CMake based UFS Weather Model build system

Dusan Jovic, Dom Heinzeller, George Gayno, Mark Potts, Kyle Gerheiser, Rahul Mahajan

Not a CMake tutorial.

There are plenty of information online. Documentation. Tutorial

Not a comprehensive description of build options, compiler flags, etc.

UFS Weather Model has two build systems

The “official” build system is still hand written GNU make:

- used in production (most production libraries are not (yet) converted to cmake)
- used by all real-time parallels and development workflows
- used by other UFS applications, HAFS, FV3-SAR, s2s coupled etc (many ufs components still do not support cmake, MOM6, WW3, …)

The new cmake based system is:

- used in MRW public release, and soon SRW App, UFS s2s model.
- in regression testing when Rocoto and ecFlow workflow managers are used

CMake is a build system generator

- cmake does not ‘build’ (compile, link) anything
- cmake generates build files (Makefiles), which actually run the build steps
- cmake runs in two phases
  - configure (parses CMakeLists.txt, detects compilers)
  - generate (creates Makefiles)
- then, you run the `make`, `make install`, …
- no need to rerun cmake after existing source code files is modified, just rerun make
- cmake must be rerun to change any build option or after removing/adding new files
- cmake generates (mostly) dependencies between fortran modules
- out-of-source build, multiple builds can be running simultaneously from the same source tree, keeps the source tree clean

Usage

Load required modules if you are on a system that uses modules

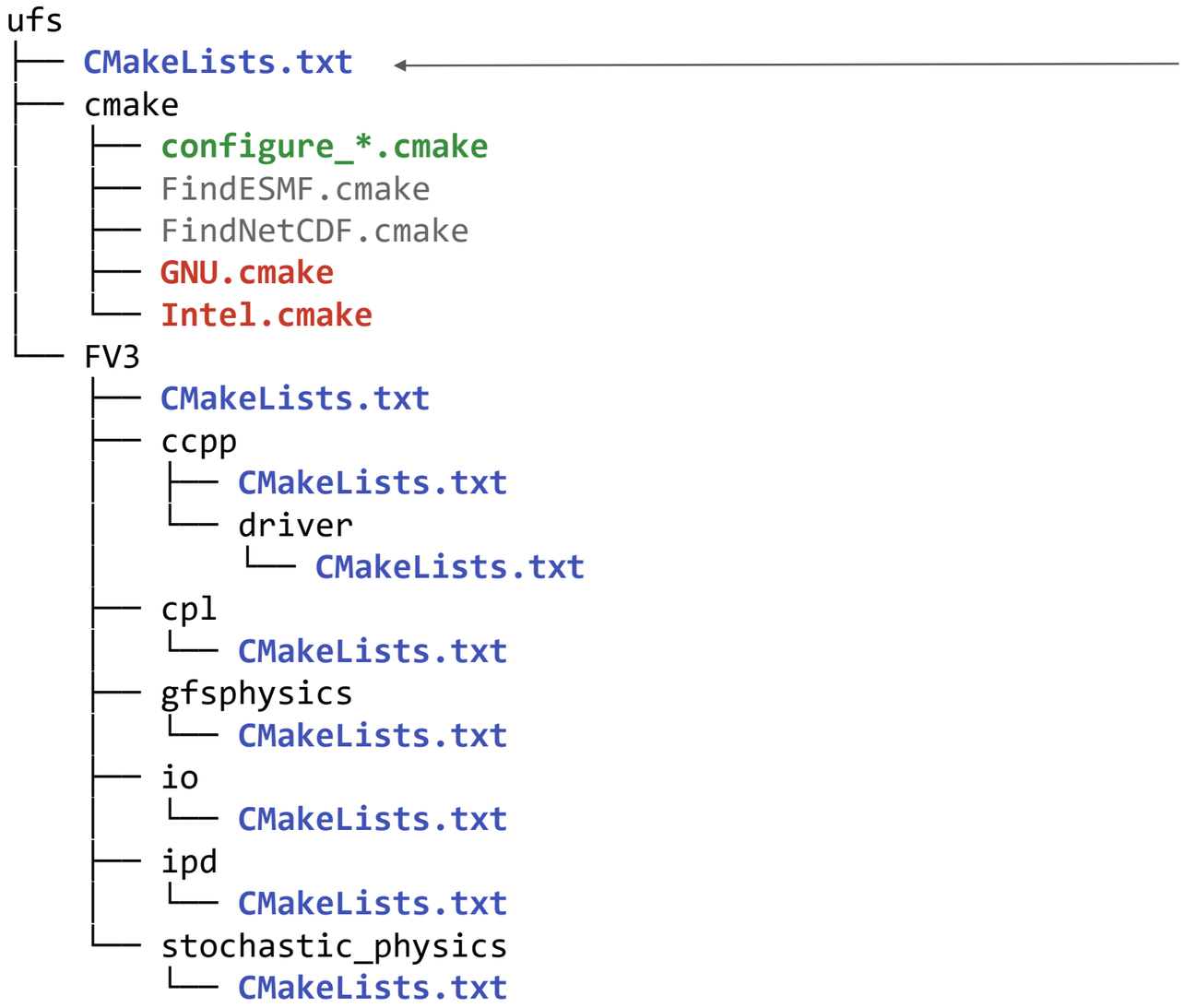
```
$ mkdir build && cd build
$ cmake <path>/ufs-weather-model build_options
```

