ATHAM-Fluidity: How to Install and Run

Contents

1 Downloading ATHAM-Fluidity

The ATHAM-Fluidity repository is found on GitHub at `https://github.com/FluidityProject/ATHAM-Fluidity`. To download the repository onto a Linux machine (that has `git` installed), open up a terminal window and navigate to the directory under which you wish to store your local copy of ATHAM-Fluidity. Then type the following command (after the command prompt '`$`'):

```
$ git clone https://github.com/FluidityProject/ATHAM-Fluidity.git
```

2 Dependencies and environment variables

ATHAM-Fluidity requires a certain number of dependencies (external packages and libraries) to be installed before it can be compiled. An essential step before compiling ATHAM-Fluidity is therefore to make sure that all these dependencies are present and can be reached by the code. There are three possible ways to achieve this, depending on the system you are using:

- **Install the `fluidity-dev` package**. This installs all the dependencies automatically, using default versions and install locations. This procedure is described in Section 2.1

- **Self-installation**. Each dependency can be installed separately if desired (e.g. if the user wants to install non-default versions of the dependencies or choose different install locations). This procedure is described in Section 2.2
- **Loading modules**. This procedure can be followed if you are running ATHAM-Fluidity on a cluster that uses environment modules, and the dependencies have already been centrally built by a system administrator. Section 2.3 describes how to load the relevant modules on the University of Cambridge's Darwin HPC.

2.1 Installing the `fluidity-dev` package

With the Ubuntu operating system, the `fluidity-dev` package can be installed from the terminal window by issuing the following three commands (assuming that the user has administrative '`sudo`' privileges):

```
$ sudo apt-add-repository -y ppa:fluidity-core/ppa
$ sudo apt-get update
$ sudo apt-get install fluidity-dev
```

It is also necessary to issue the following command before the configure stage:

```
$ export petsc_dir=/usr/lib/petscdir/3.6.3
```

At the time of writing this document, the `fluidity-dev` package installs PETSc version 3.6.3. It is possible that this has changed since then, in which case the '3.6.3' in the above command should be replaced with the newer version number, which can be found by issuing the command '`$ ls /usr/lib/petscdir`'. It is recommended that you also add the above export line (minus the command prompt) to your `.bashrc` file, located in your home directory, so that these environment variables are remembered for every login session.

Note that Fluidity (and thus ATHAM-Fluidity) is only supported for use with the GNU Compiler Collection (GCC) and OpenMPI executables. It is possible that a different compiler collection (e.g. Intel, Portland Group, etc.) is set up as the default on your system. Use of the supported executables can be ensured by setting the following environment variables:

```
$ export CC=gcc
$ export CXX=g++
$ export CPP=cpp
$ export FC=gfortran
$ export F90=gfortran
$ export F77=gfortran
$ export PATH=/usr/bin:$PATH
```

Again, it is recommended that you also add the above lines to your `.bashrc` if you want the same behaviour for every login session.

2.2 Self-installation

The following section describes the installation of the ATHAM-Fluidity dependencies on a Linux machine that was running Ubuntu 16 LTS. It is assumed that the user has administrative '`sudo`' privileges.

When newer versions of the dependencies are released, it takes time for their compatibility to be fully tested by the model developers, and so version support often lags behind the most recent versions of these dependencies. For this reason, it is recommended to install the dependencies from source code, with versions that are known to be stable, rather than using a package management tool (e.g. '`apt-get install`' on Ubuntu) which will install the most recent version of the dependency by default. It is recommended to install each dependency in a separate

sub-directory, under a location that can be reached by all users. Here, we will install each dependency in a sub-directory under `/usr/local/fluidity/`. The first step is therefore to create this directory:

```
$ sudo mkdir /usr/local/fluidity
```

2.2.1 GNU compilers

The GNU Compiler Collection (GCC) should come pre-installed with Ubuntu. You can check the version (and existence) of GCC installed on your system by typing:

```
$ gcc --version
```

Here, we are using gcc 5.4.0. It is possible that another compiler collection (e.g. Intel, Portland Group, etc.) is set up as the default on your system; see Section 2.1 for instructions on how to ensure that the GCC compilers are always used.

2.2.2 Python

Python should also come pre-installed with Ubuntu. Again, you can check the (default) version on your system by typing '`python --version`' at the command prompt. Here, we are using Python 2.7.12. Note that Python 3 is not yet supported - you may therefore have to change the symbolic link `/usr/bin/python` to point to an earlier version of the python executable, e.g. python2.7. There are also a number of Python packages that are required for ATHAM-Fluidity to build. Again, these packages are typically installed by default. One important package is NumPy. It is possible to check whether NumPy is installed by typing the following at the command line:

```
$ python -c "import numpy; print numpy.__version__"
```

Here, we are using NumPy 1.11.0. If you see an import error, you should install the NumPy package manually.

2.2.3 OpenMPI

ATHAM-Fluidity must be built with MPI support, even if the model will only be run on a single processor rather than in parallel. It is recommended to install OpenMPI. Here, we install OpenMPI 1.6.5. The source code can be download from `https://www.open-mpi.org/software/ompi/v1.6/` (choose the gzipped tarfile, i.e. openmpi-1.6.5.tar.gz). Untar the file somewhere locally and move into the untarred directory. Also, create a sub-directory under `/usr/local/fluidity` ready for installation:

```
$ tar -xzvf openmpi-1.6.5.tar.gz
$ cd openmpi-1.6.5
$ sudo mkdir /usr/local/fluidity/openmpi-1.6.5
```

OpenMPI is then built by issuing the following three commands:

```
$ ./configure --prefix=/usr/local/fluidity/openmpi-1.6.5
$ make
$ sudo make install
```

Once installation is complete, you may (if you wish) delete the local openmpi-1.6.5 directory and tarfile. In order that the OpenMPI executables can be found when building ATHAM-Fluidity (as well as some of the dependencies below), you will need to issue the following command (and add it to your `.bashrc` file if you want the same behaviour for every login session):

```
$ export PATH=/usr/local/fluidity/openmpi-1.6.5/bin:$PATH
```

2.2.4 BLAS and LAPACK

The linear algebra solvers BLAS and LAPACK are both required by the model. Netlib provide an implementation of these packages. The source codes can be downloaded from `http://www.netlib.org/blas/` and `http://www.netlib.org/lapack/`, respectively. Here, we download and install BLAS 3.6.0 (blas-3.6.0.tgz) and LAPACK 3.4.2 (lapack-3.4.2.tgz). First, untar BLAS locally, move into the untarred directory, and prepare an install directory:

```
$ tar -xzvf blas-3.6.0.tgz
$ cd BLAS-3.6.0
$ sudo mkdir /usr/local/fluidity/BLAS-3.6.0
```

Next, open the file `make.inc` using your preferred text editor (e.g. `gedit`) and modify the variables `OPTS` and `BLASLIB` so that they read:

```
OPTS = -O3 -fPIC
BLASLIB = /usr/local/fluidity/BLAS-3.6.0/libblas.a
```

