# Contents


<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Chapter Summary and Highlights</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Introduction to the Real-ESSI Simulator Program</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Real-ESSI Program Executables Download and Install</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Real-ESSI Simulator System Install</td>
<td>6</td>
</tr>
<tr>
<td>1.5 Quick Build Procedures for Sequential and Parallel Versions of the Real-ESSI Program</td>
<td>6</td>
</tr>
<tr>
<td>1.5.1 Sequential Real ESSI Build</td>
<td>7</td>
</tr>
<tr>
<td>1.5.2 Parallel Real ESSI Build</td>
<td>10</td>
</tr>
<tr>
<td>1.6 Build Procedures for Sequential and Parallel Versions of the Real-ESSI Program</td>
<td>13</td>
</tr>
<tr>
<td>1.6.1 Libraries and Application Build Process</td>
<td>13</td>
</tr>
<tr>
<td>1.6.2 Installing Libraries</td>
<td>14</td>
</tr>
<tr>
<td>1.6.3 Obtaining Real-ESSI and Source organization</td>
<td>14</td>
</tr>
<tr>
<td>1.6.4 Real-ESSI Dependencies Build Process</td>
<td>15</td>
</tr>
<tr>
<td>1.6.5 Building Dependencies from Source</td>
<td>24</td>
</tr>
<tr>
<td>1.6.6 Compiling Real-ESSI Program Source</td>
<td>26</td>
</tr>
<tr>
<td>1.6.7 Sequential Version</td>
<td>26</td>
</tr>
<tr>
<td>1.6.8 Parallel Version</td>
<td>27</td>
</tr>
<tr>
<td>1.7 Code Verification After the Build Process</td>
<td>28</td>
</tr>
<tr>
<td>1.7.1 Run all verification test cases</td>
<td>28</td>
</tr>
<tr>
<td>1.7.2 Test Sequential Real-ESSI</td>
<td>28</td>
</tr>
<tr>
<td>1.7.3 Test Parallel Real-ESSI</td>
<td>29</td>
</tr>
<tr>
<td>1.7.4 Version Stability Test</td>
<td>29</td>
</tr>
<tr>
<td>1.7.5 Memory Management Test</td>
<td>29</td>
</tr>
<tr>
<td>1.8 Compiling Real-ESSI Utilities</td>
<td>30</td>
</tr>
<tr>
<td>1.8.1 Installation of gmsh and gmESSI</td>
<td>31</td>
</tr>
</tbody>
</table>
1.8.2 Installation of ParaView and pvESSI ........................................ 32
1.9 Sublime Text Editor ........................................................................... 35
1.10 Model Conversion/Translation using FeConv .................................... 35
1.11 Build Procedures on Amazon Web Service ....................................... 35
  1.11.1 Sign In to AWS .......................................................................... 36
  1.11.2 Copy an Existing Image ............................................................... 36
  1.11.3 Create a New Image .................................................................. 36
  1.11.4 Build AWS ESSI Image from Scratch ......................................... 38
  1.11.5 Update an Existing Image ......................................................... 42
  1.11.6 Upload an Existing Real ESSI Simulator Image to AWS MarketPlace 43
Chapter 1

Software Platform Build and Install Procedures

(In collaboration with Dr. José Abell, Mr. Sumeet Kumar Sinha, Mr. Yuan Feng, Dr Han Yang and Mr 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. Pronunciation of Real ESSI is similar to “real easy”, while translation of Real ESSI to other languages (languages of developers and users) is also available: Врло Просто, Muy Fácil, Molto Facile, अति सहज, 真简单, 本当に簡単, Πραγματικά Εύκολο, बहुत ही आसान, Très Facile, Вистински Лесно, Wirklich Einfach, سهل جداً.

1.3 Real-ESSI Program Executables Download and Install

Executables for the Real ESSI Simulator program are available online. Pre-built executables are available for Linux, Ubuntu 18.04, and can be downloaded and installed by analyst.

