CFD theory Sessions

Víctor Martínez Viol

October 14, 2016

Abstract

This paper has been written with the effort on getting comfortable when using LaTeXso please don't hate. It includes some notes taken in *Application of CFD to engineering problems.* NANDO NANDO

1 Session 4

1.1 Using Salome Meca

The first hour consisted on building the geometry of an Y-pipe with a software called *Salome Meca*. A complete guide on video (with an older version of the program) can be found in:

`http://caelinux.org/wiki/downloads/docs/Pipe2007/PipeGeom.htm`

We open Salome by going to the folder that contains it, then we right-click on the `runAppli` file, select copy, and then select 'paste filename' in a terminal window. It should append the next command to the window '/opt/salome_meca/appli_V2016/runAppli'

When we have our geometry constructed we have to export the files produced as `.stl` files. This files are based in a bunch of coordinates refering to the points of the piece.

1.2 Constructing the mesh

We want to make the mesh using **ParaView**. We can also use directly *Salome-Meca* to mesh the pipe but its mesher is not as good as the one we're going to use. First we have to create a case. A good option is to look for a similar tutorial case as a basis for our mesh (and of course similar to the case we want to run). For the Y-pipe problem we can start with the motorbike case. To do so, create a folder and name it as Ypipe, then copy inside it the folders contained in the motorbike case. It can be done in a terminal by typing:

```
mkdir -p Ypipe
cp -r /opt/openfoam4/tutorials/
incompressible/simpleFoam/motorBike/* ./Ypipe
```

Once we have the folder created the first we are going to do is to copy our geometry inside the case. (Copy the .stl an paste it in /constant/triSurface). The next step is go to the `system` folder, inside there is a file named

`BlockMeshDict`. We must modify the block geometry to put the pipe inside it and mesh only the domain taht really interest us. The new values for the vertices of the block are:

```
vertices
(
     (-1  -2.5  -0.5)
     (3  -2.5  -0.5)
     (3   3  -0.5)
     (-1   3  -0.5)
     (-1  -2.5  0.5)
     (3  -2.5  0.5)
     (3   3  0.5)
     (-1   3  0.5)
);
```

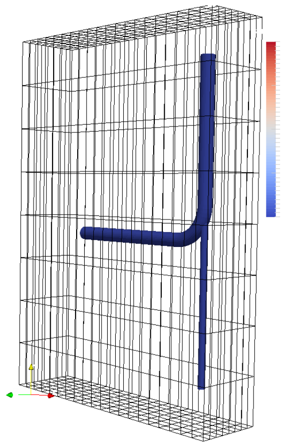
Figure 1: First mesh. Toooooo coarse

1.3 Refining the mesh

The next steps involve refining the mesh. To do so we are going to use **snappyHexMesh**(See reference). We must define the level 0 of the mesh. In this level we must have a meh as regular as possible. Open `BlockMeshDict` an edit the following line:

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (44 64 10) simpleGrading (1 1 1)
);
```

The values (44 64 10) define the number of cells on the x, y and z respectively. When we achieve a regular and dense enough mesh we must edit the lines below to have all the faces in the same type patch.

```
boundary
(
    frontAndBack
    {
        type patch;
        faces
        (
            (3 7 6 2)
            (1 5 4 0)
            (0 4 7 3)
            (2 6 5 1)
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

The next step is to take `SnappyHexMesh` dictionary. Rather than create it is better to take it from `openfoam4/applications/uilities/mesh/generation/snappyhexmesh` and paste it to the `Ypipe/system` folder. Inside the file there are a lot of comments and explanations about the mesh configuration and parameters (it's some kind of template).

Let's see the geometry section of `snappyHexMeshDict` . It is possible to define an upper level of discretization inside our blockMesh using `searchableBox`. But for this case is not necessary so we can delete it from the dictionary. Then we have to define all the solids we have in the problem. Every `.stl` file we have stored in the triSurface has to be defined here as follows. We can also include sub-solids in the same file but we saved them in different files sothe regions section should we empty.

```
geometry
{
    Inlet.stl
    {
        type triSurfaceMesh;
        regions
        {
        }
    }

    Inlet2.stl
    {
        type triSurfaceMesh;
        regions
        {
        }
    }

    Outlet.stl
    {
        type triSurfaceMesh;
        regions
        {
        }
    }

    Walls.stl
    {
        type triSurfaceMesh;
        regions
        {
        }
    }
    sphere2
    {
        type searchableSphere;
        centre  (1.5 1.5 1.5);
        radius  1.03;
    }
};
```

SnappyHexMesh is one of the only meshers that can perform a parallel mesh of the domain (using multiple cores). This feature is enabled by *castellatedMesh* feature. The parameter `maxLocalCells` and `maxGlobalCells` define the maximum number of cells per core and the maximum number of cells on the global mesh respectively. Another important parameter is the *Surface based refinement*. with this we can modify the maximum and minimum level of refinement of the solids. It detects when to surfaces are intersecting and let us apply an upper level of refinement to this edges. We have to define here all the solids as we've done before[1].

