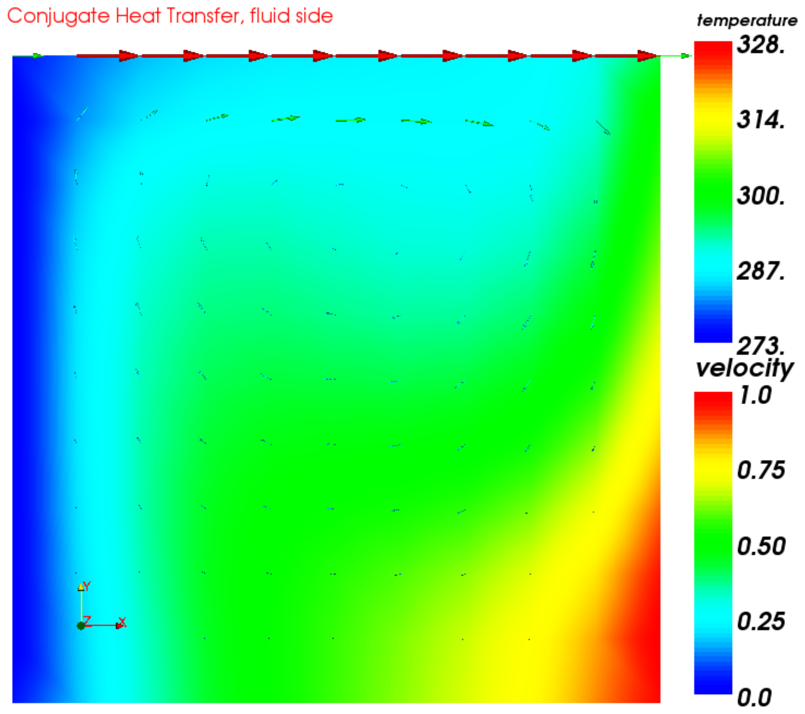
```
coupledFvScalarMatrix TEqns(2);

TEqns.hook
(
    fvm::ddt(T) + fvm::div(phi, T)
  - fvm::laplacian(DT, T)
);

TEqns.hook
(
    fvm::ddt(Tsolid) - fvm::laplacian(DTsolid, Tsolid)
);

TEqns.solve();
```

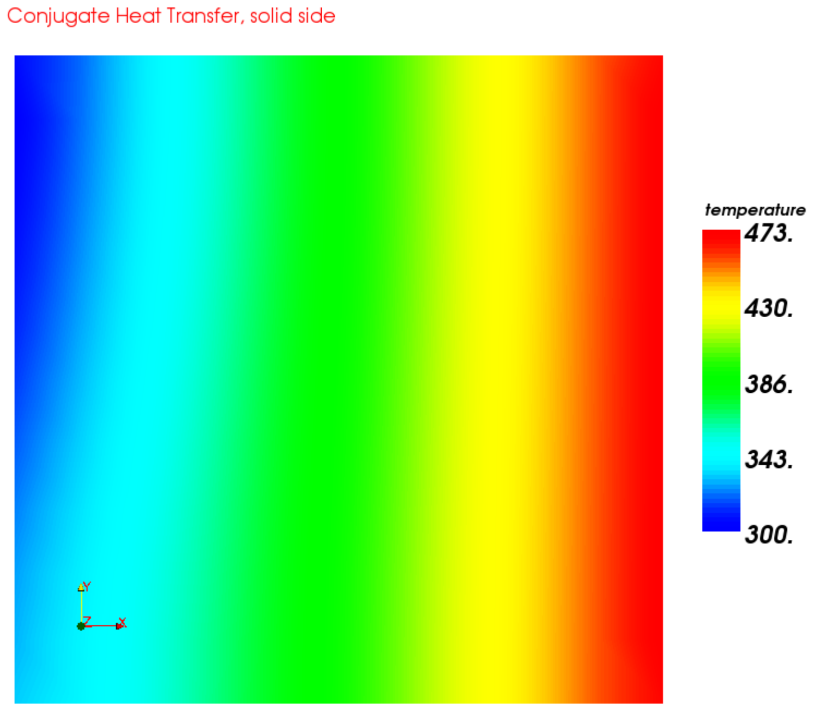
- Coupled solver handles multiple matrices together in internal solver sweeps

Linear Algebra and Solver Support

Example: Conjugate Heat Transfer

- Coupling may be established geometrically: adjacent surface pairs
- Each variable is stored only on a mesh where it is active: (U, p, T)
- Choice of conjugate variables is completely arbitrary: e.g. catalytic reactions
- Coupling is established only per-variable: handling a general coupled complex physics problem rather than conjugate heat transfer problem specifically





Close Coupling: Matrix Level

Matrix Level Coupling: Components of Implementation

- List of matrices for each coupled component mesh
- Coupled boundary condition, where the coupling may be established on a per-variable basis
- Mesh support for separate or coupled discretisation
- Special coupled boundary condition, derived from `coupledFvPatchField`
- Set of new multi-matrix solvers: all algorithms generalise without issues
- Arbitrary equation s and variables can be coupled on demand: much better that plain-vanilla conjugate heat transfer!

Some Limitations

- Currently, matrix-to-matrix coupling may only be established on boundaries: limitation of coupled LDU interfaces. This is not sufficient for “best” multi-phase solvers with volume coupling
- No AMG coarsening across matrices
- The code is new – needs some validation and debugging

Future Work

Generalisation of Matrix-Level Support

- Generalisation of matrix coefficients to be completed and merged with the block matrix work
- Improve the handling of coupled LDU interfaces: volume and surface coupling
- Improve and generalise linear solver support for block-matrix systems
- Work on improved boundary condition support in discretisation

Generalisation of Conjugate Physics Simulations

- Pack up each physics solver into a class, constructed given a mesh
- Create coupling-specific classes, e.g. FSI
- The above will allow run-time selection of multiple coupled physics systems in a single solver. Example: free surface VOF + elasto-plastic solid