Real-ESSI Simulator

Executable Build and Install Procedures

Yuan Feng, Han Yang, Hexiang Wang, José Abell

and

Boris Jeremić

University of California, Davis, CA

Version: August 8, 2022, 13:27

http://real-essi.us/

This document is an excerpt from: http://sokocalo.engr.ucdavis.edu/~jeremic/LectureNotes/

please use google-chrome to view this PDF so that hyperlinks work

Contents

Chapter 1

Software Platform Build and Install Procedures

1993-1994-1996-1999-2003-2005-2007-2008-2009-2010-2011-2015-2019

(In collaboration with Dr. José Abell, Mr. Sumeet Kumar Sinha, Mr. Yuan Feng, Dr Han Yang and Dr Hexiang Wang)

1.1 Chapter Summary and Highlights

1.2 Introduction to the Real-ESSI Simulator Program

The Real-ESSI Simulator systems consists of the Real-ESSI Program, Real-ESSI Computer and Real-ESSI Notes. Alternative name for the Real-ESSI Simulator system is Real-ESSI Simulator system. The name Real-ESSI, is explained in section 201.2.6 on page 684.

1.3 Real-ESSI Simulator System Install

In adition to the Real-ESSI Program, Real-ESSI Simulator system consists of a pre-processing modules and post-processing modules. Installation of pre-processing modules is described in Chapter 207, on page 1189 of the main document, lecture notes (Jeremić et al., 1989-2022). Installation of post-processing modules is described in Chapter 208, on page 1255 of the main document, lecture notes (Jeremić et al., 1989-2022).

Both pre and post processing manuals are also available through the main Real-ESSI Simulator web site: http://real-essi.info/.

1.4 Build Procedures for the Real-ESSI Program and Modules

Note: This section describes build procedure for the Global Release 22.07 version of Real-ESSI.

These build procedures are meant for users that have access to Real-ESSI Program source code. Required operating system is Ubuntu 22.04 LTS (Jammy Jellyfish), unless otherwise specified. Building Real-ESSI on older versions of Ubuntu is no longer supported.

1.4.1 System Libraries Update/Upgrade

```
sudo apt update
sudo apt upgrade
sudo apt dist-upgrade
sudo apt autoremove
```

1.4.2 Install Build Dependencies

```
sudo apt install -y bison
sudo apt install -y build-essential
sudo apt install -y cmake
sudo apt install -y flex
sudo apt install -y git
sudo apt install -y mpich
```

```
sudo apt install -y ssh
sudo apt install -y valgrind
sudo apt install -y wget
sudo apt install -y zlib1g-dev
sudo apt install -y libboost-all-dev
sudo apt install -y libgmp3-dev
sudo apt install -y libhdf5-serial-dev
sudo apt install -y liblapack-dev
sudo apt install -y libmpfr-dev
sudo apt install -y libopenblas-dev
sudo apt install -y libopenmpi-dev
sudo apt install -y libpthread-workqueue-dev
sudo apt install -y libssl-dev
sudo apt install -y libtbb-dev
```

You may need to manually link the HDF5 libs to their proper names so that the compiler can find them. The HDF5 maybe in different versions. For example:

```
cd /usr/lib/x86_64-linux-gnu
sudo ln -s libhdf5_serial.so.103.3.0 libhdf5.so
sudo ln -s libhdf5_serial_cpp.so.103.3.0 libhdf5_cpp.so
```

If the libs `libhdf5.so` and `libhdf5_cpp.so` are already there, just move on.

1.4.3 Download Real-ESSI Source

Make a directory where all the sources will reside and go there:

```
cd
mkdir Real-ESSI
cd Real-ESSI
```

Obtain Real-ESSI sources from the github:

```
git clone git@github.com:BorisJeremic/Real-ESSI.git
```

Go to the Real-ESSI source directory:

```
cd Real-ESSI
```

Remember to 'git checkout' to the proper branch.

1.4.4 Download and Compile Real-ESSI Dependencies

Make directories for the dependencies:

```
mkdir -p ../RealESSI_Dependencies
mkdir -p ../RealESSI_Dependencies/include
mkdir -p ../RealESSI_Dependencies/lib
```

```
mkdir -p ../RealESSI_Dependencies/bin
mkdir -p ../RealESSI_Dependencies/SRC
```

Go to the directory, download and extract the sources of the dependencies:

```
cd ../RealESSI_Dependencies
wget ↩
   http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Dependencies_SRC.tar.gz
tar -xzvf ./Dependencies_SRC.tar.gz -C ./SRC --strip-components 1
```

Go to the Real-ESSI directory and compile the dependencies:

```
cd ../Real-ESSI
./build_libraries suitesparse
./build_libraries arpack
./build_libraries lapack
./build_libraries parmetis
./build_libraries petsc_itself
```

1.4.5 Configure, Build, and Install the Real-ESSI Program

Configure and build the sequential version of Real-ESSI:

```
mkdir build
cd build
cmake ..
make -j 10
cd ..
```

Configure and build the parallel version of Real-ESSI:

```
mkdir pbuild
cd pbuild
cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ ↩
   -DPROGRAMMING_MODE=PARALLEL ..
make -j 16
cd ..
```

Copy the Real-ESSI executables to system directory:

```
sudo cp build/essi /usr/local/bin/essi-sequential
sudo cp pbuild/essi /usr/local/bin/essi-parallel
```

1.4.6 Install Sublime Text and Real-ESSI Packages

Sublime Text (https://www.sublimetext.com/) is the recommended editor for Real-ESSI input files and pre-processing files. Install Sublime Text following the official installation steps, or using the following com-

mand:

```
wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | gpg ↩
   --dearmor | sudo tee /etc/apt/trusted.gpg.d/sublimehq-archive.gpg
echo "deb https://download.sublimetext.com/ apt/stable/" | sudo tee ↩
   /etc/apt/sources.list.d/sublime-text.list
sudo apt-get update
sudo apt-get install sublime-text
```

Open Sublime Text. Open the 'Tools' menu and select 'Install Package Control...'. Open the 'Preferences' menu, select 'Package Control', then select 'Package Control: Install Package'.

In the opened search bar, type the package name and click on the package to install it. Three packages should be installed:

FEI Syntax-n-Snippets, Real-ESSI syntax and auto completion plugin for [].fei files (input files for Real-ESSI program).

gmsh-Tools, syntax and autotext completion for Gmsh model development tools for Real-ESSI.

gmESSI-Tools, syntax and autotext completion for gmESSI model development tools for Real-ESSI.

1.4.7 Install HDFView

`HDFView` can be used to open Real-ESSI output files, which are in HDF5 format. Download the latest version of `HDFView` from https://support.hdfgroup.org/ftp/HDF5/releases/HDF-JAVA/. Click on the latest version, which is `hdfview-3.2.0` as of June 2022. Go to `bin/`, click `HDFView-3.2.0-ubuntu2004_64.tar.gz`, and save the file in your `./Downloads/` directory. Then extract and install HDFView:

```
cd
tar -xvf ./Downloads/HDFView-3.2.0-ubuntu2004_64.tar.gz -C ./Downloads
sudo apt install -y ./Downloads/hdfview_3.2.0-1_amd64.deb
sudo ln -s /opt/hdfview/bin/HDFView /usr/local/bin/hdfview
```

Now you can use `HDFView` from a terminal. To be able to use `HDFView` when you click on a Real-ESSI output file, do the following additional steps. First open the file using the following command:

```
sudo gedit /usr/share/applications/hdfview-HDFView.desktop
```

Find the line:

```
Exec=/opt/hdfview/bin/HDFView
```

Replace it with:

```
Exec=/opt/hdfview/bin/HDFView %F
```

Save the file and close it.

Go to a Real-ESSI output file, which should have the suffix 'h5.feioutput'. Right click on the file and select 'Open with Other Application'. Click 'View All Applications' and choose `HDFView` from the list. Note that you only need to do this once. Next time when you click on a Real-ESSI output file, it will be opened automatically using `HDFView`.

1.4.8 Compile ParaView and PVESSIReader for Post-Processing

Install the build dependencies for ParaView:

```
sudo apt install -y libgl1-mesa-dev
sudo apt install -y libxt-dev
sudo apt install -y libqt5x11extras5-dev
sudo apt install -y libqt5help5
sudo apt install -y qttools5-dev
sudo apt install -y qtxmlpatterns5-dev-tools
sudo apt install -y libqt5svg5-dev
sudo apt install -y libtbb-dev
sudo apt install -y python3-dev
sudo apt install -y python3-numpy
sudo apt install -y ninja-build
```

Go to the directory where you want to install ParaView. Suggested location is the parent directory of the Real-ESSI source. If you are continuing from the previous subsection, do the following:

```
cd ..
```

Download the ParaView source from GitHub:

```
git clone --recursive https://gitlab.kitware.com/paraview/paraview.git
```

Make the build directory:

```
cd paraview
mkdir paraview_build
```

Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `CMakeLists.txt` in the ParaView source directory. Find "set(paraview_default_plugins" and add "PVESSIReader" to the end of the list of plugins. Download the PVESSIReader source from GitHub:

```
cd Plugins
git clone git@github.com:BorisJeremic/Real-ESSI-pvESSI.git
mv Real-ESSI-pvESSI PVESSIReader
```

Go to the build directory and compile ParaView:

```
cmake -GNinja -DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON ↩
   -DVTK_SMP_IMPLEMENTATION_TYPE=TBB -DCMAKE_BUILD_TYPE=Release ..
ninja
```

Copy the ParaView executable to system directory:

```
sudo cp bin/paraview /usr/local/bin/paraview
```

Start `ParaView` and click 'Tools' → 'Manage Plugins...'. Click 'Load New...' and find the plugin named 'PVESSIReader.so' under directory `paraview/lib/paraview-5.10/plugins/PVESSIReader/`. Also check the box 'Auto Load' then close `ParaView`.

The procedures described in this subsection are based on the official build instruction of ParaView (https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md).

1.5 Build Real-ESSI Debian Package

Note: This section describes Debian package build procedure for the Global Release 22.07 version of Real-ESSI.

Starting from the Global Release 22.07 version, the Real-ESSI Simulator system is distributed as a Debian package for Linux users. This section documents the build procedure of a Real-ESSI Debian package. Note that the steps described here are for building a "basic" or "quick" stand-alone Debian package containing the already-compiled Real-ESSI program and modulus. This is different from building a Debian package containing the program sources. For more information see:

- https://wiki.debian.org/Packaging
- https://www.internalpointers.com/post/build-binary-deb-package-practical-guide
- https://ubuntuforums.org/showthread.php?t=910717

1.5.1 Build the Real-ESSI Program and Modules

Before starting to build the Debian package, you should have finalized building the Real-ESSI program and modules from source. To build Real-ESSI from source, please follow the build procedure described in section 1.4.

1.5.2 Build the Debian Package

Package Name

Standard Debian notation is all lowercase in the following format:

```
<project>_<major version>.<minor version>-<package revision>
```

The current Real-ESSI Debian package has the name:

```
real-essi_22.07-1_amd64
```

Note that the version names will change and be consistent with the version of Real-ESSI program, as decribed at http://real-essi.info/.