The BLAS library is then installed by issuing the following command:

```
$ sudo make
```

Now untar LAPACK locally, move into the untarred directory, and prepare an install directory:

```
$ tar -xzvf lapack-3.4.2.tgz
$ cd lapack-3.4.2
$ sudo mkdir /usr/local/fluidity/lapack-3.4.2
```

Next, make a copy of the file `make.inc.example`, calling it `make.inc`, i.e. type:

```
$ cp make.inc.example make.inc
```

Open `make.inc` using your preferred text editor and modify the following lines so that they read:

```
OPTS = -O2 -fPIC
LOADOPTS = -fPIC
CFLAGS = -O3 -fPIC
BLASLIB = /usr/local/fluidity/BLAS-3.6.0/libblas.a
```

The LAPACK library file is then built by issuing the '`make`' command, after which the file must be manually copied into the install directory:

```
$ make
$ sudo cp liblapack.a /usr/local/fluidity/lapack-3.4.2/.
```

Note that no environment variables need to be set for BLAS and LAPACK - you will tell ATHAM-Fluidity where to find these libraries directly in the configure command.

2.2.5 PETSc

PETSc is used to provide matrix solvers in ATHAM-Fluidity. Here, we install PETSc 3.5.4. The source code can be downloaded from `http://www.mcs.anl.gov/petsc/download/` (petsc-3.5.4.tar.gz). Untar this file locally, move into the untarred directory, and prepare an install directory:

```
$ tar -xzvf petsc-3.5.4.tar.gz
$ cd petsc-3.5.4
$ sudo mkdir /usr/local/fluidity/petsc-3.5.4
```

PETSc has some dependencies of its own that need to be installed before configuring and installing PETSc itself. Of these dependencies, those that are most likely not on your system by default are `cmake`, `bison` and `flex`. It should be safe to install the latest releases of these dependencies, and so the command line package management tool can be used. On Ubuntu, you would issue the following commands:

```
$ sudo apt-get update
$ sudo apt-get install cmake bison flex
```

Next, issue the following command to configure PETSc:

```
$ ./configure --prefix=/usr/local/fluidity/petsc-3.5.4 --download-fblaslapac
k=1 --download-blacs=1 --download-scalapack=1 --download-ptscotch=1 --download-
mumps=1 --download-hypre=1 --download-suitesparse=1 --download-ml=1 --with-fort
ran-interfaces=1
```

After a successful configuration, you should see something similar to the following output in the terminal window:

```
Configure stage complete.  Now build PETSc libraries with (gnumake build):
   make PETSC_DIR=/home/jjo31/tarfiles/petsc-3.5.4 PETSC_ARCH=arch-linux2-c-d
ebug all
```

Copy and paste the entire line starting with '`make`' from your terminal into the command prompt and press Enter. After this stage has completed successfully, you should see something similar to the following output in the terminal window:

```
Now to install the libraries do:
make PETSC_DIR=/home/jjo31/tarfiles/petsc-3.5.4 PETSC_ARCH=arch-linux2-c-debu
g install
```

Again, copy and paste the entire line starting with '`make`' from your terminal into the command prompt, but this time add '`sudo`' to the beginning of the line before pressing Enter. You should then see something similar to the following output in the terminal window:

```
Install complete.
Now to check if the libraries are working do (in current directory):
make PETSC_DIR=/usr/local/fluidity/petsc-3.5.4 PETSC_ARCH="" test
```

You can copy-paste the final line from your terminal into the command prompt to make sure that PETSc has been installed successfully. Finally, it is necessary to set the following environment variable (and add it to your `.bashrc` file for future login sessions) so that ATHAM-Fluidity knows where to find the PETSc files during compilation:

```
$ export PETSC_DIR=/usr/local/fluidity/petsc-3.5.4
```

2.2.6 ParMETIS and Zoltan

Domain decomposition for parallel runs with ATHAM-Fluidity is performed through the partitioning and load balancing software Zoltan. Zoltan must be configured with the mesh-partitioning library ParMETIS, which should therefore be installed first. Here, we install ParMETIS 3.1.1. The source code can be downloaded from `http://glaros.dtc.umn.edu/gkhome/fsroot/sw/parmetis/OLD` (ParMetis-3.1.1.tar.gz). Untar this file locally, move into the untarred directory, and prepare an install directory:

```
$ tar -xzvf ParMetis-3.1.1.tar.gz
$ cd ParMetis-3.1.1
```

```
$ sudo mkdir /usr/local/fluidity/ParMetis-3.1.1
```

ParMETIS is then built by issuing the 'make' command, after which the library and header files must be manually copied into the install directory:

```
$ make
$ sudo cp lib*.a parmetis.h /usr/local/fluidity/ParMetis-3.1.1/.
```

Next, download the Zoltan source code from http://www.cs.sandia.gov/Zoltan/Zoltan_download.html. Here, we install Zoltan 3.83. Untar the downloaded tarfile locally, create a directory called Zoltan-build (Zoltan should be built in a separate directory to the source directory), move into this directory, and prepare an install directory:

```
$ tar -xzvf zoltan_distrib_v3.83.tar.gz
$ mkdir Zoltan-build
$ cd Zoltan-build
$ sudo mkdir /usr/local/fluidity/Zoltan_v3.83
```

It is also necessary to specify whether you are installing Zoltan on a 32- or 64-bit system. This can be determined by issuing the following command:

```
$ uname -i
```

For a 64-bit system this will return something like 'x86_64', whereas for a 32-bit system it will return something like 'i386'. To configure Zoltan, issue the following command (replacing the 'x86_64' to be consistent with your system, if necessary):

```
env CC='' CXX='' CPP='' FC='' F90='' F77='' ../Zoltan_v3.83/configure x86_64-linux-gnu --prefix=/usr/local/fluidity/Zoltan_v3.83 --enable-mpi --enable-f90interface --disable-examples --with-parmetis --with-parmetis-libdir=/usr/local/fluidity/ParMetis-3.1.1 --with-parmetis-incdir=/usr/local/fluidity/ParMetis-3.1.1
```

Note that the text before the configure command temporarily 'unsets' the compiler environment variables that we set previously - this is required because the configure script would otherwise think that these were the names of the MPI compilers, which is not the case. Next, issue the following two commands to compile and install Zoltan on your system:

```
$ make
$ sudo make install
```

Finally, it is necessary to set the following environment variables (and add them to your .bashrc file for future login sessions) so that ATHAM-Fluidity knows where to find the ParMETIS and Zoltan files during compilation:

```
$ export LDFLAGS='-L/usr/local/fluidity/ParMetis-3.1.1 -L/usr/local/fluidity/Zoltan_v3.83/lib'
$ export CPPFLAGS='-I/usr/local/fluidity/ParMetis-3.1.1 -I/usr/local/fluidity/Zoltan_v3.83/include'
```

