OpenFOAM - Cavity Tutorial

OpenFOAM

From Wikipedia's article about OpenFOAM:

> *OpenFOAM is a C++ toolbox for the development of customized numerical solvers, and pre-/post-processing utilities for the solution of continuum mechanics problems, including computational fluid dynamics (CFD).*

This is a small introduction guide to OpenFOAM. The batch scripts presented here can be used on the HPC computer *Vilje*, but this tutorial can be done on any modern laptop or workstation. The cases in this document is based on the tutorials supplied with the OpenFOAM package. All tutorials are for instructional purposes, and the results are not validated in any way. The first chapter is to quickly help you to run your first simulation, and then we will build upon this with parallel jobs and a more fine-grained mesh in the next chapters.

Lid-driven cavity flow

The *lid-driven cavity flow* tutorial is the easiest and most well documented tutorial on OpenFOAM. The original tutorial is found in the OpenFOAM documentation. Before you start you should log on to *Vilje*.

1. Load the OpenFOAM module:

   Terminal window

   ```
   module load openfoam
   ```

   It might be wise to note what version of OpenFOAM you are loading. This can be done by listing the different software packages available, and look for the default OpenFOAM version:

   Terminal window

   ```
   module avail
   ```

   This guide is written for version 2.1.1, but should apply for other versions as well (at least the 2.x.x versions of OpenFOAM).
2. Since we don't want to write all the input files from scratch, we copy the tutorial files to a directory of your choice:

   Terminal window

   ```
   cp -R $FOAM_TUTORIALS/incompressible/icoFoam/cavity ./
   cd cavity
   ```

   On *Vilje* you have access to the folders `/home/ntnu/username` and `/work/username`. The difference is the storage quotas and how long you are allowed to have files there. On `/work` there is a larger quota than on `/home`, but the files on `/work` are automatically deleted after a certain time after last access.
3. Then we need a job script to run OpenFOAM (in no cases should you run OpenFOAM on the login node, independent on the simulation size). Based on the template in Old Job Execution a small job script has been created:

OFjob1.sh

```
#!/bin/bash
#PBS -A nn1234k
#PBS -N OF_tutorial_1
#PBS -l walltime=00:01:00
#PBS -l select=1:ncpus=1:mpiprocs=1

# Load modules
module load openfoam

# Enter the directory where the user is when submitting the job
cd $PBS_O_WORKDIR

# Run OpenFOAM
blockMesh
icoFoam
```

Create a new textfile, for example in your home directory, and give it an appropriate name. For simplicity, we assume you called this file *OFjob1.sh*. Copy the job script into the textfile, and replace nn1234k with your account number. The file must be made executable the usual way:

Terminal window

```
chmod +x OFjob1.sh
```

4. Then this file can be submitted to the job queue on *Vilje*:

Terminal window

```
qsub OFjob1.sh
```

When the job is finished (should be quick, depending on the queue), several folders have appeared in the directory. The names are *0.1*, *0.2* etc, and they are the folders where the results from the different timesteps are stored.

5. Run Paraview and inspect the results as described in the end of this article.

Parallel lid-driven cavity flow with MPI

The next step is to do a parallel run, i.e. use more than one processor (core) to do the calculations. The method chosen by OpenFOAM is domain decomposition. That means that the computational domain is divided into smaller subdomains, and each process is assigned one subdomain. The different processes then need to communicate to exchange data at the boundaries and other control and synchronization information. To do this decomposition, OpenFOAM has a tool called *decomposePar*. In OpenFOAM, the norm is that each tool that need configuration, has a separate configuration file, called a dictionary. The configuration file (dictionary) for *decomposePar* is called `decomposeParDict` (this is the usual naming convention) and is located in the `system` directory in the case folder.

As *vilje* has 16 CPU cores on each node, we will use all these in our analysis (independent of how wise it is to decompose a 400-cell case in 16 domains).

To create the dictionary, create a textfile called `decomposeParDict` in the `system` directory of your case folder. Paste the following into the file:

system/decomposeParDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 16;

method          simple;

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

distributed     no;

roots           ( );


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

To learn more about the domain decomposition, you can look at the dam break tutorial in the OpenFOAM documentation.

Our job script also needs to be modified so we create a new one, called *OFjob2*. Paste the following into the file (again remember to change the account number):

OFjob2

```
#!/bin/bash
#PBS -A nn1234k
#PBS -N OF_tutorial_2
#PBS -l walltime=00:05:00
#PBS -l select=1:ncpus=32:mpiprocs=16

# Load modules
module load mpt
module load openfoam

cd $PBS_O_WORKDIR

# Run OpenFOAM
blockMesh
decomposePar -force    # Force here will overwrite any existing mesh
mpiexec_mpt icoFoam -parallel # You do not need to specify the number of MPI processes in this command
reconstructPar
```

The reason why we have specified `ncpus=32` and `mpiprocs=16` is that the Sandy Bridge microarchitecture on *Vilje* has a feature called hyper-threading. This feature makes each physical core look like it is two logical cores, while it is not. The usefulness of this feature for CFD simulations are questionable, and we won't use it here. But to be on the safe side, we ask for 32 "cpus" on each node (total available logical cores) and specify that we will use 16 MPI processes. This is the default on *Vilje* for applications that do not utilize OpenMP (OpenFOAM don't).