Create Directory

Create a directory to make your package in. The name should be the same as the package name.

```
mkdir real-essi_22.07-1_amd64
```

Create Internal Structure

Pretend that the packaging directory is actually the root of the file system. Put the files of your program where they would be installed on a linux system.

```
mkdir real-essi_22.07-1_amd64/usr
mkdir real-essi_22.07-1_amd64/usr/local
mkdir real-essi_22.07-1_amd64/usr/local/bin
mkdir real-essi_22.07-1_amd64/usr/lib
mkdir real-essi_22.07-1_amd64/usr/lib/x86_64-linux-gnu
mkdir real-essi_22.07-1_amd64/opt
mkdir real-essi_22.07-1_amd64/opt/gmESSI
mkdir real-essi_22.07-1_amd64/opt/paraview
```

Copy Files

Copy the files to the packaging directory. Note that you should use your own directory paths.... For example:

```
cp /home/han/Real-ESSI/Real-ESSI/build/essi ↩
   real-essi_22.07-1_amd64/usr/local/bin/essi-sequential
cp /home/han/Real-ESSI/Real-ESSI/pbuild/essi ↩
   real-essi_22.07-1_amd64/usr/local/bin/essi-parallel
cp -r /home/han/Real-ESSI/gmESSI/build ↩
   real-essi_22.07-1_amd64/opt/gmESSI
cp /home/han/Real-ESSI/gmESSI/lib/* ↩
   real-essi_22.07-1_amd64/usr/lib/x86_64-linux-gnu/
cp -r /home/han/Real-ESSI/paraview/paraview_build/bin ↩
   real-essi_22.07-1_amd64/opt/paraview/bin
cp -r /home/han/Real-ESSI/paraview/paraview_build/lib ↩
   real-essi_22.07-1_amd64/opt/paraview/lib
```

```
cp -r /home/han/Real-ESSI/paraview/paraview_build/Plugins ↩
   real-essi_22.07-1_amd64/opt/paraview/Plugins
cp -r /home/han/Real-ESSI/paraview/paraview_build/share ↩
   real-essi_22.07-1_amd64/opt/paraview/share
```

Create the `control` File

Now create a special metadata file that is used by the package manager to install program. The control file lives inside the `DEBIAN` directory. Mind the uppercase: a similar directory named `debian` (lowecase) is used to store source code for the so-called source packages. This tutorial is about binary packages, so we don't need source code.

Create the empty `control` file:

```
mkdir real-essi_22.07-1_amd64/DEBIAN
touch real-essi_22.07-1_amd64/DEBIAN/control
```

Open the file previously created with text editor of your choice. The `control` file is just a list of data fields.

```
Package: real-essi
Version: 22.07
Architecture: amd64
Authors: Han Yang <hhhyang@ucdavis.edu>, Boris Jeremic ↩
   <jeremic@ucdavis.edu>
Maintainer: Han Yang <hhhyang@ucdavis.edu>
Depends: libboost-all-dev, libhdf5-dev, libtbb-dev, libssl-dev, ↩
   libopenmpi-dev, mpich, libgl1-mesa-dev, libxt-dev, ↩
   libqt5x11extras5-dev, libqt5help5, qttools5-dev, ↩
   qtxmlpatterns5-dev-tools, libqt5svg5-dev, libtbb-dev, python3-dev, ↩
   python3-scipy, python3-numpy, python3-matplotlib, python3-pip, ↩
   python3-pygments, liboctave-dev, python2.7-dev, gmsh
Section: misc
Priority: optional
Provides: real-essi
Description: The Real-ESSI Simulator.
 The Real-ESSI Simulator (Realistic Modeling and Simulation
 of Earthquakes, and/or Soils, and/or Structures and their
 Interaction) is a software, hardware and documentation
 system for high performance, sequential or parallel, time
 domain, linear or nonlinear, elastic and inelastic,
 deterministic or probabilistic, finite element modeling and
 simulation of
  - statics and dynamics of soil,
  - statics and dynamics of rock,
  - statics and dynamics of structures,
  - statics of and dynamics of soil-structure systems,
  - dynamics of earthquakes, and
  - dynamic earthquake-soil-structure interaction.
```

```
Homepage: http://real-essi.info
```

Create the Post-Installation and Post-Remove Files

```
touch real-essi_22.07-1_amd64/DEBIAN/postinst
touch real-essi_22.07-1_amd64/DEBIAN/postrm
```

The postinst file/script is executed after a successful installation of the Debian package. This script looks like this:

```
#!/bin/sh
# postinst script for real-essi

set -e

case "$1" in
    configure)
ln -s /opt/gmESSI/build/bin/gmessy /usr/local/bin/
ln -s /opt/paraview/bin/paraview /usr/local/bin/

update-alternatives --install /usr/bin/python python ↩
   /usr/bin/python2.7 1
pip install h5py
    ;;

    abort-upgrade|abort-remove|abort-deconfigure)
    ;;

    *)
        echo "postinst called with unknown argument \`$1'" >&2
        exit 1
    ;;
esac

exit 0
```

The postrm file/script is executed after a successful removal of the Debian package. This script looks like this:

```
#!/bin/sh
# postrm script for real-essi

set -e

case "$1" in
    ↩
   purge|remove|upgrade|failed-upgrade|abort-install|abort-upgrade|disappear)
rm /usr/local/bin/gmessy
rm /usr/local/bin/paraview
```

```
    ;;

    *)
        echo "postrm called with unknown argument \`$1'" >&2
        exit 1
    ;;
esac

exit 0
```

Build the Package

Now you just need to build the package:

```
dpkg-deb --build --root-owner-group real-essi_22.07-1_amd64
```

The `--root-owner-group` flag makes all deb package content owned by the root user, which is the standard way to go. Without such flag, all files and folders would be owned by your user, which might not exist in the system the deb package would be installed to.

The command above will generate a nice `.deb` file alongside the working directory or print an error if something is wrong or missing inside the package. If the operation is successful you have created debian package ready for distribution.

1.6 Quick Build Procedures for Sequential and Parallel Versions of the Real-ESSI Program

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

These build procedures are meant for users that have access to Real-ESSI Program source code. Procedures assume Ubuntu version 16.04, or 18.04.

1.6.1 Update Compiler for Ubuntu 16.04, while 18.04 and Later Already have the Latest Compilers

Ubuntu 16.04 does not have the gcc-7 and thus this new version of a compiler needs to be installed needs to be installed.

```
1 sudo apt-get install build-essential
2 sudo add-apt-repository ppa:ubuntu-toolchain-r/test
3 sudo apt-get update
4 sudo apt-get install gcc-7 g++-7 cpp-7 gfortran-7 cmake
```

To change the default version of compilers, that executes with a call to gcc command, we have to configure the alternatives in the system. First remove any previous configuration with:

```
sudo update-alternatives --remove-all gcc
sudo update-alternatives --remove-all g++
sudo update-alternatives --remove-all cpp
sudo update-alternatives --remove-all gfortran
```

This might produce an error: *update-alternatives: error: no alternatives for gcc* if no previous alternatives were configured. Pay no attention to this error and continue. Alternatives will be configured according to the versions available on the computer. Each alternative will have a priority associated with it. When a link group is in automatic mode, the alternatives pointed to by members of the group will be those which have the highest priority. To configure alternatives, execute:

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 90 ↩
   --slave /usr/bin/g++ g++ /usr/bin/g++-7 --slave /usr/bin/gfortran ↩
   gfortran /usr/bin/gfortran-7 --slave /usr/bin/cpp cpp /usr/bin/cpp-7
```

It does help to verify that proper compiler version is installed:

```
gcc --version
g++ --version
cpp --version
gfortran --version
```

1.6.2 Sequential Real-ESSI Build

Libraries Required for Building Dependencies for Ubuntu 16.04 and 18.04.

```
sudo apt-get install -y cmake

sudo apt-get install build-essential
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install libboost-all-dev
sudo apt-get install libtbb-dev
sudo apt-get install valgrind
sudo apt-get install libopenblas-dev
sudo apt-get install liblapack-dev
sudo apt-get install libpthread-workqueue-dev
sudo apt-get install zlib1g-dev
sudo apt-get install libssl-dev
sudo apt-get install mpich
sudo apt-get install libopenmpi-dev
```

It is noted that Real-ESSI requires libboost version 1.70 or above. For ubuntu 16.04, the system default

libboost from *sudo apt-get install* might not meet this requirement.

Check the version of libboost by running:

```
cat /usr/include/boost/version.hpp | grep BOOST_LIB_VERSION
```

If the output version number is lower than 1.70. A newer version that meets the above requirement can be manually installed by running:

```
sudo rm -rf /usr/include/boost
wget ↩
   https://dl.bintray.com/boostorg/release/1.70.0/source/boost_1_70_0.tar.gz
tar -xzvf boost_*.tar.gz
cd boost_*
sudo cp -r boost /usr/include/
```

Libraries Required for Building Utilities for Ubuntu 16.04 and 18.04 and for Later Versions

```
sudo apt-get install liboctave-dev
sudo apt-get install libhdf5-serial-dev
sudo apt-get install hdf5-tools
sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial.so ↩
   /usr/liblibhdf5.so
sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial_hl.so ↩
   /usr/liblibhdf5_hl.so
sudo apt-get install libphonon-dev
sudo apt-get install libphonon4
sudo apt-get install qt4-dev-tools
sudo apt-get install qt4-qmake
sudo apt-get install libxt-dev
sudo apt-get install libqt4-opengl-dev
sudo apt-get install mesa-common-dev
sudo apt-get install python-dev
sudo apt-get install python-h5py
sudo apt-get install python-matplotlib
sudo apt-get install python-scipy
```

Obtain Real-ESSI Sources

Make a directory where all the sources will reside and go there:

```
mkdir RealESSI_ROOT/
cd    RealESSI_ROOT/
```

Obtain Real-ESSI sources from the github:

```
mkdir RealESSI_ROOT
git reset --hard
```

```
3 git pull
```

or alternatively:

```
1 mkdir RealESSI_ROOT
2 #
3 # using curly brackets to help in checking scripts, that rely on these
4 # brackets being available around URL
5 #
6 git clone {https://github.com/BorisJeremic/Real-ESSI.git}  # Need ↩
   permission from Boris Jeremic for Real-ESSI on github
```

if you have access to the Real-ESSI program archived sources, copy them here and unpack the archive (replace _archive_name_.tgz with the actual archive name, for example _Real-ESSI_Complete.03_Feb_2019_15h_16m_22s_

```
1 tar -xvzf _archive_name_.tgz
```

or, for the example above:

```
1 tar -xvzf _Real-ESSI_Complete.03_Feb_2019_15h_16m_22s__Sunday.tgz
```

Go to the Real-ESSI source directory:

```
1 cd  Real-ESSI/
```

Download dependencies:

```
1  mkdir -p ../RealESSI_Dependencies
2  mkdir -p ../RealESSI_Dependencies/include
3  mkdir -p ../RealESSI_Dependencies/lib
4  mkdir -p ../RealESSI_Dependencies/bin
5  mkdir -p ../RealESSI_Dependencies/SRC
6  cd ../RealESSI_Dependencies
7  #
8  rm -rf Dependencies_SRC.tar.gz
9  #
10 #
11 # using curly brackets to help in checking scripts, that rely on these
12 # brackets being available around URL
13 #
14 wget ↩
      {http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Dependencies_SRC.tar.gz}
15 #
16 tar -xzvf ./Dependencies_SRC.tar.gz -C ./SRC --strip-components 1
17 #
18 cd ../Real-ESSI
```

Then, download utilities:

```
1 mkdir -p ../RealESSI_Utilities
```

```
mkdir -p ../RealESSI_Utilities/include
mkdir -p ../RealESSI_Utilities/lib
mkdir -p ../RealESSI_Utilities/bin
mkdir -p ../RealESSI_Utilities/SRC
cd ../RealESSI_Utilities
#
rm -rf Utilities_SRC.tar.gz
#
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget ↩
   {http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Utilities_SRC.tar.gz}

#
tar -xzvf ./Utilities_SRC.tar.gz -C ./SRC --strip-components 1
#
cd ../Real-ESSI
```

Compile Real-ESSI Dependency Libraries

Start building dependency libraries, if needed:

```
time ./build_libraries suitesparse
time ./build_libraries arpack
time ./build_libraries hdf5_sequential
time ./build_libraries tbb
time ./build_libraries lapack
time ./build_libraries parmetis
```

Check that all prerequisite libraries are built

```
time ./build_libraries check_sequential
```

Compile and link Real-ESSI Program

Create directories for the main Real-ESSI build program:

```
mkdir bin
mkdir lib
rm -f -r build_sequential
mkdir build_sequential
cd build_sequential
```

Build and install the executable, using 16 CPUs in this case. Of course, if you have more CPUs available, you can use most of them.