2.2.7 VTK

VTK is required by ATHAM-Fluidity for its data output methods. The VTK install described below requires the following packages to be present on your system, which can be installed using 'apt-get' with Ubuntu:

```
$ sudo apt-get update
$ sudo apt-get install cmake-curses-gui tcl-dev tk-dev
```

Next, download the VTK source code from `http://www.vtk.org/download/`. Here we install VTK 5.10.1 (vtk-5.10.1.tar.gz). Untar the downloaded tarfile locally, create a directory called `VTK-build` (VTK should be built in a separate directory to the source directory), move into this directory, and prepare an install directory:

```
$ tar -xzvf vtk-5.10.1.tar.gz
$ mkdir VTK-build
$ cd VTK-build
$ sudo mkdir /usr/local/fluidity/VTK5.10.1
```

It is also necessary to set the following environment variable (and add it to your `.bashrc` file for future login sessions):

```
$ export PYTHONPATH=/usr/local/fluidity/VTK5.10.1/lib/python2.7/site-packag
es:$PYTHONPATH
```

To start the VTK configuration process, issue the following command:

```
$ ccmake ../VTK5.10.1
```

This opens up an interactive configure window inside the terminal. Press `c` to start the initial configure. Once this is complete, press `e` to be given a list of additional configuration options. Using the arrow keys, Enter key, and keyboard, modify the following options (the rest can be left as they are):

```
BUILD_SHARED_LIBS = ON
CMAKE_INSTALL_PREFIX = /usr/local/fluidity/VTK5.10.1
VTK_USE_CHARTS = OFF
VTK_USE_GEOVIS = OFF
VTK_USE_INFOVIS = OFF
VTK_USE_N_WAY_ARRAYS = OFF
VTK_USE_VIEWS = OFF
VTK_WRAP_PYTHON = ON
```

Press `c` to reconfigure. Once complete, press `e`, followed by `c` and `e` again. You should now have the option to 'generate' by pressing `g`. This will take you back to the standard command line, where you can then build and install VTK by issuing the following commands:

```
$ make
$ sudo PYTHONPATH=/usr/local/fluidity/VTK5.10.1/lib/python2.7/site-packages make
install
```

Note that the '`sudo`' command does not preserve user environment variables, which is why PYTHONPATH has to be passed in to '`make install`' despite having been added to the user environment previously. Finally, it is necessary to set the following environment variables (and add them to your `.bashrc` file for future login sessions) so that ATHAM-Fluidity knows where to find the VTK files during compilation:

```
$ export VTK_INCLUDE=/usr/local/fluidity/VTK5.10.1/include/vtk-5.10
$ export LDFLAGS=$LDFLAGS‘ -L/usr/local/fluidity/VTK5.10.1/lib/vtk-5.10’
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/fluidity/VTK5.10.1/lib/v
tk-5.10
```

2.3 Loading modules on Darwin

The dependencies required by ATHAM-Fluidity have already been installed centrally on Darwin. All that is required to prepare the model is therefore to load the relevant modules and set some environment variables. Note that the Intel compiler collection (not GCC) was used

to install these dependencies. Specifically, the 2014 version-set was used. As this is not the version-set loaded by default at login, the first step is to unload these default modules and load the appropriate ones. This can be done by issuing the following commands (and adding them to your `.bash_profile` file in your home directory so that this is done for every login session):

```
$ module unload intel/impi/4.1.3.045
$ module unload intel/fce/12.1.10.319
$ module unload intel/cce/12.1.10.319
$ module unload intel/mkl/10.3.10.319
$ module load intel/impi/5.0.2.044
$ module load intel/fce/15.0.1.133
$ module load intel/cce/15.0.1.133
$ module load intel/mkl/11.2.1.133
```

Note that the first four lines correspond to the Intel modules that are loaded by default on Darwin at the time of writing this document. It is possible that these modules have changed since then, in which case you should modify these lines accordingly. It is possible to see the list of modules loaded by default by typing '`module list`' into the command line after login. The rest of the dependencies are then loaded by issuing the following commands (add them to your `.bash_profile` too):

```
$ module load petsc/intel15/3.5.3
$ module load zoltan/3.83
$ module load vtk/5.6.1
```

Finally, although some of the required environment variables are automatically set when the modules are loaded, there are still a few variables that need to be set manually (add them to your `.bash_profile` too):

```
$ export CPPFLAGS='-I/usr/local/Cluster-Apps/zoltan/3.83/include'
$ export VTK_INCLUDE=/usr/local/Cluster-Apps/vtk/5.6.1/include/vtk-5.6
$ export LDFLAGS='-L/usr/local/Cluster-Apps/vtk/5.6.1/lib/vtk-5.6'
```

3 Installing ATHAM-Fluidity

ATHAM-Fluidity must first be properly configured so that it can be linked to the relevant libraries outlined in Section 2. A minimum configuration is achieved by issuing the following command from inside the main directory of your ATHAM-Fluidity repository:

```
$ ./configure --enable-2d-adaptivity
```

For users that have installed the BLAS and LAPACK libraries manually, their location should be passed to the configure command as follows:

```
$ ./configure --enable-2d-adaptivity --with-blas=/usr/local/fluidity/BLAS-3.
6.0/libblas.a --with-lapack=/usr/local/fluidity/lapack-3.4.2/liblapack.a
```

This assumes that the BLAS and LAPACK libararies were installed in the locations as speficied in Section 2.2; if they were installed elsewhere, modify the paths accordingly. A debug build of ATHAM-Fluidity is obtained by adding the flag `--enable-debugging` to the configure command. Information about further configure flags can be obtained using the command:

```
$ ./configure --help
```

To compile ATHAM-Fluidity after a successful configure, simply type:

```
$ make
```

In addition to the standard '`make`' above, a set of useful tools (including the '`flredecomp`' executable, required for domain decomposition for parallel runs) should also be compiled by typing:

```
$ make fltools
```

If not otherwise specified, the model and tool executables will be placed in the following directory: `<A-F_install_path>/bin`, where `<A-F_install_path>` is the path to your local ATHAM-Fluidity repository. It can be useful to add this directory to your `$PATH` environment variable so that the executables can be launched simply by typing their name rather than their entire path. This is done by issuing the following command (and adding it to your `.bashrc` file for future login sessions):

```
$ export PATH=<A-F_install_path>/bin:$PATH
```

Diamond, the GUI used to generate ATHAM-Fluidity input parameter (`.flml`) files, is built and then linked to the appropriate options-tree ('schema') file by typing the following two commands:

```
$ make install-diamond
$ make install-user-schemata
```

If not specified otherwise, the '`diamond`' executable will be placed in the following directory: `<A-F_install_path>/libspud/diamond/bin`. Again, to be able to launch the executable simply by typing '`diamond`', you can modify your `$PATH` environment variable by typing (and adding to your `.bashrc` file for future login sessions) :

```
$ export PATH=<A-F_install_path>/libspud/diamond/bin:$PATH
```

Note that the location of the default `.flml` can be found in the following directory: `$HOME/.diamond/schemata/flml`.

