User guide, outline

- Available in `$WM_PROJECT_DIR/doc/Guides-a4`
- Introduction to OpenFOAM
- Tutorials showing how to run some *solvers* and *utilities*
- Pre- and post-processing using blockMesh, paraFoam and third-party products
- A look at available solvers and utilities

UserGuide, Ch.1, Introduction

- OpenFOAM is first and foremost a *C++ library*, used primarily to create executables, known as *applications*. The applications fall into two categories: *solvers*, that are each designed to solve a specific continuum mechanics problem; and *utilities*, that are designed to perform tasks that involve data manipulation.

- OpenFOAM is distributed with a large number of applications, but soon any advanced user will start developing new applications for his/ her special needs. The basic way to do this is to find and copy an application that almost does what is needed, and then to modify it by copy/paste from other applications that has some features that are needed.

- Special applications for pre- and post-processing are included in OpenFOAM. Converters to/from other pre- and post-processors are available.

- OpenFOAM has a graphical user interface (GUI) named FoamX, but most advanced users don't use it. It is much better to edit the files directly using Emacs.

UserGuide, Ch.2, Tutorials

- Set-up, simulation and post-processing of some cases.
- Some solvers and utilities are introduced.
- You will later on go through this chapter yourself and do the tutorials.
- We will quickly go through the basic procedure now. You will specifically see how to do the tutorials without FoamX.
- Also view the tutorials in Chapter 2 as examples of the tutorial you should produce in your project.
- OpenFOAM provides 'tutorials' in `$FOAM_TUTORIALS`, but most of those are just case files without explanations. Hopefully this course will generate explanations to those 'tutorials'. There is an alias to go to `$FOAM_TUTORIALS`, which is `tut`

UserGuide, Ch.2, Tutorials

- Other available and useful alias:

```
app      cd $FOAM_APP
foam     cd $WM_PROJECT_DIR
foamfv   cd $FOAM_SRC/finiteVolume
foamsrc  cd $FOAM_SRC/$WM_PROJECT
lib      cd $FOAM_LIB
run      cd $FOAM_RUN
sol      cd $FOAM_SOLVERS
src      cd $FOAM_SRC
tut      cd $FOAM_TUTORIALS
util     cd $FOAM_UTILITIES
```

- We will have a look at environment variables when looking at installation of OpenFOAM. Find out what an environment variable means by: `echo $FOAM_APP`. See all the environment variables by: `env`.

UserGuide, Ch.2, FoamX - a quick look

- I will now quickly show how to use FoamX for the cavity tutorial, but then we will simply edit the files instead. We will discuss most of Chapters 4 and 5 in the UserGuide. Read them yourselves.
- When you do the tutorials you can either use FoamX or edit the files.
- Experienced users usually edit the files directly.

UserGuide, Ch.2, Lid driven cavity

- Basic procedure when running a tutorial, in this case `cavity`:
  `cp -r $FOAM_TUTORIALS/icoFoam/cavity $FOAM_RUN`
  `run`
  You have copied the `cavity` tutorial and moved to `$FOAM_RUN`

- The grid is defined by a dictionary that is read by the `blockMesh` utility
  `blockMesh $FOAM_RUN cavity`
  You have now generated the grid in OpenFOAM format.

- Check the mesh by
  `checkMesh $FOAM_RUN cavity`
  You see the grid size, the geometrical size and some grid checks.

- This is a case for the `icoFoam` solver, so run
  `icoFoam $FOAM_RUN cavity > $FOAM_RUN/cavity/log&`
  You now run the simulation in background using the settings in the case, and forward the output to the `$FOAM_RUN/cavity/log` file, where the Courant numbers and the residuals are shown. To run nice, do:
  `nohup nice -n 19 icoFoam $FOAM_RUN cavity > $FOAM_RUN/cavity/log&`

UserGuide, Ch.2, Lid driven cavity - postprocess

- View the results using paraFoam:
  `paraFoam $FOAM_RUN cavity`
  Choose the final time step (0.5) and click Accept.
  Chose which variable to color by with `Display/Color by`
  Move, rotate and scale the visualization using the mouse
- We will learn how to use paraFoam more further on.
- Exit paraFoam: `File/Exit`
- The results may also be viewed using third-party products

UserGuide, Ch.2, Lid driven cavity - What did we do?