```
time cmake ..
time make -j 16
make install
```

Rename essi to essi.sequential just so to distinguish it from the parallel executable:

```
cd ../bin
cp essi essi.sequential
```

Finally, install essi.sequential in system binary directory so that others can use it:

```
sudo rm /usr/bin/essi /usr/bin/essi.sequential
sudo cp essi.sequential  /usr/bin/essi.sequential
sudo chmod a+x /usr/bin/essi.sequential
```

1.6.3 Parallel Real-ESSI Build

Libraries Required for Building Dependencies for Ubuntu 16.04 and 18.04 and for Later Versions

```
sudo apt-get install -y cmake

sudo apt-get install build-essential
sudo apt-get install flex bison
sudo apt-get install libboost-all-dev
sudo apt-get install libtbb-dev
sudo apt-get install valgrind
sudo apt-get install libopenblas-dev
sudo apt-get install liblapack-dev
sudo apt-get install libpthread-workqueue-dev
sudo apt-get install zlib1g-dev
sudo apt-get install libssl-dev
sudo apt-get install mpich
sudo apt-get install libopenmpi-dev
```

It is noted that Real-ESSI requires libboost version 1.70 or above. For ubuntu 16.04, the system default libboost from *sudo apt-get install* might not meet this requirement.

Check the version of libboost by running:

```
cat /usr/include/boost/version.hpp | grep BOOST_LIB_VERSION
```

If the output version number is lower than 1.70. A newer version that meets the above requirement can be manually installed by running:

```
sudo rm -rf /usr/include/boost
wget ↩
   https://dl.bintray.com/boostorg/release/1.70.0/source/boost_1_70_0.tar.gz
tar -xzvf boost_*.tar.gz
```

```
cd boost_*
sudo cp -r boost /usr/include/
```

Libraries Required for Building Utilities for Ubuntu 16.04 and 18.04 and for Later Versions

```
sudo apt-get install liboctave-dev
sudo apt-get install libhdf5-serial-dev
sudo apt-get install hdf5-tools
sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial.so ↩
   /usr/liblibhdf5.so
sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial_hl.so ↩
   /usr/liblibhdf5_hl.so
sudo apt-get install libphonon-dev
sudo apt-get install libphonon4
sudo apt-get install qt4-dev-tools
sudo apt-get install qt4-qmake
sudo apt-get install libxt-dev
sudo apt-get install libqt4-opengl-dev
sudo apt-get install mesa-common-dev
sudo apt-get install python-dev
sudo apt-get install python-h5py
sudo apt-get install python-matplotlib
sudo apt-get install python-scipy
```

Obtain Real-ESSI Sources

Make a directory where all the sources will reside and go there:

```
mkdir RealESSI_ROOT/
cd    RealESSI_ROOT/
```

Copy Real-ESSI program archive to that location and unpack the archive (replace `_archive_name_.tgz` with the actual archive name, for example `_Real-ESSI_Complete.03_Feb_2019_15h_16m_22s__Sunday.tgz`):

```
tar -xvzf _archive_name_.tgz
```

or, for the example above:

```
tar -xvzf _Real-ESSI_Complete.03_Feb_2019_15h_16m_22s__Sunday.tgz
```

Go to the Real-ESSI source directory:

```
cd  Real-ESSI/
```

Download dependencies:

```
mkdir -p ../RealESSI_Dependencies
```

```
mkdir -p ../RealESSI_Dependencies/include
mkdir -p ../RealESSI_Dependencies/lib
mkdir -p ../RealESSI_Dependencies/bin
mkdir -p ../RealESSI_Dependencies/SRC
cd ../RealESSI_Dependencies
#
rm -rf Dependencies_SRC.tar.gz
#
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget ↩
   {http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Dependencies_SRC.tar.gz}
#
tar -xzvf ./Dependencies_SRC.tar.gz -C ./SRC --strip-components 1
#
cd ../Real-ESSI
```

Then, download utilities:

```
mkdir -p ../RealESSI_Utilities
mkdir -p ../RealESSI_Utilities/include
mkdir -p ../RealESSI_Utilities/lib
mkdir -p ../RealESSI_Utilities/bin
mkdir -p ../RealESSI_Utilities/SRC
cd ../RealESSI_Utilities
#
rm -rf Utilities_SRC.tar.gz
#
#
# using curly brackets to help in checking scripts, that rely on these
# brackets being available around URL
#
wget ↩
   {http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Utilities_SRC.tar.gz}

#
tar -xzvf ./Utilities_SRC.tar.gz -C ./SRC --strip-components 1
#
cd ../Real-ESSI
```

Compile Real-ESSI Dependency Libraries

Start building dependency libraries:

```
 time ./build_libraries petsc
 time ./build_libraries initialize
 time ./build_libraries hdf5_sequential
 time ./build_libraries lapack
 time ./build_libraries tbb
```

```
6 time ./build_libraries parmetis
```

Check that all prerequisite libraries are built

```
1 time ./build_libraries check_parallel
```

Compile and link Real-ESSI Program

Create directories for the main Real-ESSI build program:

```
1 mkdir bin
2 mkdir lib
3 rm -f -r build_parallel
4 mkdir build_parallel
5 cd build_parallel
```

Build and install the executable, using 16 CPUs in this case. Of course, if you have more CPUs available, you can use most of them.

```
1 time cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ ↩
     -DPROGRAMMING_MODE=PARALLEL ..
2 time make -j 16
3 make install
```

Rename `essi` to `essi.sequential` just so to distinguish it from the parallel executable:

```
1 cd ../bin
2 cp essi essi.parallel
```

Finally, install `essi.sequential` in system binary directory so that others can use it:

```
1 sudo rm /usr/bin/essi /usr/bin/essi.parallel
2 sudo cp essi.parallel   /usr/bin/essi.parallel
3 sudo chmod a+x /usr/bin/essi.parallel
```

1.7 Build Procedures for Sequential and Parallel Versions of the Real-ESSI Program

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

1.7.1 Libraries and Application Build Process

The Real-ESSI Program was designed and developed (primarily) for parallel, high performance computations, while a (secondary) sequential version is also available. Both version (same source code, small changes during

compilation process, and different main application source code and compilation) are designed and developed (primarily) for the Real-ESSI Computer (or similar, distributed memory parallel computers).

Building Real-ESSI simulator requires some basic tools be present in the target system such as C++ (must support C++11 standard) and Fortran compilers, as well as some widely available libraries (BoostC++ (Abrahams and Gurtovoy, 2005; Ramey, 2005), and an MPI-2 implementation). We provide further dependencies needed by the program which must be compiled separately and are not version controlled.

Described below in some detail, are procedures that are necessary for compilation of Real-ESSI program/application. Since we use Ubuntu GNU/Linux system, our installation procedures are using Debian/Ubuntu syntax for installing libraries or additional sources, for example `apt-get install ...`. For Red-Hat based systems, one would use `yum install ...` or similar...

The build procedures for Real-ESSI and its dependencies and utilities are available for all the major Linux Ubuntu platforms. The sections below will go in depth about installing libraries and building dependencies, Real-ESSI and its utilities.

1.7.2 Installing Libraries

Currently the `gcc-7` compiler (or a version above) is recommended, although any standard C++ compiler can be used. The requirement is a compiler that fully supports the C++11 standard. Please note that a compiler other than GNU gcc or LLVM clang would require modification of makefiles and has not been tested. Some dependencies rely on the tool `cmake` for their build system. A development version of BOOST library (`http://www.boost.org/`) and TBB (Threading Building Blocks) library (`https://www.threadingbuildingblocks.org/`) also need to be installed (see below).

Previous sections provide up to date instructions for building Real-ESSI sequential and parallel version. Sections below provide some additional information that might be useful.

1.7.3 Obtaining Real-ESSI and Source organization

Real-ESSI program sources are available to developers/collaborators under a restrictive open source license. Real-ESSI sources can be cloned from github for the developers who have direct access to the code. The source code is placed in the Real-ESSI folder. For others, please contact Boris Jeremić for Real-ESSI distribution options.

```
1 mkdir RealESSI_ROOT
2 #
3 # using curly brackets to help in checking scripts, that rely on these
4 # brackets being available around URL
5 #
6 git clone {https://github.com/BorisJeremic/Real-ESSI.git}  # Need ↩
    permission from Boris Jeremic for Real-ESSI on github
```

If you are not interested in sources and just want to run the program, you will need to have linux installed with appropriate libraries available (as detailed below). Installation of the Real-ESSI Program is most optimal if remote users have available an up to date linux system (sequential and/or parallel) and if arrangements can be made for a temporary, simple/regular user, remote login (through a secure shell) for Prof. Jeremic. This will allow us to compile and install all the necessary libraries and the executable. Distribution of executable without remote login is available also, however in this case we distribute an un-optimized (slow), version of the Real-ESSI Program.

Real-ESSI distribution must be organized in a folder structure like follows:

```
1 RealESSI_ROOT
2     |
3     + Real-ESSI
4     |
5     + RealESSI_Dependencies
6     |
7     + RealESSI_Utilities
```

Where `RealESSI_ROOT` can be any directory within the filesystem, `Real-ESSI` contains the (version controlled) source of the main code, `RealESSI_Dependencies` contains the sources for the software Real-ESSI depends on (distributed in a tar-ball) and `RealESSI_Utilities` contains sources of the utilities that help make Real-ESSI really easy.

1.7.4 Real-ESSI Dependencies Build Process

Real-ESSI Simulator build process depends on three software sources:

1. Common software available through the Linux distribution (available through the distribution package manager) which we already installed, using guidance in the above section
2. Software libraries that Real-ESSI depends on (provided via a tar-ball), described in this section
3. Real-ESSI Simulator source code

The current release of Real-ESSI includes the following dependencies.

- **ATLAS 3.10.3** Provides an efficient BLAS implementation.
- **HDF5 1.8.17** For Real-ESSI output
- **LAPACK 3.6.1** Standard linear algebra suite
- **ParMETIS 4.0.3** Software for graph partitioning used in parallel Real-ESSI

- **PETSc 3.7.3** High performance, parallel suite of system of linear equations solvers used in parallel Real-ESSI.
- **SuiteSparse 4.5.3** Provides interfaces into system of equations linear solvers used in sequential Real-ESSI.
- **Blas 3.6.0** Basic Linear Algebra Subprograms providing standard building blocks for performing basic vector and matrix operations.
- **Cmake 3.7.0-rc2** Provides latest tools designed to build cmake and makefiles

Real-ESSI source comes up with **build_libraries** script that can build all the necessary dependencies required by Real-ESSI for both sequential and parallel case. The script is located in the **RealESSI_ROOT/Real-ESSI** folder. The build_libraries is a bash script which calls a makefile that has targets defined to build the required dependencies.

The first step is to download all the sources of dependencies that needs to be build. In addition a directory where Real-ESSI sources will be placed is to be created. To do this, one has to run

```
cd Real-ESSI
./build_libraries download
```

This would download all the libraries in *tar.gz* format and would place them in **/SRC** of *RealESSI_Dependencies* directory. The script accepts targets that can be used to build a particular library or all libraries at once. The available options to the scripts can be found by running the target help as shown below.