options:

```
  -DCCPP=YES -DCCPP_SUITES=”.....” -DAVX2=ON -DINLINE_POST=ON
    -DWW3=OFF -D...
```

```
$ make -j 8
$ make install
```

CMakeLists.txt



Top-level CMakeLists.txt defines project, finds libraries, adds subdirectories, defines the executable

Set compilers and ‘platform’

Following cmake variables must be set:

```
$ more modulefiles/wcoss_cray/fv3

…

setenv CMAKE_C_COMPILER cc

setenv CMAKE_CXX_COMPILER CC

setenv CMAKE_Fortran_COMPILER ftn

setenv CMAKE_Platform wcoss_cray
```

Compiler flags

Only Intel and GNU are currently supported

Compiler flags are set (based on build options) in:

- `cmake/GNU.cmake`
- `cmake/Intel.cmake`

and one of those two files will be used (included) in the project based on chosen compiler family

```
set(CMAKE_Fortran_FLAGS "${CMAKE_Fortran_FLAGS} -fpp -fno-alias -auto -safe-cray-ptr -ftz
                                                -assume byterecl -nowarn -sox -align array64byte")
set(CMAKE_Fortran_FLAGS "${CMAKE_Fortran_FLAGS} -qno-opt-dynamic-align")


if(32BIT)
    set(CMAKE_Fortran_FLAGS "${CMAKE_Fortran_FLAGS} -i4 -real-size 32")
    add_definitions(-DOVERLOAD_R4)
    add_definitions(-DOVERLOAD_R8)
else()
    set(CMAKE_Fortran_FLAGS "${CMAKE_Fortran_FLAGS} -i4 -real-size 64")
    if(NOT REPRO)
        set(CMAKE_Fortran_FLAGS "${CMAKE_Fortran_FLAGS} -no-prec-div -no-prec-sqrt")
    endif()
endif()
```

Libraries (system, 3rdparty and NCEPLIBS) and packages

UFS Weather Model depends on a number of system, third party libraries and NCEPLIBS libraries

System libraries: (MPI, MKL)

Third party: (NetCDF, HDF5, ESMF, soon FMS)

NCEPLIBS: bacio, g2, g2tmp, nemiso, sigio, sp, w3emc

Libraries are ‘made available’ to the build system by ‘finding’ corresponding package, which either defines imported target or sets variables that point to the library include locations and library archive files (.a)

Finding packages (ie. libraries)

```
find_package(NetCDF REQUIRED C Fortran)
find_package(MPI REQUIRED)
find_package(ESMF MODULE REQUIRED)

find_package(bacio REQUIRED)
find_package(w3emc REQUIRED)
find_package(sigio REQUIRED)
find_package(sp REQUIRED)
find_package(nemsio REQUIRED)
find_package(g2 REQUIRED)
find_package(g2tmpl REQUIRED)
```

Build options

Based on CMAKE_Platoform variable, a predefined set of build options is loaded from `cmake/configure_${CMAKE_Platform}.cmake`, for example:

`$ cat cmake/configure_hera.intel.cmake`

```
option(DEBUG           "Enable DEBUG mode" OFF)
option(REPRO           "Enable REPRO mode" OFF)
option(VERBOSE         "Enable VERBOSE mode" OFF)
option(32BIT           "Enable 32BIT (single precision arithmetic in dycore)" OFF)
option(OPENMP          "Enable OpenMP threading" ON)
option(AVX2            "Enable AVX2 instruction set" OFF)
option(QUAD_PRECISION "Enable QUAD_PRECISION (for certain grid metric terms)" ON)
option(INLINE_POST     "Enable inline post" ON)
```

External libraries

In current `develop` branch ncep libraries use environment variables: (`$W3_LIB4, $BACIO_INC,` etc). On production and R&D systems these variables are set by modules.

In “near” future all of the NCEP libraries will be installed using cmake and provide imported targets on all supported platforms (Tier-1). Modules will also export environment variables for backward compatibility with older applications.