In order for prebuild executables to be able to run on a user/analyst computer, system libraries have to be brought up to date and additional libraries installed. System libraries update/upgrade:

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

For sequential and/or parallel version of Real-ESSI, additional libraries are needed, as described below:

**Sequential Version of Real-ESSI Program.** Libraries required to be installed for using sequential version of the Real ESSI program:

```
sudo apt-get install libboost-all-dev
sudo apt-get install libhdf5-dev
sudo apt-get install libtbb-dev
```

**Parallel Version of Real-ESSI Program.** Libraries required to be installed for executing parallel version of the Real ESSI program:

```
sudo apt-get install libboost-all-dev
sudo apt-get install libhdf5-dev
sudo apt-get install libtbb-dev
sudo apt-get install mpich
sudo apt-get install libopenmpi-dev
```
Real-ESSI Executable Downloads. The Real-ESSI program executables can be downloaded from Real-ESSI Simulator web site: http://real-essi.info/. Alternatively, contact Prof. Jeremić to arrange for customized Real-ESSI executables.

1.4 Real-ESSI Simulator System Install

In addition 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 1045 of the main document, lecture notes (Jeremić et al., 1989-present). Installation of post-processing modules is described in Chapter 208, on page 1110 of the main document, lecture notes (Jeremić et al., 1989-present).

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

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

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.

Update Compiler for Ubuntu 16.04, while 18.04 already has the latest version: Ubuntu 16.04 does not have the gcc-7 and thus this new version of a compiler needs to be installed:

```
sudo apt-get install build-essential
sudo add-apt-repository ppa:ubuntu-toolchain-r/test
sudo apt-get update
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:

```bash
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:

```bash
1 gcc --version
2 g++ --version
3 cpp --version
4 gfortran --version
```

1.5.1 Sequential Real ESSI Build

**Libraries Required for Building Dependencies for Ubuntu 16.04 and 18.04.**

```bash
1 sudo apt-get install -y cmake
2
3 sudo apt-get install build-essential
4 sudo apt-get install flex
5 sudo apt-get install bison
6 sudo apt-get install libboost-all-dev
7 sudo apt-get install libtbb-dev
8 sudo apt-get install valgrind
9 sudo apt-get install libopenblas-dev
10 sudo apt-get install liblapack-dev
11 sudo apt-get install libpthread-workqueue-dev
12 sudo apt-get install zlib1g-dev
13 sudo apt-get install libopenmpi-dev
```

**Libraries Required for Building Utilities for Ubuntu 16.04 and 18.04.**

```bash
1 sudo apt-get install liboctave-dev
2 sudo apt-get install libhdf5-serial-dev
3 sudo apt-get install hdf5-tools
4 sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial.so /usr/lib/libhdf5_serial.so
5 sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial_hl.so /usr/lib/libhdf5_serial_hl.so
6 sudo apt-get install libphonon-dev
7 sudo apt-get install libphonon4
8 sudo apt-get install qt4-dev-tools
9 sudo apt-get install qt4-qmake
10 sudo apt-get install libx11-dev
11 sudo apt-get install libqt4-opengl-dev
```
12 sudo apt-get install mesa-common-dev
13 sudo apt-get install python-dev
14 sudo apt-get install python-h5py
15 sudo apt-get install python-matplotlib
16 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
git pull
```

or alternatively:

```
mkdir RealESSI_ROOT
#
# 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
```  

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__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, 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

```
1 time ./build_libraries check_sequential
```

**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_sequential
4 mkdir build_sequential
5 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.

```
1 time cmake ..
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.sequential
```

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

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

### 1.5.2 Parallel Real ESSI Build

**Libraries Required for Building Dependencies for Ubuntu 16.04 and 18.04.**

