Installing SW4 version 3.0

N. Anders Petersson* Björn Sjögreen* Houjun Tang†

September 21, 2022

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 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-741310.

†Lawrence Berkeley National Laboratory

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. an MPI library. This library provides support for message passing on parallel machines. Examples of open source implementations include Mpich and OpenMPI. Note that the MPI 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*.

In order to use geographic projection and material models stored in the *rfile* format, you need to install the PROJ before building *SW4*:

- The PROJ library (version 6+), `https://proj.org/download.html`

If you also wish to use material models using the *sfile* or *GMG* format, you also need to download and install the HDF5 library:

- The HDF5 library (version 1.12+), `https://www.hdfgroup.org/downloads/hdf5/source-code`

To simplify the build process, all libraries should be installed under the same directory, such that the library files (.so, .a, etc.) are in the lib sub-directory and the include files (.h) end up in the include sub-directory. 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 gcc11
shell> sudo port install mpich-gcc11
shell> sudo port install hdf5
```

Here, `gcc11` refers to version 11 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 `mpich` package using the gcc11 compilers, which includes a compatible Fortran compiler. Alternatively, you can use the `openmpi` 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`, `mpif90`, and `mpirun`, respectively.

The `lapack` and `blas` libraries are preinstalled on recent Macs and can be accessed using the `-framework Accelerate` 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. For Ubuntu systems, one can simply change `yum` to `apt-get`

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 `mpich` and `hdf5` library and include files with the command

```
yum install mpich-devel
yum install hdf5
```

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

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

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

```
setenv PATH ${PATH}:/usr/lib64/mpich/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-v3.0.tar.gz` in the desired directory and issue the following command:

```
shell> tar xzf sw4-v3.0.tar.gz
```

As a result a new sub-directory named `sw4-v3.0` 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.
- `pytest` Python script and input files for testing the *SW4* installation.
- `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 `quartz` (at LLNL), `Cori` (at LBNL), and `Summit` (at ORNL). We have successfully used the following versions of Gnu, Intel, and IBM compilers:

```
Gnu:   g++/gcc/gfortran      versions 4.5+
Intel: icpc/icc/ifort        versions 16.0+
IBM:   xlcxx/xlc/xlf         versions 12.1+
```

SW4 uses the message passing interface (MPI) standard 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 `mpich` package build wrappers for the underlying C++ and Fortran compilers called `mpicxx` and `mpif90`, 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 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 Fortran 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
proj = no
hdf5 = no
FC = mpif90
CXX = mpicxx
EXTRA_FORT_FLAGS =
EXTRA_LINK_FLAGS = -framework Accelerate -L/opt/local/lib/gcc11 -lgfortran
```

In this case, the `blas` and `lapack` libraries are assumed to be provided by the `-framework Accelerate` option. The `libgfortran` library is located in the directory `/opt/local/lib/gcc11`, 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
proj = no
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

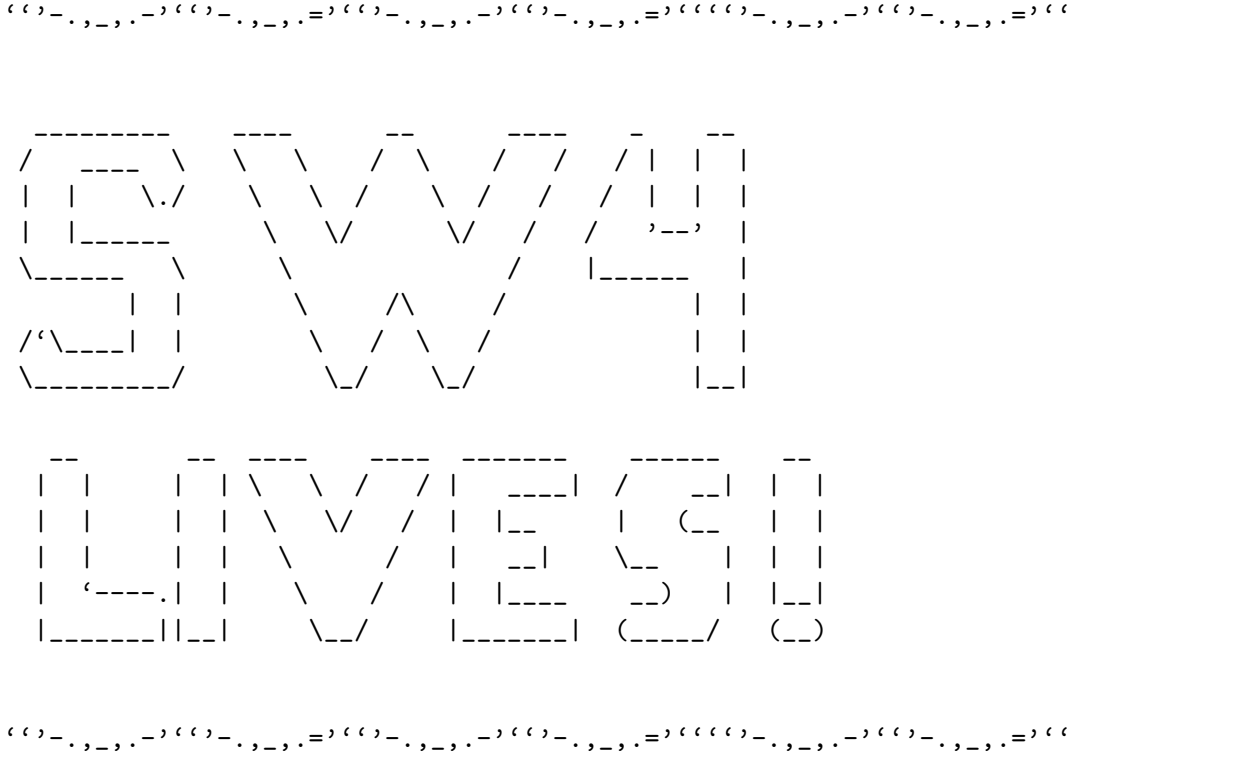
```
EXTRA_LINK_FLAGS = -Wl,-rpath=/usr/lib64/mpich/lib  -llapack -lblas -lgfortran
```

4.1.3 Using make

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