Some 3rdparty or system libraries still use cmake or environment variables

Extra slides

Building non-cmake components

Not all UFS components support CMake.

For example WW3 is still using custom scripts and Makefiles.

It is possible to define a custom target that executes arbitrary command from during the build.

This custom target that can be later used to define dependencies

```
if(WW3)
  set(WW3_COMP ${CMAKE_Platform})
  if(${CMAKE_Platform} STREQUAL "hera.intel")
    set(WW3_COMP "hera")
  endif()
  if(${CMAKE_Platform} STREQUAL "orion.intel")
    set(WW3_COMP "orion")
  endif()
  message("Build WW3:")
  message("   run: ${CMAKE_BUILD_TOOL} WW3_PARCOMPN=4 WW3_COMP=${WW3_COMP} ww3_nemslibonly")
  message("   in:  ${PROJECT_SOURCE_DIR}/WW3/model/esmf")

  add_custom_target(ww3_nems
    WORKING_DIRECTORY ${PROJECT_SOURCE_DIR}/WW3/model/esmf
    COMMAND ${CMAKE_BUILD_TOOL} WW3_PARCOMPN=4 WW3_COMP=${WW3_COMP} ww3_nemslibonly > make.log 2>&1
  )

  set(WW3_LIBS ${PROJECT_SOURCE_DIR}/WW3/model/obj/libww3_multi_esmf.a)
endif()
```

```
if(WW3)
  target_compile_definitions(NEMS.exe PRIVATE -DFRONT_WW3=WMESMFMD)
  set_target_properties(NEMS.exe PROPERTIES Fortran_MODULE_DIRECTORY ${PROJECT_SOURCE_DIR}/WW3/model/mod)
  add_dependencies(NEMS.exe ww3_nems)
endif()
```

Code generation

Sometimes code that is going to be compiled needs to be generated depending on the build options

For example:

- CCPP generates code based on selected physics suites
- WW3 generates code based on enabled “switches”

Code can be generated during the configuration/generation phase (cmake) or during the build phase (make)

```
if(CCPP)
    find_package(Python 3 QUIET COMPONENTS Interpreter)
    if (NOT Python_Interpreter_FOUND)
      find_package(Python 2.7 QUIET REQUIRED COMPONENTS Interpreter)
    endif()
    message("Found Python: ${Python_EXECUTABLE}")

    if(DEFINED CCPP_SUITES)
        message("Calling CCPP code generator (ccpp_prebuild.py) for SUITES = ${CCPP_SUITES}")
        execute_process(COMMAND FV3/ccpp/framework/scripts/ccpp_prebuild.py
                                "--config=FV3/ccpp/config/ccpp_prebuild_config.py"
                                "--suites=${CCPP_SUITES}"
                                "--builddir=${PROJECT_BINARY_DIR}/FV3"
                        WORKING_DIRECTORY ${PROJECT_SOURCE_DIR}
                        OUTPUT_FILE ${PROJECT_BINARY_DIR}/ccpp_prebuild.out
                        ERROR_FILE ${PROJECT_BINARY_DIR}/ccpp_prebuild.err
                        RESULT_VARIABLE RC )
    else()
        message("Calling CCPP code generator (ccpp_prebuild.py) ...")
        execute_process(COMMAND FV3/ccpp/framework/scripts/ccpp_prebuild.py
                                "--config=FV3/ccpp/config/ccpp_prebuild_config.py"
                                "--builddir=${PROJECT_BINARY_DIR}/FV3"
                        WORKING_DIRECTORY ${PROJECT_SOURCE_DIR}
                        OUTPUT_FILE ${PROJECT_BINARY_DIR}/ccpp_prebuild.out
                        ERROR_FILE ${PROJECT_BINARY_DIR}/ccpp_prebuild.err
                        RESULT_VARIABLE RC )
```

```
set(SRCS
  constants.ftn
  w3adatmd.ftn
  w3arrymd.ftn
    . . .
)
add_executable(w3adc ../aux/w3adc.f)

foreach(src_file ${SRCS})
  STRING(REGEX REPLACE ".ftn" ".F90" gen_src_file ${src_file})
  STRING(REGEX REPLACE "/" "_" gen_log_file ${gen_src_file})
  add_custom_command(
    OUTPUT  ${gen_src_file}
    DEPENDS w3adc ${ftn_dir}/${src_file}
    COMMAND ${CMAKE_CURRENT_SOURCE_DIR}/run_w3adc.sh ${ftn_dir} ${src_file} > ${gen_log_file}.w3adc.log
    COMMENT "Running w3adc ${src_file}")
  list(APPEND SRCS_F90 ${gen_src_file})
endforeach()

add_library(ww3_multi_esmf STATIC ${SRCS_F90})
```