```
./build_libraries help

#--------- result from the terminal -------------------
#
#Miscellaneous:
#  list_dependencies              Lists all the available dependencies version from SRC folder
#  list_build_dependencies        Lists all the dependencies library already build in lib folder
#  help                           Show this help.
#  download                       Downloads the Dependencies Sources
#
#Parallel:
#  parallel                       Builds all the necessary libraries for parallel Real-ESSI
#  hdf5_parallel                  Builds parallel hdf5
#  petsc                          Builds petsc
#  clean_parallel                 Cleans parallel libraries
#
#Sequential:
#  sequential                     Builds all the necessary libraries for sequential Real-ESSI
```

```
#  parmetis                         Builds parmetis and metis
#  suitesparse                      Builds suitesparse
#  arpack                           Builds arpack
#  hdf5_sequential                  Builds sequential hdf5
#  lapack                           Builds lapack
#  atlas                            Builds tuned LAPACK and BLAS
#  clean_sequential                 Cleans sequential libraries
#
#Default:
#  all                              Builds all the necessary libraries for both sequential and #parallel Real-ESSI
#  all_libs                         Builds all the necessary libraries for both sequential and #parallel Real-ESSI
#  clean_all                        Cleans everything
#
#Check:
#  check_sequential                 Checks if all sequential libraries are build
#  check_parallel                   Checks if all parallel libraries are build
#
#Clean:
#  clean_parmetis                   Clean parmetis and metis
#  clean_suitesparse                Clean suitesparse libraries
#  clean_arpack                     Clean arpack libraries
#  clean_hdf5_sequential            Cleans hdf5_sequential libraries
#  clean_lapack                     Cleans lapack
#  clean_hdf5_parallel              Cleans hdf5_parallel libraries
#  clean_petsc                      Cleans petsc
```

Advanced users play with the various targets and their uses. Simple action is to build libraries one at a time, for a sequential version:

```
time ./build_libraries suitesparse
time ./build_libraries arpack
time ./build_libraries hdf5_sequential
time ./build_libraries tbb
time ./build_libraries lapack
time ./build_libraries parmetis
```

or all at once for sequential version (it helps if they are build one at a time, to observe any potential compilation messages):

```
./build_libraries sequential
```

Similar procedures is used for building parallel libraries:

```
./build_libraries parallel
```

The user can also check whether all the libraries are successfully built:

```
./build_libraries check_sequential
./build_libraries check_parallel
```

Before building libraries one needs to follow procedure to tune ATLAS based on local system configuration. The tuned atlas has advantage over the regular atlas with an almost 10-fold increase in speed. If Real-ESSI program is used for testing purpose only, one does not need to follow this (rather involved) step (of tuning of ATLAS), rely on the default installation of ATLAS and jump directly to section 1.7.5 on page 33.

[Optional Atlas Tuning :: Recommended for High Performance]

Real-ESSI is meant to take maximum advantage of your platform's hardware capabilities to provide a high-performance finite element implementation for real earthquake-soil-structure interaction simulations. An efficient implementation of BLAS (Basic Linear Algebra Subprograms) is crucial to attain this goal. Therefore, we have opted to use ATLAS (Automatically Tuned Linear Algebra Software) to provide a working, portable, high-performance BLAS and LAPACK as default for Real-ESSI.

A side-effect of this election is that end-users will have to go through the ATLAS auto-tuning process for their system in order to get performance out of Real-ESSI. If you want to skip this step, and provide your own version of BLAS and LAPACK, please see next section. The most import step for ATLAS auto-tuning process is

Disabling Auto CPU Frequency Scaling Auto scaling of CPU frequencies has to be turned OFF for ATLAS to be properly configured. Modern CPUs can scale their clock frequencies up and down, in order to respond to computational load and also save energy. This scaling needs to be turned off so that ATLAS can properly evaluate CPU performance, and tune its performance using all the features of a CPU.

This scaling is found on all current modern CPUs. So in order to turn off CPU frequency scaling you will need to install this utility:

```
sudo apt-get install cpufrequtils
```

Then edit the following file (if it doesn't exist, create it):

```
sudo nano /etc/default/cpufrequtils
```

and add the following line to it:

```
GOVERNOR="performance"
```

then save and exit. You also need to disable ondemand daemon, otherwise after you reboot the settings will be overwritten.

```
sudo update-rc.d ondemand disable
```

Now you can reboot.

You will be able to restore the old ondemand daemon (which will control frequencies of your CPU(s), just like it did before, and have your CPU save energy when it is not needed), by doing

```
sudo /etc/init.d/cpufrequtils restart
```

This will enable temporarily the "performance" governor, until next reboot.

After setup is done, check:

```
cpufreq-info
```

and make sure that the current CPU-frequency for all CPUs is the hardware maximum. If not you can do:

```
sudo /etc/init.d/cpufrequtils restart
```

Then re-check for the hardware maximum.

For more info see http://askubuntu.com/questions/523640/.

If you now run a command

```
cpufreq-info
```

and you note that your CPU(s) (are) is at the maximum frequency already, you can probably skip steps below and jump to page 31 where we continue to describe ATLAS build process.

A Note on Intel CPUs and Linux Kernel $\geq$ 3.9: For new Intel processors running the Linux Kernel 3.9 or above the default driver for the CPU frequency scaling is `intel_pstate`. This has to be disabled via the Linux kernel command:

```
intel_pstate=disable
```

so that the driver `acpi-cpufreq` takes over, allowing constant CPU frequency at the max frequency. For example, on Ubuntu 14.04 one edits `/etc/default/grub` and change the line

```
GRUB_CMDLINE_LINUX_DEFAULT="nomodeset  quiet splash"
```

to

```
GRUB_CMDLINE_LINUX_DEFAULT="nomodeset quiet splash intel_pstate=disable"
```

The new file `/etc/default/grub` should look something like this (this is example from one of our computers, your file will look similar but not the same, except for the addition of `intel_pstate=disable` to that line):

```
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.
# For full documentation of the options in this file, see:
#   info -f grub -n 'Simple configuration'

GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="nomodeset quiet splash intel_pstate=disable"
GRUB_CMDLINE_LINUX=""

# Uncomment to enable BadRAM filtering, modify to suit your needs
# This works with Linux (no patch required) and with any kernel that obtains
# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
#GRUB_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_RECOVERY="true"

# Uncomment to get a beep at grub start
#GRUB_INIT_TUNE="480 440 1"
```

Then re-configure grub with

```
sudo grub-mkconfig -o /boot/grub/grub.cfg
```

Reboot and confirm that this worked by running

```
cpufreq-info
```

and look at the driver information that should read acpi-cpufreq rather than intel_pstate.

Separating "Real" CPUs from Hyperthreaded CPUs This is probably not of any interest for regular users, so please skip to the next section.

If you want to configure ATLAS to use multiple cores and hyperthreading[1] than the above configure options would look like this:

```
../configure -b 64 -D c -DPentiumCPS=2000 ↩
  --prefix=$RealESSI_Dependencies_PATH  ↩
  --with-netlib-lapack-tarfile=$RealESSI_Dependencies_PATH/lapack-3.5.0.tgz ↩
  --force-tids="4 0 1 2 3"
```

All the options, as described above are the same with an addition of

Explanation of options:

- `--force-tids="4 0 1 2 3"` tells ATLAS to only use those core whose ids are given (first number is the number of cores to use).

For nagoyqqatsi computer:

```
../configure -b 64 -D c -DPentiumCPS=2200 ↩
  --prefix=$RealESSI_Dependencies_PATH  ↩
  --with-netlib-lapack-tarfile=$RealESSI_Dependencies_PATH/lapack-3.5.0.tgz ↩
  --force-tids="8 0 1 2 3 4 5 6 7"
```

For an Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz:

```
../configure -b 64 -D c -DPentiumCPS=4000 ↩
  --prefix=$RealESSI_Dependencies_PATH  ↩
  --with-netlib-lapack-tarfile=$RealESSI_Dependencies_PATH/lapack-3.5.0.tgz ↩
  --force-tids="4 0 1 2 3 "
```

Why is this important:

Modern processors provide hyperthreading https://en.wikipedia.org/wiki/Hyper-threading feature that creates virtual processor cores in an attempt to parallelize instruction execution when possible. This feature **may or may not** affect a particular platform, so some experimentation on part of the user is needed. If unsure, then just use only the amount of real cores available in your system.

Look at `cat /proc/cpuinfo` and look at the core ids. Pick processors which are on different cores.

On José's laptop (quad core, Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz)

```
--force-tids="4 0 1 2 3"
```

On nagoyqqatsi computer (oct core, AMD Opteron(TM) Processor 6274 @ 2.20 GHz)

```
--force-tids="8 0 1 2 3 4 5 6 7"
```

[1] THIS IS NOT TO BE USED FOR PARALLEL VERSION OF Real-ESSI AS YOU WILL BE EXPLICITLY USING THOSE MULTIPLE CORES YOURSELF!

It might help if you do:

```
cat /proc/cpuinfo | grep "core id"
```

It will show the number of real cores (and `cat /proc/cpuinfo` will show the processor Ids associated with them).

Also READ the ATLAS manual under "Handling hyperthreading, SMT, modules, and other horrors".

Continuation of ATLAS tuning/build process Continue tuning/building ATLAS

- Tune ATLAS to *your* system (example is for a Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz laptop, using the maximum turbo frequency of 4.40GHz)

  For bash do:

```
export RealESSI_Dependencies_PATH=set_to_appropriate_path
```

  or for tcsh do:

```
set RealESSI_Dependencies_PATH=(set_to_appropriate_path)
```

  For example in my case (Boris) this previous command looks like

```
set ↩
   RealESSI_Dependencies_PATH=(/home/jeremic/oofep/Rad_na_Sokocalu/ESSIforO
```

  check again your CPU frequency:

```
cpufreq-info
```

  and provided that your CPU is at full throttle, start building ATLAS:

```
mv ATLAS ATLAS3.10.x
cd ATLAS3.10.x
mkdir MY_CPU_type
cd MY_CPU_type
time ../configure -b 64 -D c -DPentiumCPS=2900 ↩
  --prefix=$RealESSI_Dependencies_PATH/lib  ↩
  --with-netlib-lapack-tarfile=$RealESSI_Dependencies_PATH/lapack-3.5.0.tg
```

  Explanation of options for configure:

  - -b is the pointer bitwidth, 64 is standard

- `-D c -DPentiumCPS=2000` is a system dependent settings, and it sets the CPU clock rate so that ATLAS can use CPU cycles for timing. You can get that measure running `cpufreq-info` and recording the highest possible frequency that your CPU supports. Unit for this argument is MHz (so for 2.20GHz you would write 2200)
- `--prefix` where to install, a good idea is $RealESSI_Dependencies_PATH/lib
- `-with-netlib-lapack-tarfile=` where is the lapack tarball (provided in the Real-ESSI dependencies tarball), and that is where we have it in Real-ESSI dependencies and

For nagoyqqatsi computer, this last line is

```
../configure -b 64 -D c -DPentiumCPS=2200 ↩
   --prefix=$RealESSI_Dependencies_PATH  ↩
   --with-netlib-lapack-tarfile=$RealESSI_Dependencies_PATH/lapack-3.5.0.tg
```

For an José's laptop, Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz this looks like:

```
../configure -b 64 -D c -DPentiumCPS=4000 ↩
   --prefix=$RealESSI_Dependencies_PATH  ↩
   --with-netlib-lapack-tarfile=$RealESSI_Dependencies_PATH/lapack-3.5.0.tg
```

For Boris' laptop:

```
time ../configure -b 64 -D c -DPentiumCPS=2900 ↩
   --prefix=$RealESSI_Dependencies_PATH  ↩
   --with-netlib-lapack-tarfile=$RealESSI_Dependencies_PATH/lapack-3.5.0.tg
```

You are now ready to build ATLAS. This process will take some time (between 7 and 30 minutes on our computers), do not run anything in parallel with this, let your computer devote full attention to ATLAS. In addition do not use parallel compile (as in `make -j 8` or similar, as this will skew ATLAS tuning. Proceed to build ATLAS:

```
 time make

```

You can also perform some testing:

- sanity check correct answer:

```
make check
```

- sanity check parallel

```
make ptcheck
```

– check if lib is fast

```
make time
```

Once this is done, install:

```
make install

```

This creates the 'lib' and 'include' directories in the `RealESSI_Dependencies_PATH` which are expected by the Real-ESSI build system and needed for placing the rest of the libraries dependencies. Compilation will not continue without these.

1.7.5 Building Dependencies from Source

Now, we have all that we need to build dependencies. To build all the dependencies at once we need to run the **build_libraries script**. To compile the libraries for sequential or parallel version:

- **Build all Libraries**

```
      ./build_libraries NPROC=8

```

- **Build only sequential Libraries**

```
      ./build_libraries sequential NPROC=8

```

- **Build only parallel libraries Libraries**

```
      ./build_libraries parallel NPROC=8

```

where argument **NPROC** is the number of processes, the user wants to run their makefile on. This is equivalent to **"-j"** option in makefile. By default,**NPROC=1**.

With Tuned ATLAS

On the top of that if the use wants to build a *TUNED ATLAS*, then one needs to add extra arguments to the build_libraries script as the following

- TUNED_ATLAS=ON

- CPU_MAX_FREQ=max_cpu_frequency_of_the_system

By default, Atlas tuning is off i.e. **TUNED_ATLAS=OFF**. With the addition of the above parameters (assuming the max_cppu_freq = 2000 MH) the commands would be

- **Build all Libraries**

