DUNE-PDELab-Tutorials Exercise Workflow

How to build and run exercises

You will learn here how to get and build all necessary dune-modules for building the dune-pdelab-tutorials module. Besides that you will learn how to build and run the code from the tutorials and exercises.

1 Installation

First you need to install all the necessary Dune modules. The dependencies for installing DUNE can be found in the installation notes [1]. You will need the core modules [2] and the dune-pdelab and dune-typetree modules [3] for discretization. If you are familiar with git you can instead clone the git repositories [4] [5] from the DUNE homepage. Besides these dune-modules many examples need ALUGrid or UGGrid for grid creation. Both modules can be downlodaded through our gitlab instance [6] [7] . If you enter a terminal and go to your dune-modules you should see the following:

```
[user@localhost]$ cd path/to/dune/folder
[user@localhost]$ ls
dune-alugrid
dune-common
dune-geometry
dune-grid
dune-istl
dune-localfunctions
dune-pdelab
dune-typetree
dune-uggrid
```

2 Building dune-pdelab-tutorials

After downloading all dune-modules and installing the dependencies we want to build the dune-pdelab-tutorials module. In the following we will describe a simple

[1] http://www.dune-project.org/doc/installation/
[2] http://www.dune-project.org/releases/
[3] http://www.dune-project.org/modules/dune-pdelab/
[4] http://www.dune-project.org/dev/downloadgit/
[5] http://www.dune-project.org/modules/dune-pdelab/
[6] https://gitlab.dune-project.org/extensions/du ne-alugrid
[7] https://gitlab.dune-project.org/staging/dune-uggrid

setup that should work in most cases. If you have problems you should first refer to the installation instructions referenced above and see if you can find help there.

Building your DUNE modules is done by a script from dune-common. The easiest way to specify options for the build process is passing an opts file to dune-common. This way you can e.g. switch optimization flags for the compiler. Here are two simple otps files that can be used for debugging and release build:

Listing 1: "Opts file for debuging."

```
CMAKE_FLAGS="
-DCMAKE_C_COMPILER='/usr/bin/gcc'
-DCMAKE_CXX_COMPILER='/usr/bin/g++'
-DCMAKE_CXX_FLAGS_DEBUG='-O0 -ggdb -Wall'
-DCMAKE_BUILD_TYPE=Debug
-DDUNE_SYMLINK_TO_SOURCE_TREE=1
"
```

Listing 2: "Opts file with optimization."

```
CMAKE_FLAGS="
-DCMAKE_C_COMPILER='/usr/bin/gcc'
-DCMAKE_CXX_COMPILER='/usr/bin/g++'
-DCMAKE_CXX_FLAGS_RELEASE='-O3 -DNDEBUG -g0 -Wno-deprecated-
   ↪ declarations -funroll-loops'
-DCMAKE_BUILD_TYPE=Release
-DDUNE_SYMLINK_TO_SOURCE_TREE=1
"
```

If you want to use a different compiler just change the corresponding paths in your opts files. If you save these two files as debug.opts and release.opts in your dunefolder you can use dune-common to build your sources:

```
[user@localhost]$ ./dune-common/bin/dunecontroll --opts=debug
   ↪ .opts --builddir=release-build --module=dune-pdelab-
   ↪ tutorials all
[user@localhost]$ ./dune/dune-common/bin/dunecontrol --opts=
   ↪ debug.opts -builddir=debug-build --module=dune-pdelab-
   ↪ tutorials all
```

3 How to run and change the code

Dune uses cmake as a build system. This generates out of source builds, in our case we specified the two build directories `release-build` and `debug-build`. In these directories the whole structure of your modules is reproduced with your build targets instead of the sources.

In the following we will use the exercise of tutorial00 as an example for basic build system usage. If we want to run the code we have to go to the build directory of this exercise:

```
[user@localhost]$ cd release-build/dune-pdelab-tutorials/
   ↪ tutorial00/task
[user@localhost]$ ./exercise00
```

This program uses an ini file to set some program parameters. If we want to change some parameters we can do that by going to the source directory and changing the `tutorial00.ini` file:

```
[user@localhost]$ cd src\_dir
[user@localhost]$ xdg-open tutorial00.ini
```

After changing the ini file we can rerun the program:

```
[user@localhost]$ cd ..
[user@localhost]$ ./exercise00
```

We can also change the sources of the program by modifying the file `exercise00` or one of the `.hh` file.

```
[user@localhost]$ cd src\_dir
[user@localhost]$ xdg-open exercise00.cc
```

After changing the source you need to rebuild your program before running it:

```
[user@localhost]$ cd ..
[user@localhost]$ make
[user@localhost]$ ./exercise00
```