Debugging OpenFOAM implementations

Debugging OpenFOAM implementations

(Acknowledgements to Dr. Fabian Peng-Kärrholm)

- 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)
  - Debugging using the Gnu debugger, GDB (`http://www.gnu.org/software/gdb/`) ...with `emacs` or `ddd`
- 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:

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

What is a DebugSwitch?

- Let's have a look at the `lduMatrix DebugSwitch`, which is set to `1`.
- The `lduMatrix` class is implemented in
  `$FOAM_SRC/OpenFOAM/matrices/lduMatrix/lduMatrix`
- Looking inside `lduMatrix.C`, we see a line:

```
defineTypeNameAndDebug(Foam::lduMatrix, 1);
```

  This line defines the `DebugSwitch` name `lduMatrix`, and sets its default value to `1`.
- In `$FOAM_SRC/OpenFOAM/matrices/lduMatrix/solvers/PBiCG/PBiCG.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 setting it to `2` in your own copy of the global `controlDict`, and run the `cavity` case.

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, yielding something like:

```
DILUPBiCG:  Iteration 0 residual = 0.0157314
DILUPBiCG:  Iteration 1 residual = 0.00199487
DILUPBiCG:  Iteration 2 residual = 0.000810123
DILUPBiCG:  Iteration 3 residual = 0.000115163
DILUPBiCG:  Iteration 4 residual = 3.877e-05
DILUPBiCG:  Iteration 5 residual = 9.59773e-06
DILUPBiCG:  Solving for Ux, Initial residual = 0.0157314, Final residual = 9.59773e-06, No Iterations 5
```

- Now, remove your copy of the global `controlDict` to go back to normal...
- In summary, the `DebugSwithes` only control different levels of `Info`-statements. 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 OpenFOAM implementations with 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: I have compiled it for you here at Chalmers.
- More 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 `OF23xDebug` alias, which help us set that environment variable:

```
alias OF23xDebug=\
    'export FOAM_INST_DIR=/chalmers/sw/unsup64/OpenFOAM;\
    . $FOAM_INST_DIR/OpenFOAM-2.3.x/etc/bashrcHani WM_COMPILE_OPTION=Debug'
```

- In `foam-extend-3.1` 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):
  `OF23xDebug`
  `which icoFoam`
  which should point at a path containing `linux64GccDPDebug`.

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 `OF23x`
- Copy the `icoFoam` solver, re-name the executable name and location, and modify the first line in `Make/options` to:

  ```
  EXE_INC = -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 `OF23xDebug`

Debugging the icoFoam solver

- Let's practice GDB on the `icoFoam` solver, using `OF23xDebug`
- The objective is to find out what a part of the code does.
- In `$FOAM_SOLVERS/incompressible/icoFoam/icoFoam.C` there is a line saying:
  `adjustPhi(phiHbyA, U, p);`
  What does this line do?
- Make sure that you have an `icoFoam/cavity` case, that you have run `blockMesh` on it, and you are inside that case directory while running GDB.
- Start `icoFoam` under GDB in the `cavity` case directory:
  `gdb icoFoam`
- Set a break point at the line in `icoFoam.C` where `adjustPhi(phiHbyA, U, p);` is used:
  `b icoFoam.C:80`
- Start the execution by typing:
  `run` (this will take some time...)
- The exectution will stop at the breakpoint saying something like:

