Installing SW4 version 1.18

N. Anders Petersson* Björn Sjögreen* Eric Heien†

October 16, 2014
Revised December 11, 2015

Contents

1 Introduction

The sole purpose of this document is to describe the installation process of the seismic wave propagation code *SW4*. A comprehensive user's guide of this code is provided in the report by Petersson and Sjogreen [1].

*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, PO Box 808, Livermore CA 94551. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. This is contribution LLNL-SM-662989.

†Computational Infrastructure for Geodynamics, University of California, Davis.

2 Compilers and third party libraries

Before you can build *SW4* on your system, you must have

1. the `lapack` and `blas` libraries. These libraries provide basic linear algebra functionality and are pre-installed on many machines;

2. a MPI-2 library. This library provides support for message passing on parallel machines. Examples of open source implementations include Mpich-2 and OpenMPI. Note that the MPI-2 library must be installed even if you are only building *SW4* for a single core system.

To avoid incompatibility issues and linking problems, we recommend using the same compiler for the libraries as for *SW4*.

For a complete installation that supports projections from the Proj.4 library and material models from an e-tree database, you need to download and install three additional libraries:

- The Proj.4 library, `http://trac.osgeo.org/proj`
- The Euclid e-tree library, `http://www-2.cs.cmu.edu/ euclid`
- The cencalvm library,
  `http://earthquake.usgs.gov/data/3dgeologic/cencalvm_doc/index.html`

These libraries need to be installed under the same directory, such that each library installs its files in the `lib` and `include` sub-directories. See Section 6 for details.

Mac computers We recommend using the MacPorts package manager for installing the required compilers and libraries. Simply go to www.macports.org, and install macports on your system. With that in place, you can use the `port` command as follows

```
shell> sudo port install gcc48
shell> sudo port select --set gcc mp-gcc48
shell> sudo port install openmpi +gcc48
shell> sudo port select --set mpi openmpi-gcc48-fortran
```

Here, `gcc48` refers to version 4.8 of the Gnu compiler suite. Compiler versions are bound to change in the future, so the above commands will need to be modified accordingly. Before starting, make sure you install a version of gcc that is compatible with the MPI library package. The above example installs the `openmpi` package with the `gcc48` variant, which includes a compatible Fortran compiler. Alternatively, you can use the `mpich2` package. Note that the `port select` commands are used to create shortcuts to the compilers and MPI environment. By using the above setup, the Gnu compilers can be accessed with `gcc` and `gfortran` commands, and the MPI compilers and execution environment are called `mpicxx`, `mpif77`, and `mpirun`, respectively.

The `lapack` and `blas` libraries are preinstalled on recent Macs and can be accessed using the `-framework vecLib` link option. If that is not available or does not work on your machine, you can download `lapack` and `blas` from www.netlib.org.

Linux machines We here give detailed instructions for installing the third part libraries under 64 bit, Fedora Core 18 Linux. Other Linux variants use similar commands for installing software packages, but note that the package manager `yum` is specific to Fedora Core.

You need to have root privileges to install precompiled packages. Start by opening an xterm and set your user identity to root by the command

```
su -
```

Install the compilers by issuing the commands

```
yum install gcc
yum install gcc-c++
yum install gcc-gfortran
```

You install the `mpich2` library and include files with the command

```
yum install mpich2-devel
```

The executables and libraries are installed in `/usr/lib64/mpich2/bin` and `/usr/lib64/mpich2/lib` respectively. We suggest that you add `/usr/lib64/mpich2/bin` to your path. This is done with the command

```
export PATH=${PATH}:/usr/lib64/mpich2/bin
```

if your shell is `bash`. For `tcsh` users, the command is

```
setenv PATH ${PATH}:/usr/lib64/mpich2/bin
```

It is convenient to put the path setting command in your startup file, `.bashrc` or `.cshrc`., for bash or csh/tcsh respectively.

