TACC

Building projects with CMake

Victor Eijkhout

Fall 2022

Justification

CMake is a portable build system that is becoming a *de facto* standard for C++ package management.
Also usable with C and Fortran.

The build/make cycle

Building software the old way

Using ‘GNU Autotools’:

```
./configure
make
make install
```

User vs system packages

The `make install` often tries to copy to a system directory. If you're not the admin, do:

```
./configure --prefix=/home/yourname/mypackages
```

with a location of your choice.

Building with CMake

- Either replace only the configure stage

```
cmake ## arguments
make
make install
```

or

- do everything with CMake:

```
cmake ## arguments
cmake --build ## stuff
cmake --install ## stuff
```

(The second one is portable to non-Unix environments.)

The build/make cycle

CMake creates makefiles;
makefiles ensure minimal required compilation

```
cmake                ## make the makefiles
make                 ## compile your project
emacs onefile.c ## edit
make                 ## minimal recompile
```

Only if you add (include) files do you rerun CMake.

Using packages through pkgconfig

Problem

You want to install a package/application
. . . which needs 2 or 3 other packages.

```
cmake \
    -D PACKAGE1_INC=/users/my/package1/include \
    -D PACKAGE1_LIB=/users/my/package1/lib \
    -D PACKAGE2_INC=/users/my/package2/include/packaage \
    -D PACKAGE2_LIB=/users/my/package2/lib64 \
    ../newpackage
```

Can this be make simpler?

Finding packages with ‘pkg config’

- Many packages come with a `package.pc` file
- Add that location to `PKG_CONFIG_PATH`
- That defines variables in your own cmake file

Example: PETSc
add `$PETSC_DIR/$PETSC_ARCH/lib/pkgconfig` to config path, then

```
find_package( PkgConfig REQUIRED )
pkg_check_modules( PETSC REQUIRED petsc )
target_include_directories(
        program PUBLIC
        ${PETSC_INCLUDE_DIRS} )
```

Eigen

```
1  cmake_minimum_required( VERSION 3.12 )
2  project( eigentest )
3
4  find_package( PkgConfig REQUIRED )
5  pkg_check_modules( EIGEN REQUIRED eigen3 )
6
7  add_executable( eigentest eigentest.cxx )
8  target_include_directories(
9        eigentest PUBLIC
10       ${EIGEN_INCLUDE_DIRS})
```

Where do these settings come from?

```
find $TACC_EIGEN_DIR -name \*.pc
${TACC_EIGEN_DIR}/share/pkgconfig/eigen3.pc
```

Pkgconfig outside CMake

`.pc` files are also useful outside CMake:

```
$ pkg-config --cflags tbb
-I/opt/intel//oneapi/tbb/latest/lib/pkgconfig/../..//include
$ pkg-config --libs tbb
-L/opt/intel//oneapi/tbb/latest/lib/pkgconfig/../..//lib/intel64
    /gcc4.8 -ltbb
```

(useful for packages where there is no module)

Other discovery mechanisms

Some packages come with `FindWhatever.cmake` or similar files.
Pity that there is not just one standard.
These define some macros, but you need to read the docs to see which.
Pity that there is not just one standard.
Some examples follow.

MPI from C

```
1  cmake_minimum_required( VERSION 3.12 )
2  project( ${PROJECT_NAME} VERSION 1.0 )
3
4  find_package( MPI )
5
6  add_executable( ${PROJECT_NAME} ${PROJECT_NAME}.c )
7  target_include_directories(
8         ${PROJECT_NAME} PUBLIC
9         ${MPI_C_INCLUDE_DIRS} ${CMAKE_CURRENT_SOURCE_DIR} )
10 target_link_libraries(
11        ${PROJECT_NAME} PUBLIC
12        ${MPI_C_LIBRARIES} )
13
14 install( TARGETS ${PROJECT_NAME} DESTINATION . )
```


MPI from Fortran

```
1  cmake_minimum_required( VERSION 3.12 )
2  project( ${PROJECT_NAME} VERSION 1.0 )
3
4  enable_language(Fortran)
5
6  find_package( MPI )
7
8  if( MPI_Fortran_HAVE_F08_MODULE )
9  else()
10   message( FATAL_ERROR "No f08 module for this MPI" )
11 endif()
12
13 add_executable( ${PROJECT_NAME} ${PROJECT_NAME}.F90 )
14 target_include_directories(
15         ${PROJECT_NAME} PUBLIC
16         ${MPI_Fortran_INCLUDE_DIRS} ${CMAKE_CURRENT_SOURCE_DIR}
       )
17 target_link_directories(
18         ${PROJECT_NAME} PUBLIC
19         ${MPI_LIBRARY_DIRS} )
20 target_link_libraries(
21         ${PROJECT_NAME} PUBLIC
```

OpenMP

```
1  cmake_minimum_required( VERSION 3.12 )
2  project( ompprogram VERSION 1.0 )
3
4  find_package(OpenMP)
5  if(OpenMP_CXX_FOUND)
6  else()
7          message( FATAL_ERROR "Could not find OpenMP" )
8  endif()
9
10 add_executable( program program.cxx )
11 target_link_libraries( program
12     PUBLIC OpenMP::OpenMP_CXX)
13
14 install( TARGETS program DESTINATION . )
```