Lastly, we are going to modify the `locationInMesh` parameter. This define if we want to study the outside or the inside of the solid (and so we mesh the specified region of the domain). For this case we are going to study the flow inside the pipe so we put a point inside it (`(2.522222332 1.2324 0.0343)` is a good point) Now we save and close the file, and then type texttttSnappyHexMesh in the terminal window. The we can view the results by opening paraFoam.

2 Session 5

2.1 Constructing the mesh II

In the last session we worked with three different `.stl` files but we can also put all of these surfaces in a single file. To do so, we must modify the one of the files in order to append all he geometries. The Pdf `pipe.pdf` which can be found in atenea explains all the process.

1. Put all the solids into a single file

[1]It's important to include the full file name e.g. file.stl if we are dealing with separated files. If we work with regions only the patch name

2. Modify SnappyHexMesh dictionary to specify these solids into the regions section.

The process is nearly the same as in the session 4. When executing a command we can print the terminal output of the command to a log (e.g `snappyHexMesh > snappy.log` or `foamJob -s snappyHexMesh, this one prints also the output to the terminal`. we can clean the polymesh folder directy by tipping `foamCleanPolyMesh`. If we want to only have a single mesh stored in the constant folder we can do it by tipping `snappyHexMesh -owerwrite`. It's important to have the final mesh stored in the constant folder.

2.2 Simulating the case

A good advice before simulating a case is to make a copy of the 0 folder in order to be able to reset the case.

Now we are going to edit the `0` inside the 0.orig folder to put our boundary conditions. Inside the boundary section of this file we have to modify the code to appear as following. inlet1 0 -1 0 inlet2 -10 0 0

```
internalField uniform(0 0 0);

.... %delete all the include lines under initialConditions

patchName
{
        type            fixedValue;
        value           uniform(Ux Uy Uz);
}
```

the `inletOutlet;` option in the type field means that in the oultet when the flow is going out of the domain the boundary condition is Zero Gradient avoiding the flow to returning inside of the domain.

Now the computer know the velocity on different points of the geometry. And the computer will compute the relative pressure between the points on the domain. Hence, it has to know the pressure in one of the points. We modify the `p` file to have pressure equal to 0 in outlet and a zerogradient in the rest. Then we have to modify the file turbulenceProperties to perform a laminar simulation.

3 Session 6

3.1 Solvers

- incompressible
  - Simple Foam
    - steady
    - turbulent
  - PisoFoam
    - transient
  - PimpleFoam (piso + simple)
- compressible
  - rhoSimpleFoam
    - steady
    - compressible

It's interesting to differentate `pimpleDyMFoam` which is based in the use of Dynamic Mesh. It's not easy to implement but it's interesting.

3.2 Parallel Computation with OpenFoam

Our aim is to decompose the problem we are dealing with in four or whatever is the number of cores that we have. The instruction that we have to use is `decompose Par`. This methodology needs to know the Boundary conditions between the differents parts of the decomposed domain.

Using this method doesn't reduce the time as a linear proportion. This is because the major part of the time is used on communication between the cores.

3.3 Taking a tutorial

We are going to simulate the motorBike case using parallel computation.

First of all we have to copy the case in a new folder using:

```
mkdir -p Parallel
cp -r /opt/openfoam4/tutorials/incompressible/simpleFoam/motorBike/* ./
```

Then we have to copy the surface of the motorBike in our folder. To do so type:

```
cp -r /opt/openfoam4/tutorials/resources/geometry/motorBike.obj.gz
./Parallel/constant/triSurface/
```

Now we must perform a surfaceFeatureExtract because we have a complex surface. This is used to tell snappyHexMesh to pay special attention to critical points in the surface of the motorbike. It writes a lot of files, the most interesting is the one named motorByke.emesh. We can specify the level of discretization in those critical points by `features` in **snappyHexMeshdict**.

Meshing the case only with one core takes 246s. Now we are going to clone the case and perform it with 2 cores. Type:

```
 foamCloneCase Parallel/ Parallel2cores
 rm -rf 1
```

Open the file decomposeParDict. There are some interesting parameters. Modify it as follows and then run `decomposePar`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 2;
```

```
method              hierarchical;
// method              ptscotch;

simpleCoeffs
{
    n                   (4 1 1);
    delta               0.001;
}

hierarchicalCoeffs
{
    n                   (2 1 1);
    delta               0.001;
    order               xyz;
}

manualCoeffs
{
    dataFile            "cellDecomposition";
}

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

tWhen using parallel computation the cores we are usng needs to communicate between them. To do so, we have to use the library MPI (Multiple Processor Interface). It's difficult to implement, so OpenFoam has a function to make it easier. We only have to type: foamJob −s −p snappyHexMesh. This command means that we want to run snappyHexMesh in parallel. With two cores the mesh takes $xxxs$

Now the computer know the velocity on different points of the geometry. And the computer will compute the relative pressure between the points on the domain. Hence, it has to know the pressure in one of the points. We modify the `p` file to have pressure equal to 0 in outlet and a zerogradient in the rest. Then we have to modify the file turbulenceProperties to perform a laminar simulation.

References