ATHAM-Fluidity: How to Install and Run

**December 2017**



Prepared by

James Jordan, James O'Neill, Julien Savre & Michael Herzog

Department of Geography, University of Cambridge

Approved by

UNESCO-IHE

Project Coordinator

Westvest 7

2611 AX, Delft

The Netherlands

www.unesco-ihe.org

www.pearl-fp7.eu

# Downloading the ATHAM-Fluidity repository from GitHub

ATHAM-Fluidity is part of the Fluidity project (<http://fluidityproject.github.io/>). Fluidity is an open source, general purpose, multiphase computational fluid dynamics code capable of numerically solving the Navier-Stokes equation and accompanying field equations on arbitrary unstructured finite element meshes in one, two and three dimensions. Fluidity has been used in a number of different scientific areas including geophysical fluid dynamics, computational fluid dynamics, ocean modelling and mantle convection. ATHAM-Fluidity is a development of the main Fluidity project better suited for high resolution atmospheric processes. The ATHAM-Fluidity repository can be 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, e.g. $HOME/ATHAM-Fluidity. Then type the following command (after the command prompt `$'):

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

# 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. This procedure is described in Section 2.3

## 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

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

It is recommended that you add the above line (minus the command prompt) to your .bashrc fille, which is located in your home directory, so that these environment variables are remembered for every login session.

## Self-installation

The following section describes the installation of the ATHAM-Fluidity dependencies on a Linux machine that was running Ubuntu 16.04 LTS (Xenial Xerus). However, it is expected that the same procedure will apply on a machine running Ubuntu 14.04 LTS (Trusty Tahr). 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

### 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 4.8.4. 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.

### 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 2.7.6. 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 1.8.2. If you see an import error, you should install the NumPy package manually.

### 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.openmpi.org/software/ompi/v1.6/> (choose the gzipped tar file, 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 tar file. 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

### 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/](http://www.netlib.org/blas/%20) and [http://www.netlib.org/lapack/](http://www.netlib.org/lapack/%20), 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.

### 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 OK 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-fblaslapack=1 \

--download-blacs=1 --download-scalapack=1 --download-ptscotch=1 \

--download-mumps=1 --download-hypre=1 --download-suitesparse=1 \

--download-ml=1 --with-fortran-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/[user]/tarfiles/petsc-3.5.4 PETSC ARCH=arch-linux2-c-debug 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/[user]/tarfiles/petsc-3.5.4 PETSC ARCH=arch-linux2-c-debug 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 \_le 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

### 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](http://glaros.dtc.umn.edu/gkhome/fsroot/sw/parmetis/OLD%20) (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 tar file 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'

### 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/](http://www.vtk.org/download/%20). Here we install VTK 5.10.1 (vtk-5.10.1.tar.gz). Untar the downloaded tar file 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-packages:$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/vtk-5.10

## Loading Modules

If you are intending to run ATHAM-Fluidity on a cluster that uses environmental variables then the dependencies required may have already been installed centrally by a system administrator. All that is required to prepare the model is therefore to load the relevant modules and set some environment variables. Check with your system administrator exactly which versions of the dependencies in question were used during compilation of ATHAM-Fluidity. A list of all currently loaded modules can be obtained by typing `module list’ on the command line. If this differs from those used during compilation out of date modules can be unloaded via

$ module unload <MODULE NAME>

Alternatively, modules can be loaded with

$ module load <MMODULE NAME>

It is recommended that you add the correct configuration of modules to your .bashrc profile.

