Compiling applications and libraries

- Applications are divided into *solvers* and *utilities*, and are compiled to executables that can be run.
- Libraries containing the OpenFOAM classes are dynamically linked during the compilation of the solvers and utilities.
- Dynamic linking means that the libraries are not part of the *solver* or *utility* executable, so they must exist in an appropriate path on the computer where the *solver* or *utility* is run. If you have installed OpenFOAM on the computer according to the instructions, all should be fine. Be careful if you run in parallel - all the CPUs must see the OpenFOAM installation!
- You can check which libraries are used by `icoFoam` by typing:
  `` ldd `which icoFoam` ``
- If a library is not found something like the following will be printed:
  `libfiniteVolume.so => not found`
- A library may also be dynamically linked to other libraries.

Compiling applications and libraries

- The `wmake` compilation script that is based on `make` helps you compile all or parts of OpenFOAM.
- `wmake` can be executed in any directory containing a `Make` directory. The `Make` directory contains instructions for `wmake`.
- We will now learn how to compile our own `icoFoam` solver, named `myIcoFoam`.
- First copy the source code to your working directory to make sure that we don't modify the original code:

  ```
  cd $WM_PROJECT_DIR
  cp -r --parents applications/solvers/incompressible/icoFoam $WM_PROJECT_USER_DIR
  ```

  (This way we use the same structure as in the source code!)

  ```
  cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible/icoFoam
  ```

  `wclean` (to remove the files from the previous compilation)
- Modify line 3 of `Make/files` to: `EXE = $(FOAM_USER_APPBIN)/myIcoFoam`
- Compile using `wmake` ...(continued)

Compiling applications and libraries

- You get an exectuable `myIcoFoam` in `$FOAM_USER_APPBIN`. Renaming the executable to `myIcoFoam` makes sure that there aren't two executables with the same name, and putting it in `$FOAM_USER_APPBIN` makes sure that we don't contaminate the originally downloaded directories. For central installations of OpenFOAM you also don't have write-permisson, so you must put the executable in your own working directory.

- To run your executable you might have to `rehash`, so that the contents of the `path` is updated.

- `which myIcoFoam` now gives: `$FOAM_USER_APPBIN/myIcoFoam`

- `myIcoFoam` is an exact copy of `icoFoam`, so we can test it on the `cavity` tutorial:

```
run
cd cavity
myIcoFoam >& log_myIcoFoam &
```

Compiling applications and libraries

- The `wmake` script may take arguments:
  `wmake <optionalArguments> <optionalDirectory>`,
  where `<optionalDirectory>` is the directory where `wmake` should be executed (if other than the current).
  `<optionalArguments>` is used when compiling libraries:

Argument	Type of compilation
`lib`	Build a statically-linked library
`libso`	Build a dynamically-linked library
`libo`	Build a statically-linked object file library
`jar`	Build a JAVA archive
`exe`	Build an application independent of the specified project library

  In this course we will only use the `libso` argument.

- Environment variables used by `wmake` are shown by:
  `env | grep WM_`
  (see description in the UserGuide)

- `wclean` deletes the files generated by the compilation in the local application/library.

- `rmdepall` recursively removes dependency files.

Clean-up scripts

- There are some different clean-up scripts in `$WM_PROJECT_DIR/bin`:

```
wclean
wcleanAlmostAll
wcleanMachine
wcleanAll
wcleanLnIncludeAll
```

- Have a look inside the scripts to find out what they do.
- There are other interesting scripts as well.

Debug messaging and optimization switches

- Debug messaging and optimization switches can be viewed and modified in `$WM_PROJECT_DIR/etc/controlDict`.
  Let us have a look in that file...
- Debug messaging is activated for a specific class by setting the switch to 1 (sometimes 2 and 3 also).
- You do not have write access to the pre-installed file...
- Do the following to get a personal global controlDict file:

```
mkdir -p $HOME/.OpenFOAM/1.6.x
cp $WM_PROJECT_DIR/etc/controlDict $HOME/.OpenFOAM/1.6.x
```

- Read yourself in the UserGuide.

Running applications in parallel

- `decomposePar` makes a decomposition of your case.
- `reconstructPar` reconstructs your results.
- You need a `decomposeParDict` to tell these applications how to decompose/reconstruct. See example in the `interFoam/damBreak` tutorial. Read more about the decomposition methods in the UserGuide.
- `openMPI` is the default MPI implementation in OpenFOAM.
- You can use a `hostfile`, containing the computers that take part of the computation, one computer per line, and one line for each process. The number of cpus that should be used on multi-cpu computers should be specified by cpu=2 after the name of the computer.
- Run the `interFoam/damBreak` tutorial in parallel by:

```
run
cp -r $FOAM_TUTORIALS/interFoam/damBreak damBreak
cd damBreak ; blockMesh ; setFields
```
(three commands on the same line!)
```
decomposePar
mpirun -np 4 interFoam -parallel >& log &
reconstructPar
```

- You can use other MPI implementations, such as MPICH and LAM.

Standard solvers, utilities and libraries

- The UserGuide lists the standard solvers, utilities and libraries. This is sometimes an incomplete list!
- You can also list the standard solvers by looking in the `$WM_PROJECT_DIR/applications/solvers` directory. Move there using the `sol` alias. The descriptions of the solvers are available in the source files.
- You can also list the standard utilities by looking in the `$WM_PROJECT_DIR/applications/utilities` directory. Move there using the `util` alias. The descriptions of the utilities are available in the source files.
- You can also list the standard libraries by looking in the `$FOAM_LIB/$WM_OPTIONS` directory. Move there using the `lib` alias. You can't however find the descriptions there - see the UserGuide or the source code.