Programming in OpenFOAM

Lecture 7

Balogh Miklós

March 25, 2014

Table of Contents

1 Basic concept

2 Programming

3 Variables

4 Loops

Difference between commercial and open-source

- Commercial solvers (Fluent, CFX, StarCCM++)
  - Black-box (way of implementation is not known)
  - UDF - User Defined Functions (ANSI C, Fortran)
  - Modification is limited
- OpenFOAM
  - The source code is accessible
  - Direct implementation is possible
  - No limitation (limited by common sense)

Applications and libraries

- Applications - standalone objects
  - Solvers - icoFoam, simpleFoam, etc.
  - Utilities - blockMesh, setFields, etc.
- Libraries
  - Turbulence models
  - Specific inlet/outlet boundary conditions
  - Specific wall functions
  - Additional source terms

Programming language of OpenFOAM

- OpenFOAM is a C++ library (toolbox)
  - Object-oriented, such as C++
  - Provide the mechanism (classes) to declare types and associated operations
  - In verbal and mathematical languages used in science and engineering
- Interpretation
  - Physics: Velocity field and magnitude
  - Math: U and $|U|$
  - OpenFOAM: volVectorField U and volScalarField Umag = mag(U)

Equation representation in OpenFOAM

Example: $\dfrac{\partial \rho U}{\partial t} + \nabla \cdot \phi U - \nabla \cdot \mu \nabla U = -\nabla p$

Listing 1: solver code

```
    solve
    (
        fvm::ddt(rho, U)
      + fvm::div(phi, U)
      - fvm::laplacian(mu, U)
        ==
      - fvc::grad(p)
    );
```

Variables

- Local single variables (e.g. in a loop)
  - scalar
  - vector
  - tensor
- Dimensioned single variables
  - dimensionedScalar
  - dimensionedVector
  - dimensionedTensor
- Dimensioned field variables
  - volScalarField (scalarField)
  - volvectorField (vectorField)
  - volTensorField (tensorField)

Listing 2: single variables

```
    dimensionedScalar nu(transportProperties.lookup("nu"));
    dimensionedVector flowDir(transportProperties.lookup("flowDir"));
```

Variables - volScalarField

Listing 3: volScalarField

```
    Info<< "Reading field p\n" << endl;
    volScalarField p
    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

Variables - volVectorField

Listing 4: volVectorField

```
    Info<< "Reading field U\n" << endl;
    volVectorField U
    (
        IOobject
        (
            "U",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

Variables - volTensorField

Listing 5: volTensorField

```
    Info<< "Reading field Permeability\n" << endl;
    volTensorField Permeability
    (
        IOobject
        (
            "Permeability",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

Loops and cycles over cells

Listing 6: loop over cells

```
    volScalarField centres_ = mag(wallNorm_ & mesh_.C());

    // Limitation of centres to the ABL height in the cells
    forAll(centres_,cellI)
    {
      if(centres_[cellI] > hABL_.value())
      {
        centres_[cellI] = hABL_.value();
      }
    }

    // Similar to the following
    volScalarField centres_ = min(mag(wallNorm_ & mesh_.C()),hABL_);
```

Loops and cycles over faces

Listing 7: loop over faces of patches

```
    // Limitation of centres to the ABL height at the boundaries
    forAll(patches, patchI)
    {
      const fvPatch& curPatch = patches[patchI];
      forAll(curPatch, faceI)
      {
        vector cf = mesh_.Cf().boundaryField()[patchI][faceI];
        scalar ctemp = mag(wallNorm_.value() & cf);
        if(ctemp > hABL_.value())
        {
          centres_.boundaryField()[patchI][faceI] = hABL_.value();
        }
        else
        {
          centres_.boundaryField()[patchI][faceI] = ctemp;
        }
      }
    }
```