# 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, e.g. $HOME/ATHAM-Fluidity:

$ ./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 libraries were installed in the locations as specified 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

# Running ATHAM-Fluidity

Once the ATHAM-Fluidity model options have been appropriately set for a given simulation (see Section 5), or if running one of the supplied example test cases, running the model is straight forward. The general command (run from inside a simulation subdirectory that contains the flml configuration file’) reads:

fluidity <input file>.flml

The above command assumes that you have added <A-F install path>/bin to your $PATH environment variable. <input file>.flml should be replaced with the actual name of the .flml file (generated by diamond) for your simulation. Option flags can also be added. 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). For a parallel run, the numerical domain must first be decomposed using the flredecomp tool:

flredecomp -i X -o Y <serial input file> <parallel input file>

Here, -i is used to specify the initial number of processes (sub-domains) X (typically X=1), and -o is used to specify the output number of processes (sub-domains) Y on which the model will run. <serial input file> is the name of the original .flml file (without the extension) while <parallel input file> is the name of the .flml file (without the extension) generated by flredecomp that will be used as input for the parallel run. After the decomposition is complete, ATHAM-Fluidity is run in the same way as above but specifying the parallel .flml file (with the extension again) rather than the serial file, i.e.:

$ fluidity <parallal input file>.flml

For parallel runs, flredecomp and fluidity need to be executed through the MPI wrapper executable. If this is not done automatically (e.g. through the job submission script that you are using), you should run them in the following way:

mpiexec -np Y flredecomp -i X -o Y <serial input file> <parallel input file>

mpiexec -np Y fluidity <parallel input file>.flml

# ATHAM-Fluidity model options

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 P1DG -P2 atmospheric simulation (the recommended finite element discretization method, in which pressure and density are solved on a 2nd-order continuous grid while momentum and all other scalars are solved on a 1st-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. Example .fml files for the test cases described in section 7 can be found under $HOME/ATHAM-Fluidity/tests. 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. In the following section options in the diamond interface will appear in **bold.**

## General

The **simulation** name will be used to prefix model output files. The **problem type** must be set to `fluids', whilst the **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.

## Geometry

**Geometry/mesh (CoordinateMesh)** defines the reference mesh, reads from the mesh file(s) (therefore use 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. Therefore use **Geometry/mesh (VelocityMesh)/from mesh/mesh** and select (**CoordinateMesh**). **Geometry/mesh (VelocityMesh)/from mesh/** **mesh\_shape/polynomial degree** should be set to 1 and **Geometry/mesh (VelocityMesh)/from mesh/mesh continuity** set to **discontinuous** (i.e. a P1DG configuration). **Geometry/mesh (PressureMesh)** is the pressure and density default mesh which again should be defined from the coordinate mesh, in the same manner as the Velocity mesh above. However in this case **mesh shape/polynomial degree** should be set to 2 and **mesh continuity** set to **continuous** (i.e. a P2 configuration).

Additional derived meshes can be defined following the above procedure. For example, if temperature is to be solved prognostically, a new mesh called, for example, TemperatureMesh should be added, by first selecting and renaming **Geometry/mesh** to **Geometry/TemperatureMesh** and then, similar to before, selecting from **Geometry/TemperatureMesh/mesh/from mesh/mesh** and choosing CoordinateMesh. The .mesh shape/polynomial degree should be set to 1 and mesh continuity set to discontinuous (i.e. a P1DG configuration). **Geometry/quadrature/degree** should be set to at least 2N, where N is the maximum polynomial degree. With the P1DG P2 method, N=2 and **so quadrature/degree** should be set to 4 (or greater).

## I/O (input/output)

Input/Output options can be set, with **io/dump period/constant** specifying the time period (in seconds) between each set of output files generated by the model. For the best quality output **io/output mesh (PressureMesh)** should be selected in order to generate outputs on the 2nd-order pressure/density mesh. To enable checkpointing and produce files for model restarts **io/checkpointing** needs to be selected (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.

The time at the start of the simulation can be set using **timestepping/current** **time**. This is typically set to 0, unless restarting from a checkpoint where it is set to the time of the desired checkpoint. The model time step (in seconds) is set using **timestepping/timestep**. This value will depend on the modelled flow-field characteristics and the minimum grid mesh size. If adaptive timestepping is used (often employed with adaptive grid meshing) then this is the initial time-step, as it will change throughout the run. The end time of the simulation can be set using **timestepping/finish time**, which specifies the simulation end-time (in seconds).