- We will have a look at what we did when running the `cavity` tutorial by looking at the case files.
- First of all it should be noted that `icoFoam` is a *Transient solver for incompressible, laminar flow of Newtonian fluids*
- The case directory originally contains the following sub-directories: `0`, `constant`, and `system`. After our run it also contains the output `0.1`, `0.2`, `0.3`, `0.4`, `0.5`, and log
- The `0*` directories contain the values of all the variables at those time steps. The `0` directory is thus the initial condition.
- The `constant` directory contains the mesh and a dictionary for the kinematic viscosity transportProperty.
- The `system` directory contains settings for the run, discretization schemes, and solution procedures.
- The `icoFoam` solver reads the files in the case directory and runs the case according to those settings.

UserGuide, Ch.2, Lid driven cavity - The constant directory

- The transportProperty file is a dictionary for the dimensioned scalar `nu`.
- The `polyMesh` directory originally contains the `blockMeshDict` dictionary for the `blockMesh` grid generator, and now also the mesh in OpenFOAM format.
- We will now have a quick look at the `blockMeshDict` dictionary in order to understand what grid we have used.

UserGuide, Ch.2, Lid driven cavity - blockMeshDict dictionary

- The blockMeshDict dictionary first of all contains a number of vertices:

```
convertToMeters 0.1;
vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
    (0 1 0)
    (0 0 0.1)
    (1 0 0.1)
    (1 1 0.1)
    (0 1 0.1)
);
```

- There are eight vertices defining a 3D block. OpenFOAM always uses 3D grids, even if the simulation is 2D.

- `convertToMeters 0.1;` multiplies the coordinates by 0.1.

UserGuide, Ch.2, Lid driven cavity - blockMeshDict dictionary

- The blockMeshDict dictionary secondly defines a block and the mesh from the vertices:

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);
```

- `hex` means that it is a structured hexahedral block.
- `(0 1 2 3 4 5 6 7)` is the vertices used to define the block. The order of these is important (read the UserGuide yourself).
- `(20 20 1)` is the number of grid *cells* in each direction.
- `simpleGrading (1 1 1)` is the expansion ratio, in this case equidistant. (read the UserGuide yourself).

UserGuide, Ch.2, Lid driven cavity - blockMeshDict dictionary

- The blockMeshDict dictionary finally defines three patches:

```
patches
(
    wall movingWall
    (
        (3 7 6 2)
    )
    wall fixedWalls
    (
        (0 4 7 3)
        (2 6 5 1)
        (1 5 4 0)
    )
    empty frontAndBack
    (
        (0 3 2 1)
        (4 5 6 7)
    )
);
```

UserGuide, Ch.2, Lid driven cavity - blockMeshDict dictionary

- Each patch defines a type, a name, and a list of boundary faces
- Let's have a look at the fixedWalls patch:

```
wall fixedWalls
(
    (0 4 7 3)
    (2 6 5 1)
    (1 5 4 0)
)
```

- `wall` is the type of the boundary.
- `fixedWalls` is the name of the patch.
- The patch is defined by three sides of the block according to the list, which refers to the vertex numbers. The order of the vertex numbers is important (Read the UserGuide, chapter 6 yourself).

UserGuide, Ch.2, Lid driven cavity - blockMeshDict dictionary

- To sum up, the blockMeshDict dictionary generates a block with:
  x/y/z dimensions 0.1/0.1/0.01
  20×20×1 cells
  `wall fixedWalls` patch at three sides
  `wall movingWall` patch at one side
  `empty frontAndBack` patch at two sides

- The type `empty` tells OpenFOAM that it is a 2D case.

- Read more about `blockMesh` yourself in the UserGuide (section 6.3).

- You can also convert mesh files from third-party products. (section 6.4).

UserGuide, Ch.2, Lid driven cavity - blockMeshDict dictionary

- `blockMesh` uses the `blockMeshDict` to generate some files in the `constant` directory:

  ```
  boundary
  faces
  neighbour
  owner
  points
  ```

- `boundary` shows the definitions of the pathches, for instance:

```
movingWall
{
    type wall;
    physicalType wall;
    nFaces 20;
    startFace 760;
}
```

- The other files defines the points, faces, and the relations between the cells.

UserGuide, Ch.2, Lid driven cavity - The system directory

- The `system` directory consists of three set-up files:

  `controlDict  fvSchemes  fvSolution`

- `controlDict` contains general instructions on how to run the case.
- `fvSchemes` contains instructions on which discretization schemes that should be used for different terms in the equations.
- `fvSolution` contains instructions on how to solve each discretized linear equation system. It also contains instructions for the PISO pressure-velocity coupling.

UserGuide, Ch.2, Lid driven cavity - The controlDict dictionary

- The `controlDict` dictionary consists of the following lines:

```
application     icoFoam;
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         0.5;
deltaT          0.005;
writeControl    timeStep;
writeInterval   20;
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression uncompressed;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
```

UserGuide, Ch.2, Lid driven cavity - The controlDict dictionary

- `application icoFoam;` tells FoamX to use the set-up specifications of the icoFoam solver.
- The following lines tells `icoFoam` to start at `startTime`=0, and stop at `endTime`=0.5, with a time step `deltaT`=0.005:

```
startFrom       startTime;
startTime       0;
stopAt          endTime;
endTime         0.5;
deltaT          0.005;
```

UserGuide, Ch.2, Lid driven cavity - The controlDict dictionary

- The following lines tells `icoFoam` to write out results in separate directories (`purgeWrite 0;`) every 20 `timeStep`, and that they should be written in `uncompressed ascii` format with `writePrecision` 6. `timeFormat` and `timePrecision` are instructions for the names of the time directories.

```
writeControl      timeStep;
writeInterval     20;
purgeWrite        0;
writeFormat       ascii;
writePrecision    6;
writeCompression  uncompressed;
timeFormat        general;
timePrecision     6;
```

- `runTimeModifiable yes;` allows you to make modifications to the case while it is running.

UserGuide, Ch.2, Lid driven cavity - A dictionary hint

- If you don't know which entries are available for a specific key word in a dictionary, just use a dummy and the solver will list the alternatives, for instance:

```
stopAt              dummy;
```

When running icoFoam you will get the message:

```
--> FOAM FATAL IO ERROR : dummy is not in enumeration
4
(
nextWrite
writeNow
noWriteNow
endTime
)
```

and you will know the alternatives.

UserGuide, Ch.2, Lid driven cavity - A dictionary hint

- Note that

```
startFrom               dummy;
```

only gives the following message without stopping the simulation:

```
--> FOAM Warning :
    From function Time::setControls()
    in file db/Time/Time.C at line 134
        expected startTime, firstTime or latestTime found 'dummy' in
    Setting time to 0
```

and the simulation will start from time 0.

- You may also use C++ commenting in the dictionaries:

```
// This is my comment
/* My comments, line 1
   My comments, line 2 */
```

UserGuide, Ch.2, Lid driven cavity - The fvSchemes dictionary

- The `fvSchemes` dictionary defines the discretization schemes, in particular the time marching scheme and the convections schemes:

```
ddtSchemes
{
    default         Euler;
}
divSchemes
{
    default         none;
    div(phi,U)      Gauss linear;
}
```

- Here we use the `Euler` implicit temoral discretization, and the `linear` (central-difference) scheme for convection.
- You usually don't change any other scheme.
- Find the available convection schemes using a 'dummy' dictionary entry. There are 46 alternatives, and the number of alternatives are increasing!

UserGuide, Ch.2, Lid driven cavity - The fvSolution dictionary

- The `fvSolution` dictionary defines the solution procedure.
- The solutions of the p linear equation systems is defined by:

```
p PCG
{
    preconditioner   DIC;
    tolerance        1e-06;
    relTol           0;
};
```

- The p linear equation system in solved using the Conjugate Gradient solver `PCG`, with the preconditioner `DIC`.
- The solution is considered converged when the residual has reached the `tolerance`, or if it has been reduced by `relTol` at each time step.
- `relTol` is here set to zero since we use the PISO algorithm. The PISO algorithm only solves each equation once per time step, and we should thus solve the equations to `tolerance 1e-06` at each time step. `relTol 0;` disables `relTol`.

UserGuide, Ch.2, Lid driven cavity - The fvSolution dictionary

- The solutions of the U linear equation systems is defined by:

```
U PBiCG
{
    preconditioner   DILU;
    tolerance        1e-05;
    relTol           0;
};
```

- The U linear equation system in solved using the Conjugate Gradient solver `PBiCG`, with the preconditioner `DILU`.
- The solution is considered converged when the residual has reached the `tolerance 1e-05` for each time step.

UserGuide, Ch.2, Lid driven cavity - The fvSolution dictionary

- The settings for the PISO algorithm are specified in the `PISO` entry:

```
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

- `nCorrectors` is the number of PISO correctors. You can see this in the log file since the p equation is solved twice, and the pressure-velocity coupling is thus done twice.

- `nNonOrthogonalCorrectors` adds corrections for non-orthogonal grids, which may sometimes influence the solution.

- The pressure is set to `pRefValue 0` in cell number `pRefCell 0`. This is over-ridden if a constant pressure boundary condition is used for the pressure.

UserGuide, Ch.2, Lid driven cavity - The 0 directory

- The `0` directory contains the dimensions, and the initial and boundary conditions for all primary variables, in this case `p` and `U`. `U`-example:

```
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{   movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }
    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    frontAndBack
    {
        type            empty;
    }}
```

UserGuide, Ch.2, Lid driven cavity - The 0 directory

- `dimensions [0 1 -1 0 0 0 0];` states that the dimension of `U` is m/s. We will have a look at this further later on.
- `internalField uniform (0 0 0);` sets `U` to zero internally.
- The boundary patches `movingWall` and `fixedWalls` are given the `type fixedValue; value uniform (1 0 0);` and `(0 0 0)` respectively, i.e. $U_x = 1m/s$, and $U = 0m/s$ respectively.
- The `frontAndBack` patch is given `type empty;`, indicating that no solution is required in that direction since the case is 2D.
- You should now be able to understand `0/p` also.
- The resulting 0.* directories are similar but the `internalField` is now a `nonuniform List<scalar>` containing the results. There is also a `phi` file, containing the resulting face fluxes that are needed to yield a perfect restart. There is also some time information in `0.*/uniform/time`. The `0.*/uniform` directory can be used for uniform information in a parallel simulation.

UserGuide, Ch.2, Lid driven cavity - The log file

- If you followed the earlier instructions you should now have a `log` file. That file contains mainly the Courant numbers and residuals at all time steps:

```
Time = 0.09

Courant Number mean: 0.116099 max: 0.851428
DILUPBiCG:  Solving for Ux, Initial residual = 0.000443324,
                    Final residual = 8.45728e-06, No Iterations 2
DILUPBiCG:  Solving for Uy, Initial residual = 0.000964881,
                    Final residual = 4.30053e-06, No Iterations 3
DICPCG:  Solving for p, Initial residual = 0.000987921,
                    Final residual = 5.57037e-07, No Iterations 26
time step continuity errors : sum local = 4.60522e-09,
                    global = -4.21779e-19, cumulative = 2.97797e-18
DICPCG:  Solving for p, Initial residual = 0.000757589,
                    Final residual = 3.40873e-07, No Iterations 26
time step continuity errors : sum local = 2.81602e-09,
                    global = -2.29294e-19, cumulative = 2.74868e-18
ExecutionTime = 0.08 s  ClockTime = 0 s
```

UserGuide, Ch.2, Lid driven cavity - The log file

- Looking at the `Ux` residuals

```
DILUPBiCG:  Solving for Ux, Initial residual = 0.000443324,
                    Final residual = 8.45728e-06, No Iterations 2
```

- We see that we used the `PBiCG` solver with `DILU` preconditioning.
- The `Initial residual` is calculated before the linear equation system is solved, and the `Final residual` is calculated afterwards.
- We see that the `Final residual` is less than our `tolerance` in `fvSolution` (`tolerance 1e-05;`).
- The `PBiCG` solver used 2 iterations to reach convergence.
- We could also see in the log file that the pressure residuals and continuity errors were reported twice each time step. That is because we specified `nCorrectors 2;` for the `PISO` entry in `fvSolution`.
- The `ExecutionTime` is the elapsed CPU time, and the `ClockTime` is the elapsed wall clock time for the latest time step.

UserGuide, Ch.2, Lid driven cavity - The log file

- It is of interest to have a graphical representation of the residual development.
- The `foamLog` utility is basically a script using `grep`, `awk` and `sed` to extract values from a log file.
- `foamLog` uses a database (`foamLog.db`) to know what to extract. The `foamLog.db` database can be modified if you want to extract any other values that `foamLog` doesn't extract by default.
- `foamLog` is executed on the `cavity` case with log-file `log` by:

```
foamLog $FOAM_RUN cavity log
```

- A directory `logs` has now been generated, with extracted values in ascii format in two columns. The first column is the `Time`, and the second column is the value at that time.
- Type `foamLog -h` for more information.
- The graphical representation is then given by Matlab, `xmgrace -log y Ux_0 p_0` or `gnuplot`: `set logscale y`, `plot "Ux_0","Uy_0","p_0"`.

UserGuide, Ch.2, Lid driven cavity - The icoFoam solver

- The `icoFoam` solver source code is located in `$WM_PROJECT_DIR/applications/solvers/incompressible/icoFoam` where you can find two files (`createFields.H` and `icoFoam.C`) and two directories (`FoamX` and `Make`). We don't bother about `FoamX` here.
- The `Make` directory contains two files (`files` and `options`) that specifies how `icoFoam` should be compiled. We will have a look at that later.
- In `icoFoam.C` you basically see a `runTime` loop, the specification and solution of the `UEqn`, and the `PISO` loop.
- In `createFields.H` the kinematic viscosity, the velocity field, and the pressure fields are read from the `startTime` directory. The face convections, `phi` are computed if they are not available in the `startTime` directory. Finally, the reference pressure is set if there are no constant pressure boundary conditions.
- We will study these files further later on, and we will learn how to copy a solver, modify it slightly and compile it.

UserGuide, Ch.2, Lid driven cavity - Tutorial

- You are now ready to do the lid driven cavity tutorial in the UserGuide:
  `acroread $WM_PROJECT_DIR/doc/Guides-a4/UserGuide.pdf`

- We have basically done everything until section 2.1.3, but we have been more detailed. You can make sure that you understand everything we have done, and then do 2.1.4-10 and 2.3 in the UserGuide (i.e. skip 2.2). Then you will also use the `turbFoam` solver, which basically adds turbulence and non-Newtonian fluids to `icoFoam`. The turbulence model is then specified in the `constant/turbulenceProperties` dictionary. You will also use the `interFoam` solver.

- In sections 2.1.4, 2.1.5.6-7, 2.1.10 and 7.1 paraFoam is desribed in more detail.

- You can skip the descriptions on how to modify `blockMeshDict` dictionaries for now.

- Here follows a number of useful utilities to study. Some of them are described in the UserGuide tutorials. If you find a utility that is not described you can include a description of it in your tutorial.

The mapFields utility

- The mapFields utility maps the results of one case to another case. See section 6.5 in the UserGuide for more information.
- Usage:
  `mapFields <source root> <source case> <target root> <target case>`
  Optional flags:
  `[-consistent] [-parallelSource] [-parallelTarget]`
  (You get this information by simply typing `mapFields`)
- The time used for the mapping is specified by `startFrom/startTime` in the *target* case.
- The flag `-consistent` is used if the geometry and boundary conditions are identical in both cases. This is useful when modifying the mesh density of a case. For non-consistent cases a `mapFieldsDict` dictionary must be edited, see section 2.1.19 in the UserGuide.
- The flags `-parallelSource` and `-parallelTarget` are used if any, or both, of the cases are decomposed for parallel simulations.

The sample utility

- This is used to produce graphs for publication.
- See section 7.5 and section 2.2.3 in the UserGuide.
- Usage:
  `sample <root> <case> [-parallel] [-constant] [-latestTime] [-time time]`
- Copy and modify `sampleDict` in the `solidDisplacementFoam plateHole` tutorial. In this case a line with 100 sample points is defined along the `leftPatch`, between `(0 0.5 0.25);` and `(0 2 0.25);`
- An example of a gnuplot command that uses the output and compares it to an analytical solution:
  `plot [0.5:2] '<datafile>', 1e4*(1+(0.125/(x**2))+(0.09375/(x**4)))`

The magU utility

- This utility reads the U velocity vector field from the time directories and writes the magnitude of the U velocity vector field to the time directories.
- Usage:
  `magU <root> <case> [-parallel] [-constant] [-latestTime] [-time time]`
- The source code is really simple, and can be found in:
  `$WM_PROJECT_DIR/applications/utilities/postProcessing/velocityField/magU`

The Ucomponents utility

- This utility reads vector field `U` and writes scalar fields `Ux`, `Uy` and `Uz` in the corresponding time directories.
- Usage:

```
Ucomponents <root> <case> [-parallel] [-constant]
                  [-latestTime] [-time time]
```

- The source code is really simple, and can be found in:

```
$WM_PROJECT_DIR/applications/utilities/postProcessing/
                  velocityField/Ucomponents
```

The setFields utility

- The `setFields` utility
- Usage:
  `setFields <root> <case> [-parallel] [-constant] [-latestTime] [-time time]`
- A `setFieldsDict` dictionary is used. Find an example in the `interFoam damBreak` tutorial. The `defaultFieldValues` sets the default values of the fields. A `boxToCell` bounding box is used to define a set of cells where the `fieldValues` should be different than the `defaultFieldValues`. `numberOfSubdomains` specifies the number of subdomains the grid should be decomposed into.

The funkySetFields utility

- The `funkySetFields` utility is a user contributed utility available in the OpenFOAM Wiki:
  `http://openfoamwiki.net`
  Click Contributions/Utilities/prePocessing
- `funkySetFields` is a development of the `setFields` utility.

Probes

- Probes the development of the results during a simulation, writing to a file
- Copy and modify the `functions` part at the end of the `controlDict` for the `oodles` `pitzDaily` tutorial.
- Visualize the file in Matlab after removing the first line in the output file. It is probably easy to visualize also with gnuplot and xmgrace.

decomposePar

- `decomposePar` makes a domain decomposition for parallel computations
- Usage:
  `decomposePar <root> <case> [-cellDist] [-fields] [-filterPatches] [-copyUniform]`
- A `decomposeParDict` specifies how the grid should be decomposed. An example can be found in the `interFoam damBreak` tutorial.
- There are some different alternatives for which method to use for the decomposition. See the UserGuide, section 3.4.
- We will discuss parallel processing later on.

reconstructPar

- `reconstructPar` is the reverse of `decomposePar`, reassembling the grid and the results.
- Usage:
  `reconstructPar <root> <case> [-constant] [-latestTime] [-time time]`
- This is usually done for post-processing, although it is also possible to post-process each domain separately by treating an individual processor directory as a separate case when starting `paraFoam`.
- We will discuss parallel processing later on.

Specifying a maximum Courant number and varying time steps

- Some solvers, like the interFoam solver allows a varying time step, based on a maximum Courant number. Some extra entries should then be added to the controlDict dictionary:

```
adjustTimeStep  yes;
maxCo           0.5;
maxDeltaT       1;
```

- The solver is told to adjust the time step so that the output still occurs at specific times using:

```
writeControl    adjustableRunTime;
writeInterval   0.05;
```

Hints on how to get convergence from scratch (In order of importance)

- Use a good grid.
- Set appropriate boundary conditions.
- Initialize turbulent quantities to their inlet average value.
- Run potentialFoam to get a good starting value for U and phi.
- Set low order discretization.
- Use hard under-relaxation (don't forget under-relaxation when running transientSimpleFoam!).
- Use short time steps, particularly when using PISO which has no under-relaxation.
- Reduce the Reynolds number temporarily by increasing the viscosity. Don't forget to change it back!

More tutorials can be found in

- The Programmer's guide, chapter 3
- The OpenFOAM Wiki
- The OpenFOAM Forum
- The Internet