```
shell> cd /enter/your/path/sw4-v3.0
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-v3.0/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-v3.0
shell> make debug=yes
```

In this case, the executable will be located in

```
/enter/your/path/sw4-v3.0/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-v3.0/Makefile
```

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

```
/my/path/sw4-v3.0/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.

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

```
proj = no
proj_6 = no
hdf5 = no
SW4ROOT = /my/path/to/installed/library

CXX = mpicxx
FC  = mpif77

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

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 and/or HDF5 support

The PROJ libraray enables the more advanced geographical mapping keywords in the `grid` command and is also required by the `rfile, sfile` and `gmg` commands. To enable the `sfile` and `gmg` commands, you have to also install the `HDF5` library.

Once you have successfully installed the PROJ, and optionally the HDF5 library, 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_6 = yes
# if you installed PROJ 5 or earlier version, set proj = yes instead
hdf5 = yes
SW4ROOT = /thid/party/basedir
...
```

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

```
shell> cd /my/installation/dir/sw4-v3.0
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.

4.3 Testing the *SW4* installation

The *SW4* source code distribution includes a python(3) script for running several tests and checking the solutions against previously verified results. Note that the same set of tests can be performed when *SW4* is built with CMake, see Section 5.2.

After *SW4* has been built with `make`, go to the `pytest` directory and run `test_sw4.py`. If the `sw4` executable resides in the `optimize` directory, you can run the basic tests by doing:

```
shell> cd pytest
shell> ./test_sw4.py ("./test_sw4.py -u 0" if HDF5 is not installed)
```

If all goes well, you should see the following output:

```
>shell test_sw4.py
Running all tests for level 0 ...
Test # 1 Input file: energy-nomr-2nd-1.in PASSED
Test # 2 Input file: energy-mr-4th-1.in PASSED
...
Test # 24 Input file: loh1-h100-mr-hdf5-sfile-1.in PASSED
Test # 25 Input file: loh1-h100-mr-restart-hdf5-1.in PASSED
Out of 25 tests, 0 failed, 23 passed, and 2 skipped
```

Some aspects of the testing can be modified by providing command line arguments to `test_sw4.py`. For a complete list of options do `test_sw4.py --help`, which currently give the output:

```
shell> ./test_sw4.py --help
usage: test_sw4.py [-h] [-v] [-l {0,1,2}] [-m MPITASKS] [-d SW4_EXE_DIR]

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         increase output verbosity
  -l {0,1,2}, --level {0,1,2}
                        testing level
  -m MPITASKS, --mpitasks MPITASKS
                        number of mpi tasks
  -t OMPTHREADS, --ompthreads OMPTHREADS
                        number of omp threads per task
  -d SW4_EXE_DIR, --sw4_exe_dir SW4_EXE_DIR
                        name of directory for sw4 executable
  -p PYTEST_DIR, --pytest_dir PYTEST_DIR
                        full path to the directory of pytest
                        (/path/sw4/pytest)
  -u {0,1,2,3,4}, --usehdf5 {0,1,2,3,4}
                        run HDF5 tests with, 0 no HDF5, 1 first HDF5 case, 2
                        second case,..., 4 all cases
  -g {0,1}, --geodynbc {0,1}
                        run Geodynbc tests with, 0 skip, 1 both cases
  -A CPU_ALLOCATION, --cpu_allocation CPU_ALLOCATION
                        name of cpu bank/allocation
```

Note that the directory name for the `sw4` executable should be given relative to the main `sw4` directory.

5 Installing *SW4* with CMake

SW4 can also be built with CMake. Compared to using regular `make`, this build process is easier to use because it is fully automated. However, it gives the user less control of which compilers, linker, and libraries to use. Similar to using regular `make`, the *SW4* CMake configuration allows automated correctness testing of the installation. The test runs the same set of cases as the `test_sw4.py` script in the `pytest` directory, see Section 5.2 for details.

To use CMake, navigate to the top `sw4` directory and run the following commands:

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

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 that are appropriate for your system. You then run `make` to compiles and link *SW4* using these makefiles. For details about the exact commands being used in compilation, run `make VERBOSE=1`. Once SW4 has been successfully built, you will see the "SW4 Lives!" banner on the screen.

NOTE: 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. Another way is to remove all files 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
PROJ_HOME	(none)	The path to the PROJ installation to use when compiling *SW4*.
HDF5_HOME	(none)	The path to the HDF5 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.

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 and HDF5 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=2 -DMPI_NUM_TEST_PROCS=36 -DMPIEXEC=/usr/bin/srun \
      -DMPIEXEC_NUMPROC_FLAG=-n -DMPIEXEC_PREFLAGS=-ppdebug \
      -DPROJ_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 PROJ: /usr/apps/wpp/lib/libproj.so
...
```