```
1       ./build_libraries TUNED_ATLAS=ON  CPU_MAX_FREQ=2000 NPROC=20
2
```

- **Build only sequential Libraries**

```
1       ./build_libraries sequential TUNED_ATLAS=ON  ↩
    CPU_MAX_FREQ=2000 NPROC=20
2
```

- **Build only parallel libraries Libraries**

```
1       ./build_libraries parallel TUNED_ATLAS=ON  ↩
    CPU_MAX_FREQ=2000 NPROC=20
2
```

Checking the dependencies build libraries

The user can finally check all the libraries that are build by running the following arguments with **build_libraries script**

- **Checking Sequential Libraries**

```
1       ./build_libraries check_sequential
2
```

- **Checking Parallel Libraries**

```
1       ./build_libraries check_parallel
2
```

- **Showing all libraries build**

```
1       ./build_libraries list_build_dependencies
2
```

1.7.6 Compiling Real-ESSI Program Source

Real-ESSI sources are available to developers/collaborators under an open source license.

If you just want to run the program, you will need to have linux installed with appropriate libraries available (as detailed above). Installation of the Real-ESSI program is most optimal if remote users have available an up to date linux system (sequential and/or parallel) and if arrangements can be made for a temporary, simple/regular user, remote login (through a secure shell) for Prof. Jeremić. This will allow us to compile and install all the necessary libraries and the executable. Distribution of executable without remote login is available also, however in this case we distribute an un-optimized (slow), version of the Real-ESSI Program.

For developers with direct access to git repository do the following:

```
1 git pull
2 cd RealESSI/
```

then recompile:

```
1 mkdir build
2 cd build
3
4 time cmake ..
5 time make -j 16
```

1.7.7 Sequential Version

Compilation of the sequential version needs to be setup in `Makefile.Compilers` file where a line

```
1 PROGRAMMING_MODE = SEQUENTIAL
```

needs to be un-commented, while line `#PROGRAMMING_MODE = PARALLEL` needs to be commented out.

In addition to the above steps, compilation of the sequential version[2] is done by executing:

```
1 mkdir build
2 cd build
3
4 time cmake ..
5 time make -j 16
```

Compilation is done in parallel using available CPUs/cores (here using 16 CPUs, command `time make -j 16`). Depending on the number of available cores/CPUs this number (16) can be changed to whatever is appropriate/desired/available. Number of CPUs will only change/improve the speed of compilation, while for building a parallel version of the Real-ESSI Program, different approach is used (to be described later). Main

[2] Using 16 CPUs/Cores, if smaller/larger number is available/desired, please change that number in make command (`time make -j 16`).

executable program for the Real-ESSI Simulator is in folder `build` and is named `essi`.

It is recommended to create a symbolic link from the Real-ESSI executable to a folder in the system's path. For example:

```
sudo ln -sf $RealESSI_PATH/build/essi   /usr/local/bin/essi
sudo chmod a+rx  /usr/local/bin/essi
```

This will make Real-ESSI available to all users of the system.

Shell script `clean.sh` removes all the object files for all libraries, while leaving the main executable `essi` untouched.

1.7.8 Parallel Version

This section describes Real-ESSI build process on a distributed memory parallel machine. In addition to the above general packages, for parallel version the following packages need to be installed:

For Ubuntu 14.04:

```
sudo apt-get install build-essential
sudo apt-get install cmake
sudo apt-get install openmpi-bin openmpi-doc libopenmpi-dev ↩
   libopenmpi1.6
```

Follow instructions for installing the dependencies. Use the script `compile_libraries_parallel.sh` after tuning ATLAS.

Compilation of parallel Real-ESSI needs execute the command in Real-ESSI source code folder.

```
mkdir pbuild && cd pbuild
cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ ↩
   -DPROGRAMMING_MODE=PARALLEL ..
make -j 32
```

Once done, parallel Real-ESSI executable (named `essi`) is located in the `build` directory that resides within the source directory.

Again, it is a good idea to create a symbolic link from the Real-ESSI executable to a folder in the system's path. For example:

```
sudo ln -sf $RealESSI_PATH/pbuild/essi   /usr/local/bin/essi_parallel
```

This will make Real-ESSI available to all users of the system. Please note that we created a different name for a link `essi_parallel` to distinguish it from the sequential version (which can reside at the same time on the same system, input files will be exactly the same...).

1.8 Real-ESSI and OpenFOAM, Connecting

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

OpenFOAM is a free, open source computational fluid dynamics (CFD) software developed primarily by OpenCFD Ltd since 2004 (https://www.openfoam.com/). Real-ESSI supports numerical interface with OpenFOAM and can perform solid/structure fluid interaction analysis through Real-ESSI – OpenFOAM connection.

1.8.1 Installation of Customized OpenFOAM

We have made in-house modifications and developments to the InterFOAM application (Deshpande et al., 2012) of OpenFOAM-v1612+ for solid fluid interaction. This section presents the installation of the Customized OpenFOAM:

Install the dependencies:

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install flex
sudo apt-get install bison
sudo apt-get install cmake
sudo apt-get install zlib1g-dev
sudo apt-get install libboost-system-dev
sudo apt-get install libboost-thread-dev
sudo apt-get install libopenmpi-dev
sudo apt-get install openmpi-bin
sudo apt-get install gnuplot
sudo apt-get install libreadline-dev
sudo apt-get install libncurses-dev
sudo apt-get install libxt-dev
sudo apt-get install qt4-dev-tools
sudo apt-get install libqt4-dev
sudo apt-get install libqt4-opengl-dev
sudo apt-get install freeglut3-dev
sudo apt-get install libqtwebkit-dev
sudo apt-get install libscotch-dev
sudo apt-get install libcgal-dev
```

Also, make sure gcc and cmake meet the following minimum version requirements:

- gcc: version 4.8.5 or above
- cmake: version 3.3 or above

Check the version of gcc and cmake by running the following commands on terminal. If you are installing on Ubuntu 16.04 and above, the system version of gcc and cmake should already meet the requirements.

```
gcc --version
cmake --version
```

Downloaded the source code of Customized OpenFOAM:

```
wget ↩
   http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/
_Chapter_SoftwareHardware_Build_Process/OpenFOAM/sources/OpenFOAM.tar.gz
```

Choose a directory and extract the downloaded compressed file to the target directory.

```
tar -xzvf OpenFOAM.tar.gz -C /target/directory
```

For example, hereafter we choose $HOME as target directory. Replace $HOME with your chosen directory accordingly.

```
tar -xzvf OpenFOAM.tar.gz -C $HOME
```

Go to the extracted folder and source OpenFOAM environment configurations:

```
cd $HOME/OpenFOAM
source $HOME/OpenFOAM/OpenFOAM-v1612+/etc/bashrc
```

Setup CGAL and Boost version for compilation:

```
cgal_version=CGAL-4.9.1
boost_version=boost-system
```

Check the system readiness

```
foamSystemCheck
```

Change to the main OpenFOAM directory:

```
foam
```

Note: if running *foam* cannot change to the main OpenFOAM directory, in this case the directory is *$HOME/OpenFOAM*, source the environment configuration again by running the following terminal command.

```
source $HOME/OpenFOAM/OpenFOAM-v1612+/etc/bashrc
```

Compile OpenFOAM:

```
./Allwmake
```

Since OpenFOAM is shipped with ParaView for post-processing OpenFOAM field results using developed plug-in paraFoam (https://cfd.direct/openfoam/user-guide/v6-paraview/). We also need to compile customized ParaView with paraFoam plug-in:

```
cd $WM_THIRD_PARTY_DIR
./makeParaView
```

1.8.2 Check the Customized OpenFOAM Installation

Open a new terminal and source the OpenFOAM environment:

```
source $HOME/OpenFOAM/OpenFOAM-v1612+/etc/bashrc
```

Validate the build by running:

```
foamInstallationTest
```

Create a user run directory:

```
mkdir -p $FOAM_RUN
```

go to the user run directory:

```
run
```

Copy a simulation case from OpenFOAM tutorial to the user run directory:

```
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily ./
```

go to the copies case directory:

```
cd pitzDaily
```

Generate the mesh:

```
blockMesh
```

Perform the analysis with the application simpleFoam:

```
simpleFoam
```

Visualize the simulation results:

```
paraFoam
```

1.8.3 Compile Real-ESSI with Link to OpenFOAM

Go to Real-ESSI source directory under directory RealESSI_ROOT and clean any previous old compilation of Real-ESSI:

```
cd RealESSI_ROOT/Real-ESSI
```

```
rm -rf bin
rm -rf lib
rm -rf  build_sequential
mkdir bin
mkdir lib
mkdir  build_sequential
cd build_sequential
```

Build and install the executable, using 16 CPUs in this case. Of course, if you have more CPUs available, you can use most of them. Please make sure to specify your OpenFOAM installation directory with CMake argument -DOPENFOAM_DIR. For example, in this case, we specify the installation directory as *$HOME-/OpenFOAM*.

```
time cmake -DUSE_OPENFOAM=TRUE -DOPENFOAM_DIR=$HOME/OpenFOAM ..
time make -j 16
make install
```

Rename essi to essi.sequential just so to distinguish it from the parallel executable:

```
cd ../bin
cp essi essi.sequential
```

Finally, install essi.sequential in system binary directory so that others can use it:

```
sudo rm /usr/bin/essi /usr/bin/essi.sequential
sudo cp essi.sequential /usr/bin/essi.sequential
sudo chmod a+x /usr/bin/essi.sequential
```

1.9 Code Verification After the Build Process

After build process, test cases to verify that installation is successful should be run. There are four groups of verification cases. The first two groups are designed for users. The last two groups are designed for developers.

1. The first group of test cases compares the sequential essi results to the analytic solutions.
2. The second group of test cases compares the parallel essi results to the analytic solutions.
3. The third group of test cases tests the version stability between two essi executables.
4. The fourth group of test cases tests the memory management of Real-ESSI with valgrind.

1.9.1 Run all verification test cases

In order to run all test cases to verify the installation, users can run

```
cd $RealESSI_PATH/
bash run_all_verification.sh
```

Please make sure that sequential essi is available as 'essi' in the PATH, and parallel essi is available as 'essi_parallel' in the PATH before running all the verification test cases.

In addition, if users want to clean the test results, users can run

```
cd $RealESSI_PATH/
bash clean_all_verification.sh
```

Finally, users can also run a single group of test cases as follows.

1.9.2 Test Sequential Real-ESSI

In order to test whether the installation of sequential essi is successful, open the sequential example folder and run the bash script.

```
cd ↩
   $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/analytic_solution
bash make_comparison.sh
```

This bash script will run all the examples automatically and compare the results to the analytic solutions. The comparison results are not only printed in the Terminal but also saved as a .log file in the same folder. Before you run the examples, make sure $essi$ is in your PATH.

1.9.3 Test Parallel Real-ESSI

In order to test whether the installation of parallel essi is successful, open the parallel example folder and run the bash script.

```
cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/parallel
bash make_comparison.sh
```

This bash script will run all the examples automatically and compare the results to the analytic solutions. The comparison results are not only printed in the Terminal but also saved as a .log file in the same folder. Before you run the examples, make sure $essi\_parallel$ is in your PATH.

1.9.4 Version Stability Test

Since new features are continuously updated and improved in Real-ESSI, the version stability test helps the developers to guarantee their modification will not affect the correct operation of other code.

In order to test version stability,

```
cd ↩
   $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/version_stability
bash generate_original.sh
```

This bash script will run all the examples automatically and save the results for reference later. This bash script above should run with the previous stable essi.

Then, to test the new essi and compare the results

```
cd ↩
   $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/version_stability
bash make_comparison.sh
```

This bash script will run all the examples again and compare the results to the previous saved results. This bash script should run with the new essi. The comparison results are not only printed in the Terminal but also saved as a .log file in the same folder.

1.9.5 Memory Management Test

Memory management is important in C/C++ programming. This group of test cases helps the developers to track the memory leak in Real-ESSI. For the details about the code stability verification, please refer to the Section 3.2 on Page 34.