```
1 sudo apt-get install -y cmake
2
3 sudo apt-get install build-essential
4 sudo apt-get install flex bison
5 sudo apt-get install libboost-all-dev
6 sudo apt-get install libtbb-dev
7 sudo apt-get install valgrind
8 sudo apt-get install libopenblas-dev
9 sudo apt-get install liblapack-dev
10 sudo apt-get install libpthread-workqueue-dev
11 sudo apt-get install zlib1g-dev
```
12 sudo apt-get install libssl-dev
13 sudo apt-get install mpich
14 sudo apt-get install libopenmpi-dev


1 sudo apt-get install liboctave-dev
2 sudo apt-get install libhdf5-serial-dev
3 sudo apt-get install hdf5-tools
4 sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial.so →
   /usr/lib/libhdf5.so
5 sudo ln -sf /usr/lib/x86_64-linux-gnu/libhdf5_serial_hl.so →
   /usr/lib/libhdf5_hl.so
6 sudo apt-get install libphonon-dev
7 sudo apt-get install libphonon4
8 sudo apt-get install qt4-dev-tools
9 sudo apt-get install qt4-qmake
10 sudo apt-get install libxt-dev
11 sudo apt-get install libqt4-opengl-dev
12 sudo apt-get install mesa-common-dev
13 sudo apt-get install python-dev
14 sudo apt-get install python-h5py
15 sudo apt-get install python-matplotlib
16 sudo apt-get install python-scipy

Obtain Real-ESSI Sources

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

1 mkdir RealESSI_ROOT/
2 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):

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:
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
#
rmdir 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
Check that all prerequisite libraries are built

 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/mpicc -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

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

1.6.1 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.

### 1.6.2 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.

### 1.6.3 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.

```bash
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
  |   
3 + Real-ESSI
  |   
5 + RealESSI_Dependencies
  |   
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.

### 1.6.4 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 RealESSI output

- **LAPACK 3.6.1** Standard linear algebra suite

- **ParMETIS 4.0.3** Software for graph partitioning used in parallel RealESSI
• **PETSc 3.7.3** High performance, parallel suite of system of linear equations solvers used in parallel RealESSI.

• **SuiteSparse 4.5.3** Provides interfaces into system of equations linear solvers used in sequential RealESSI.

• **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

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

```
1  ./build_libraries help
2  #---------- result from the terminal -------------------
3  #
4  # Miscellaneous:
5  # list_dependencies Lists all the available ↔
6  # dependencies version from SRC folder
7  # list_build_dependencies Lists all the dependencies library ↔
8  # already build in lib folder
9  # help Show this help.
10 # download Downloads the Dependencies Sources
11 #
12 # Parallel:
13 # parallel Builds all the necessary libraries ↔
14 # for parallel Real-ESSI
15 # hdf5_parallel Builds parallel hdf5
16 # petsc Builds petsc
17 # clean_parallel Cleans parallel libraries
18 #
19 # Sequential:
20 # sequential Builds all the necessary libraries ↔
21 # for sequential Real-ESSI
22 # parmetis Builds parmetis andmetis
```
<table>
<thead>
<tr>
<th>target</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td># suitesparse</td>
<td>Builds suitesparse</td>
</tr>
<tr>
<td># arpack</td>
<td>Builds arpack</td>
</tr>
<tr>
<td># hdf5_sequential</td>
<td>Builds sequential hdf5</td>
</tr>
<tr>
<td># lapack</td>
<td>Builds lapack</td>
</tr>
<tr>
<td># atlas</td>
<td>Builds tuned LAPACK and BLAS</td>
</tr>
<tr>
<td># clean_sequential</td>
<td>Cleans sequential libraries</td>
</tr>
<tr>
<td># clean_parallel</td>
<td>Cleans parallel libraries</td>
</tr>
<tr>
<td># clean_sequential</td>
<td>Cleans sequential libraries</td>
</tr>
<tr>
<td># clean_parallel</td>
<td>Cleans parallel libraries</td>
</tr>
<tr>
<td># clean_all</td>
<td>Cleans everything</td>
</tr>
<tr>
<td># check_sequential</td>
<td>Checks if all sequential libraries are build</td>
</tr>
<tr>
<td># check_parallel</td>
<td>Checks if all parallel libraries are build</td>
</tr>
<tr>
<td># clean_parmetis</td>
<td>Clean parmetis and metis</td>
</tr>
<tr>
<td># clean_suitesparse</td>
<td>Clean suitesparse libraries</td>
</tr>
<tr>
<td># clean_arpack</td>
<td>Clean arpack libraries</td>
</tr>
<tr>
<td># clean_hdf5_sequential</td>
<td>Cleans hdf5_sequential libraries</td>
</tr>
<tr>
<td># clean_lapack</td>
<td>Cleans lapack</td>
</tr>
<tr>
<td># clean_hdf5_parallel</td>
<td>Cleans hdf5_parallel libraries</td>
</tr>
<tr>
<td># clean_petsc</td>
<td>Cleans petsc</td>
</tr>
</tbody>
</table>