## Physical parameters

Gravity must be defined in **physical parameters/gravity/magnitude** (9.81 for acceleration due to Earth's gravity, for example), with the direction it acts in given by **physical parameters/gravity/vector field (GravityDirection).** For a 2D case, the constants 0 and -1 would be input if the 2nd dimension coordinate scheme was positive vertically upwards; for a 3D case, the constants 0, 0 and -1 would be input if the 3rd dimension was positive vertically upwards.

## Material phase

A new material phase must be selected and named. For the purpose of this section we assume the name chosen is “Fluids”.

### Equation of state

A compressible equation of state must be selected from **material\_phase (Fluids)/equation of state/compressible**. **Giraldo** is the default model for ideal gas, although it will eventually be replaced by the ATHAM option (pending future development). **Reference pressure** must be set and is the same as that used in the definition of the Exner function (100000 Pa). **C\_P** and **C\_V** are the default values for the heat capacity of dry air at constant pressure and volume respectively (1004.64 and 717.6 in this case). When enabled, **buoyancy\_from\_pt** allows the user to define buoyancy based on density potential temperature instead of density. The **constant\_cp\_cv**, if enabled, holds constant the heat capacities for the moist atmosphere, and sets them equal to their default dry atmosphere values. If **subtract\_out\_reference\_profile** is enabled then the buoyancy, pressure gradient and scalar diffusion terms are computed by subtracting out the hydrostatic reference profile. These reference profiles MUST be defined beforehand, as described later.

### Pressure

The pressure field is set by enabling **material\_phase (Fluids)/scalar\_field (Pressure).** The mesh should be set to **prognostic/mesh (PressureMesh**), with **spatial\_discretisation/continuous\_galerkin/remove\_stabilisation** also enabled, and **scheme/poisson\_pressure\_solution** set to never. Your preferred solver should be set (**iterative\_ method (gmres)** works fine as a default) with relative error and max iterations for the chosen solver (suggested values are 1.0e-7 and 1000). I**nitial\_conditions** can be specified either via a python script (for idealized cases) or read from a file (non-idealised cases). Boundary conditions are set individually for each boundary with **boundary\_conditions (name)**. Setting Dirichlet boundary conditions on all sides of the model, as well as the top, (chosen via selecting ids to match those of the mesh generated in gmsh) was found to work correctly. The method (**python** or **from\_file**) must again be selected. Time varying boundary conditions can be set with **boundary\_conditions (name)/from\_file/time\_dependent**. The number of input files must then be prescribed (see Section 6 for the required file name formats).

### Density

A density field must be enabled by selecting **material\_phase (Fluids)/scalar\_field (Density).** The mesh should be set to be the same as the pressure mesh with **prognostic/mesh (PressureMesh)**, i.e. the density must be defined on the same continuous mesh as pressure. As the density is diagnosed, **equation** is not required. The **spatial\_discretisation/continuous\_galerkin/stabilisation/no\_stabilisation** option should be selected, whilst **spatial\_discretisation/continuous\_galerkin/conservative advection** is, again, not used as the density is diagnosed. Time advancing is set to **temporal\_discretisation/theta.** It is recommend to choose the same solver as for the pressure solver, setting relative error and max iterations as before (suggested values are 1.0e-7 and 1000, respectively). Initial conditions are specified either via a python script (idealized cases), read from file or from the equation of state (the density is initially diagnosed using pressure, energy and equation of state. The latter is recommended for non-idealized cases for consistency between thermodynamic variables). Boundary conditions are not necessary if pressure boundary conditions are defined (which is recommended). It is not necessary to specify absorption.