Finally, local Python scripts should also be made accessible by setting the following environment variable (and adding to your `.bashrc` file for future login sessions):

```
$ export PYTHONPATH=$PYTHONPATH:<A-F_install_path>/python
```

4 Running ATHAM-Fluidity

A typical workflow when setting up and running an ATHAM-Fluidity simulation consists of the following four steps:

1. Creating an ATHAM-Fluidity model options (`.flml`) file. This procedure is described in Section 5.
2. Defining the geometry of the numerical modelling domain and creating the (initial/static) mesh. This procedure is described in Section 6.
3. Decomposing the numerical domain into a number of smaller sub-domains in preparation for a parallel run on multiple processes. This procedure is described below.
4. Invoking the main model executable on the appropriate number of processes. This procedure is described below.

After the first two steps have been completed, for a parallel run, the numerical domain must first be decomposed using the `flredecomp` tool. This is done by issuing the following command from within the main simulation directory (where the model options (`.flml`) file is located):

```
$ mpiexec -n Y flredecomp -i X -o Y <original_input_file> <parallel_input_file>
```

The above command assumes that `<A-F_install_path>/bin` has been added to your `$PATH`

environment variable. The argument `-i` is used to specify the initial number of processes (sub-domains) `X` (typically `X`=1), and the arguments `-n` and `-o` are used to specify the output number of processes (sub-domains) `Y` on which the model executable will be run. `<original_input_file>` is the name of the original model options (`.flml`) file *without the extension*, while `<parallel_input_file>` is the name of the `.flml` file (again without the extension) that will be generated by `flredecomp` and used as the input file for the parallel run.

After the domain decomposition is complete, the main ATHAM-Fluidity executable is run by issuing the following command from within the same directory:

```
$ mpiexec -n Y fluidity <parallel_input_file>.flml
```

Note that the extension `.flml` *is* included at the end of the parallel model options file in this case. Further option flags can also be added to the above command (after the `fluidity` keyword). For example, `-v?` (where `?` is either `1`, `2` or `3`) is a verbose option with 3 different levels (from the least to the most verbose), and `-l` generates log and error files (one per processor).

If performing a serial run, the `flredecomp` stage is not required and the main model executable is simply run using:

```
$ fluidity <original_input_file>.flml
```

Finally, it is noted that for parallel runs, there is a second workflow option available that uses the depreciated `fldecomp` rather than the `flredecomp` tool:

```
$ fldecomp -n Y <mesh_file>
$ mpiexec -n Y fluidity <original_input_file>.flml
```

where `<mesh_file>` is the name of the numerical mesh file (see Section 6) without the extension, including its relative path if the file is not in the same directory as the `.flml` file. This is only mentioned here because this second workflow has been found to overcome an as-yet unresolved memory issue with `flredecomp` when running on a large number (hundreds) of processes.

5 ATHAM-Fluidity model options

******Note that ATHAM-Fluidity is still under development, and so many of the options in this section are likely to change or be added to before the model's general release******

In this section, you will find a description of all the useful ATHAM-Fluidity model options accessible through the diamond GUI. The options listed below correspond to a generic $\mathbb{P}_{1DG} - \mathbb{P}_2$ atmospheric simulation (the recommended finite element discretization method, in which pressure and density are solved on a 2^{nd}-order continuous grid while momentum and all other scalars are solved on a 1^{st}-order discontinuous grid) for compressible flow, including turbulence and cloud microphysics parameterizations. Refer to the main Fluidity manual for a more detailed description of all the Fluidity options and capabilities. For any option that does not appear below, it can typically be left turned off, or the default option used.

To launch the diamond GUI type:

```
$ diamond <filename>.flml
```

The above command assumes that you have added `<A-F_install_path>/libspud/diamond/bin` to your `$PATH` environment variable. If `<filename>.flml` is omitted, diamond will start with a blank `.flml` file. Note that when saving the file for the first time, the file extension `.flml` must be explicitly typed, otherwise the default extension `.xml` will be added and diamond will complain the next time the file is opened.

General:

- `simulation_name` will be used to prefix model output files
- `problem_type` must be set to '`fluids`'
- `geometry/dimension` should be set to 2 or 3 depending on whether the test case is a two- or three-dimensional problem. Note that once the `.flml` file has been saved, this number may not be modified later.

Meshes:

- `geometry/mesh (CoordinateMesh)` is the reference mesh, read from the mesh file(s) (thus select `from_file`). In the `Value` field of the `Attributes` section, type the (relative) path to the mesh file(s), e.g. `src/MyMesh` if the mesh file(s) are called `MyMesh` and are located within the subdirectory `src`. It is recommended to use the software package `gmsh` to generate the initial model mesh, in which case `format (gmsh)` should be selected.
- `geometry/mesh (VelocityMesh)` is the default mesh for the velocity (and non-pressure /density scalar) fields, which should be defined from the coordinate mesh. Thus, select `from_mesh/mesh (CoordinateMesh)`. `mesh_shape/polynomial_degree` should be set to 1 and `mesh_continuity` set to `discontinuous` (i.e. a $\mathbb{P}_{1DG}$ configuration).
- `geometry/mesh (PressureMesh)` is the pressure and density default mesh which again should be defined from the coordinate mesh. Thus, select `from_mesh/mesh(Coordinate Mesh)`. `mesh_shape/polynomial_degree` should be set to 2 and `mesh_continuity` set to `continuous` (i.e. a $\mathbb{P}_2$ configuration).
- Additional derived meshes can be defined following the above procedure. For example, if temperature is to be solved prognostically, a new mesh called, e.g., `TemperatureMesh` should be added, with `from_mesh/mesh (CoordinateMesh)` selected, `mesh_shape/polyno mial_degree` set to 1 and `mesh_continuity` set to `disontinuous` (i.e. a $\mathbb{P}_{1DG}$ configuration).
- `quadrature/degree` should be set to at least 2N, where N is the maximum polynomial degree. With the $\mathbb{P}_{1DG} - \mathbb{P}_2$ method, N=2 and so `quadrature/degree` should be set to 4 (or greater).

Input/Output:

- `io/dump_period/constant` specifies the time period (in seconds) between each set of output files generated by the model.
- `io/output_mesh (PressureMesh)` should be selected in order to generate outputs on the 2^{nd}-order pressure/density mesh for better quality.
- `io/checkpointing` can be used to generate files for a later restart of the model from a given simulation time (these files consist of gridded data files as well as an updated `.flml` file). Typically a value would be selected for `checkpoint_period_in_dumps`. For example, if this is set to 4 then restart files will be generated every 4 dump periods.

Time:

- `timestepping/current_time` specifies the time at the start of the simulation. This is typically set to 0.
- `timestepping/timestep` specifies the model time-step (in seconds). This value will depend on the modelled flow-field characteristics and the minimum grid mesh size. If adaptive time stepping is used (often employed with adaptive grid meshing) then this is the initial time-step.
- `timestepping/finish_time` specifies the simulation end-time (in seconds).