The `blas` and `lapack` libraries are installed with

```
yum install blas
yum install lapack
```

On our system, the libraries were installed in `/usr/lib64` as `libblas.so.3` and `liblapack.so.3`. For some unknown reason, the install program does not add links to these files with extension `.so`, which is necessary for the linker to find them. We must therefore add the links explicitly. If the libraries were installed elsewhere on your system, but you don't know where, you can find them with the following command:

```
find / -name "*blas*" -print
```

After locating the directory where the libraries reside (in this case `/usr/lib64`), we add links to the libraries with the commands:

```
cd /usr/lib64
ln -s libblas.so.3 libblas.so
ln -s liblapack.so.3 liblapack.so
```

Note that you need to have `root` privileges for this to work.

3 Unpacking the source code tar ball

To unpack the *SW4* source code, you place the file `sw4-v1.1.tgz` in the desired directory and issue the following command:

```
shell> tar xzf sw4-v1.1.tgz
```

As a result a new sub-directory named `sw4-v1.1` is created. It contains several files and sub-directories:

- `LICENSE.txt` License information.
- `INSTALL.txt` A link to this document.
- `README.txt` General information about *SW4*.
- `configs` Directory containing `make` configuration files.
- `src` C++ and Fortran source code of *SW4*.
- `tools` Matlab/Octave scripts for post processing and analysis.
- `examples` Sample input files.
- `Makefile` Main makefile (don't change this file!).
- `CMakeLists.txt` CMake configuration file (don't change this file either!).
- `wave.txt` Text for printing the "SW4 Lives" banner at the end of a successful build.

4 Installing *SW4* with make

The classical way of building *SW4* uses `make`. We recommend using GNU make, sometimes called `gmake`. You can check the version of make on you system with the command

```
shell> make -v
```

If you don't have GNU make installed on your system, you can obtain it from www.gnu.org.

We have built *SW4* and its supporting libraries on Intel based laptops and desktops running LINUX and OSX. It has also been built on several supercomputers such as the Intel machines `cab` (at LLNL) and `edison` (at LBNL), as well as the IBM BGQ machine `vulcan` at LLNL. We have successfully used the following versions of Gnu, Intel, and IBM compilers:

```
Gnu:   g++/gcc/gfortran      versions 4.5 to 4.8
Intel: icpc/icc/ifort        version 12.1
IBM Blue Gene: xlc++/xlc/xlf version 12.1
```

SW4 uses the message passing interface (MPI) standard (MPI-2 to be specific) for communication on parallel distributed memory machines. Note that the MPI library often includes wrappers for compiling, linking, and running of MPI programs. For example, the `mpich2` package includes the `mpicxx` and `mpif77` compilers, as well as the `mpirun` script. We highly recommend using these programs for compiling, linking, and running *SW4*.

4.1 Basic compilation and linking of *SW4*

The best way of getting started is to first build *SW4* without the proj.4 and cencalvm libraries. This process should be very straight forward and the resulting *SW4* executable will support all commands except `efile` and the `proj`/`ellps` options in the `grid` command. If you need to use these options, you can always recompile *SW4* after the proj.4 and cencalvm libraries have been installed. See § 6 for details.

The basic build process is controlled by the environmental variables `FC`, `CXX`, `EXTRA_FORT_FLAGS`, `EXTRA_CXX_FLAGS`, and `EXTRA_LINK_FLAGS`. These variables should hold the names of the Fortan-77 and C++ compilers, and any extra options that should be passed to the compilers and linker. The easiest way of assigning these variables is by creating a file in the `configs` directory called `make.inc`. The `Makefile` will look for this file and read it if it is available. There are several examples in the `configs` directory, e.g. `make.osx` for Macs and `make.linux` for Linux machines. You should copy one of these files to your own `make.inc` and edit it as needed.

4.1.1 Mac machines

If you are on a Mac, you could copy the setup from `make.osx`,

```
shell> cd configs
shell> cp make.osx make.inc
shell> cat make.inc
etree = no
proj = no
FC = mpif77
CXX = mpicxx
EXTRA_FORT_FLAGS =
EXTRA_LINK_FLAGS = -framework vecLib -L/opt/local/lib/gcc48 -lgfortran
```

In this case, the `blas` and `lapack` libraries are assumed to be provided by the `-framework vecLib` option. The `libgfortran` library is located in the directory `/opt/local/lib/gcc48`, which is where `macports` currently installs it.