Before you run the test cases, make sure Valgrind is installed. You can install Valgrind by this command.

```
sudo apt-get install valgrind
```

You can also download the source of Valgrind and compile it from scratch.

It is important to test memory leak in parallel simulations.

```
mpirun -np 3 valgrind --log-file='log_%p.valgrind' --leak-check=yes ↩
   essi-parallel-debug -f main.fei
```

A few important things to mention here:

- To test memory leak in parallel simulation, you obviously need a parallel version of Real-ESSI.
- Real-ESSI needs to be compiled in debug mode. This is important for Valgrind to capture and location the source of memory leaks.
- Running Real-ESSI in debug mode and in Valgrind means the simulation will be very slow. So it's not practical to run memory leak test using a large model. You should have a model with only a few elements/nodes (but more than 1 element so that it runs in parallel) that includes the specific functions you want to test.

- Valgrind log files will be saved in the location where you run the model. There will be multiple log files named as `log_processID.valgrind`. Each process will have its own Valgrind log file. There might be a few empty Valgrind log files generated, you can just ignore those. The number of Valgrind log files that actually contain memory leak information should be the same as the number of cores you use in your simulation.

- Valgrind is a powerful tool with many options. The command shown above is rather basic but serves as a good starting point. Memory leaks can be very tricky to track and fix. You should learn and experiment with Valgrind options for different issues you want to fix.

Valgrind log file can be very long and hard to read. At the bottom, there is a leak summary that looks like this: You should primarily focus on the 'definitely lost' result. 'Indirectly lost' and 'possibly lost' can also

```
==24551== LEAK SUMMARY:
==24551==    definitely lost: 1,329,224 bytes in 13,211 blocks
==24551==    indirectly lost: 674,019 bytes in 2,395 blocks
==24551==      possibly lost: 31,496 bytes in 9 blocks
==24551==    still reachable: 60,929,829 bytes in 8,999 blocks
==24551==         suppressed: 0 bytes in 0 blocks
```

Figure 1.1: Valgrind log file: Memory leak summary.

be problematic but should go away once you fix the source of 'definitely lost'. 'Still reachable' is usually not considered as actual memory leak but is something that can be optimized. Refer to the Valgrind User Manual for more information.

Valgrind log file contains detailed information on each memory leak. A typical leak detail looks like this. Following the trail, you should be able to locate the source of a specific leak and then fix it properly.

```
==24551== 493,960 bytes in 12,349 blocks are definitely lost in loss record 6,136 of 6,140
==24551==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==24551==    by 0xB1BACD: Marray<double, 2, TinyArray_base<double, 2> >::compute_Determinant() (Marray_rank2.h:628)
==24551==    by 0x19BA8E8: ClassicElastoplasticMaterial<LinearIsotropic3D_EL, VonMises_YF<ArmstrongFrederickTensor_EV, LinearHardeningScala
==24551==    by 0x19AF12A: ClassicElastoplasticMaterial<LinearIsotropic3D_EL, VonMises_YF<ArmstrongFrederickTensor_EV, LinearHardeningScala
==24551==    by 0xC386B0: EightNodeBrick::update() (MasterEightNodeBrick.cpp:1950)
==24551==    by 0x15B3A3C: Domain::update() (Domain.cpp:3891)
==24551==    by 0x165C689: ActorSubdomain::update() (ActorSubdomain.cpp:1297)
==24551==    by 0x15766C2: AnalysisModel::updateDomain() (AnalysisModel.cpp:534)
==24551==    by 0x1572CFA: Newmark::update(Vector const&) (Newmark.cpp:570)
==24551==    by 0x157FA55: NewtonLineSearch::solveCurrentStep() (NewtonLineSearch.cpp:154)
==24551==    by 0x156926C: TransientDomainDecompositionAnalysis::analyze(double) (TransientDomainDecompositionAnalysis.cpp:242)
==24551==    by 0x1569AAC: TransientDomainDecompositionAnalysis::newStep(double) (TransientDomainDecompositionAnalysis.cpp:460)
```

Figure 1.2: Valgrind log file: Memory leak detail.

A serious memory leak issue caused by external solvers used by PETSc was found. As shown in Figure 1.3, when the `mumps` option was used in parallel solver, a significant amount of memory leak was detected by Valgrind. More importantly, such memory leak was observed to increase with the number of time steps. This means large-scale, long-duration simulation could be interrupted due to not enough memory in the operating

system. Note that this issue was also reported in other occasions where the mumps package is used within PETSc, as recent as June 2020.

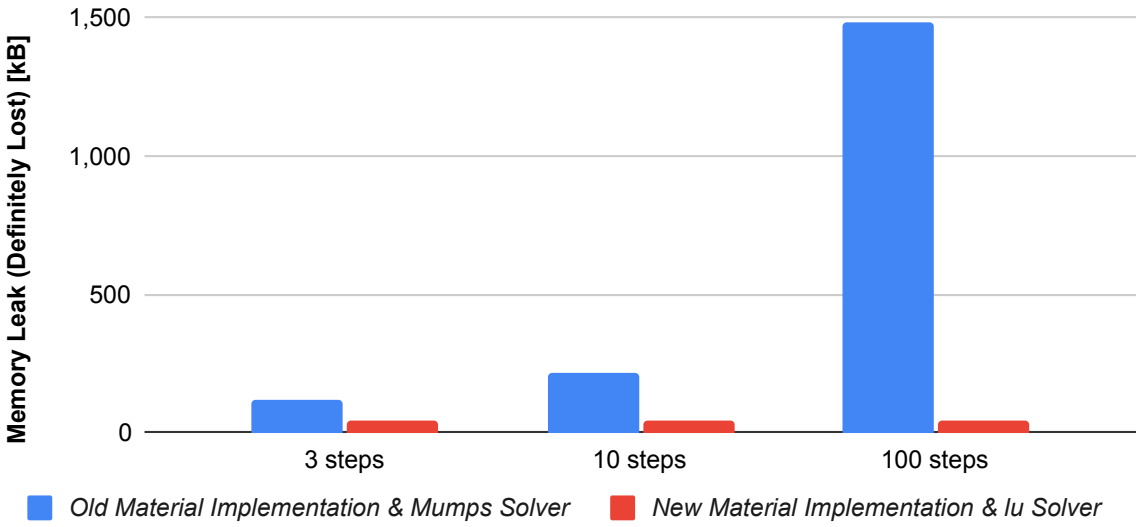

Figure 1.3: Comparison of memory leak between different PETSc solver options.

After extensive tests, it has been found out that other options/packages in PETSc don't have the memory leak issue mentioned above. Therefore it is recommended to use options other than mumps for large-scale, long-duration simulations. For example, the following command calls the default direct solver of PETSc:

```
1 define solver parallel petsc "-ksp_type preonly -pc_type lu" ;
```

1.10 Compiling Real-ESSI Utilities

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

Real-ESSI comes with a lot of utilities to help the users speed up the simulation process. It provides mesh building, auto-input generation and visualization features which makes it quite nice.

Real-ESSI source code contains **build_utilities** script which can be used to build all the avialble utilities. We will go through the following subsection to introduce each utility and how to compile them.

The first step is to download all the sources of utilities that needs to be build. To do this, one has to run

```
1 cd Real-ESSI
2 ./build_utilities download
```

This would download all the utilities sources in tar.gz format and would place them in "/SRC" of RealESSI_Utilities directory. The script is very powerfull and accepts targets that can be used to build a

particular utility or all utilities at once. The available options to the scripts can be found by running the target help as shown below. A snippet is shown below

```
./build_utilities help

#usage: make [target]
#
#Utilities:
#  gmessi                          Builds gmessi
#  paraview                        Builds paraview
#  pvessi                          Builds pvessi
#  gmsh                            Builds gmsh
#  visit                           Builds visit
#  visitessi                       Builds visitessi
#
#Sequential:
#  clean_utilities                 Cleans all utilities
#
#Default:
#  all                             Builds all the necessary utilities for REAL-ESSI
#  all_utils                       Builds all the necessary utilities for REAL-ESSI
#  clean_all                       Cleans everything
#  clean                           Cleans everything
#
#Check:
#  check_utilities                 Checks if all utilities libraries are build
#
#Miscellaneous:
#  list_utilities                  Lists all the available utilities version from SRC folder
#  list_build_utilities            Lists all the utilities library allready build in lib folder
#  help                            Show this help.
#  download                        Downloads the Utilities Sources
#
#Update:
#  update_gmessi                   update gmessi utility
#  update_pvessi                   update pvessi utility
#  update_visitessi                update visitessi utility
#
#Clean:
#  clean_gmessi                    Clean gmessi utility
#  clean_paraview                  Clean paraview utility
#  clean_pvessi                    Clean pvessi utility
#  clean_gmsh                      Cleans gmsh utility
#  clean_visit                     Cleans visit
#  clean_visitessi                 Cleans visitessi utility
```

The user can compile individual utilities by running just running

```
./build_utilities <utilitu_name>
```

Note: All the binaries of the utilities after build gets linked/copied to the **RealESSI_Utilities/bin** directory inside RealESSI_ROOT.

1.10.1 Installation of `gmsh` and `gmESSI`

gmsh is a 3-D finite element mesh generator for academic problems with parametric input and advanced visualization capabilities. It can be downloaded and installed from http://geuz.org/gmsh/. Additionally, the user can also install gmsh from terminal:

```
sudo apt-get install gmsh
```

gmESSI is effective pre-processor for generating Real-ESSI input files directly for the mesh file provided by gmsh. More information about gmESSI and how it works is given in Chapter 207 of the main document, lecture notes (Jeremić et al., 1989-2022). The gmESSI package is available from the main repository site: http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Pre_Processing/Real-ESSI_gmESSI.tgz. Before installing gmESSI, please install required libraries, as explained in Section 1.6.2 on Page 16. To install gmESSI, go to the Real-ESSI directory and then run

```
./build_utilities gmessi
```

To update the utility at the user just needs to run

```
./build_utilities update_gmessi
```

Refer to section 1.11 on page 52 for instructions on what and how to install autocompletion and syntax coloring for gmESSI and Real-ESSI syntax on sublime text editor.

1.10.2 Installation of `ParaView` and `PVESSIReader`

ParaView package http://www.paraview.org/ is a powerfull multi-platform data analysis and visualization application avialable in Open Source. It can be run on supercomputers to analyze datasets of petascale size as well as on laptops for small datasets. ParaView can be used to visualize results of Real-ESSI simulations. A plug-in was developed for ParaView so that all the simulations results from Real-ESSI finite elements, material models and analysis types can be directly visualized, animated, etc.

Building ParaView and PVESSIReader Plugin from Source on Linux System

Note that ParaView, as well as its building procedure, has recently (during 2020) gone through some major changes. The building procedures shown in this section are mostly based on the information available at: https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md.

1. Install Dependencies

```
sudo apt-get install libgl1-mesa-dev libxt-dev qt5-default ↩
   libqt5x11extras5-dev libqt5help5 qttools5-dev ↩
   qtxmlpatterns5-dev-tools libqt5svg5-dev python3-dev ↩
   python3-numpy ninja-build
```

2. Obtain the source of ParaView

```
git clone --recursive ↩
   https://gitlab.kitware.com/paraview/paraview.git
cd paraview
git checkout v5.8.1
git submodule update --init --recursive
mkdir paraview_build
```

3. Obtain the source of PVESSIReader plugin

   - The source of PVESSIReader plugin can be downloaded from http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Post_Processing/_Real_ESSI_PVESSIReader_.zip
   - Extract the files and move the `PVESSIReader` folder to `./Plugins/`

4. Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `paraview/CMakeLists` using your choice of text editor. Find "`set(paraview_default_plugins`" and add "`PVESSIReader`" to the end of the list of plugins.

5. Build

```
cd paraview_build
cmake -GNinja -DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON ↩
   -DVTK_SMP_IMPLEMENTATION_TYPE=TBB -DCMAKE_BUILD_TYPE=Release ..
ninja
```

6. Load PVESSIReader plugin into ParaView

   - Run the ParaView executable and click on Tools → Manage Plugins → Load New ...

