BUILD-RUN-DOCUMENT

Saurabh Sharma, Hugo Solera Licona, Iñaki Martin Soroa, Noud van Herpen

November 18, 2019

This document shows the steps needed to install the prerequisite packages, clone, build, run, and do post-processing analysis for the FWI code.

1 Pre-requisites

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

```
sudo apt install git qt5-default libeigen3-dev eog cmake python3.7
sudo apt install python3.7-dev python3-tk python3-numpy python3-matplotlib python3-skimage
```

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

In this section, the process to build is explained. It is worth noticing that there is a `Build` and a `BuildAndRun` python scripts which does the build of the project automatically.

3.1 Install Google Test

First we need to install Google Test using the following commands:

```
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.

```
cd ~
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*)

4 Run

To run, first check out the *inputFiles* folder in the parallelized-fwi folder and copy the *default* folder to the *FWIInstall* folder. This can be done by issuing the following command:

```
cp -r ../parallelized-fwi/inputFiles/default/ ../FWIInstall
```

Also, make sure that there is a *bin* folder in *FWIInstall*, if there is not you can copy it from *Build/runtime/*. You can do this with the command:

```
cp -r ../Build/runtime/bin/ ../FWIInstall
```

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

```
cd ~/FWIInstall/bin/
./FWI_PreProcess ../default/
./FWI_UnifiedProcess ../default/ conjugateGradientInversion
```

For both executables we have to pass one argument which is the path to the case to be run (in this case *../default/*. It is important to notice that for the UnifiedProcess, a second argument is needed, this argument corresponds to the name of the inversion method you want to use. Note that the arguments are the locations relative to the executable position. The default input parameters are stored in the *default/* which is located in *~/parallelized-fwi/inputFiles/*. Users can create their own set of input cards by copying the default folder into a new folder, modifying the input cards and giving the new folder's location argument to both PreProcess and Process.

5 Post-Processing

For post-processing (i.e. generation of image using the estimated chi values and the residual plot), the python script `postProcessing-python3.py` can be used. Note that when using the `BuildAndRun` and `Run` scripts, the postprocessing is done automatically. The `postProcessing-python3.py` script is located inside the *parallelized-fwi/pythonScripts* folder. This can be copied to the FWIInstall folder and then executed using the following commands:

```
cp ~/parallelized-fwi/pythonScipts/postProcessing-python3.py ..
cd ..
python3 postProcessing-python3.py default/
```

The run case folder 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:

```
cd default/output/
eog defaultResult.png
```

6 Unit Test

Unit tests are written in one file per class. The file must be located in the `test` directory of the respective library. To make the unit tests available for Google Test, make sure `CMakeLists.txt` in `test` has the following line:

```
build_test(TARGET myClassTest SOURCE myClassTest.cpp LIBRARIES core_library)
```

In this case, myClassTest covers a class that is part of the core library.
After building, the test can be ran by writing in the terminal:

```
cd ../../bin/test
./myClassTest
```

7 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 the regression data folder, the default folder, the input cards and the regression test python scripts are copied to this *test* folder. Then finally the python scripts can be run in succession.

```
cd ..
mkdir test
cp -r ~/parallelized-fwi/tests/regression_data/fast/ test/
cp ~/parallelized-fwi/tests/testScripts/*.py test/
cp -r default/ test/
cd test/
python3 regressionTestPreProcessing_python3.py fast default
python3 regressionTestProcessing_python3.py fast default
```

8 Troubleshooting

8.1 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 toolbar (make sure full-screen view is disabled), 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`
   Where `user` is the account in the virtual machine, before the '@' sign, in the terminal.
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.