BUILD-RUN-DOCUMENT

Saurabh Sharma

30 November 2018

This document shows the steps needed to clone, build and run the FWI code and to install the prerequisite packages.

1 Pre-requisites

The prerequisite development tools needed can be installed using the following commands.

1. `sudo apt-get install git`
2. `sudo apt-get install qt5-default`
3. `sudo apt-get install libeigen3-dev`
4. `sudo apt-get install python2.7-dev`
5. `sudo apt-get install python2.7`
6. `sudo apt-get install python-tk`
7. `sudo apt-get install python-numpy`
8. `sudo apt-get install python-matplotlib`
9. `sudo apt-get install eog`
10. `sudo apt-get install cmake`

2 Cloning the Repository

To clone the FWI repository using git,

```
git clone -o redmine https://git.alten.nl/parallelized-fwi.git
```

This will create a copy of the repository, in a folder named *parallelized-fwi*
Any branch as needed can then be checked out from inside the *parallelized-fwi* folder, e.g. the develop branch

```
git checkout develop
```

3 Build/Run

In this section the process to build the code is explained in two steps, namely downloading and installing Google Test and thereafter building the code and finally running it.

3.1 Install Google Test

First go back to your home directory, then make the googletest directory. Then download Google test from Github. Then execute cmake, make and make install commands. Finally set the gtest root to the path of the working directory `$/googletest/install`.

```
cd ..
mkdir googletest
cd googletest
git clone https://github.com/google/googletest source
mkdir build
cd build
cmake = -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=~/googletest/install/ ../source
```

```
make
make install
cd ..
cd install/
export GTEST_ROOT=$PWD
```

Another way of installing GTest

```
sudo apt-get install libgtest-dev
cd /usr/src/gtest
sudo cmake CMakeLists.txt
sudo make
sudo cp *.a /usr/lib
```

3.2 Build

To build the project, first create a folder titled *build* outside the *parallelized-fwi* folder. Afterwards the code is built and run.

```
NOTE: This folder should be exactly 1 level outside the parallelized-fwi folder.
mkdir Build
cd Build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX= ~/FWIInstall ..parallelized-fwi/
make install
```

The first flag in the cmake command above enforces that the release version of the code be built and the 2nd flag implies that all the executables of the program will be placed in a folder *FWIInstall* (executables are in *FWIInstall/bin*)

3.3 Run

First enter the *inputFiles* folder parallelized-fwi folder and copy its contents to a folder named *input* inside *FWIInstall* folder. Also, a folder named *output* is created to store all the output files of the program.

```
mkdir input output
cp parallelized-fwi/inputFiles/* input/
```

Now, the individual scripts for the preProcessing and the processing part can be run as shown below:

```
cd FWIInstall/bin
./FWI_PreProcess ../input ../output/ default
./FWI_Process ../input ../output/ default
```

In the both the above executables the 1st argument (i.e. ../input) provides the directory where the input card and the temple/skull chi values are located , the 2nd argument (i.e. ../output) provides the directory where all the output files from each application will be stored and the 3rd argument is the name of the input card (which is default in this case). Also note that the first 2 arguments are the locations relative to the executable position. The default input parameters for the code are provided in the input card i.e. default.in. User can create his/her own input card with a new name e.g. `newCard.in`.

For post-processing (i.e. generation of image using the estimated chi values and the residual plot), the python script `postProcessing.py` can be used. This script is located inside the *parallelized-fwi/pythonScripts* folder. This can be copied to the FWIInstall folder and then used as,

```
cp parallelized-fwi/pythonScipts/postProcessing.py FWIInstall/
python postProcessing.py output/
```

The folder where all the output files are located is provided as the argument for the python script. The pre-processing, processing and the post-processing can all be grouped together using the python wrapper `wrapper.py` located inside the *parallelized-fwi/pythonScripts* folder.
The postprocessed data can then be visualized using EOG from the output folder:

```
eog defaultResult.png
```

4 Unit Test

The unit test executable is stored in the *FWIInstall/bin* folder after the make install command. The following command can be used to run the test.

```
cd FWIInstall/bin
./unittest
```

5 Regression Test

This section details the comparison of a *default* run with the *fast* regression data. In order to run a regression test the following steps need to be taken. First, a *test* folder is created, then the contents of a regression data folder, the output folder, the input card (default.in) and the regression test python scripts are copied to this *test* folder. Then finally the python scripts can be run in succession.

```
mkdir test
cp parallelzed-fwi/tests/regression_data/fast/* test/
cp parallelzed-fwi/tests/pythonScripts/* test/
cp input/default.in test/
cd test
python regressionTestPreProcessing.py fast default
python regressionTestProcessing.py fast default
```

6 Troubleshooting

6.1 SSH Tunneling

In case the Ubuntu Virtual Machine is very slow and has long response times, try setting up an SSH connection between the **Ubuntu host** & **Windows guest**. This way, you can use Visual Studio Code for development with fast response times from Windows. This post explains how to set up such a connection.

1. In the **host** Ubuntu virtual machine, go to `Devices > Network > Network Settings > Advanced > Port Forwarding`

2. Add a new port forwarding rule with the following parameters:

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
ssh	TCP		3022		22

3. In the **host**, install an ssh server:
   `sudo apt-get install openssh-server`

4. In the **guest**, install Visual Studio Code with the Remote Development extension.

5. In the **guest**, open cmd and write:
   `ssh -p 3022 user@127.0.0.1`

6. Run Visual Studio Code & click on the Remote Development icon in the bottom-left corner.

7. From the drop-down menu, go to Remote-SSH: Open Configuration File. Select one, remember which one.

8. Add to this document the following lines:
   ```
   Host 127.0.0.1
   HostName 127.0.0.1
   Port 3022
   User user
   ```
   Where `user` is the account in the virtual machine, before the '@' sign.

9. Click the Remote Development button in the bottom-left corner again & select Remote-SSH: Connect To Host. Select the SSH configuration file you just modified.

10. Don't forget to provide the **host** password in an easily overlooked password field, located at the top of the editor.