- Find `PVESSIReader.so` at `paraview/paraview_build/lib/paraview-5.8/plugins/PVESSIReader` and click OK to load it.
- Now you should see PVESSIReader loaded in the list of plugins. Double click on it to expand advanced options and check Auto Load.
- Close the ParaView application and reopen it. Now the PVESSIReader plugin should be automatically loaded and ready to use.

Building ParaView and PVESSIReader Plugin from Source on Windows System

Note that ParaView, as well as its building procedure, has recently (during 2020) gone through some major changes. The building procedures shown in this section are mostly based on the information available at: https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md. It is noted that user should be prepared to spend some time (perhaps hours) on installing procedure...

1. Install Dependencies

   - Download and install git bash for windows. Use the latest release version.
   - Download and install cmake. Use the lastest release version.
   - Download and install Visual Studio 2015 Community Edition. If this is the first time Visual Studio is installed in your system, some developer tools are probably missing. Open Visual Studio and attempt to build a new Visual C++ project. If you see any options saying "Install ... for C++", click on that/them and necessary tools will be installed.
   - Download ninja-build and drop `ninja.exe` in `C:\Windows\`. Use the latest release version.
   - Download and install both `msmpisetup.exe` and `msmpisdk.msi` from Microsoft MPI. Use the latest release version from Microsoft.
   - Download and install Python for Windows. Latest release version should work fine. To avoid potential compatibility issues, install the same Python version that is used for the latest release of ParaView.
   - Download and install Qt 5.12.3 for Windows, make sure to check the MSVC 2015 64-bit component during installation, make sure to add `C:\Qt\Qt5.12.3\5.12.3\msvc2015_64\bin` to your `PATH` environnement variable. Note that Qt for Windows is x86 but it works for x64 machine as well.

2. Obtain the source of ParaView

   - Open your preferred Windows command prompt. Windows PowerShell is a nice tool for people usually work with Linux system.

- To build ParaView developement version (usually refered as "master"), run the following commands:

```
cd C:
mkdir pv
cd pv
git clone --recursive ↩
   https://gitlab.kitware.com/paraview/paraview.git
mv paraview pv
mkdir pvb
```

- To build a specific ParaView version, please refer to https://gitlab.kitware.com/paraview/paraview/blob/master/Documentation/dev/build.md.

3. Obtain the source of PVESSIReader plugin

   - The source of PVESSIReader plugin can be downloaded from http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Post_Processing/_Real_ESSI_PVESSIReader_.zip
   - Extract the files and move the `PVESSIReader` folder to `C:\pv\Plugins\`

4. Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `C:\pv\CMakeLists.tx` using your choice of text editor. Find "`set(paraview_default_plugins`" and add "`PVESSIReader`" to the end of the list of plugins.

5. Build

   - Open VS2015 x64 Native Tools Command Prompt and run the following commands

```
cd C:\pv\pvb
cmake -GNinja -DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON ↩
   -DCMAKE_BUILD_TYPE=Release ..\pv
ninja
```

   - This step could be take a few hours. If no configuration or compilation error is encountered, you should have the ParaView executable at `C:\pv\pvb\bin\`.

6. Load PVESSIReader plugin into ParaView

   - Run the ParaView executable and click on Tools → Manage Plugins → Load New ...
   - Find `PVESSIReader.dll` at `C:\pv\pvb\bin\paraview-5.8\plugins\PVESSIReader\` and click OK to load it.

- Now you should see PVESSIReader loaded in the list of plugins. Double click on it to expand advanced options and check Auto Load.
- Close the ParaView application and reopen it. Now the PVESSIReader plugin should be automatically loaded and ready to use.

Building ParaView and PVESSIReader Plugin from Source on AWS

Currently, the AWS image has Ubuntu 18.04. This may change in the future. Because AWS is a remote server, properly running ParaView needs more steps in compilation. Note that most information here are based on this discussion.

1. Install Dependencies

```
sudo apt-get install libgl1-mesa-dev libxt-dev qt5-default ↩
   libqt5x11extras5-dev libqt5help5 qttools5-dev ↩
   qtxmlpatterns5-dev-tools libqt5svg5-dev python3-dev ↩
   python3-numpy ninja-build gettext python-mako
```

2. Download, Build and Install LLVM

```
wget http://releases.llvm.org/7.0.1/llvm-7.0.1.src.tar.xz
mkdir llvm
cd llvm
tar -xvf /path/to/llvm-7.0.1.src.tar.xz
mkdir llvm_build
mkdir llvm_install
cd  llvm_build
cmake                                                \
  -DCMAKE_BUILD_TYPE=Release                         \
  -DBUILD_SHARED_LIBS=ON                             \
  ↩
   -DCMAKE_INSTALL_PREFIX=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llv
    \
  -DLLVM_ENABLE_RTTI=ON                              \
  -DLLVM_INSTALL_UTILS=ON                            \
  -DLLVM_TARGETS_TO_BUILD:STRING=X86                 \
  ../llvm-7.0.1.src
make -j8 install
```

3. Download, Build and Install Mesa

```
wget ↩
   https://gitlab.freedesktop.org/mesa/mesa/-/archive/mesa-18.3.3/mesa-mesa
mkdir mesa
cd mesa
tar -xvf /path/to/mesa-mesa-18.3.3.tar.bz2
cd mesa-mesa-18.3.3
```

```
autoreconf --force --verbose --install
cd ../
mkdir mesa_build
mkdir mesa_install
cd mesa_build
../mesa-mesa-18.3.3/configure ↩
   --prefix=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/mesa/mesa_install
   \
--enable-opengl --disable-osmesa --disable-gallium-osmesa \
--enable-glx --with-platforms=x11 --disable-gles1 --disable-gles2 \
--disable-va --disable-gbm --disable-xvmc --disable-vdpau \
--disable-shared-glapi --disable-dri --with-dri-drivers= \
--enable-llvm  ↩
   --with-llvm-prefix=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llvm/ll
    \
--with-gallium-drivers=swrast,swr --with-swr-archs=avx,avx2 ↩
   --disable-egl
make -j8 install
```

4. Obtain the source of ParaView

```
git clone --recursive ↩
   https://gitlab.kitware.com/paraview/paraview.git
cd paraview
git checkout v5.8.1
git submodule update --init --recursive
```

5. Obtain the source of PVESSIReader plugin

   - The source of PVESSIReader plugin can be downloaded from http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_notes_online_material/_Chapter_SoftwareHardware_Post_Processing/_Real_ESSI_PVESSIReader_.zip
   - Extract the files and move the `PVESSIReader` folder to `./Plugins/`

6. Modify the cmake file to include PVESSIReader plugin in the building process. Open the file `paraview/CMakeLists` using your choice of text editor. Find "`set(paraview_default_plugins`" and add "`PVESSIReader`" to the end of the list of plugins.

7. Build ParaView with Mesa

```
mkdir paraview_build
cd paraview_build
cmake -GNinja ↩
   -DOPENGL_gl_LIBRARY=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/mesa/m
   \
-DOPENGL_INCLUDE_DIR=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/mesa/mes
   \
```

```
-DOPENGL_EGL_INCLUDE_DIR= -DOPENGL_GLES2_INCLUDE_DIR= \
-DOPENGL_GLES3_INCLUDE_DIR= -DOPENGL_GLX_INCLUDE_DIR= \
-DOPENGL_egl_LIBRARY= -DOPENGL_gles2_LIBRARY=  \
-DOPENGL_gles3_LIBRARY= -DOPENGL_glu_LIBRARY= \
-DOPENGL_glx_LIBRARY= -DOPENGL_opengl_LIBRARY= ../paraview \
-DPARAVIEW_USE_PYTHON=ON -DPARAVIEW_USE_MPI=ON \
-DVTK_SMP_IMPLEMENTATION_TYPE=TBB -DCMAKE_BUILD_TYPE=Release ../paraview
LD_LIBRARY_PATH=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llvm/llvm_ins
   ninja
```

Note: There are many warnings showed up when I was doing this step, but it doesn't seem to terminate the build process.

8. Run ParaView with Mesa

```
LD_LIBRARY_PATH=/home/ubuntu/RealESSI_ROOT/RealESSI_Utilities/llvm/llvm_ins
   paraview
```

9. Load PVESSIReader plugin into ParaView

   - Run the ParaView executable and click on Tools → Manage Plugins → Load New ...
   - Find `PVESSIReader.so` at `paraview/paraview_build/lib/paraview-5.8/plugins/PVESSIReader` and click OK to load it.
   - Now you should see PVESSIReader loaded in the list of plugins. Double click on it to expand advanced options and check Auto Load.
   - Close the ParaView application and reopen it. Now the PVESSIReader plugin should be automatically loaded and ready to use.

1.11 Sublime Text Editor

Note: This section describes build procedure for old versions of Real-ESSI and/or its modules.

Install `sublime` text editor from http://www.sublimetext.com/. Then install package control to sublime in order to install plugins. (go to preferences, package control, install package.) Then install two packages:

FEI Syntax-n-Snippets, Real-ESSI syntax and auto completion plugin for [].fei files (input files for Real-ESSI program).

gmsh-Tools, syntax and autotext completion for gmsh model development tools for Real-ESSI.

gmESSI-Tools, syntax and autotext completion for gmESSI model development tools for Real-ESSI.

1.12 Model Conversion/Translation using FeConv

FeConv allows conversion/translation of input files (models) between Real-ESSI and SASSI, Sofistik, Ansys, OpenSees and Strudyn. FeConv was developed and is maintained by Mr. Viktor Vlaski.

1.13 Build Procedures on Amazon Web Service

This section shows the steps to install a new Real-ESSI image on Amazon Web Service (AWS). This document is only intended for Real-ESSI developers, not for general users. For using Real-ESSI on AWS, please refer to Chapter 211, on page 1362 in Jeremić et al. (1989-2022).

Noted that when creating a new image, the instance type should be consistent with future usage. For example, if the user intend to launch a Real-ESSI instance using the instance type "General Purpose", such as the T2 series, the image should also be created with the same instance type. If the image is created with a different instance type, Real-ESSI will not be able to run, and the following error message will be observed:

```
1 Illegal instruction (core dumped)
```

1.13.1 Sign In to AWS

Here is the link to the AWS sign in page. Click "Sign In to the Console" button on the upper right corner of the page. No need to register a new account. You should already have the account ID, IAM user name, and password for AWS sign in. If not, please contact an administrator to add you to the developers' group.

After sign in, go to the "EC2" tab under "Service". Here you can view all your instances and AMIs. This is where you can start new simulations or install new images.

Note that you probably also need to choose the correct region. On the upper right corner of the page, you can see your current region and switch to another one if necessary.

1.13.2 Copy an Existing Image

Since we already have a few images for Real-ESSI, the most efficient way to create a new image is to simply copy an existing one. To do this, go to the "AMIs" tab under "IMAGES" on the left part of the page. Now you should be able to view all existing images.

Select the image that you want to copy. Click the "Actions" button and choose "Copy AMI". On the pop-up window, enter the informations of this new image that you want to create. Then just click the "Copy AMI" button.

Now you should have a new image that has been installed with all the Real-ESSI components. To make any change inside this image, you need to launch it as a new instance and access it using X2GO. Procedures to

install and use X2GO can be found in Chapter 211. For cloud server, on AWS or similar, the build procedures are the same as those for local installation, which can be found in previous sections of this chapter.

1.13.3 Create a New Image

If you need to create a new Real-ESSI image from scratch, this section shows the steps to do so. First sign in to AWS and go to "EC2". Choose the correct region. Click the "Instance" tab under "INSTANCES" on the left part of the page. Choose "Launch Instance" to start a new instance that later will be saved as your new image.

Then, follow these steps:

1. Choose AMI: Ubuntu Server 16.04 LTS (HVM), SSD Volume Type.
2. Choose Instance Type: Family = Compute optimized, Type = c5.4xlarge, vCPUs = 16, Memory (GiB) = 32.
3. Keep other options as default, and click "Review and Launch".
4. Review the information of the new instance, and click "Launch".

Next, you are asked to choose a key pair for your instance. It's recommended to create a new key pair for the first time, then use it in the future. First, choose "Create a new key pair", and enter a name. Click the "Download Key Pair" button. Save the key in a secure directory in your local computer for future use, for example in .ssh directory.