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

```bash
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):

```bash
./build_libraries sequential
```

Similar procedures is used for building parallel libraries:

```bash
./build_libraries parallel
```

The user can also check whether all the libraries are successfully built:
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 1.6.5 on page 24.

**[Optional Atlas Tuning :: Recommended for High Performance]**

RealESSI 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 RealESSI.

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 RealESSI. 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:

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

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

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

and add the following line to it:

```
1 GOVERNOR="performance"
```

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

```
1 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 22 where we continue to describe ATLAS build process.

**A Note on Intel CPUs and Linux Kernel ≥ 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:

```
in tel_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 = "0x01234567,0xefefefef,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

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

Reboot and confirm that this worked by running

```bash
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:

```bash
./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:

```bash
./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:

```bash
./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](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)

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

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

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

It might help if you do:

\(^1\)THIS IS NOT TO BE USED FOR PARALLEL VERSION OF Real-ESSI AS YOU WILL BE EXPLICITLY USING THOSE MULTIPLE CORES YOURSELF!
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:

```bash
export RealESSI_Dependencies_PATH=set_to_appropriate_path
```

or for tcsh do:

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

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

```bash
set RealESSI_Dependencies_PATH=/home/jeremic/oofep/Rad_na_Sokocalu/ESSIforOT
```

cHECK AGAIN YOUR CPU FREQUENCY:

```bash
cpufreq-info
```

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

```bash
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.tgz
```

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 RealESSI dependencies tarball), and that is where we have it in RealESSI dependencies and

For nagoyqqatsi computer, this last line is

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

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

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

For Boris' laptop:

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

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:

```
1 time make
```

You can also perform some testing:

- sanity check correct answer:

```
1 make check
```

- sanity check parallel

```
1 make ptcheck
```

- check if lib is fast
Once this is done, install:

1 make install

This creates the ‘lib’ and ‘include’ directories in the RealESSI_Dependencies_PATH which are expected by the RealESSI build system and needed for placing the rest of the libraries dependencies. Compilation will not continue without these.

### 1.6.5 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**

  1. ./build_libraries NPROC=8

- **Build only sequential Libraries**

  1. ./build_libraries sequential NPROC=8

- **Build only parallel libraries Libraries**

  1. ./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_cpu_freq = 2000 MH) the commands would be

• Build all Libraries

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

• Build only sequential Libraries

1 ./build_libraries sequential TUNED_ATLAS=ON ←
   CPU_MAX_FREQ=2000 NPROC=20

• Build only parallel libraries Libraries

1 ./build_libraries parallel TUNED_ATLAS=ON ←
   CPU_MAX_FREQ=2000 NPROC=20

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

• Checking Parallel Libraries

1 ./build_libraries check_parallel

• Showing all libraries build

1 ./build_libraries list_build_dependencies
1.6.6 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  time cmake ..
4  time make -j 16
```

1.6.7 Sequential Version

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

```
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\textsuperscript{2} is done by executing:

```
1  mkdir build
2  cd build
3  time cmake ..
4  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

\textsuperscript{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 RealESSI executable to a folder in the system’s path.
For example:

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

This will make RealESSI 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.

### 1.6.8 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:

```bash
1 sudo apt-get install build-essential
2 sudo apt-get install cmake
3 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 RealESSI needs execute the command in Real-ESSI source code folder.