4.1.2 Linux machines

If you are on a Linux machine, we suggest you copy the configuration options from `make.linux`,

```
shell> cd configs
shell> cp make.linux make.inc
shell> cat make.inc
FC = gfortran
CXX = mpicxx
EXTRA_LINK_FLAGS = -L/usr/lib64 -llapack -lblas -lgfortran
```

This setup assumes that the `blas` and `lapack` libraries are located in the `/usr/lib64` directory. In the case of Fedora Core 18, we needed to set the link flag variable to

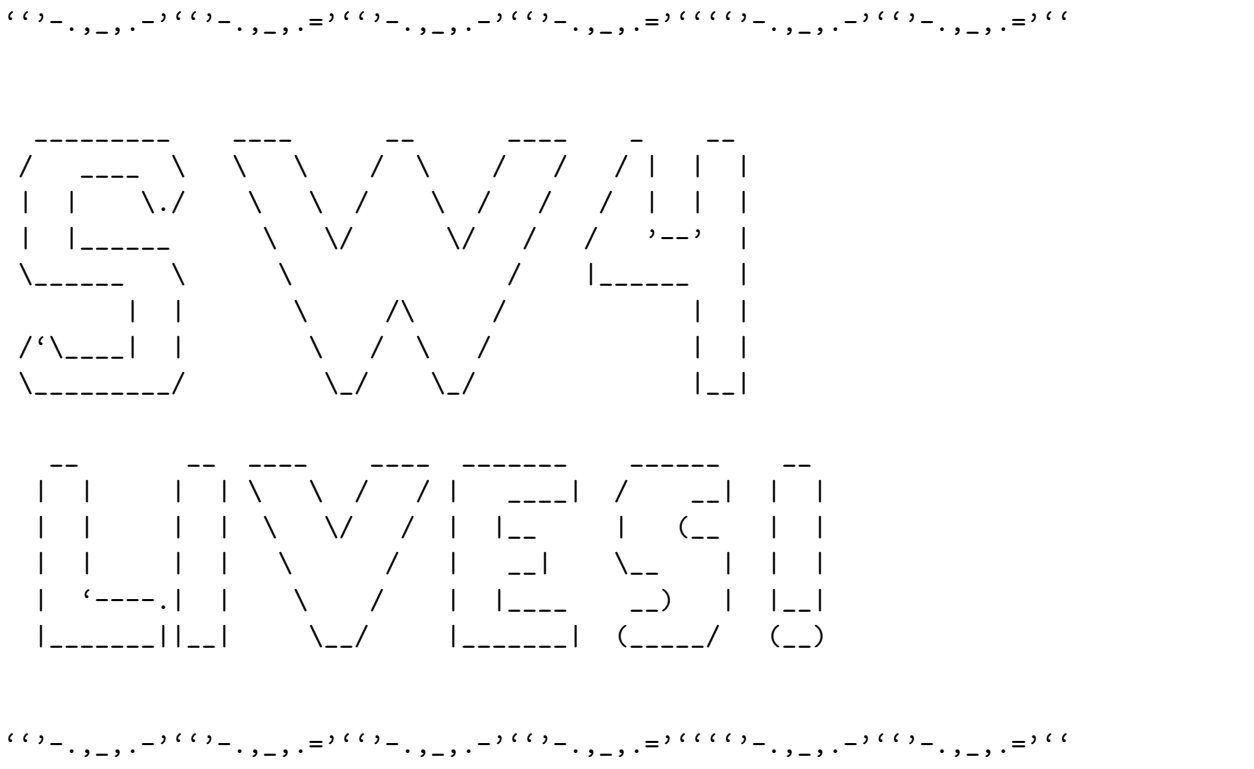
```
EXTRA_LINK_FLAGS = -Wl,-rpath=/usr/lib64/mpich2/lib  -llapack -lblas -lgfortran
```

4.1.3 Using make

You build *SW4* with the "make" command from the main directory.

```
shell> cd /enter/your/path/sw4-v1.1
shell> make
```

If all goes well, you will see the SW4 Lives banner on your screen after the compilation and linking has completed,



By default, `make` builds an optimized `sw4` executable. It is located in

```
/enter/your/path/sw4-v1.1/optimize/sw4
```

You can also build an executable with debugging symbols by adding the `debug=yes` option to `make`,

```
shell> cd /enter/your/path/sw4-v1.1
shell> make debug=yes
```

In this case, the executable will be located in

```
/enter/your/path/sw4-v1.1/debug/sw4
```

It can be convenient to add the corresponding directory to your `PATH` environment variable. This can be accomplished by modifying your shell configuration file, e.g. `~/.cshrc` if you are using C-shell.