Physics:

- `physical_parameters/gravity/magnitude` must be defined, e.g. 9.81 for acceleration due to Earth's gravity.
- `physical_parameters/gravity/vector_field (GraviryDirection)` is used to prescribe the direction in which gravity acts. For a 2D case, the constants 0 and -1 would be input if the 2nd dimension pointed vertically upwards; for a 3D case, the constants 0, 0 and -1 would be input if the 3rd dimension pointed vertically upwards.
- `material_phase [name]/equation_of_state/compressible` must be selected.
- `equation_of_state/compressible/giraldo` is the default model for ideal gas and should therefore be selected. (ATHAM will eventually be the default option, but it is currently not operational.)
- `equation_of_state/compressible/giraldo/reference_pressure` is the pressure used in the definition of the Exner function (100000 Pa).
- `equation_of_state/compressible/giraldo/C_P` and `C_V` are the default values for the heat capacities of dry air (constants).
- `equation_of_state/compressible/giraldo/buoyancy_from_pt` allows the user to define buoyancy based on density potential temperature instead of density.
- `equation_of_state/compressible/giraldo/constant_cp_cv`: If turned on, the heat capacities for the moist atmosphere are held constant and equal to their default dry atmosphere values.
- `equation_of_state/compressible/giraldo/subtract_out_reference_profile`: If turned on, the buoyancy, pressure gradient and scalar diffusion terms are computed by subtracting out the hydrostatic reference state. These reference profiles MUST be defined beforehand, as described in the following.

`material_phase [name]` **prognostic fields:**

- `scalar_field (Pressure)`:
  - `scalar_field (Pressure)/prognostic/mesh (PressureMesh)` should be selected.
  - `spatial_discretisation/continuous_galerkin/remove_stabilisation_term` should be selected.
  - `scheme/poisson_pressure_solution` should be set to `never`.
  - `solver`: Your preferred solver. `Relative_error` and `max_iterations` are required for the chosen solver (suggested values are 1.0e-7 and 1000).
  - `initial_condition`: Specified either via a python script (idealized cases) or read `from_file` (non-idealised cases).
  - `boundary_conditions (name)`: Setting Dirichlet boundary conditions on all sides and top (chosen via `select_ids`) was found to work correctly. The method (`python` or `from_file`) must again be selected.
  - `boundary_conditions (name)/from_file/time_dependent` allows the user to prescribe time-varying boundary conditions. The number of input files must then be prescribed (see Section 8 for the required file name formats).
- `scalar_field (Density)`
  - `scalar_field (Density)/prognostic/mesh (PressureMesh)` should be selected, i.e. the density must be defined on the same continuous mesh as pressure.
  - `scalar_field(Density)/prognostic/equation` is not required. The density is diagnosed.
  - `spatial_discretisation/continuous_galerkin/stabilisation/no_stabilisation` should be selected.

- `spatial_discretisation/continuous_galerkin/conservative_advection` is not used as the density is diagnosed.
- `temporal_discretisation/theta`: The value used for time advancing with theta-scheme.
- `solver` is necessary if the initial condition is determined from the equation of state. It is recommend to choose the same as the pressure solver. `relative_error` and `max_iterations` are required for the chosen solver (suggested values are 1.0e-7 and 1000). ***We should change the code so that if not selected, the pressure solver is used automatically/the simulation stopped nicely rather than crashing.***
- `initial_condition`: Specified either via a python script (idealized cases), read `from_file` or `from_equation_of_state` (the density is initially diagnosed using pressure, energy and equation of state; recommended for non-idealized cases for consistency between thermodynamic variables).
- `boundary_conditions` are not necessary if pressure boundary conditions are defined (which is recommended). `Absorption` specification is not required either.

- `vector_field (Velocity)`
  - `vector_field (Velocity)/prognostic/mesh (VelocityMesh)` should be selected.
  - `vector_field (Velocity)/prognostic/mesh (VelocityMesh)/equation(LinearMomentum)` should be selected.
  - `spatial_discretisation/discontinuous_galerkin` should be selected.
  - `spatial_discretisation/discontinuous_galerkin/viscosity_scheme/bassi_rebay` with `partial_stress_form` is recommended for LES or with fixed viscosity.
  - `spatial_discretisation/discontinuous_galerkin/advection_scheme/project_velocity_to_continuous` should be selected.
  - `spatial_discretisation/discontinuous_galerkin/advection_scheme/integrate_advection_by_parts/twice` should be selected.
  - `spatial_discretisation/discontinuous_galerkin/conservative_advection` should be set to zero.
  - `temporal_discretisation/relaxation` should be set to zero.
  - `temporal_discretisation/discontinuous_galerkin/maximum_courant_number_per_subcycle` should be specified.
  - `boundary_conditions`: For the surface, either use `surface_ocean_COARE3` or `type (dirichlet)/align_bc_with_cartesian` with only normal velocity component selected and set to 0 (free slip condition, often sed at domain lid) or all three velocity components selected and set to 0 (no slip condition, often used at domain surface)
  - `boundary_conditions`: At the inlet, set the normal velocity components as appropriate. Nothing needs to be set for other boundaries (except possibly top BC as a rigid lid with normal velocity set to 0). There is also the option to use time dependent boundary conditions. BC values are read in from external files (`from_file` option) and the number of input files is prescribed (see Section 8 for the required file name formats).
  - `vector_field (Absorption)/diagnostic/algorithm (atmosphere_forcing_vector)`: Optional for sponge layers. Possibility to use time dependent conditions consistent with the BC.

- `scalar_field (PotentialTemperature)` (This is the default thermal variable used for ATHAM-Fluidity, although the absolute temperature or internal energy may also be used)
  - `scalar_field (PotentialTemperature)/prognostic/mesh (VelocityMesh)`: VelocityMesh is the default mesh for any scalar, but a different ScalarMesh can also be defined if desired.
  - `scalar_field(PotentialTemperature)/prognostic/mesh(VelocityMesh)/equation(AdvectionDiffusion)`: AdvectionDiffusion is the default equation type for any scalar.

