Debugging OpenFOAM implementations

Debugging OpenFOAM implementations

- It is impossible to do bug-free programming (trust me!), so you should always verify your implementations.
- When you run into problems, such as code crash, or mysterious behaviour, you also need some debugging approach.
- There are many debugging approaches, and we will discuss three alternatives here:
  - `Info` statements in the code
  - Built-in `DebugSwitch` option in OpenFOAM (similar to the above - you will see)
  - In separate document:
    Debugging using the Gnu debugger, GDB (`http://www.gnu.org/software/gdb/`)
- We will now go through these alternatives...

Debugging using Info statements

- The simplest way to do debugging is to write out intermediate results to the screen, and check that those results are correct.
- In OpenFOAM this is done using `Info` statements.
- This kind of debugging does not allow any control of the running of the code while debugging. It will just print out additional information.
- `Info` debugging requires that new lines are inserted in the source code, and that the source code must be re-compiled whenever a new `Info` statement has been added.
- When the code has been debugged, all the `Info` statements must be deleted, or commented, so that you don't get all that information when you are no longer debugging.
- OpenFOAM provides an alternative to removing all the `Info` statements, so that these `Info` statements can be activated again later.
- This brings us to the next level of debugging in OpenFOAM...

Debugging using OpenFOAM DebugSwitches

- In `$WM_PROJECT_DIR/etc/controlDict` (global `controlDict` dictionary), you can find a list of `DebugSwitches`:

```
DebugSwitches
{
    Analytical          0;
    APIdiffCoefFunc     0;
    Ar                  0;
    ...
}
```

- Each class thus has its own `DebugSwitch`.
- `DebugSwitches` set to zero will produce no debug information.
- Different levels of debugging can be chosen by setting a `DebugSwitch` to `1`, `2`, `3` ...
- It is recommended to make a copy of that file to a specific location and make any modifications there. This file will override the original file (searched paths in `foamEtcFile -list`):

```
mkdir -p $HOME/.OpenFOAM/2006
cp $WM_PROJECT_DIR/etc/controlDict $HOME/.OpenFOAM/2006
```

However, now there is a better way to do this...

Debugging using OpenFOAM DebugSwitches

A better way to activate DebugSwitches:

- List all flags, to find the ones used below:
  `icoFoam -help-full`
- List all registered switches:
  `icoFoam -listRegisteredSwitches`
  Have a look under `DebugSwitches`
- Activate a debug switch:

```
icoFoam -debug-switch <name=val>
                    Specify the value of a registered debug switch. Default is 1
                    if the value is omitted. (Can be used multiple times)
```

What is a DebugSwitch?

- Let's have a look at the `lduMatrix DebugSwitch`, which is set to `1` in `etc/controldict`.
- The `lduMatrix` class is implemented in
  `$FOAM_SRC/OpenFOAM/matrices/lduMatrix/lduMatrix`
- Looking inside `lduMatrix.C`, we see a line:
  ```
  defineTypeNameAndDebug(lduMatrix, 1);
  ```
  This line defines the `DebugSwitch` name `lduMatrix`, and sets its default value to `1`.
- In `$FOAM_SRC/OpenFOAM/matrices/lduMatrix/solvers/PCG/PCG.C`, you can find:

```
if (lduMatrix::debug >= 2)
    {
        Info<< "   Normalisation factor = " << normFactor << endl;
    }
```

- Boolean `debug` corresponds to the `lduMatrix DebugSwitch`, and it is `true` if the `DebugSwitch` is *greater than* `0`, and the above if-statement is done if it is *equal to or greater than* `2`.
- The default value, both in the class definition, and in the global `controlDict` is `1`. Try:
  `icoFoam -debug-switch lduMatrix=2 -case $FOAM_RUN/cavity`

The SolverPerformance DebugSwitch

- Setting the `SolverPerformance` DebugSwitch to `2` activates (in `$FOAM_SRC/OpenFOAM/matrices/LduMatrix/LduMatrix/SolverPerformance.C`):

```
if (debug >= 2)
{
    Info<< solverName_
        << ":  Iteration " << noIterations_
        << " residual = " << finalResidual_
        << endl;
}
```

Try it with the `cavity` case (keeping the `lduMatrix DebugSwitch`):

```
icoFoam -debug-switch lduMatrix=2 \
        -debug-switch SolverPerformance=2 \
        -case $FOAM_RUN/cavity
```

- If you have copied the global `controlDict`, remove your copy to go back to normal...