```bash
1 mkdir pbuild && cd pbuild
2 cmake -DCMAKE_CXX_COMPILER=/usr/bin/mpic++ ←
   -DCMAKE_CXX_COMPILER_PARALLEL=PARALLEL ..
3 make -j 32
```

Once done, parallel RealESSI 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 RealESSI executable to a folder in the system’s path. For example:

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

This will make RealESSI 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...).
1.7 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.

1.7.1 Run all verification test cases

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

1. cd $RealESSI_PATH/
2. 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

1. cd $RealESSI_PATH/
2. bash clean_all_verification.sh

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

1.7.2 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.

1. cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/analytic_solution
2. 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.
1.7.3 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.

```
1 cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/parallel
2 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.

1.7.4 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,

```
1 cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/version_stability
2 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

```
1 cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/version_stability
2 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.

1.7.5 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.

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

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

You can also download the source of valgrind and compile it from scratch.
In order to track the memory leak,

```
1 cd $RealESSI_PATH/CompGeoMechUCD_Miscellaneous/examples/valgrind_test
2 bash make_comparison.sh
```

This bash script run all the examples in the environment of valgrind. The essi results will be compared to the analytic solution. The memory management report is saved in the same folder of the real essi example.

For the details about the code stability verification, please refer to the Section 303.2 on Page 1244.

### 1.8 Compiling Real-ESSI Utilities

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 avialable 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

```
1 ./build_utilities help
2
3 #usage: make [target]
4#
5 #Utilities:
6 #  gmeshi               Builds gmeshi
7 #  paraview            Builds paraview
8 #  pvessi              Builds pvessi
9 #  gmsh                Builds gmsh
10 #  visit               Builds visit
11 #  visitessi           Builds visitessi
12#
13 #Sequential:
14 #  clean_utilities   Cleans all utilities
15#
16 #Default:
17 #  all               Builds all the necessary utilities ←
18 #      for REAL-ESSI
19 #  all_utils         Builds all the necessary utilities ←
20 #      for REAL-ESSI
21 #  clean_all         Cleans everything
```
Jeremić et al. Real-ESSI Simulator 31 of 44

# 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

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

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

### 1.8.1 Installation of gmsh and gmESSI

gmsh is 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:

```bash
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 teh main document, lecture notes (Jeremić et al., 1989-present). 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. To install gmessi, go to the Real-ESSI directory and then run
To update the utility at the user just needs to run

```
./build_utilities update_gmessi
```

Refer to section 1.9 on page 35 for instructions on what and how to install autocompletion and syntax coloring for gmESSI and Real-ESSI syntax on sublime text editor.

### 1.8.2 Installation of ParaView and pvESSI

ParaView package [http://www.paraview.org/](http://www.paraview.org/) is a powerful multi-platform data analysis and visualization application available 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 from Source

1. Install dependencies

```
1 sudo apt-get install libphonon-dev libphonon4 qt4-dev-tools
2 sudo apt-get install libqt4-core libqt4-gui qt4-qmake libxt-dev
3 sudo apt-get install g++ gcc cmake-curses-gui libqt4-opengl-dev
4 sudo apt-get install mesa-common-dev python-dev
5 sudo apt-get install libvtk6.2
```

2. After successful installation of dependencies, download the source files of ParaView. To ensure version compatibility, ParaView v5.2.0 can be downloaded from local server. Create and to the directory where you want to put the Real-ESSI utilities and follow the commands shown below. Make sure the directory is clean before the installation.