Now, you can select "Choose an existing key pair", and select your key pair that should be visible. Check the box for acknowledging the use of a private key. Finally, your new instance is launched. Note that this new instance is a brand new Ubuntu server, which means that you need to install everything.

At this point, the new Ubuntu server on AWS does not have X2GO for remote access or a GUI desktop to operate. We will now install these necessary softwares. First, run the following command to access the remote Ubuntu server on AWS using ssh. Note that you need to change the name of your ssh key to the one you just created. The public IP address can be found on the AWS webpage where you launched your new instance. Go the description of your instance to find the "IPv4 Public IP".

```
chmod 400 your_ssh_key.pem
ssh -i your_ssh_key.pem ubuntu@your_AWS_public_IP_address
```

Run the following command to install X2GO server on Ubuntu Linux.

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:x2go/stable
sudo apt-get update
sudo apt-get install x2goserver x2goserver-xsession
```

Xfce is a lightweight desktop and ideal for usage on a remote server. Run the following command to install xfce on Ubuntu.

```
1 sudo apt-get install xfce4 xfce4-goodies
```

Now you can access your new instance (the remote Ubuntu server) using X2GO. Steps to do this can be found in Chapter 211. After you established remote control of the Ubuntu server on AWS, the build procedures are the same as those for local installation, which can be found in previous sections of this chapter.

The last step is to create a new image from this instance so that you can launch it in the future. Go the "Instances", and choose the correct instance. Click "Actions", and select "Create Image" under "Image". You can change the size of the instance volume, but it's not necessary at this moment. Give your image a name and a description, and click "Create Image". Now you have sucessfully created a new image for Real-ESSI. If you go to "AMIs", you should be able to see this new image you just created.

1.13.4 Build AWS ESSI Image from Scratch

This section is a developer guide, which presents the procedures to build AWS ESSI Image from scratch.

ESSI AWS users do not need to know the technical details in this section.

1. Launch EC2 instance from an AWS blank image: Ubuntu 16.04 Server.

   EC2 Dashboard → Instances → Instance → Launch Instance.

   Choose

   ```
   Ubuntu Server 16.04 (HVM), SSD Volume Type.
   ```

   Since there is no Desktop version available, so we have to launch the server version and install desktop by ourself.

   You will need to download a .pem key to launch the instance.

2. Login to the Remote Instance using Terminal.

   Copy the external IP address of the remote instance from the Browser.

   Use the downloaded .pem key to login to the remote instance.

   ```
   chmod 400 your_key.pem
   ssh -i your_key.pem ubuntu@your_remote_instance_IP
   ```

3. Install Desktop and git on AWS Remote Instance

   ```
   sudo apt update
   sudo apt install -y ubuntu-desktop git
   ```

4. Install remote-desktop-server (x2goserver) on AWS Remote Instance

   ```
   sudo add-apt-repository ppa:x2go/stable
   sudo apt update
   sudo apt install -y x2goserver x2goserver-xsession xfce4
   ```

5. Set up the automatic launch of remote desktop server

   ```
   sudo systemctl enable x2goserver.service
   sudo systemctl start x2goserver.service
   ```

6. Install ESSI

   ```
   # Install prerequisite
   sudo apt install -y cmake
   ```

```
sudo apt install -y build-essential
sudo apt install -y zlib1g-dev
sudo apt install -y libtbb-dev
sudo apt install -y bison flex
sudo apt install -y libboost-dev
sudo apt install -y python
sudo apt install -y gfortran
sudo apt install -y libopenblas-dev
sudo apt install -y liblapack-dev
sudo apt install -y python-scipy
sudo apt install -y libhdf5-dev libhdf5-cpp-11
sudo apt install -y python-h5py
sudo apt install -y python-matplotlib
sudo apt install -y libssl-dev

# Download ESSI
#
# using curly brackets to help in checking scripts, that rely on ↩
   these
# brackets being available around URL
#
git clone {https://github.com/BorisJeremic/Real-ESSI.git} # Need ↩
   permission from Boris Jeremic for Real-ESSI on github
cd Real-ESSI

# Build ESSI Dependencies
./build_libraries download
./build_libraries sequential
./build_libraries hdf5_sequential
./build_libraries suitesparse
./build_libraries arpack
./build_libraries parmetis
./build_libraries petsc

# Build Sequential ESSI
mkdir build
cd build
cmake ..
make -j $(nproc)
cd ..


# Build Parallel ESSI
mkdir build_parallel
cd build_parallel
cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ -DPETSC_HAS_MUMPS=TRUE ↩
   -DPROGRAMMING_MODE=PARALLEL ..
make -j $(nproc)
cd ..
```

7. Install gmsh

```
sudo apt install -y gmsh
```

8. Install gmESSI

```
# Install the prerequisite
sudo apt install -y libboost-all-dev
sudo apt install -y build-essential
sudo apt install -y python-dev
sudo apt install -y liboctave-dev

# Install gmESSI
## download the package from the main Real-ESSI repository
#
# using curly brackets to help in checking scripts, that rely on ↩
   these
# brackets being available around URL
#
wget ↩
   {http://sokocalo.engr.ucdavis.edu/~jeremic/Real_ESSI_Simulator/gmESSI/_a
mkdir Real-ESSI-gmESSI
mv _all_files_gmESSI_.tgz Real-ESSI-gmESSI
cd Real-ESSI-gmESSI

make -j $(nproc)

# Add binary PATH to ~/.bashrc
cd ./build/bin/
part1="export PATH=\""
part2=$PWD
part3=":\$PATH\""
newline=$part1$part2$part3
echo $newline >> ~/.bashrc
```

9. Install ParaView with PVESSIReader plugin

```
# Install the prerequisite
sudo apt install -y libavformat-dev
sudo apt install -y libswscale-dev
sudo apt install -y ffmpeg
sudo apt install -y libphonon-dev libphonon4 qt4-dev-tools
sudo apt install -y libqt4-core libqt4-gui qt4-qmake libxt-dev
sudo apt install -y g++ gcc cmake-curses-gui libqt4-opengl-dev
sudo apt install -y mesa-common-dev python-dev
sudo apt install -y libvtk6.2
sudo apt install -y mpich libopenmpi-dev
sudo apt install -y libxmu-dev libxi-dev

# Download the ParaView
#
# using curly brackets to help in checking scripts, that rely on ↩
```

```
   these
# brackets being available around URL
#
git clone {https://github.com/Kitware/ParaView.git}
cd ParaView
git checkout v5.1.2
git submodule update --init

# Download the Plugin
cd Plugins
#
# using curly brackets to help in checking scripts, that rely on ↩
   these
# brackets being available around URL
#
wget ↩
   {http://sokocalo.engr.ucdavis.edu/~jeremic/Real_ESSI_Simulator/pvESSI/_p
tar -xvzf _pvESSI_all_files_.tgz
cd ..

# Compile ParaView along with PVESSIReader
mkdir build && cd build
cmake -DPARAVIEW_USE_MPI=true -DPARAVIEW_ENABLE_PYTHON=true ↩
   -DPARAVIEW_ENABLE_FFMPEG=true ..
make -j $(nproc) # require Internet during ParaView compilation.

# Add binary PATH to ~/.bashrc
cd bin
part1="export PATH=\""
part2=$PWD
part3=":\$PATH\""
newline=$part1$part2$part3
echo $newline >> ~/.bashrc
```

10. Install Sublime Text 3 and ESSI plugin. Following this link.

```
#
# using curly brackets to help in checking scripts, that rely on ↩
   these
# brackets being available around URL
#
wget -qO - {https://download.sublimetext.com/sublimehq-pub.gpg} | ↩
   sudo apt-key add -
sudo apt-get install apt-transport-https
echo "deb {https://download.sublimetext.com/ apt/stable/}" | sudo ↩
   tee /etc/apt/sources.list.d/sublime-text.list
sudo apt-get update
sudo apt-get install sublime-text
```

11. Install Sublime Text Plugin:

```
# Inside Sublime Text Window

# Ctrl+Shift+P, then Type
install package control

# Ctrl+Shift+P, then Type
install package # press ENTER, then type
fei syntax-n-snippets

# Ctrl+Shift+P, then Type
install package # press ENTER, then type
gmESSI-Tools

# Ctrl+Shift+P, then Type
install package # press ENTER, then type
gmsh-Tools
```

12. Create Image inside Browser.

    Select the launched Image with the above software installed.

    Choose Actions → Image → Create Image.

    Type your Image Name and descriptions.

    You will then see your image in EC2 Dashboard → Images → AMIs

1.13.5 Update an Existing Image

For updating an existing image, for example for a new version or release follow instruction below. First sign in to AWS and go to "EC2". Choose the correct region. Click the "Instance" tab under "INSTANCES" on the left part of the page. Choose "Launch Instance" to start a new instance that later will be saved as your new image.

Then, follow these steps:

1. Choose an existing AMI, for example GlobalRelease...
2. Choose Instance Type, for example: Family = Compute optimized, Type = c5.4xlarge, vCPUs = 16, Memory (GiB) = 32.
3. Keep other options as default, and click "Review and Launch".
4. Review the information of the new instance, and click "Launch".

Next, you are asked to choose a key pair for your instance. It's recommended to create a new key pair for the first time, then use it in the future. That is the keypair that is saved, for example in .ssh.

Now, you can select "Choose an existing key pair", and select your key pair that should be visible. Check the box for acknowledging the use of a private key. Finally, your new instance is launched. Note that this new instance is an already existing Ubuntu server/image. This image is the one we will update.

Now you can access your new instance (the remote Ubuntu server) using X2GO. Steps to do this can be found in Chapter 211. After you established remote control of the Ubuntu server on AWS, the build procedures are the same as those for local installation, which can be found in previous sections of this chapter.

The last step is to create a new image from this instance so that you can launch it in the future. Go the "Instances", and choose the correct instance. Click "Actions", and select "Create Image" under "Image". You can change the size of the instance volume, but it's not necessary at this moment. Give your image a (new) name and a description, and click "Create Image". Now you have sucessfully created a new image for Real-ESSI. If you go to "AMIs", you should be able to see this new image you just created.

Now you can go to `Software` directory and follow install procedures from section 1.6 on page 14.

After compiling and linking both sequential and parallel Real-ESSI, and install them on /usr/bin (follow procedures for build), and delete source code (!), one can make this instance into a new image. Create new image inside AWS EC2 Management Console Browser window. Select the launched Image with the above software installed. Choose Actions → Image → Create Image. Type your Image Name and descriptions. Click Create Image. This might take some time. You will then see your image in EC2 Dashboard → Images → AMIs (on the left side bar).

Make sure that you terminate all the running instances so that you do not get charged. Find: Action, Instance State, Terminate.

1.13.6 Upload an Existing Real-ESSI Simulator Image to AWS MarketPlace

- Copy to private image for region North Virginia
- Go to the AWS market place `https://aws.amazon.com/marketplace`,
- Choose sell in AWS marketplace,
- Choose AMIs selection the new private in Region North Virginia to publish.
- Proceed until finalizing the AWS Marketplace Image.

Bibliography

D. Abrahams and A. Gurtovoy. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*. C++ in Depth Series. Addison-Wesley., 2005. ISBN 0321227255.

S. S. Deshpande, L. Anumolu, and M. F. Trujillo. Evaluating the performance of the two-phase flow solver interfoam. *Computational Science & Discovery*, 5(1):014016, 2012.

B. Jeremić, Z. Yang, Z. Cheng, G. Jie, N. Tafazzoli, M. Preisig, P. Tasiopoulou, F. Pisanò, J. Abell, K. Watanabe, Y. Feng, S. K. Sinha, F. Behbehani, H. Yang, and H. Wang. *Nonlinear Finite Elements: Modeling and Simulation of Earthquakes, Soils, Structures and their Interaction*. University of California, Davis, CA, USA, 1989-2022. ISBN 978-0-692-19875-9. URL: http://sokocalo.engr.ucdavis.edu/~jeremic/LectureNotes/.

R. Ramey. Making a boost library. In D. Musser, editor, *Library-Centric Software Design LCSD'05*. Object-Oriented Programming, Systems, Languages and Applications, October 2005.