Introduction to CMake

Dr. Axel Kohlmeyer

Assistant Dean for High-Performance Computing
Associate Director, ICMS
College of Science and Technology
Temple University, Philadelphia

http://sites.google.com/site/akohlmey/

a.kohlmeyer@temple.edu

MHPC
Master in High Performance Computing

What is wrong with “make”? Why use a tool like CMake?

- Nothing much, if you just use a single platform and compiler, have a rather simple project, and don’t need any fancy “bells and whistles”
- Makefiles need to be adapted for different compilers, platforms, compilation settings etc. This requires technical knowledge and there are no checks whether it works correctly. CMake can automate a lot of these steps.
- A tool like CMake can construct files for different build systems on different platforms and knows about different compilers and IDEs

Basic Steps with CMake

- There are 3 phases: 1. configuration,
  2. generate build files, 3. build with build tool
- On Linux we can build with “make” and “ninja”
- IDE support for: Kate, Eclipse, CodeBlocks,…
- Configuration program in **`CMakeLists.txt`** file
- **<u>Recommended</u>** to do “out-of-source” build
- Supports command line (**`cmake`**), TUI (**`ccmake`**) and GUI (**`cmake-gui`**) for phases 1 and 2
- **`mkdir build;cd build; cmake ..; cmake --build .`**
  or: **`cmake -S . -B build; cmake --build build`**

Common CMake Options

- Use make: **`cmake -G ‘Unix Makefiles’ ..`**
- Use ninja: **`cmake -G ‘Ninja’ ..`**
- Use clang: **`cmake -DCMAKE_CXX_COMPILER=clang++ .`**
- Enable building/linking to shared libraries:
  `cmake -DBUILD_SHARED_LIBS=on .`
- Select build type (default is Debug):
  `cmake -DCMAKE_BUILD_TYPE=Release .`
- Enable tests: **`cmake -DENABLE_TESTING=on .`**

MHPC
Master in High Performance Computing

A Minimal Example

```
#include <iostream>

int main(int, char **)
{
        std::cout << "Hello, World!\n";
        return 0;
}
```

```
##############################################
cmake_minimum_required(VERSION 3.10)

project(canvas-draw VERSION 0.1 LANGUAGES CXX)

add_executable(canvas-draw main.cpp)
```

Using Multiple Source Files / Library

```
##############################################
cmake_minimum_required(VERSION 3.10)

project(canvas-draw VERSION 0.2 LANGUAGES CXX)

#add_executable(canvas-draw main.cpp canvas.cpp)

add_executable(canvas-draw main.cpp)

# build libcanvas library
add_library(canvas canvas.cpp)
target_link_libraries(canvas-draw PUBLIC canvas)
```

Using Configuration Options

```
##############################################
cmake_minimum_required(VERSION 3.10)
project(canvas-draw VERSION 0.3 LANGUAGES CXX)

option(BUILD_SHARED_LIBS "Build shared lib" OFF)
if(BUILD_SHARED_LIBS)
    set(CMAKE_POSITION_INDEPENDENT_CODE ON)
endif()

add_executable(canvas-draw main.cpp)
# build libcanvas library
add_library(canvas canvas.cpp)
target_link_libraries(canvas-draw PUBLIC canvas)
```

Using a Configuration File

```
###########################################
cmake_minimum_required(VERSION 3.10)
project(canvas-draw VERSION 0.4 LANGUAGES CXX)

# …

add_executable(canvas-draw main.cpp)

configure_file(canvas_config.h.in canvas_config.h)

# build libcanvas library
add_library(canvas canvas.cpp)
target_include_directories(canvas PUBLIC
                           ${CMAKE_BINARY_DIR})
target_link_libraries(canvas-draw PUBLIC canvas)
```

Using a Configuration Filen (2)

```
########### canvas_config.h.in ################
// the configured options and version definitions
#define CANVAS_MAJOR @canvas-draw_VERSION_MAJOR@
#define CANVAS_MINOR @canvas-draw_VERSION_MINOR@
########### main.cpp #########################
// -*- c++ -*-
#include "canvas_config.h"
#include "canvas.h"

#include <iostream>

int main(int, char**)
{
  std::cout << "canvas-draw version "
        << CANVAS_MAJOR << "." << CANVAS_MINOR << "\n";
```

Master in High Performance Computing

Detecting/Using an Optional Feature

```
project(canvas-draw VERSION 0.5 LANGUAGES CXX)
# …
add_executable(canvas-draw main.cpp)

# enable optional JPEG support automatically, if found
find_package(JPEG QUIET)
option(USE_JPEG "Enable JPEG support" ${JPEG_FOUND})

add_library(canvas canvas.cpp)

# if JPEG support is enabled, required and add include/libs
if(USE_JPEG)
  find_package(JPEG REQUIRED)
  target_compile_definitions(canvas PRIVATE -DHAVE_JPEG_LIB)
  target_include_directories(canvas PRIVATE ${JPEG_INCLUDE_DIRS})
  target_link_libraries(canvas PRIVATE ${JPEG_LIBRARIES})
endif()

target_link_libraries(canvas-draw PRIVATE canvas)
```

Adding Tests

```
project(canvas-draw VERSION 0.6 LANGUAGES CXX)
# …
add_executable(canvas-draw main.cpp)
add_library(canvas canvas.cpp)

target_link_libraries(canvas-draw PRIVATE canvas)

if(ENABLE_TESTING)
  enable_testing()

  # does the application run?
  add_test(NAME Runs COMMAND canvas-draw 5)

  # does it create the file write.ppm?
  file(WRITE ${CMAKE_BINARY_DIR}/test_write.sh "rm -f *.ppm &&
                           ./canvas-draw && test -f white.ppm")
  add_test(NAME Writes COMMAND bash test_write.sh 5)
endif()
```

Master in High Performance Computing

Use Custom CMake Script Code

```
project(canvas-draw VERSION 0.7 LANGUAGES CXX)
# …
add_executable(canvas-draw main.cpp)
add_library(canvas canvas.cpp)
target_link_libraries(canvas-draw PRIVATE canvas)

if(ENABLE_TESTING)
  enable_testing()
  # include GTest.cmake file to build googletest library
  set(CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR})
  include(GTest)

  add_executable(test_color test_color.cpp)
  target_link_libraries(test_color PRIVATE
                        GTest::GTestMain GTest::GTest)
  add_test(NAME Color COMMAND test_color)

  add_test(NAME Runs COMMAND canvas-draw)
endif()
```

Support for MPI and OpenMP

```
project(pi VERSION 0.1 LANGUAGES C)

# look for MPI C interface and add MPI executable
find_package(MPI REQUIRED)
add_executable(pi_mpi pi_mpi.c)
target_link_libraries(pi_mpi PRIVATE MPI::MPI_C)

# look for OpenMP header and runtime library
find_package(OpenMP REQUIRED)
include(CheckIncludeFile)
check_include_file(omp.h HAVE_OMP_H)
if(!HAVE_OMP_H)
  message(FATAL_ERROR "Must have omp.h header file")
endif()
add_executable(pi_omp pi_omp.c)
target_link_libraries(pi_omp PRIVATE OpenMP::OpenMP_C)
```