Sometimes CMake doesn't pick up the correct compiler. Say, for example that the C++ compiler on your system is called `mpicxx` and the Fortran compiler is `mpiifort`. You can tell `cmake` to use those compilers by setting the following envoronment variables *before* running cmake (assuming a `csh` shell),

```
> setenv CXX mpicxx
> setenv FC mpiifort
```

5.2 CTest

The *SW4* CMake configuration includes several test cases that are used to verify the correctness of the *SW4* installation. Each test consists of two parts. First it runs a case using an input file in the `pytest` directory. Secondly, it checks that the results are within a reasonable error tolerance from previously recorded results.

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

```
build > ctest
Test project /Users/petersson1/src/sw4-cig/build
      Start  1: Run_twilight/flat-twi-1
 1/24 Test  #1: Run_twilight/flat-twi-1 .....................   Passed    0.49 sec
      Start  2: Check_Result_twilight/flat-twi-1
 2/24 Test  #2: Check_Result_twilight/flat-twi-1 ............   Passed    0.03 sec
      Start  3: Run_twilight/flat-twi-2
...
      Start 23: Run_pointsource/pointsource-sg-1
23/24 Test #23: Run_pointsource/pointsource-sg-1 ............   Passed   89.56 sec
      Start 24: Check_Result_pointsource/pointsource-sg-1
24/24 Test #24: Check_Result_pointsource/pointsource-sg-1 ...   Passed    0.03 sec

100\% tests passed, 0 tests failed out of 24

Total Test time (real) = 230.91 sec
```

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

```
build > ctest -R meshrefine
Test project /Users/petersson1/src/sw4-cig/build
    Start 15: Run_meshrefine/refine-el-1
1/6 Test #15: Run_meshrefine/refine-el-1 ................   Passed   25.61 sec
    Start 16: Check_Result_meshrefine/refine-el-1
2/6 Test #16: Check_Result_meshrefine/refine-el-1 ........   Passed    0.03 sec
    Start 17: Run_meshrefine/refine-att-1
```

```
3/6 Test #17: Run_meshrefine/refine-att-1 ..............   Passed   22.00 sec
    Start 18: Check_Result_meshrefine/refine-att-1
4/6 Test #18: Check_Result_meshrefine/refine-att-1 .......   Passed    0.03 sec
    Start 19: Run_meshrefine/refine-att-2nd-1
5/6 Test #19: Run_meshrefine/refine-att-2nd-1 ............   Passed   17.63 sec
    Start 20: Check_Result_meshrefine/refine-att-2nd-1
6/6 Test #20: Check_Result_meshrefine/refine-att-2nd-1 ...   Passed    0.03 sec

100% tests passed, 0 tests failed out of 6

Total Test time (real) =  65.35 sec
```

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

6 Installing the PROJ and HDF5 packages

If you are interested in using the advanced geographical mapping options of the `grid` command, the `sfile`, or the `gmg` command, you need to install the PROJ package. For `sfile` and `gmg`, the HDF5 library is also required.

The following instructions describe how to install the two packages. For simplicity all packages are installed under the same top directory. If you are using `cmake`, you may optionally put the PROJ and HDF5 packages 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 PROJ library should be configured to be installed in `${SW4ROOT}`. This is accomplished by

```
shell> cd proj-9.0.0
shell> ./configure --prefix=${SW4ROOT}
shell> make
shell> make install
```

Installing HDF5 can be done similarly:

```
shell> cd hdf5-1.12.2
shell> ./autogen.sh
shell> ./configure --prefix=${SW4ROOT} --enable-parallel
shell> make
shell> make install
```

Alternatively they can be installed using `macports` (if you are using MacOS).

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 lib64
libproj.so libhdf5.so ...
```

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

```
shell> cd ${SW4ROOT} %$
shell> ls include
proj.h hdf5.h ...
```

7 Disclaimer

References

[1] N. A. Petersson and B. Sjögreen. User's guide to SW4, version 2.0. Technical Report LLNL-SM-741439, Lawrence Livermore National Laboratory, 2022. (Source code available from `https://github.com/geodynamics/sw4`).