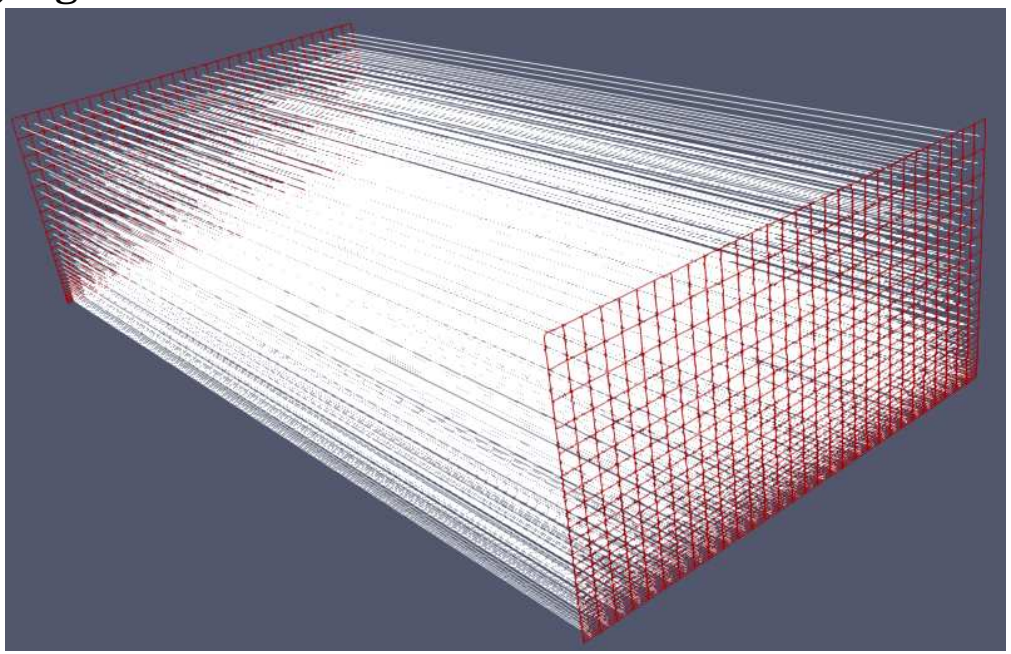
Verify cyclic coupling

You can verify that your cyclic coupling is set up correctly:

- Copy the `channel395` case and run `blockMesh`:
  ```
  cp  -r $FOAM_TUTORIALS/incompressible/pimpleFoam/LES/channel395 $FOAM_RUN
  cd $FOAM_RUN/channel395; blockMesh
  ```
- Activate the cyclic `DebugSwitch` and just run the start-up of the simulation:
  `pimpleFoam -debug-switch cyclic >& log` (and press `CTRL-c` after a few seconds)
  You get some metrics of the coupling in the log file.
- Read in and visualize the obj files in `paraview`, e.g.:

```
inout1_half0_faces.obj
inout1_half1_faces.obj
inout1_half0_to_inout1_half1.obj
```

(switch to 3D view)



Debugging using DebugSwitches

- In summary, the `DebugSwithes` only control different levels of `Info`-statements and in some cases activates/deactivates some additional output. No re-compilation is needed when switching the level, but if new `Info`-statements are included, re-compilation is needed. You can use this feature in your own development.
- This still offers no control of the running of the code while debugging...

Debugging using GDB

- Now it is time for some real debugging with the Gnu debugger (`www.gnu.org/software/gdb/`)
- GDB can be used for
  - Programs written in C, C++, Ada, Pascal etc
  - Running and stopping at specific positions in the code
  - Examining variables at run-time
  - Changing your program at run-time
- Bad news: The complete code needs to be re-compiled with a debug flag. This will produce ∼1Gb extra of OpenFOAM binaries.
- More bad news: The code will run much slower in debug mode, in particular when running under GDB. The reason is that nothing can be optimized, and that there is much more information to keep track of.
- Good news: You can compile only your development in debug-mode.
- Best news: GDB can help you implement a bug-free code, which can then be run fast in an optimized version (unless the compiler optimization introduces some bug).

Running (and compiling) OpenFOAM in Debug mode

- In `$WM_PROJECT_DIR/etc/bashrc` you find an environment variable `WM_COMPILE_OPTION` that can be set to `Debug`. That is what you need to do if you want to compile using the debug flag, or use the `Debug` version. Have a look at our `OFv2006Debug` alias, which help us set that environment variable:

```
alias OFv2006Debug=\
    '. $HOME/OpenFOAM/OpenFOAM-v2006/etc/bashrc WM_COMPILE_OPTION=Debug'
```

- In `foam-extend-4.1` you instead do `export WM_COMPILE_OPTION=Debug` before sourcing the OpenFOAM bashrc file.

- Make sure that you use the `Debug` mode by typing (in a new terminal):

```
OFv2006Debug
which icoFoam
```

  which should point at a path containing `linux64GccDPInt32Debug`.

Compiling an application (or library) in Debug mode

If you only want to debug your own development, you can compile only that in debug mode:

- Here use a terminal window with `OFv2006`
- Copy the `icoFoam` solver, re-name the executable name and location, and modify the first line in `Make/options` to:

```
EXE_INC = -O0 -fdefault-inline -ggdb3 -DFULLDEBUG \
```

  With this you can debug *only* this application.

- Now you know how to compile all, or parts of, OpenFOAM in `Debug` mode.
- Now it is time to learn the basics of GDB...
- Do this in `OFv2006Debug`

Investigate the laplacianFoam/flange TEqn.solve() function using GDB

You will do this yourself using another document...