### Velocity

Velocity should be set by selecting **vector\_field (Velocity)/prognostic.** The mesh is chosen to be on the **(VelocityMesh).** The **equation** to be used should be specified as **LinearMomentum.** The spatial discretisation should be **spatial\_discretisation/discontinuous\_galerkin** with **viscosity\_scheme/bassi\_ebay** enabled. **Partial\_stress\_form** is recommended for LES or with fixed viscosity. For the advection scheme, **spatial\_discretisation/discontinuous galerkin/advection\_scheme/project\_velocity\_to\_continuous** should be selected. With **spatial\_discretisation/discontinuous\_galerkin/advection\_scheme/integrate\_advection\_by\_parts/twice** also enabled. **Conservative\_advection** should be set to zero (nonconservative). The **temporal\_discretisation/relaxation** should be set to zero whilst **temporal\_discretisation/discontinuous\_galerkin/maximum\_courant\_number\_per\_subcycle** should be specified. For surface boundary conditions either **use surface\_ocean\_COARE3** or **type(dirichilet)/align\_bc\_with\_cartesian** with only the normal velocity component selected and set to zero for a free slip condition. Inlet boundary conditions should be set as appropriate for the normal velocity components. Nothing needs to be set for the other boundaries (with the possible exception of the top boundary condition as a rigid lid with normal velocity set to zero). Time dependent boundary conditions are also available, with values read in from external files (from file option) and the number of input files **prescribed** (see Section 6 for the required file name formats). It is possible to set up a “sponge” layer with **vector\_field (Absorption)/diagnostic/algorithm (atmosphere forcing vector).**

### Potential Temperature

Potential Temperature is the default thermal variable used for ATHAM-Fluidity, although the absolute temperature or internal energy may also be used. It is set with **scalar\_field (PotentialTemperature)** and should set its mesh to be on the velocity mesh with **/prognostic/mesh (VelocityMesh**). Note that the velocity mesh is the default mesh for any scalar, but a different ScalarMesh can also be defined if desired. The default **equation type** is **AdvectionDiffusion,** the default type for any scalar. The **spatial discretisation** should be set to **discontinuous\_galerkin** withan **upwind** advection scheme. **Advection\_ scheme/project\_velocity** should be continuous with **advection\_by\_parts/twice** and **spatial\_discretisation/discontinuous\_galerkin/slope\_limiter: Vertex\_Based.** This is the most diffusive but safest option, **Hermite\_WENO** is less diffusive and works well in most situations, and as such it is the best choice to use if stable. Constant subsidence can be added with **spatial\_discretisation/subsidence** in the form of a large-scale horizontal divergence (with a possibility to apply subsidence to horizontally averaged fields only). **Conservative\_advection** should be set to zero (non-conservative). It is unstable if set to 1 (conservative). Conservation can be achieved with **include\_continuity\_residual** and **temporal\_discretisation/discontinuous\_galerkin** with max CFL. Initial conditions can be set using python (idealised) or from a file **(non-idealised\_ boundary conditions).** For the surface boundary conditions, either use **surface\_ocean\_COARE3, type (dirichlet) (python or from file), or nothing** (zero-gradient). At the inlet, use **Dirichlet** (do not apply weakly). Other boundary conditions should not need to be set. It is possible to use time dependent boundary conditions. Boundary condition values are read from external files (from file option) and the number of input files is prescribed (see section 6). An absorption layer can be set up with **scalar\_field (Absorption)/diagnostic/algorithm** (atmosphere forcing scalar). If doing so **galerkin\_projection/discontinuous** should be selected.