4.1.4 How do I setup the `make.inc` file?

The input file for `make` is

`sw4-v1.1/Makefile`

Do *not* change this `Makefile`. It should only be necessary to edit your configuration file, that is,

`/my/path/sw4-v1.1/configs/make.inc`

Note that you must create this file, for example by copying one of the `make.xyz` files in the same directory. The `make.inc` file holds all information that is particular for your system, such as the name of the compilers, the location of the third party libraries, and any extra arguments that should be passed to the compiler or linker. This file also tells `make` whether or not the `cencalvm` and `proj.4` libraries are available and where they are located.

The following `make.inc` file includes all configurable options:

```
etree = no
proj = no
SW4ROOT = /Users/petersson1

CXX = mpicxx
FC  = mpif77

EXTRA_CXX_FLAGS = -DUSING_MPI
EXTRA_FORT_FLAGS = -fno-underscoring
EXTRA_LINK_FLAGS = -framework vecLib
```

The `etree` variable should be set to `yes` or `no`, to indicate whether or not the cencalvm and related libraries are available. The `SW4ROOT` variable is only used when `etree=yes`. The `CXX` and `FC` variables should be set to the names of the C++ and Fortran compilers, respectively. Finally, the `EXTRA_CXX_FLAGS`, `EXTRA_FORT_FLAGS`, and `EXTRA_LINK_FLAGS` variables should contain any additional arguments that need to be passed to the C++ compiler, Fortran compiler, or linker, on your system.

4.2 Building `sw4` with proj.4 and/or efile support

The installation of the proj.4, euclid, and cencalvm libraries is discussed in Section 6. Note that the proj.4 libraray enables the more advanced geographical mapping keywords in the `grid` command and is also required by the `rfile` command. To enable the `efile` command, you have to also install the `euclid` and `cencalvm` libraries. Note that the latter two libraries are only needed by the `efile` command. If you are not planning on using that command, there is no need to install those libraies. This is a change from *SW4* version 1.0.

Once you have successfully installed the proj.4, and optionally the euclid and cencalvm libraries, it should be easy to re-configure *SW4* to use them. Simply edit your configuration file (`make.inc`) by adding two lines to the top of the file, setting the `etree` keyword to `yes` or `no`, as appropriate.

```
proj = yes
etree = no
SW4ROOT = /thid/party/basedir
...
```

You then need to re-compile *SW4*. Go to the *SW4* main directory, clean up the previous object files and executable, and re-run make:

```
shell> cd /my/installation/dir/sw4-v1.1
shell> make clean
shell> make
```

If all goes well, the "SW4 lives" banner is shown after the make command is completed. As before, the `sw4` executable will be located in the `optimize` or `debug` directories.

5 Installing *SW4* with CMake

SW4 also allows building using CMake. The *SW4* CMake configuration also automates the testing of the code, which ensures that it functions correctly. The tests run a number of input scripts in the `examples` directory and evaluates the accuracy of the numerical solutions using various analytical solutions, see the Appendix of the *SW4* user's guide [1] for details.

To use CMake, change to the sw4 directory and run the following commands:

```
shell> mkdir build
shell> cd build
shell> cmake [options] ..
shell> make
```

The two dots after `cmake [options]` are essential and instructs it to look in the parent directory for the `CMakeLists.txt` file.

The `cmake` command searches for the necessary libraries and other dependencies, then creates makefiles appropriate to your system. Running `make` compiles *SW4* using these makefiles. For details about the exact commands being used in compilation, run `make VERBOSE=1`. Once SW4 is successfully compiled and linked you will see the "SW4 Lives!" banner on the screen. The `sw4` executable resides in the `build/bin` directory.

If you want to rebuild `sw4` with a new set of options, you can force `cmake` to start from scratch by removing the file `CMakeCache.txt` in the `build` directory.

