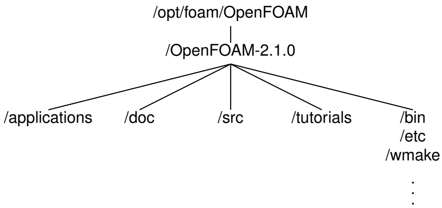
Write your own solver

Prof Gavin Tabor

Friday 25th May 2018

Installation directory structure



Easiest to modify code from core library

Applications (apps) in `/applications` subdirectory

Copy to user directory! (alongside user case directory)

App directory structure

Example – `icoFoam`

```
<grtabor@emps-copland>ls
createFields.H  Make  icoFoam.C  icoFoam.dep
```

Directory contains

- File `icoFoam.C`
- Other files (`.H`, `.C`)
- Directory `Make`

To compile, type `wmake`

Compilation

`wmake` is a make system – directs the compiler to compile specific files in particular ways. Controlled through files in `Make` :

- `files` – specifies which user-written files to compile and what to call the result
- `options` – specifies additional header files/libraries to be included in the compilation.

`files` :

```
icoFoam.C
EXE = $(FOAM_APPBIN)/icoFoam
```

1. Need to change `$(FOAM_APPBIN)` to `$(FOAM_USER_APPBIN)`
2. Probably need to change executable name!
3. Might need to change name of `.C` file

Example – Boussinesq approximation

For buoyancy-driven flow we often make use of the *B*oussinesq approximation : air modelled as incompressible with a body force proportional to $\Delta\theta$. Can we implement this into `icoFoam`?

Need to solve standard heat conduction equation :

$$\frac{\partial \theta}{\partial t} + \nabla.(\underline{u}\theta) = \frac{\kappa}{\rho_0 C_V}\nabla^2\theta$$

and alter the momentum equation

$$\frac{\partial \underline{u}}{\partial t} + \nabla.\underline{u}\ \underline{u} = -\nabla p + \nu\nabla^2\underline{u} - \beta\underline{g}(\theta_0 - \theta)$$

to accommodate this.

Standard `icoFoam`

```
int main(int argc, char *argv[])
{
    #include "setRootCase.H"

    #include "createTime.H"
    #include "createMesh.H"
    #include "createFields.H"
    #include "initContinuityErrs.H"

    while (runTime.loop())
    {
        Info<< "Time␣=␣" << runTime.timeName()

        #include "readPISOControls.H"
        #include "CourantNo.H"

        fvVectorMatrix UEqn
        (
            fvm::ddt(U)
          + fvm::div(phi, U)
          - fvm::laplacian(nu, U)
        );
        solve(UEqn == -fvc::grad(p));
```

Include files – `createFields.H`

Solve

$$a_p U_p = H(U) - \nabla p$$

to find U_p – Momentum predictor

```
        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);

                fvScalarMatrix pEqn
                (
                    fvm::laplacian(rUA, p)
                 == fvc::div(phi)
                );

                pEqn.setReference(pRefCell, pRefValue);
                pEqn.solve();

#           include "continuityErrs.H"

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

Enter pressure loop – set up variables

Solve

$$\nabla.\left(\frac{1}{a_p}\nabla p\right) = \sum_f \underline{S}.\left(\frac{H(\underline{U})}{a_p}\right)_f$$

to find p – Pressure corrector

Update flux using

$$F = \underline{S}.\left[\left(\frac{H(\underline{U})}{a_p}\right)_f - \left(\frac{1}{a_p}\right)_f (\nabla p)_f\right]$$

boussinesqFoam

Modify this in the following way :

1. Open `createFields.H` and read in the various properties :

```
    dimensionedScalar kappa
    (
        transportProperties.lookup("kappa")
    );
```

(similar lines for `rho0, Cv, theta0` and `beta`). Also worth introducing `hCoeff` :

```
  dimensionedScalar hCoeff = kappa/(rho0*Cv);
```

(...cont)

2. Introduce gravitational accelleration $\underline{g}$; read in from the same dictionary, but is a `dimensionedVector` rather than a `dimensionedScalar`.
3. Create a temperature field `theta` as a `volScalarField` and read it in. This is very similar to the pressure field, so make a copy of this and modify accordingly.
4. Modify the momentum equation (`UEqn`) to add the term