### Water Vapour

Water vapour is represented by **scalar\_field (VapourWaterQ),** the water vapour fraction, recommended for moist processes. Scalar\_field(**TotalWaterQ)**, the total water vapour (concentration multiplied by volume), is also a possible choice. The mesh should be set to the velocity mesh in **scalar\_field (VapourWaterQ)/prognostic/mesh (VelocityMesh**) and the **equation** set to **AdvectionDiffusion.** Spatial discretisation should be **spatial\_discretisation/discontinuous\_galerkin** should be selected with an **upwind\_advection\_scheme. Advection\_ scheme/project\_velocity** should be continuous with **advection\_by\_parts/twice** and **spatial\_discretisation/discontinuous\_galerkin/slope\_limiter: Vertex\_Based.** This is the most diffusive but safest option, **Hermite\_WENO** is less diffusive and works well in most situations. It is the best choice to use if stable. If enabled **spatial\_discretisation/subsidence** adds a constant subsidence under the form of a large-scale horizontal divergence with the possibility to apply subsidence to horizontally averaged field only. **Spatial\_discretisation/discontinuous\_galerkin/conservative\_advection** should be set to zero (non conservative). It is unstable if set to 1 (conservative). Conservation can be achieved with **include\_continuity\_residual**. Initial conditions can be set using python (idealised) or from a file (non-idealised boundary conditions). For the surface boundary conditions, either use **surface\_ocean\_COARE3**, type (dirichlet) (python or from file), or nothing (**zero-gradient**). At the inlet, use **Dirichlet** (do not apply weakly). Other boundary conditions should not need to be set. It is possible to use time dependent boundary conditions. Boundary condition values are read from external files (**from\_file** option) and the number of input files is prescribed (see section 6). An absorption layer can be set up with **scalar\_field (Absorption)/diagnostic/algorithm** (**Atmosphere\_forcing\_scalar**). If doing so **galerkin\_projection/discontinuous** should be selected. If using an optional sponge layer then **galerkin\_projection** should be set to **discontinuous.** Any other prognostic scalar can be defined in the same way as those described above, material phase [name] prescribed fields:

### Hydrostatic reference states

The hydrostatic reference pressure is enabled with **scalar\_field (HydrostaticReferencePressure)** defined on the Pressure Mesh and initialized in the same way as the pressure but without perturbations. It is used in the evaluation of the pressure gradient term. Similarly, the hydrostatic reference density is enabled with **scalar\_field (HydrostaticReferenceDensity)** anddefined on the Pressure Mesh and initialized in the same way as the density but without perturbations. It is used in the buoyancy calculation. The hydrostatic reference potential temperature is enabled with **scalar\_field (HydrostaticReferencePotentialTemperature)** defined on the Velocity Mesh (or ScalarMesh) and initialized in the same way as the potential temperature but without perturbations. It is used in the buoyancy calculation (if selected) and in the turbulent diffusion operator. Finally, the scalar field (HydrostaticReferenceScalarName) defined on VelocityMesh (or ScalarMesh) and initialized in the same way as the scalar but without perturbations. It is used in the turbulent diffusion operator. It may be necessary to choose a solver even for prescribed fields (which have no solver by default) in the case when projections will be performed.

### Cloud microphysics:

The default integration of the cloud microphysics component with the rest of the model is enabled with **cloud\_microphysics/time\_integration (Direct)**. This integrates microphysics source terms directly within the advection step whilst **time\_integration (Splitting)** uses a time-split method for microphysics where microphysics is integrated independently after the main time marching loop. **Time\_integration (Strang)** is another 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. Finally, **time\_integration/limit\_after\_advance** applies a slope limiter after the microphysics step (when computed independently from the rest). The relaxation parameter, **relaxation (optional)** sets the relaxation parameter for the calculation of microphysics sources (when it is zero the values from the previous time-step are used). **Condensation\_evaporation(saturation adjustment)** uses the saturation adjustment procedure to calculate liquid content whilst with **condensation\_evaporation (Simple)** diffusion/evaporation sources are computed and treated implicitly. When using **condensation\_evaporation (Analytic)** diffusion/evaporation sources are computed and treated analytically (this is the recommended setting) whilst with **condensation\_evaporation (Adaptive)** diffusion/evaporation sources are computed with a 2nd order backward difference method with variable step size to guarantee 2nd order accuracy. To prevent unphysical negative concentrations, enable **no\_negative\_concentrations** to apply a strong limit to concentrations. The full microphysics scheme is found within **fortran\_microphysics. Scalar\_field (MicrophysicsSource**) defines options related to the microphysics sources (must be diagnostic and defined on the same mesh as the microphysical quantities, likewise for **scalar field (SinkingVelocity). Two\_moment\_microphysics (Morrison)** is the default microphysical scheme. Five prognostic quantities must be defined as scalar fields **(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. **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. **Autoconversion** radius sets the auto conversion radius for the scheme (default is 40 microns), whilst with **simple\_activation** droplets are systematically formed in supersaturated regions and Ndrop is directly set to the prescribed number of CCN (from CCN scalar field). **Detailed\_activation** simulates theactivation of new cloud droplets using a simple model based on Kohler theory **One\_moment\_microphysics (kessler)** is a one moment scheme where only the mass of cloud and rain droplets are prognostic while the number of cloud droplets is prescribed and the number of rain droplets diagnostic. It uses Kessler auto conversion plus parts of Thompson's scheme. **Mass\_threshold** is for auto conversion