5.1 CMake Options

CMake provides several options to allow customized configuration of *SW4*. To use any option, add `-D<option>=<value>` to the options in the `cmake` command. For example:

```
cmake -DTESTING_LEVEL=1 -DCMAKE_BUILD_TYPE=Debug ..
```

configures *SW4* with testing level 1, to be compiled with debugging symbols in the object files. A list of options is shown in the table below.

Option	Default	Details
PROJ4_HOME	(none)	The path to the Proj.4 installation to use when compiling *SW4*.
CENCALVM_HOME	(none)	The path to the cencalvm installation to use when compiling *SW4*.
CMAKE_BUILD_TYPE	Release	The type of build to setup. Can be either `Debug`, `Release`, or `RelWithDebInfo`. This affects the type of optimization and debug flags used in compiling *SW4*.
TESTING_LEVEL	0	Specifies the testing level for automated tests. Level 0 corresponds to tests that run in roughly a minute or less (7 total), level 1 to tests that run in roughly 10 minutes or less (13 total) and level 2 to tests that may require up to an hour or more (17 total).
MPI_NUM_TEST_PROCS	4	Number of MPI processes to use in tests. Generally using more processes will result in the tests finishing faster, but there is no point exceeding the number of available cores on your system. We strongly recommend at least 8 processes if TESTING_LEVEL is 1 or higher.
MPIEXEC	mpirun	UNIX command for running an MPI application.
MPIEXEC_NUMPROC_FLAG	-np	MPI command option for specifying the number of processes.
MPIEXEC_PREFLAGS	(none)	Extra MPI command option.

5.2 CTest

The *SW4* CMake configuration includes several test cases, which can be used to confirm that the code is working correctly. Each test consists of two parts. First it runs a case using an input file in the `examples/` directory. Secondly, it confirms that the results are within a reasonable error tolerances from previously recorded results.

To run the tests, use either the command `make test` or `ctest` as follows:

```
build > make test
Running tests...
Test project /.../sw4/build
      Start  1: twilight/flat-twi-1.in
 1/14 Test  #1: twilight/flat-twi-1.in ....................   Passed    0.70 sec
      Start  2: twilight/flat-twi-1.in_result_check
 2/14 Test  #2: twilight/flat-twi-1.in_result_check .......   Passed    0.07 sec
      Start  3: twilight/flat-twi-2.in
...
      Start 13: twilight-att/tw-topo-att-1.in
13/14 Test #13: twilight-att/tw-topo-att-1.in .............   Passed    5.23 sec
```

```
      Start 14: twilight-att/tw-topo-att-1.in_result_check
14/14 Test #14: twilight-att/tw-topo-att-1.in_result_check    Passed    0.02 sec

100% tests passed, 0 tests failed out of 14

Total Test time (real) =  50.49 sec
```

You can run tests selectively using `ctest -R <regex>`, for example:

```
build > ctest -R twi-2
Test project /.../sw4/build
    Start 3: twilight/flat-twi-2.in
1/4 Test #3: twilight/flat-twi-2.in ..................   Passed    5.77 sec
    Start 4: twilight/flat-twi-2.in_result_check
2/4 Test #4: twilight/flat-twi-2.in_result_check ....   Passed    0.02 sec
    Start 7: twilight/gauss-twi-2.in
3/4 Test #7: twilight/gauss-twi-2.in ..................   Passed    9.08 sec
    Start 8: twilight/gauss-twi-2.in_result_check
4/4 Test #8: twilight/gauss-twi-2.in_result_check ...   Passed    0.02 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) =  14.90 sec
```

If a test fails you can check the details in the output log at `Testing/Temporary/LastTest.log`.

Modifying the MPI execution commands. By default, `mpirun` is used to start parallel runs when you do `make test`. However, on Livermore computing (LC) machines the command for running MPI programs is `srun`, not `mpirun`. Also, the flag for specifying the number of processors is different, and you must give an additional flag for running interactive jobs on the debug partition. For example, you would say

```
srun -ppdebug -n 128 sw4 inputfile.in
```

to run on the debug partition using 128 cores. To modify the default MPI execution program and other runtime parameters, the variables MPIEXEC, MPIEXEC_NUMPROC_FLAG, and MPIEXEC_PREFLAGS can be set as in the following example:

```
cmake -DTESTING_LEVEL=2 -DMPI_NUM_TEST_PROCS=128 -DMPIEXEC=/usr/bin/srun \
      -DMPIEXEC_NUMPROC_FLAG=-n -DMPIEXEC_PREFLAGS=-ppdebug ..
```

After the proj.4, euclid and cencalvm libraries have been installed (see next section), you need to tell `cmake` where to find them. On the LC-machines, all three libraries are currently installed under `/usr/apps/wpp`, and you can use the following command options to configure `sw4`:

```
cmake -DTESTING_LEVEL=1 -DMPI_NUM_TEST_PROCS=16 -DMPIEXEC=/usr/bin/srun \
      -DMPIEXEC_NUMPROC_FLAG=-n -DMPIEXEC_PREFLAGS=-ppdebug \
      -DPROJ4_HOME=/usr/apps/wpp -DCENCALVM_HOME=/usr/apps/wpp ..
```

To verify that `cmake` actually found the libraries, pay attention to the following lines of the output from the `cmake` command:

```
...
-- Found PROJ4: /usr/apps/wpp/lib/libproj.so
-- Found CENCALVM: /usr/apps/wpp/lib/libcencalvm.so;/usr/apps/wpp/lib/libetree.so
...
```

6 Installing the proj.4, euclid, and cencalvm packages

If you are only interested in using the advanced geographical mapping options of the `grid` command, or the `rfile` command, you only need to install the proj.4 package.

The following instructions describe how to install all three packages. For simplicity all packages are installed under the same top directory. If you are using `cmake`, you may optionally put the proj.4 package in a separate directory. In the following we shall assume that all packages are installed under the same top directory, and that you assign the name of that directory to the environment variable `SW4ROOT`. When you are finished installing the packages, the corresponding include and library files should be in the sub-directories `${SW4ROOT}/include` and `${SW4ROOT}/lib`, respectively.

The cencalvm library was developed by Brad Aagaard at USGS. Instructions for building the cencalvm library as well as downloading the Etree database files for Northern California, can currently be downloaded from

`http://earthquake.usgs.gov/data/3dgeologic/cencalvm_doc/INSTALL.html`

The installation process for cencalvm, which is outlined below, is described in detail on the above web page. Note that cencalvm relies on both the euclid and the proj.4 libraries.

The euclid library must be installed manually by explicitly copying all include files to the include directory and all libraries to the lib directory,

```
shell> cd euclid3-1.2/libsrc
shell> make
shell> cp *.h ${SW4ROOT}/include
shell> cp libetree.* ${SW4ROOT}/lib
```

The proj4 library should be configured to be installed in `${SW4ROOT}`. This is accomplished by

```
shell> cd proj-4.7.0
shell> configure --prefix=${SW4ROOT}
shell> make
shell> make install
```

We remark that the `proj.4` library can alternatively be installed using `macports` (if you are on a Mac OSX machine).

The cencalvm library should also be configured to be installed in `${SW4ROOT}`. You also have to help the configure script finding the include and library files for the proj.4 and etree libraries,

```
shell> cd cencalvm-0.6.5
shell> configure --prefix=${SW4ROOT} CPPFLAGS="-I${SW4ROOT}/include" \
                 LDFLAGS="-L${SW4ROOT}/lib"
shell> make
shell> make install
```

To verify that the libraries have been installed properly, you should go to the `SW4ROOT` directory and list the `lib` sub-directory (`cd ${SW4ROOT}; ls lib`). You should see the following files (on Mac OSX machines, the .so extension is replaced by .dylib):

```
shell> cd ${SW4ROOT}
shell> ls lib
libetree.so libetree.a
libproj.so libproj.a libproj.la
libcencalvm.a libcencalvm.la libcencalvm.so
```

Furthermore, if you list the include sub-directory, you should see include files such as

```
shell> cd ${SW4ROOT} %$
shell> ls include
btree.h etree.h etree_inttypes.h
nad_list.h projects.h proj_api.h
cencalvm
```

Note that the include files for cencalvm are in the sub-directory with the same name.

7 Disclaimer

References

[1] N. A. Petersson and B. Sjögreen. User's guide to SW4, version 1.1. Technical Report LLNL-SM-662014, Lawrence Livermore National Laboratory, 2014. (Source code available from `geodynamics.org/cig/software/sw4/`).