- spatial_discretisation/discontinuous_galerkin
- spatial_discretisation/discontinuous_galerkin/advection_scheme/advection_scheme/upwind is the default upwind flux. lax_friedrichs may also be used but integrate_advection_by_parts must then be set to once.
- spatial_discretisation/discontinuous_galerkin/advection_scheme/project_velocity_to_continuous
- spatial_discretisation/discontinuous_galerkin/advection_scheme/integrate_advection_by_parts/twice
- spatial_discretisation/discontinuous_galerkin/advection_scheme/include_continuity_residual: Optional. This option allows to solve for conservative scalar equations via a correction term added after the pressure solve.
- spatial_discretisation/discontinuous_galerkin/slope_limiter: Vertex_Based is the most diffusive but safest option. Hermite_WENO is less diffusive and works very well in most applications (best choice if it works).
- spatial_discretisation/subsidence adds constant subsidence under the form of a large-scale horizontal divergence (possibility to apply subsidence to horizontally averaged field only).
- spatial_discretisation/discontinuous_galerkin/conservative_advection: 0 (non conservative). Unstable if set to 1 (conservative). Conservation can be achieved with include_continuity_residual.
- temporal_discretisation/discontinuous_galerkin with max CFL.
- initial_condition: Use python (idealised) or from_file (non-idealised).
- boundary_conditions: For the surface, either use surface_ocean_COARE3, type (dirichlet) (python or from_file), or nothing (zero-gradient).
- boundary_conditions: At inlet, use Dirichlet (do not apply_weakly). Nothing to be set for other boundaries (do not define other boundaries). Possibility to use time dependent boundary conditions. BC values are read in external files (from_file option) and the number of input files is prescribed (see section 8).
- scalar_field (Absorption)/diagnostic/algorithm (atmosphere_forcing_scalar): Optional for sponge layers.
- galerkin_projection/discontinuous should be selected.

- scalar_field (VapourWaterQ) (water vapor fraction, recommended for moist processes. TotalWaterQ is also a possible choice.)
  - scalar_field (VapourWaterQ)/prognostic/mesh (VelocityMesh): VelocityMesh is the default mesh for any scalar, but a different ScalarMesh can also be defined.
  - scalar_field(VapourWaterQ)/prognostic/mesh(VelocityMesh)/equation(AdvectionDiffusion)
  - spatial_discretisation/discontinuous_galerkin should be selected.
  - spatial_discretisation/discontinuous_galerkin/advection_scheme/advection_scheme/upwind is the default upwind flux. lax_friedrichs may also be used but integrate_advection_by_parts must then be set to once.
  - spatial_discretisation/discontinuous_galerkin/advection_scheme/project_velocity_to_continuous
  - spatial_discretisation/discontinuous_galerkin/advection_scheme/integrate_advection_by_parts/twice
  - spatial_discretisation/discontinuous_galerkin/advection_scheme/include_continuity_residual: Optional. This allows to solve for conservative scalar equations via a correction term added after the pressure solve.
  - spatial_discretisation/discontinuous_galerkin/slope_limiter: Vertex_Based is the most diffusive but safest option. Hermite_WENO is less diffusive and works very well in most applications (best choice if it works).

- `spatial_discretisation/subsidence` adds constant subsidence under the form of a large-scale horizontal divergence (possibility to apply subsidence to horizontally averaged field only).
- `spatial_discretisation/discontinuous_galerkin/conservative_advection`. 0 (non conservative). Unstable if set to 1 (conservative). Conservation can be achieved with `include_continuity_residual`.
- `temporal_discretisation/discontinuous_galerkin` with max CFL
- `initial_condition`: Use `python` (idealised) or `from_file` (non-idealised).
- `boundary_conditions`: For the surface, either use `surface_ocean_COARE3`, `type(dirichlet)` (`python` or `from_file`), or nothing (zero-gradient).
- `boundary_conditions`: At inlet use Dirichlet (do not `apply_weakly`). Nothing to be set for other boundaries (do not define other boundaries). Possibility to have `time_dependent` BCs.
- `scalar_field (Absorption)/diagnostic/algorithm (atmosphere_forcing_scalar)`: Optional for sponge layers.
- `galerkin_projection/discontinuous`

- Any other prognostic scalar can be defined in the same way.

`material_phase [name]` **prescribed fields:**

- Typically used to specify an initial reference (hydrostatic) state
- `scalar_field (HydrostaticReferencePressure)` defined on `PressureMesh` and initialized in same way as the pressure but without perturbations. Used in evaluation of pressure gradient term.
- `scalar_field (HydrostaticReferenceDensity)` defined on `PressureMesh` and initialized in the saem way as the density but without perturbations. Used in buoyancy calculation.
- `scalar_field (HydrostaticReferencePotentialTemperature)` defined on `VelocityMesh` (or `ScalarMesh`) and initialized in the same way as the potential temperature but without perturbations. Used in buoyancy calculation (if selected) and in the turbulent diffusion operator.
- `scalar_field (HydrostaticReferenceScalarName)` defined on `VelocityMesh` (or `ScalarMesh`) and initialized in the same way as the scalar but without perturbations. Used in the turbulent diffusion operator.
- If `value/from_file/time_dependent` selected, the prescribed field will vary in time (the BCs and sponge layer values should also be `time_dependent`)
- It may be necessary to choose a `solver` even for prescribed fields (no solver by default) in the case when projections will be performed.

`material_phase [name]` **diagnostic fields:**

- `vector_field (DG_CourantNumber)`: MUST be selected in list of available diagnostics with DG.
- Atmosphere specific diagnostics: `Saturation`, `Reflectivity`, `VapourWaterQ` (default is prognostic but can be diagnosed), `CompressibleContinuityResidual` (for conservative form). All are present in default list, with `algorithm (internal)`.
- Atmospheric surface diagnostics: `SurfaceTemperature`, `SurfacePrecipitation`, `LiquidWaterPath`... Defined on scalar mesh, with `algorithm(internal)`. `boundary_conditions (diagnostic)` field is active with `surface_ids` corresponding to surface BC, `parent_field_name` is name of reference field (e.g. `Qdrop`), and type is selected in list to match diagnostic field name. `LiquidWaterPath` and `CloudCover` require vertical integrals which are computed by first averaging in the vertical direction (using iterative Helmholtz diffusive filter with recommended length scale alpha $\approx$ domain height) and then multiplying by domain height.

LES parameterization:

- `field/prognostic/spatial_discretisation/disontinuous_galerkin/les_model` (specific implementation for DG).
- `les_model/tensor_field (EddyViscosity)` for velocity or `les_model/tensor_field(ScalarNameDiffusivity)` for scalars.
- `les_model/tensor_field (name)/diagnostic/algorithm (internal)` (and must be defined on same mesh as the main variable).
- `les_model/tensor_field(SFSLeonard)` is optional, to be defined with non_linear model.
- Default methods are `second_order (Smagorinsky)` or `non_linear` (Kosovic, for velocity only) (with appropriate constants as suggested)
- For `second_order`: `buoyancy_correction` (turbulence inhibition or enhancement depending on stability, `Brown` is default)

Sponge layers and nudging:

- `scalar_field (Absorption)/diagnostic/algorithm (atmospher_forcing_scalar)` (or equivalent for vector - velocity).
- `algorithm (atmospher_forcing_scalar)/from_file` or `python` (depending on initialization method). Possibility to define time varying layers with `time_dependent`.
- `algorithm (atmospher_forcing_scalar)/sponge_layer_scalar_absorption/coefficient`: relaxation coefficient in sponge layer (0.25?)
- Possibility to define layers along each boundary: top (`z_sponge`), or sides (`x_sponge` and `y_sponge`). The position of the sponge layer (base) must be specified (in m).
- `algorithm(atmospher_forcing_scalar)/scalar_nudging`: `time_scale` is relaxation time and `use_horizontal_mean` applies nudging to horizontal averages instead of local scalar values (horizontal means are computed by iterating a Helmholtz (diffusion operator) filter in the horizontal with recommended length scale alpha $\approx$ domain length).

Cloud microphysics:

- `cloud_microphysics/time_integration (Direct)`, default integration for microphysics, integrates microphysics source terms directly within the advection step
- `cloud_microphysics/time_integration (Splitting)`, time-split method for microphysics where microphysics is integrated independently after the main time marching loop
- `cloud_microphysics/time_integration (Strang)`, time-split method for microphysics where microphysical variables are advanced by half a time-step before the main integration sequence and half a time-step after
- `cloud_microphysics/time_integration/limit_after_advance` applies slope limiter after microphysics step (when computed independently from the rest)
- `cloud_microphysics/relaxation` (optional) sets the relaxation parameter for the calculation of microphysics sources (when 0, values from previous time-step are used)
- `cloud_microphysics/condensation_evaporation(saturation_adjustment)` uses the saturation adjustment procedure to calculate liquid content
- `cloud_microphysics/condensation_evapration (Simple)` diffusion/evaporation sources are computed and treated implicitly
- `cloud_microphysics/condensation_evapration (Analytic)` diffusion/evaporation sources are computed and treated analytically (recommended)

- `cloud_microphysics/condensation_evapration (Adaptive)` diffusion/evaporation sources are computed and a 2nd order backward difference method with variable step size is used to guarantee 2nd order accuracy
- `cloud_microphysics/no_negative_concentrations` applies a strong limit to concentrations
- `cloud_microphysics/diagnostic`: Only mass mixing ratios of liquid drops can be included (`Qdrop` or `Qrain`) and although they are advected (prognostic quantities), only `saturation_adjustment` to diagnose microphysical processes
- `cloud_microphysics/fortran_microphysics`: Full microphysical schemes
- `cloud_microphysics/fortran_microphysics/scalar_field (MicrophysicsSource)` defines options related to the microphysics sources (must be diagnostic, defined on the same mesh as the microphysical quantities)
- `cloud_microphysics/fortran_microphysics/scalar_field (SinkingVelocity)` same as above
- `cloud_microphysics/fortran_microphysics/two_moment_microphysics (seifert_beheng)` is the default microphysical scheme from Seifert and Beheng (only warm phase microphysics implemented so far). 5 prognostic quantities must be defined (CCN, Ndrop, Qdrop, Nrain and Qrain). Some can be directly prescribed (CCN and Ndrop in particular, in which case only one moment is computed for droplets). See prognostic scalar definition above to setup these fields
- `two_moment_microphysics (seifert_beheng)/cold_microphysics` switches on ice particles and cold microphysics. Although no proper ice microphysics is implemented, there is already the possibility to define ice particle quantities.
- `two_moment_microphysics (seifert_beheng)/autoconversion_radius` sets the auto conversion radius for the scheme (default is 40 microns)
- `two_moment_microphysics (seifert_beheng)/simple_activation` droplets are systematically formed in supersaturated regions and Ndrop is directly set to the prescribed number of CCN (from CCN scalar field)
- `two_moment_microphysics (seifert_beheng)/detailed_activation` activation of new cloud droplets uses a simple model based on Kohler theory but for a single mono-modal aerosol type
- `two_moment_microphysics (morrison)` is the same as `seifert_beheng` but uses parts of Morrison's scheme and Kogan's scheme (not as well implemented as seifert and beheng)
- `one_moment_microphysics (kessler)` one moment scheme where only Qdrop and Qrain are prognostic while Ndrop is prescribed and Nrain diagnostic. Uses Kessler auto conversion plus parts of Thompson's scheme. `mass_threshold` is for auto conversion

Others:

- `mesh_adaptivity` contains the options defining a grid adaptive simulation
- `radiation_model` is a template for configuration options related to a radiation transfer model. The current options are actually not used in the code as the RRTM is not fully coupled yet.
- `flredecomp` contains options relative to domain decomposition using flredecomp. If problems related to the ParMETIS library arise (see Section 2), typically observed when executing flredecomp, a different partitioner can be used. In this case, `flredecomp/final_partitioner/zoltan/method[hypergraph]` will be preferred. If nothing is specified, the ParMETIS graph is used for partitioning.

6 Creating a numerical mesh

The numerical meshes for ATHAM-Fluidity are generated using the external software `Gmsh`. The `Gmsh` binaries (or source code) can be downloaded from `http://gmsh.info/#Download`. If building `Gmsh` from source, following the installation instructions within the included README file.

Generating the mesh is a two stage process. The geometry of the modelling domain and mesh resolution parameters (plus any additional options) must first be specified in an ASCII text file with the extension `.geo`. Guidance on the format of these geometry files, as well as some examples, can be found in the `Gmsh` reference manual, available here: `http://gmsh.info/doc/texinfo/gmsh.pdf`. Users could also look at the `.geo` files included with the ATHAM-Fluidity test cases/examples (see Section 7). The modelling domain and mesh can also be visualised within the `Gmsh` GUI, which is launched by typing:

```
$ gmsh <mesh_file>.geo
```

Once the geometry file has been created, the `Gmsh` executable should be invoked to read the contents of this `.geo` and generate a corresponding `.msh` file - a much larger file that contains a list of all the mesh elements and node coordinates:

```
$ gmsh -N <other options> <mesh_file>.geo
```

where `N` is the dimension of the modelling domain (i.e. 2 or 3). Useful '`other options`' include `-bin`, which writes the `.msh` file in binary rather than ASCII format (thereby saving disc space) and `-optimize`. This latter option can be very important when generating a fully unstructured mesh within a complex geometry (e.g. above a topographical surface) as it passes over the mesh a second time to ensure that there are no 'bad' (i.e. very small) elements that might dramatically reduce a CFL-limited model timestep.

7 ATHAM-Fluidity test cases and examples

7.1 Test cases

A number of 2-dimensional test cases have been included with the ATHAM-Fluidity source code (under the '`tests`' subdirectory). These comprise a series of benchmark and idealized atmospheric simulations that were originally performed in order to evaluate ATHAM-Fluidity. Full details and results of these simulations can be found in Savre *et al.* (2016, Monthly Weather Review, 144:4349-4372). Note, however, that due to subsequent code changes and/or tweaks to model parameters, results will be similar but not identical to those presented in this paper. A brief description of each test case is also given below:

- `warm_bubble`: A positive Gaussian potential temperature perturbation is initially imposed in an otherwise neutral and static environment in hydrostatic balance. This results in a rising smooth warm bubble with a Kelvin-Helmholtz rotor developing on each side of the bubble. The configuration from Savre *et al.* (2016) that uses a 10 m spatial resolution is specified.

- `warm_bubble_adapt`: Same as the `warm_bubble` test case, but using mesh optimisation. The minimum edge length (maximum resolution) is set to 5 m and the maximum edge length (coarsest resolution) is set to 250 m. Optimisation is computed based on a target absolute interpolation error for potential temeprature.

- `density_current`: A negative Gaussian potential temperature perturbation is initially imposed in an otherwise neutral and static environment in hydrostatic balance. This rapidly sinks, hits the solid surface and spreads out as a cold density current with three rotors on each side. The configuration from Savre *et al.* (2016) that uses a 200 m spatial resolution is specified.

- `warm_cold_bubbles`: This case is similar to the `warm_bubble` set-up with the addition of a second smaller cold bubble above and off-centre with the warm bubble. As time progresses, these bubbles collide and mix. The configuration from Savre *et al.* (2016) that uses a 10 m spatial resolution is specified.

- `inertial_gravity_waves`: This case involves a horizontally propagating nonhydrostatic gravity wave in a channel, resulting from an initial potential temperature perturbation in an otherwise uniformly stratified atmosphere with homogeneous horizontal flow. The configuration from Savre *et al.* (2016) that uses an adaptive timestep based on a CFL number of 0.25 is specified.

- `schar_mountain`: In this case, a dry atmospheric flow is forced over a five-peak mountain range with constant velocity, producing steady-state gravity waves. The configuration differs from Savre *et al.* (2016) in that the simulation time has been reduced from 8 h to 0.5 h to keep the runtime down (this could be easily changed back in the `.flml` file).

- `single_mountain`: In this case, a dry atmospheric flow is forced ocer a single linear mountain profile with constant velocity. Similarly, the configuration differs from Savre *et al.* (2016) in that the simulation time has been reduced from 5 h to 0.5 h.

As well as the `.flml` and mesh geometry files, each of the above test directories also includes an `.xml` file that can be used to automate the execution of each test case and a subsequent comparison of the model output against stored results (for the benefit of model development testing). To run a particular test case, issue the following command:

```
$ <A-F_install_path>/tools/testharness.py -f <test_case>.xml
```

where `<test_case>` should be replaced by one of the test case names above, e.g. `warm_bubble`. To run all the test cases in succession, issue the following command:

```
$ <A-F_install_path>/tools/testharness.py -t atham -l long
```

Note that all the above test cases are set up to run in parallel on *24 processes* in order to keep runtimes down. If you are using a machine that does not have this many processes (or if you want use more processes!), you should open the relevant `.xml` file(s) in a text editor and modify this value (it appears in two places near the top of each file) as appropriate.

7.2 Example cases

A number of 2- and 3-dimensional examples have also been included with the ATHAM-Fluidity source code (under the '`examples`' subdirectory). A brief description of each example is given below:

- `squall_line`: This simulation of a 2-d squall line is often used as a benchmark to evaluate cloud microphysics codes. An initially dry but convectively unstable atmosphere is perturbed by a low-level warm bubble, which initiates a moist convection cell. A vertical velocity profile with low-level wind shear aids the formation of a long-lived squall line, as precipitation from one cell causes evaporative cooling at the surface, creating a cold pool that spreads out as a density current. On the upwind side, the vorticity associated with the spreading cold pool counteracts the vorticity associated with the ambient low-level shear, producing deeper lifting and thus maintaining the cell regeneration process.

- `rain`: This 3-d boundary-layer-scale simulation involves a logarithmic velocity profile, a constant potential temperature profile up to 1000 m with an inversion above this, and a moisture profile that gives a stratified layer of saturated air (clouds) just below the inversion. Turbulence is generated at the domain inlet using Fluidity's synthetic-eddy method. A spatially varying surface boundary condition for potential temperature is defined, where the surface in the last two thirds of the domain is 5K warmer than the upwind third. This generates thermals, leading to convectively-driven cloud and precipitation formation in the downwind half of the domain later in the simulation.

- `precip_extreme`: This simulation represents a simplified version of a case study performed as part of the EU PEARL project (Preparing for Extreme And Rare events in coastaL regions). An extreme pricipitation event, as simulated by a regional climate model, is downscaled close to the region of interest (Greve, Denmark). A simplified numerical mesh without topography or distinct land-sea regions is adopted, with the horizontal domain extent also reduced from 100 x 100 km to 100 x 0.6 km to accommodate the use of modest computational resources (the set-up uses 24 cores by default). If the user has access to a larger number of cores, a mesh geometry file with the full horizontal extent of 100 x 100 km is also supplied. A version of the model input (`.flml`) file in which the inlet velocity profile is scaled to give a maximum of 10 $\mathrm{m\,s^{-1}}$ is also included - this reduces the (CFL-limited) timestep for a faster runtime and also allows for the generation of convective clouds closer to the inlet (these clouds form due to the advection of a convectively unstable atmosphere into the domain and the vertical perturbations provided by the inlet turbulence generator).
- `sealevel_extreme`: This simulation represents a simplified version of a second case study performed for PEARL. An extreme sea-level event predicted by the regional climate model is downscaled close to Greve. The same simplified numerical mesh (no topography or distinct land-sea regions, reduced horizontal extent) as in the previous example is adopted.

Any one of these examples can be run by navigating to the relevant directory and issuing the following commands:

```
$ make preprocess
$ make run
```

To clean the output from a previous run, the following command can be used:

```
$ make clean
```

8 Input/output files

By default, the main output files are exported in `.vtu` format. These are directly readable by the ParaView free software (which can also be found on Darwin). In parallel runs, as many files as there are processes are produced, and stored in a separate subdirectory for each output time. A `.pvtu` file is also created in the main directory to unify all the sub-domains in ParaView. A `.stat` file is also created and updated after each time-step. It contains runtime information on each defined variable (plus others) such as domain averages and min or max values. The file is readable directly or by using the `statplot` visualisation tool which comes with the model source code.

In addition to the `.flml` and mesh files required to run ATHAM-Fluidity, other input files may be necessary. For example, for non-idealized cases for which it is not possible to initialize the simulations using a built-in python script, initial atmospheric soundings can be provided (select the `from_file/type (sounding)` option under `intial_conditions` and/or `boundary_condition`) to homogeneously initialize the domain. These soundings possess a very standard and easy to read structure: the first line contains the values of the surface pressure (hPa), near-surface potential temperature (K), near-surface vapor mixing ratio ($\mathrm{kg\,m^{-3}}$) and near-surface horizontal velocity components ($\mathrm{m\,s^{-1}}$). Then, the 1D variables are stored in columns: first the altitude (m) then the potential temperature, the vapor mixing ratio and the horizontal velocities (vertical velocities are set to 0).

If `time_dependent` boundary conditions are requested (or for sponge layers and nudging), the number of input files to be provided must be prescribed. These files all have the same base name (to be specified in diamond), with an extension indicating the time of the sounding (the first one being 0): e.g. `SOUNDING.00000` (first one), `SOUNDING.07200` (after 2h), `SOUNDING.14400` (after 4h)... Linear interpolations in time are performed between soundings. Note that this feature has not yet been fully implemented and tested in the code.