## **Flredecomp**

This section contains options related 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.

# 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](http://gmsh.info/%23Download). If building Gmsh from source, follow 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.

# ATHAM-Fluidity test cases and examples

## Test cases

A number of 2-dimensional test cases have been included with the ATHAM-Fluidity source code (under the ‘tests’ subdirectory, located in $HOME/ATHAM-Fluidity/tests). 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. Example results from these cases can be found in the ‘test\_results’ subdirectory of the tests folder. A brief description of each test case, as well as a snapshot image of the result at the end of the simulation using paraview, are 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.

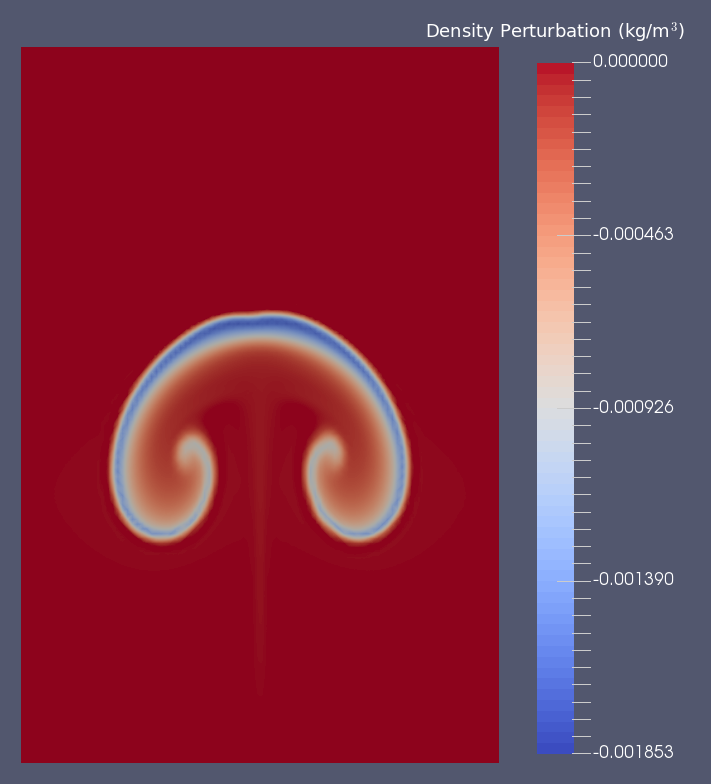


Figure 1: Density perturbation relative to hydrostatic reference density for the warm\_bubble test case. Snapshot is from the end of the simulation.

• **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.



Figure 2: Density perturbation relative to hydrostatic reference density from the density current test case. Snapshot is from the end of the simulation.

**• Inertial gravity waves:** This case involves a horizontally propagating non-hydrostatic 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.