```
        + beta*g*(theta0-theta)
```

(...cont)

5. End of the PISO loop – create and solve the temperature equation :

```
fvScalarMatrix tempEqn
(
   fvm::ddt(theta)
 + fvm::div(phi,theta)
 - fvm::laplacian(hCoeff,theta)
);

tempEqn.solve();
```

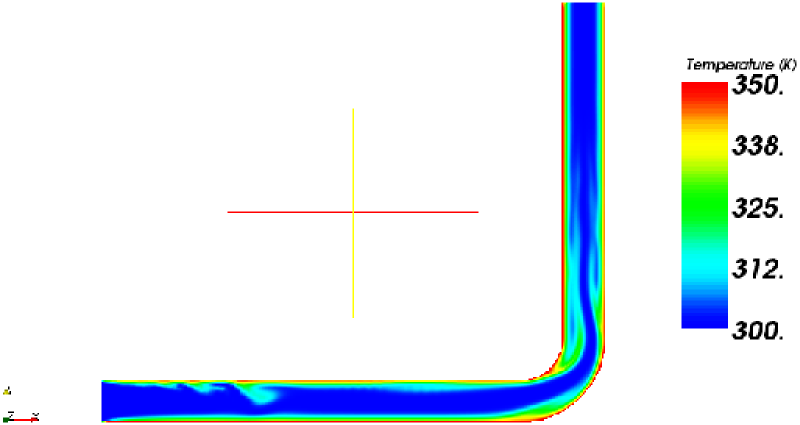
6. Compile this using `wmake` (rename executable as `boussinesqFoam`)

Case

We need to modify a case to function with `boussinesqFoam`. Use a pipe flow problem – modify as follows;

1. Create a `theta` file in the `0` timestep directory. This is best done by creating a copy of `U` and editing it. Don't forget to change the dimensions of `theta` as well.
2. Introduce the physical parameters. `boussinesqFoam` looks for the thermophysical constants in `transportProperties`; check that these are in there and that the values are correct.
3. The differencing schemes need to be specified for the `theta` equation. These are in `fvSchemes`; check that they are appropriate.
4. Finally, `solvers` in `fvSolution` needs an entry for the `theta` equation. Again, this has been provided, but you should check that it is correct.

Results



Example 2 – Casson model

The Casson model is a non-Newtonian viscosity model – used for chocolate, blood, etc. Can we implement in OpenFOAM?

Stress-strain relation for a fluid

$$\tau = \rho \nu \dot{\gamma} \qquad \text{where} \qquad \dot{\gamma} = \frac{1}{2}\left(\nabla \underline{u} + \nabla \underline{u}^T\right) \qquad \text{is rate of strain tensor}$$

$\nu = const$ is a Newtonian fluid. $\nu = \nu(\dot{\gamma}, \ldots)$ is non-Newtonian.

Casson model :

$$\nu(J_2) = \frac{\left[\left(\eta^2 J_2\right)^{1/4} + \sqrt{\frac{\tau_y}{2}}\right]^2}{\rho\sqrt{J_2}} \qquad \text{where} \qquad J_2 = \|\frac{1}{2}\left(\nabla \underline{u} + \nabla \underline{u}^T\right)\|^2$$

Implementation

To implement this convert the viscosity `nu` from `dimensionedScalar` into `volScalarField`. In `createFields` we create an appropriate `volScalarField` :

```
    Info << "Reading␣field␣nu" << nl << endl;
    volScalarField nu
    (
        IOobject
        (
            "nu",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );
```

and remove the original definition of `nu` as a `dimensionedScalar`.

Also need to read in Casson model coefficients in `createFields.H`.

Calculate the value of `nu` somewhere within the iterative loop :

```
volScalarField J2 = 0.5*magSqr(symm(fvc::grad(U))) +
  dimensionedScalar("tiny",dimensionSet(0,0,-2,0,0,0,0),0.0001);

nu = sqr(pow(sqr(eta)*J2,0.25) + pow(tau_y/2,0.5))/(rho*sqrt(J2));
```

Note :

- `J2` created locally – not being saved
- Introduce "tiny" to avoid division by zero

Results : `offsetCylinder` case from tutorials