```
Breakpoint 1, main (argc=1448, argv=0x3d1) at icoFoam.C:80
80              adjustPhi(phiHbyA, U, p);
```

Line-by-line checking of adjustPhi

- There are two ways of stepping in the file (**don't do these now!**):
  `n` (next) will step to the next line in the current file, but will **not** go into functions and included files.
  `s` (step) will go to the next line, also in functions and included files.
  Both can be followed by a number specifying how many times to repeat the command.
  If you want to repeat a previous command, just type enter.
- Step to the next line by typing '`s`' (allowing GDB to go inside the `adjustPhi` function).
  We are now at line `41` in `cfdTools/general/adjustPhi/adjustPhi.C`
- Type '`where`' to see which file you are debugging, and which file called it. This is the 'stack'.
- Type '`list cfdTools/general/adjustPhi/adjustPhi.C:30,50`' to see source lines `30-50` of `adjustPhi.C`. Line `41` is the first line of the `adjustPhi` function.
- Since you are at that location, you can also simply type:
  '`l 30,50`'
- Type '`l 1`' to list the first 10 lines of the file
- Press enter (or '`l`') to march through the file, 10 lines at a time.

Line-by-line checking of adjustPhi

- Type '`n 2`' to avoid going into the evaluation of the boolean, instead we will see that we are at line `47` in `adjustPhi.C`, so the if-statement is evaluated.
- Type '`p massIn`' and then '`p fixedMassOut`'. Note that line `47` has not yet been evaluated, so `fixedMassOut` can have any value!! (BUG IN GDB 7.x))
- Type '`s 5`' to step 5 lines, allowing to enter functions.
- Again, type '`where`' to see the stack.
- Type '`f 3`', where `3` is the *frame* number.
- Again, type '`where`' to see the stack (same as above, since you are still at the same location but looking at another frame)
- Type '`l`' to see 10 lines around the one corresponding to frame 3.
- Type '`up` or '`down` to go up and down the stack
- Type '`f`' to show which frame you are looking at
- Type '`c`' to continue execution until it gets back to the breakpoint or terminates normally
- Type '`run`' to restart the degugging from the beginning if needed. Type '`quit`' to quit.
- Stepping and listing the code will show every single step that will be taken. You will understand that `adjustPhi` ensures global continuity.

Learn more on GDB

- See `http://www.gnu.org/software/gdb`
- There are some interfaces to GDB:
  - See `http://www.gnu.org/software/gdb/links/`
  - `ddd`
  - `emacs` ... next slide
- Macros for GDB/OpenFOAM: `http://openfoamwiki.net/index.php/Contrib_gdbOF`
- `eclipse` is another alternative (An Integrated Development Environment)
  See: `http://openfoamwiki.net/index.php/HowTo_Use_OpenFOAM_with_Eclipse`
- `Qt` is yet another alternative
  See: `http://openfoamwiki.net/index.php/HowTo_Use_OpenFOAM_with_QtCreator`

GDB in emacs

- In the cavity case, run `emacs`
- Click `Tools/Debugger (GDB)`
- Type `icoFoam` (appears at the bottom of emacs, as part of a GDB command)
- Type `b icoFoam.C:80`, as before.
- Click on `GO`
  The execution stops at line 80 in icoFoam.C, and emacs shows the output of icoFoam in the lower window, and the surrounding code in the upper window.
- Mark `phiHbyA` in the upper window and click `Gud/Watch Expression`, and you should get a window (Speedbar) showing which class it belongs to and the possibility to examine it more.
- Click `Next Line` until you are at the line with `if (nonOrth == nNonOrthCorr)`. Add `nonOrth` and `nNonOrthCorr` to `Gud/Watch Expression`, and you see that they are the same, i.e. the if-statement is true (check by clicking on `Next Line` again!). Let's examine that line closer by clicking `Step Line`. Emacs will ask if you want to follow a symbolic link - just type `yes`. We now see that we are in `fvMatrix.C`.

GDB in emacs

- Will we enter the if-statement? Mark in the lower window:
  `!psi_.mesh().fluxRequired(psi_.name())`
  and click on `Gud/Watch Expression`
  This returns `false`, so we will not enter (check with `Next Line`). Note that you can change from `false` to `true` in the Speedbar.
- Click `Next Line` until you are at the line with `forAll`, then click on `Step Line` and type `yes` on the question on the symbolic link.
- The `size()` function returns the size of member data `ptrs_` in the templated `PtrList` class. Click `Step Line` and type `yes`.
- The `PtrList` seems to be pointing at the `List` class to evaluate the `size()` function. Click on `Step Line` twice and we are back to the `PtrList`. Click on `Step Line` again, and we are in a `forAll` loop in the `fvMatrix.C` file.
- Click on GO (or `Gud/Continue`) to continue to the next breakpoint or to terminate normally
- Let's not dig any further, so click on `Finish Function` to get back to `icoFoam.C`, and we can continue the execution on the highest level.
- Exit `emacs`.

Floating point exception (or segmentation fault)

- Getting the error 'Floating point exception' (or 'segmentation fault') is a nightmare, since you get no information what went wrong, or where it happened.
- We will now experience a 'Floating point exception' and use `gdb` in `emacs` to find out where it happens.
- Do the following:

```
run
rm -r pitzDaily
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily .
cd pitzDaily
blockMesh
sed -i s/0.375/0/g 0/k
```

- Run `simpleFoam`, and get a 'Floating point exception'.
- Run `simpleFoam` through `gdb` in `emacs` and again get the 'Floating point exception'.
- It stops in `scalarField.C`

Floating point exception (or segmentation fault)

- March up and down in the file stack, by clicking on `Up Stack` and `Down Stack`, accepting to follow symbolic links, and realize that there is a division by `k_` in `tmp<fvScalarMatrix> epsEqn` in file `kEpsilon.C`
- If you go to the top of the file stack, you see that `kEpsilon.C` was called using `turbulence->correct();`, as we already know.
- Now we know that there is a problem with the division by k, and we realize that we initialized k to zero.

Debugging a simple C++ code

In `nonFailSafeCode.C`:

```
#include <iostream>
using namespace std;

int main()
{
const int constantInt=5;
int zeroInt;
cout << "Please type the integer number zero!" << endl;
cin >> zeroInt;
cout << "constantInt = " << constantInt << endl;
cout << "zeroInt = " << zeroInt << endl;
cout << "constantInt/zeroInt = " << constantInt/zeroInt << endl;
return(0);
}
```

Compile and debug with:

```
g++ -g nonFailSafeCode.C -o nonFailSafeCode
```