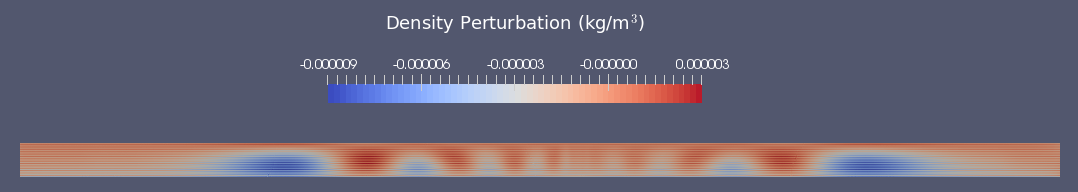


Figure 3: Density perturbation relative to hydrostatic reference density from the inertial gravity waves test case. Snapshot is from the end of the simulation.

• **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).

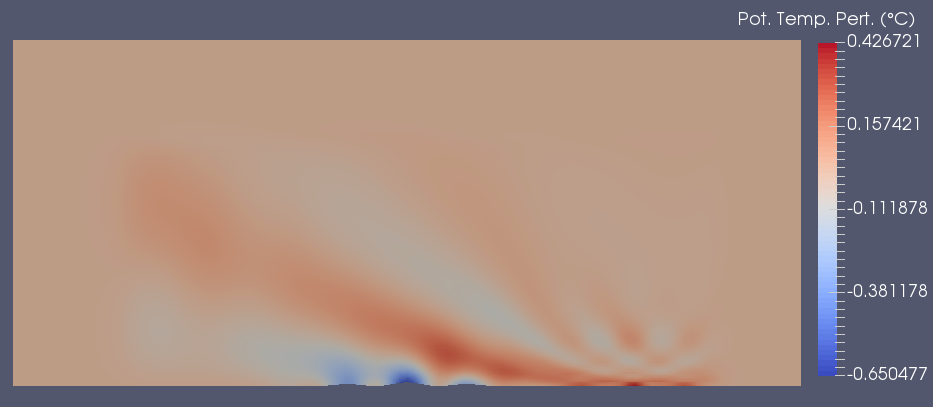


Figure 4: Potential temperature perturbation relative to hydrostatic reference temperature from the single mountain test case. Snapshot is from the end of the simulation.

**• Single mountain:** In this case, a dry atmospheric flow is forced over 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.

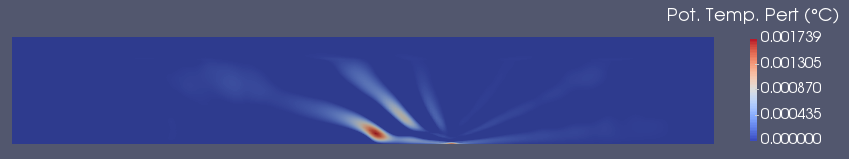


Figure 5: Potential temperature perturbation from the hydrostatic reference temperature for the single mountain test case. Snapshot is from the end of the simulation.

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 cores (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.

## 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:

* **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.

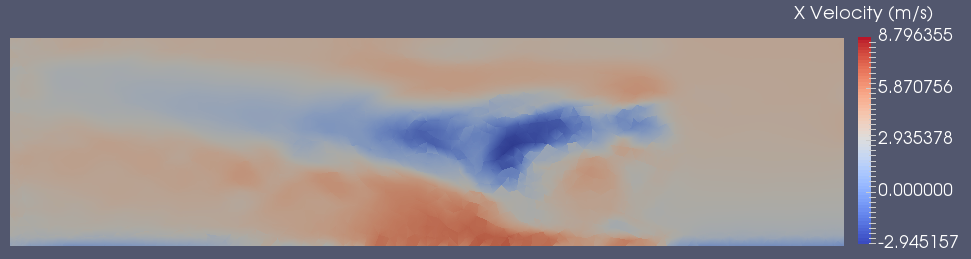


Figure 6: X velocity for the rain example case. Snapshot is from midway through the simulation.