Submit the script the usual way, and wait for the results. When the job is complete, a number of folders should be present. The difference from the previous run is that now there is also folders called *processor0*, *processor1* etc. This is the working directories for the individual processes. When the analysis is finished, the tool *reconstructPar* has the responsibility of reconstructing the previously decomposed domain into one continuously domain again. That means that the folders *processor0*, *processor1* etc. is no longer needed, and can be deleted if you want. The results for the simulation is merged together in the individual time step directories (*0.1*, *0.2* etc).

Parallel lid-driven cavity flow with fine mesh and MPI

The mesh size in the previous case was 20 by 20 by 1 (i.e. a 2D case), a total of 400 cells. Divided by four processes this gives 50 cells per process. The total solution time for such a case is totally dominated by startup overhead, and even the simplest laptop could have solved it in seconds. The Reynolds number of the previously simulation was 10, witch is extremely low. We will now refine the grid to a 40000-cell grid and go up to a Reynolds number of 100. We will do this by lowering the kinematic viscosity from 0.01 [m^2/s] to 0.001 [m^2/s] and changing the grid to a 200x200x1 case.

1. Open the file `transportProperties` in the `constant` folder. Change the kinematic viscosity to 0.001. The file should now look something like this:

constant/transportProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

nu              nu [ 0 2 -1 0 0 0 0 ] 0.001;


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

2. Open the `constant/polyMesh/blockMeshDict` file. Change the mesh to a 200x200x1 mesh. This is done in the *blocks* section of the file. This section should now look like:

constant/polyMesh/blockMeshDict

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (200 200 1) simpleGrading (1 1 1)
);
```

3. Because we have decreased the cell size, we also need to decrease the time steps to make sure that the solver converges more easily. Open the `system/controlDict` file, and set *deltaT* to one tenth of it's original value, 0.0005. While we are in this file it is worth noticing that the *writeControl* is set to *timeStep*. That means that it will write the solution to disk every *writeInterval* time steps. We still want 0.1 second write interval, so we change this to *runTime* and *0.1* respectivly. The last change we do is that we set the *runTimeModifiable* setting to *false*, as this will reduce the I/O activity. The final file should look like this:

system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     icoFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         0.5;

deltaT          0.0005;

writeControl    runTime;

writeInterval   0.1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable false;


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

You can read more about the settings in this file in the time and data input/output control chapter of the OpenFOAM documentation.

4. The job script from the previous run can be used, and no changed need to be made. Submit the job and wait for it to finish.
5. Inspect the terminal output from OpenFOAM. The output is located in a file called OF_tutorial_2.oXXXXXX, if you have kept the name in the jobscript given above. Here is the output after the final timestep from a run on *Vilje*:

```
Time = 0.5

Courant Number mean: 0.223619 max: 0.984534
DILUPBiCG:  Solving for Ux, Initial residual = 5.80291e-05, Final residual = 2.7915e-06, No Iterations 1
DILUPBiCG:  Solving for Uy, Initial residual = 7.30461e-05, Final residual = 2.19468e-06, No Iterations 1
DICPCG:  Solving for p, Initial residual = 0.00140484, Final residual = 9.74492e-07, No Iterations 278
time step continuity errors : sum local = 8.87556e-11, global = -2.25368e-21, cumulative = -4.19007e-18
DICPCG:  Solving for p, Initial residual = 0.00140093, Final residual = 9.52736e-07, No Iterations 279
time step continuity errors : sum local = 8.68566e-11, global = 6.16869e-20, cumulative = -4.12838e-18
ExecutionTime = 340.27 s  ClockTime = 341 s
```

As we can see, the solver uses a considerable number of iterations to solve the pressure equation for each timestep. Since this is a laminar steady-state problem, this indicates that the solution has not reached steady-state. The simulation should have been run for some more time. There are two options on how to do this: one is to start over again with another duration, and one is to start the simulation from where the last ended. We will explain the latter here.

1. Open the *system/controlDict* file, and change the *startFrom* variable to *latestTime*. This tells OpenFOAM to start from the latest timestep present. Then you change the *endTime* variable from *0.5* to *1.0*.
2. Now we do not want to regenerate the mesh or do the decomposition again. Take the jobscript, and remove the *blockMesh* and *decomposePar* commands. Leave the *reconstructPar* in place. The jobscript should look like this:

OFjob3

```
#!/bin/bash
#PBS -A nn1234k
#PBS -N OF_tutorial_3
#PBS -l walltime=00:05:00
#PBS -l select=1:ncpus=32:mpiprocs=16

# Load modules
module load mpt
module load openfoam

cd $PBS_O_WORKDIR

# Run OpenFOAM
mpiexec_mpt icoFoam -parallel
reconstructPar
```

3. Submit the job and wait. Inspect the output file and see if the solution has reached steady-state. If no iterations has been performed on the velocity components *Ux* and *Uy*, and only a few iterations has been performed on the pressure equations, the solution has probably reached steady-state. Note that we are using a transient solver, and that the solution actually never enter a perfectly steady state (a few iterations will always be done on the pressure equation) due to the nature of fluid problems.
4. Run Paraview and inspect the results as explained in the next page.