```
1 mkdir ParaView
2 cd ParaView
3 #
4 # using curly brackets to help in checking scripts, that rely on these
5 # brackets being available around URL
6 #
7 wget {http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Utilities_SRC.tar.gz}
8 tar -xzf ./Utilities_SRC.tar.gz
9 rm -rf Utilities_SRC.tar.gz
```
10  mkdir ParaView_INSTALL
11  tar -xzvf Utilities_SRC/ParaView-v5.2.0-RC1.tar.gz -C ParaView_INSTALL --strip-components 1
12  rm -rf Utilities_SRC

3. Compile ParaView:

1  cd ParaView_INSTALL
2  mkdir build
3  cd build
4  cmake .. - DPARAVIEW_USE_MPI=ON - DPARAVIEW_ENABLE_PYTHON=ON
5  make -j 4

4. ParaView executable is in /Paraview/build/bin. It is recommended to create a symbolic link from the ParaView executable to a folder in system path. You need to be an administrator to do so.

1  cd bin
2  sudo ln -sf ~/FULL_INSTALL_DIRECTORY/paraview /usr/local/bin/
3  # for example
4  sudo ln -sf
   ~/oofep/Rad_na_Sokocalu/ParaView/ParaView_INSTALL/build/bin/paraview
   /usr/local/bin/

Building pvESSI Plugin

1. By placing source files of pvESSI in Plugins directory, pvESSI will be automatically compiled together with ParaView. Start by going to ParaView Plugins directory

1  cd ../../Plugins

2. Obtain the latest version of the pvESSI plugin:

1  #
2  # using curly brackets to help in checking scripts, that rely on these
3  # brackets being available around URL
4  #
5  wget
   {http://sokocalo.engr.ucdavis.edu/~jeremic/RealESSI/Utilities_SRC.tar.gz
6  tar -xzvf Utilities_SRC.tar.gz
7  tar -xzvf Utilities_SRC/pvESSI-master.tar.gz
8  rm -rf Utilities_SRC.tar.gz
9  rm -rf Utilities_SRC
10 %tar -xzvf Utilities_SRC/pvESSI-master.tar.gz

The pvESSI can be downloaded from http://sokocalo.engr.ucdavis.edu/~jeremic/lecture_
3. Recompile part of ParaView with the pvESSI plugin present:

```
1 cd ../build
2 cmake .. -DPARAVIEW_USE_MPI=ON -DPARAVIEW_ENABLE_PYTHON=ON
3 make -j 4
```

4. ParaView is now compiled with pvESSI.

5. Load pvESSI plugin into ParaView:
   
   - open ParaView
   - Click on Tools $\rightarrow$ Manage Plugins (See Figure 1.1)

   ![Figure 1.1: Open ParaView plugin-manager.](image)

   - Find pvESSI and click "Load Selected", see Figure 1.2,
   - Click on pvESSI and check on the "Auto Load" option, see Figure 1.2,
   - Close ParaView
   - Start Paraview again. pvESSI is now part of automatically loaded plugins for ParaView and Real-ESSI results can be loaded by loading Real-ESSI HDF5 outp files that are named after model and stage name and have extension .h5.feioutput.
1.9 Sublime Text Editor

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 RealESSI.
- **gmESSI-Tools**, syntax and autotext completion for gmESSI model development tools for RealESSI.

1.10 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.

1.11 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 1187 in Jeremić et al. (1989-present).

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)

1.11.1 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.

1.11.2 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.

1.11.3 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:


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".

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

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

```bash
1 sudo apt-get install software-properties-common
2 sudo apt-get install ppa:x2go/stable
3 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.

```bash
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 successfully created a new image for Real-ESSI. If you go to "AMIs", you should be able to see this new image you just created.
1.11.4 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.

   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.

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

3. Install Desktop and git on AWS Remote Instance
   
   ```bash
   sudo apt update
   sudo apt install -y ubuntu-desktop git
   ```

4. Install remote-desktop-server (x2goserver) on AWS Remote Instance
   
   ```bash
   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
   
   ```bash
   sudo systemctl enable x2goserver.service
   sudo systemctl start x2goserver.service
   ```

6. Install ESSI
   
   ```bash
   # 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/ _all_files_gmESSI_.tg z
mkdir Real-ESSI-gmESSI
mv _all_files_gmESSI_.tg z 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 pvESSI plugin

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

# Download the ParaView
#
# using curly brackets to help in checking scripts, that rely on →
10. Install Sublime Text 3 and ESSI plugin. Following this link.

```
# # using curly brackets to help in checking scripts, that rely on ➝ # # 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
ds 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

### 1.11.5 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 successfully 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 1.5 on page 6.

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.

1.11.6 Upload an Existing Real ESSI Simulator Image to AWS MarketPlace

- Copy to private image for region North Virginia

- Go to the AWS marketplace 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