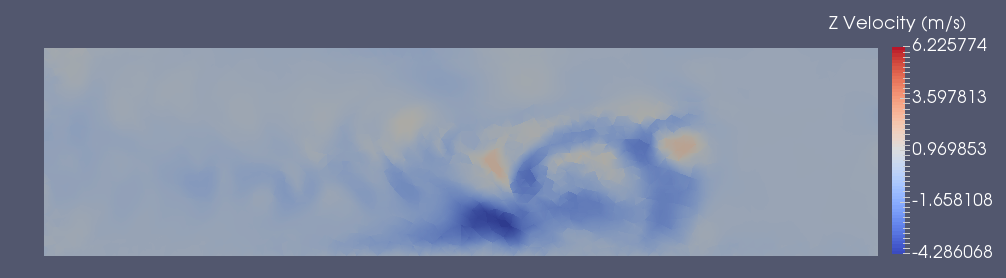


Figure 7: Z velocity for the rain example case. Snapshot is from the middle of the simulation.

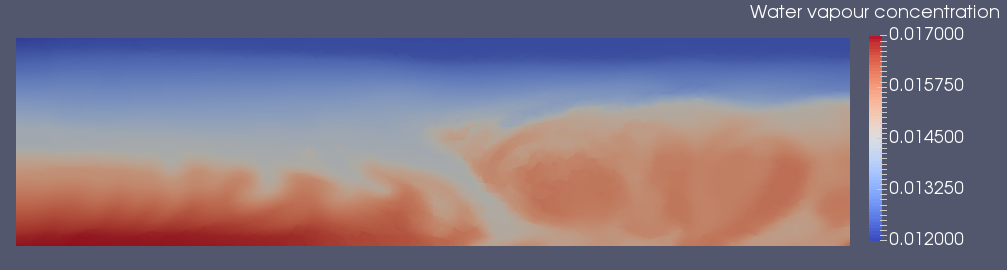


Figure 8: Water vapour concentration for the rain example case. Snapshot is from the end of the model simulation.

* **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 precipitation 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 km x 100 km to 100 km 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 km 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 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).

|  |
| --- |
| Figure 9: Cloud droplet mass concentration for the precip\_extreme example case. Snapshot is from a period of high precipitation. |
| Figure 10: Rain sinking velocity for the precip\_extreme example case. Snapshot is from a period of high precipitation. |

* **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 ( $HOME/ATHAM-Fluidity/tests/warm\_bubble, for example) 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

# Input/output files

By default, the main output files are written in .vtu format. These are directly readable by the ParaView free software (available from [https://www.paraview.org/](https://www.paraview.org/%20) ). In parallel runs, as many files as there are processes are produced and stored in a folder. A .pvtu file is also created 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 using the statplot tool.

In addition to the .flml file required to run ATHAM-Fluidity, several other input files are necessary, in particular initial condition files and mesh files (the necessary files for the cases in section 7 are all located within the repository). In a parallel run, these will be needed by flredecomp only, which will then export .vtu files containing mesh and initial field information. When restarting the parallel run, only these will be needed by ATHAM-Fluidity to restart (this is specified in the modified .flml file). Mesh files are preferably generated using the free GMSH software. GMSH is now the default mesh format for ATHAM-Fluidity. The geometry to be meshed can be defined in a .geo file readable by GMSH.

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 initial 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), surface potential temperature, surface vapour mixing ratio (kgm-3) and surface velocities. Then, the 1D variables are stored in columns: first the altitude (in m) then the potential temperature, the vapour mixing ratio (kgm-3) and the horizontal velocities (vertical velocities are set to zero).

If time dependent boundary conditions are requested for sponge layers or 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 interpolation in time is performed between soundings.

# Licensing

ATHAM-Fluidity can be used under the terms of a GNU Lesser General Public License, the text of which is reproduced below for convenience (as well as being located within the repository).

**GNU Lesser General Public License**

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

**Preamble**

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

**TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

**1.** You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

**2.** You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

**3.** You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

**4.** You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

**5.** A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

**6.** As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

**7.** You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

**8.** You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

**9.** You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

**10.** Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

**11.** If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

**12.** If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

**13.** The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

**14.** If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

**15.** BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

**16.** IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS