GSI Fundamentals (1): Setup and Compilation

Mark Potts

Environmental Modeling Center (EMC)

NOAA Center for Environmental Prediction

Tuesday 11 July, 2017

Outline

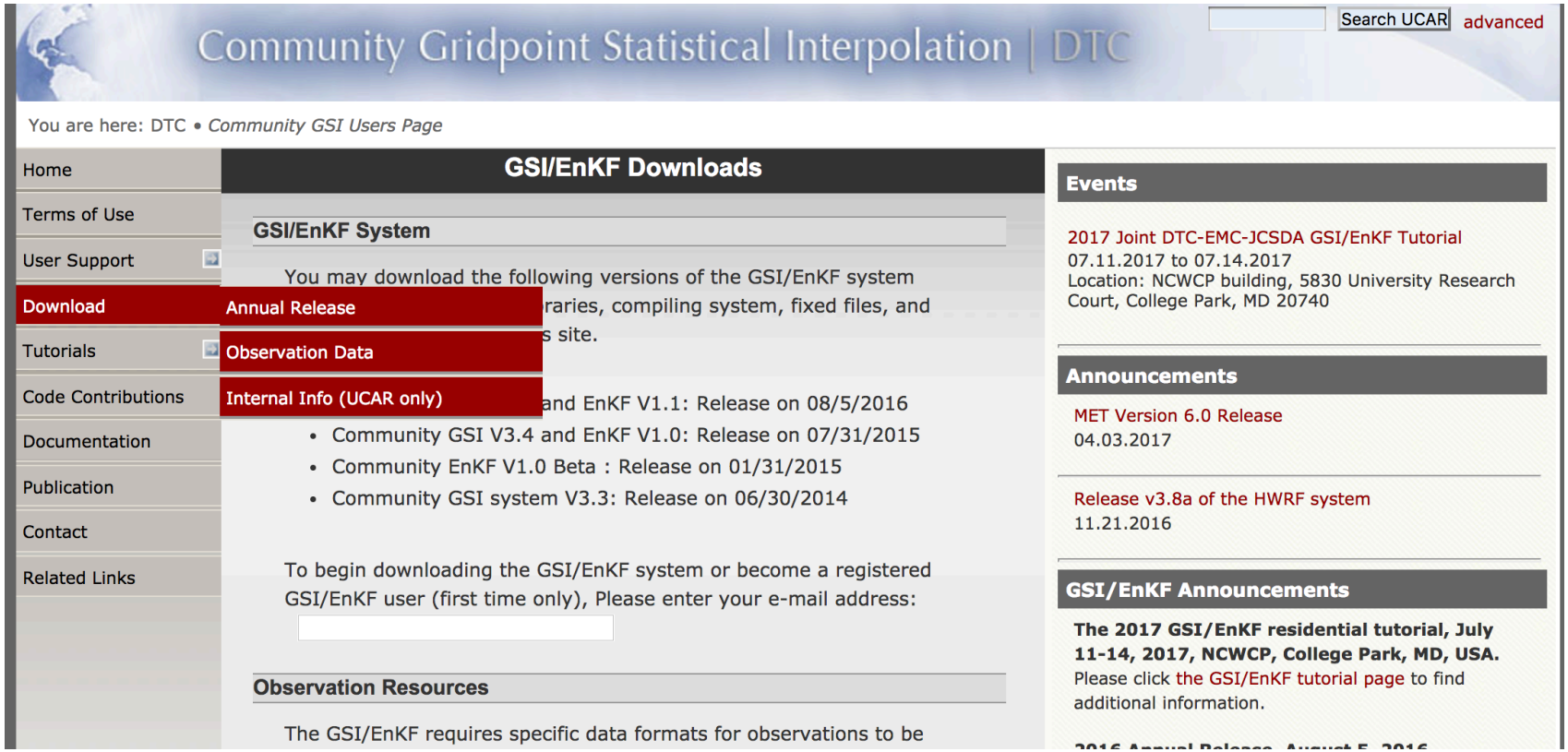
- GSI fundamentals (1): Setup and Compilation
  - Where to get the code
  - Directory structure
  - Building with the DTC build system
  - Porting build to new platforms
- Using CMake to build GSI

This talk is tailored on Chapter 2 of the GSI User's Guide for Community Release V3.5

Downloading the Source Code

- All of the GSI source code can be obtained from:
  - http://www.dtcenter.org/com-GSI/users/downloads/index.php
  - This tutorial will use GSI version 3.6-beta, which will be released in final version in September
  - Available via subversion (svn)
  - svn checkout https://vlab.ncep.noaa.gov/svn/comgsi/branches/release_V3.6beta
- NOAA users may also clone the latest EMC version of the GSI source code using git.
  - git clone --recursive https://vlab.ncep.noaa.gov/git/comgsi (email mark.potts@noaa.gov to be added to the project)

DTC Download website



Downloading Source code

GSI/EnKF Downloads

GSI/EnKF System

You may download the following versions of the GSI/EnKF system (including source codes, libraries, compiling system, fixed files, and sample run script) from this site.

- Community GSI V3.5 and EnKF V1.1: Release on 08/5/2016
- Community GSI V3.4 and EnKF V1.0: Release on 07/31/2015
- Community EnKF V1.0 Beta : Release on 01/31/2015
- Community GSI system V3.3: Release on 06/30/2014

To begin downloading the GSI/EnKF system or become a registered GSI/EnKF user (first time only), Please enter your e-mail address:

GSI/EnKF Downloads

Community GSI Version 3.5 and EnKF Version 1.1

The community GSI Version 3.5 and EnKF Version 1.1 was released on August 3, 2016.

NOTE: This tarball includes the GSI and EnKF code, libraries, fixed files, run script, and utilities. It does not include CRTM cofficients. The CRTM cofficients are available as a separate download. Both tarballs are necessary to run GSI and EnKF.

- comGSIv3.5_EnKFv1.1 tarball (15 MB)
- CRTM 2.2.3 Big_Endian coefficients tarball (937 MB)

Release notes Check
Known issues Check

Download GSI Fixed Files

To reduce the size of release GSI tar file, the following fix files are released separately from the GSI tar file:

- fix files to run Global GSI (121 MB):
  only download and untar it inside the fix directory if run global GSI.
- Little_Endian fix files (117 MB) :
  only download and untar it inside the fix directory if needed.

Unpack Downloads

- Two tar files
  - comGSIv3.5_EnKFv1.1.tar.gz
  - CRTM_Coefficients-2.2.3.tar.gz
- Unpack source code & CRTM coefficients
  - tar –xvfz comGSIv3.5_EnKFv1.1.tar.gz
  - Tar -xvfz CRTM_Coefficients-2.2.3.tar.gz

Tour of the Directory Structure

If you download from DTC or use svn to obtain the source code, you will find the following scripts and directories inside the top level of the GSI directory :

- arch/
- clean
- compile
- configure
- fix/
- makefile_DTC
- run/
- src/
- util/

DTC Build Infrastructure

- Using the DTC Build system
- **/arch** directory contains rules & scripts for build.
  - **/arch/Config.pl** perl script for parsing system info & combining together *configure.gsi* file.
  - **/arch/preamble**: uniform requirements for the code, such as word size, etc.
  - **/arch/configure.defaults** default platform settings
  - **/arch/postamble**: standard make rules & dependencies
- **./clean** script to clean the build.
- **./configure** script to create configuration file *configure.gsi*; contains info on compiler, MPI, & paths.
- **./compile** script to compile executable.
- **./makefile** top level makefile for build.

The rest

- **fix/** directory containing fixed parameter files
  - Background error covariance and observation errors
  - Observation data control files
  - BUFR tables for Prepbufr files
- **run/**
  - Run scripts and namelists
  - gsi.exe executable
- **src/** source directory
  - **libs/** supplemental library source code
  - **main/** main GSI source code
  - **main/enkf** EnKF source code
- **util/** additional community tools

Supplemental Libraries (libs/)

- **bacio/** NCEP BACIO library
- **bufr/** NCEP BUFR library
- **crtm_2.2.3/** JCSDA Commuity Radiative Transfer Model
- **gsdcloud/** GSD Cloud Analysis
- **misc/** Misc additional libraries
- **nemsio/** Support for NEMS I/O
- **sfcio/** NCEP GFS surface file I/O module
- **sigio/** NCEP GFS atmospheric file I/O module
- **sp/** NCEP spectral-grid transforms (global application only)
- **W3emc_v2.0.5/** NCEP W3 library (date/time manipulation, GRIB)
- **W3nco_v2.0.6/** NCEP W3 library (date/time manipulation, GRIB)

Building GSI

System Requirements/Libraries

- FORTRAN 90+ compiler
- C compiler
- Perl
- GNU Make
- NetCDF V4+
- HDF5
- Linear algebra library (ESSL, MKL or LAPACK/ BLAS)
- MPI V1.2+ & OpenMP
- WRF V3.5+

Supported Platforms/Compilers

Platform	F90 compiler	C compiler
IBM*	xlf	xlc
Linux	Intel (ifort)	Intel (icc)
	Intel (ifort)	Gnu (gcc)
	PGI (pgf90)	PGI (pgcc)
	PGI (pgf90)	Gnu (gcc)
	Gnu (gfortran)	Gnu (gcc)
Mac*	PGI (pgf90)	PGI (pgcc)

* Legacy support provided on platforms no longer available for testing

Building GSI using DTC build

- Build sequence
  - ./clean –a
  - Set library paths
    - setenv WRF_DIR *Location_of_WRF_directory*
    - setenv LAPACK_PATH (*typically only needed for Linux w/ ifort or gfortran*).
  - ./configure
    - Customize file *configure.gsi* if necessary
  - ./compile
- Successful compilation will produce:
  - *comGSIv3.5_EnKFv1.1/run/gsi.exe*

Clean Compilation

- To remove all object files and executables, type:

  clean

- To remove **all** built files, including the configure file, type: *clean –a*
  - A clean all needed if:
    - Compilation failed
    - Want to change configuration file

Diagnosing Build Issues

- How the build system works
- What to do when the build fails

How the build works

- Running *./configure* creates file configure.gsi by:
  - Running the Perl script /arch/Config.pl
  - Script Config.pl queries the system & selects the appropriate entry from /arch/configure.defaults
  - Results are saved to configure.gsi.

Identifying Build Errors

- Most build or run problems must be diagnosed by use of the log files.
- For build errors pipe the standard out and standard error into a log file with a command such as (for csh)
  ./compile >& build.log
- Search the log file for any instance of the word "Error." Its presence indicates a build error. Be certain to use the exact spelling with a capital "E."
- If the build fails, but the word "Error" is not present in the log file, it typically indicates a failure in link the phase. Information on the failed linking phase will be present at the very end of the log file.

Fixing Build Issues

- Most build problems are due to non-standard installation of one of the following:
  - compiler,
  - mpi,
  - or support libraries.
- Edit paths in the file configure.gsi to correctly reflect your system.
- When the build succeeds, modify file arch/configure.defaults to include new settings.
- Please report issues to gsi-help so they can be addressed in next release.

./configure

```
------------------------------------------------------------------------
Please select from among the following supported platforms.

   1.  Linux x86_64, PGI compilers (pgf90 & pgcc)
   2.  Linux x86_64, PGI compilers (pgf90 & gcc)
   3.  Linux x86_64, GNU compilers (gfortran & gcc)
   4.  Linux x86_64, Intel/gnu compiler (ifort & gcc)
   5.  Linux x86_64, Intel compiler (ifort & icc)
   6.  Linux x86_64, Intel compiler (ifort & icc) IBM POE
   7.  Linux x86_64, Intel compiler (ifort & icc) SGI MPT

Enter selection [1-7] :
```

./configure

```
------------------------------------------------------------------------
Please select from among the following supported platforms.

   1.  Linux x86_64, PGI compilers (pgf90 & pgcc)
   2.  Linux x86_64, PGI compilers (pgf90 & gcc)
   3.  Linux x86_64, GNU compilers (gfortran & gcc)
   4.  Linux x86_64, Intel/gnu compiler (ifort & gcc)
   5.  Linux x86_64, Intel compiler (ifort & icc)
   6.  Linux x86_64, Intel compiler (ifort & icc) IBM POE
   7.  Linux x86_64, Intel compiler (ifort & icc) SGI MPT

Enter selection [1-7] :
```

configure.gsi

```
SHELL          =    /bin/sh
# Listing of options that are usually independent of machine type.
# When necessary, these are over-ridden by each architecture.
#### Architecture specific settings ####

# Settings for Linux x86_64, Intel compiler (ifort & icc)  (dmpar,optimize)#
LDFLAGS      =  -Wl,-rpath,/glade/apps/opt/netcdf/4.3.0/intel/12.1.5/lib -openmp

COREDIR      =  /glade/scratch/stark/GSI/src/intel/release_V3.3_intel12.1.5
INC_DIR      =  $(COREDIR)/include
SFC          =  ifort
SF90         =  ifort -free
SCC          =  icc
INC_FLAGS    =  -module $(INC_DIR) -I $(INC_DIR) -I /glade/apps/opt/netcdf/4.3.0/
intel/12.1.5/include
```

configure.gsi

```
SHELL          =    /bin/sh
# Listing of options that are usually independent of machine type.
# When necessary, these are over-ridden by each architecture.
#### Architecture specific settings ####

# Settings for Linux x86_64, Intel compiler (ifort & icc)  (dmpar,optimize)#
LDFLAGS      =  -Wl,-rpath,/glade/apps/opt/netcdf/4.3.0/intel/12.1.5/lib -openmp

COREDIR      =  /glade/scratch/stark/GSI/src/intel/release_V3.3_intel12.1.5
INC_DIR      =  $(COREDIR)/include
SFC          =  ifort
SF90         =  ifort -free
SCC          =  icc
INC_FLAGS    =  -module $(INC_DIR) -I $(INC_DIR) -I /glade/apps/opt/netcdf/4.3.0/
intel/12.1.5/include
```

configure.gsi

```
SHELL          =    /bin/sh
# Listing of options that are usually independent of machine type.
# When necessary, these are over-ridden by each architecture.
#### Architecture specific settings ####

# Settings for Linux x86_64, Intel compiler (ifort & icc)  (dmpar,optimize)#
LDFLAGS      =  -Wl,-rpath,/glade/apps/opt/netcdf/4.3.0/intel/12.1.5/lib -openmp

COREDIR      =  /glade/scratch/stark/GSI/src/intel/release_V3.3_intel12.1.5
INC_DIR      =  $(COREDIR)/include
SFC          =  ifort
SF90         =  ifort -free
SCC          =  icc
INC_FLAGS    =  -module $(INC_DIR) -I $(INC_DIR) -I /glade/apps/opt/netcdf/4.3.0/
intel/12.1.5/include
```

configure.gsi

```
SHELL          =    /bin/sh
# Listing of options that are usually independent of machine type.
# When necessary, these are over-ridden by each architecture.
#### Architecture specific settings ####

# Settings for Linux x86_64, GNU compilers (gfortran & gcc)  (dmpar,optimize)#
LDFLAGS      =  -Wl,-noinhibit-exec

COREDIR      =  /d1/stark/GSI/src/intel/V3.2/release_V3.2
INC_DIR      =  $(COREDIR)/include
SFC          =  ifort
SF90         =  ifort –free
SCC          =  icc
INC_FLAGS    =  -I $(INC_DIR) -I /usr/local/netcdf3-ifort/include
```

Fortran Build Flags:

configure.gsi

```
FFLAGS_i4r4    =  -integer-size 32 -real-size 32
FFLAGS_i4r8    =  -integer-size 32 -real-size 64
FFLAGS_i8r8    =  -integer-size 64 -real-size 64
FFLAGS_DEFAULT =  -fp-model precise -assume byterecl -fpe0 -ftz -convert big_endian
FFLAGS_DEBUG   =  -O0 -g -traceback -check bounds -warn errors -fpstkchk -mp
FFLAGS_OPT     =  -O3
FFLAGS         =  -O3 $(FFLAGS_DEFAULT) $(INC_FLAGS) $(LDFLAGS) -DLINUX
```

configure.gsi

```
CPP          =  cpp
CPP_FLAGS    =  -P -D_REAL8_ -DWRF -DLINUX
CPP_F90FLAGS  =  -traditional-cpp

# MPI compiler wrappers
DM_FC        =  mpif90  -f90=$(SFC)
DM_F90       =  mpif90 -free -f90=$(SFC)
DM_CC        =  icc
```

configure.gsi

```
CPP          =  cpp
CPP_FLAGS    =  -P -D_REAL8_ -DWRF -DLINUX
CPP_F90FLAGS  =  -traditional-cpp

# MPI compiler wrappers
DM_FC        =  mpif90
DM_F90       =  mpif90 -free
DM_CC        =  icc
```

configure.gsi

Library Paths

NETCDFPATH = /usr/local/netcdf3-ifort

NETCDFLIBS = -L$(NETCDFPATH) -lnetcdff -lnetcdf

WRF_DIR = /d1/stark/WRF/intel/release_3-5-1

Check that your system has libraries in the specified path and with the specified names.

Getting Help

- For more detailed information on installation see: GSI User's Guide, chapter 2
  - www.dtcenter.org/com-GSI/users/docs/index.php
- Check the FAQ
  - www.dtcenter.org/com-GSI/users/support/faqs/index.php
- Check the Known Issues
  - www.dtcenter.org/com-GSI/users/support/known_issues/
  - For further assistance contact:
    - gsi-help@ucar.edu

Building GSI with CMake

Overview

- Cross platform builds
- Open source and generally already installed on most platforms
- Out of source builds ensure that the source tree is not modified and allow for simple changes to build and link options.
- Provides an easily editable Cache file (text editor or cCMake) to tweak build options and provide a full description of final build
- Automatically finds libraries, files and executables based on environment variables, system settings, command-line definitions, hard coded paths based on hostname or other variable.
- Connects seamlessly to continuous build packages like Hudson/Jenkins
- Provides a built in unit and regression testing framework
- Is easily extensible and modular.
- Automatically determines dependencies and allows for parallel builds
- Allows for easy adaptation to various compilers and even works well with TAU profiling compiler scripts

Building with CMake

- CMake command is somewhat analogous to "configure"
- Run prior to building in order to identify compilers, locate packages and libraries needed for the build
- Allows for command line options, but basic command is "cmake path-to-GSI_ROOT"
  - Note that the path is not to GSI_ROOT/src, but rather to GSI_ROOT
- After completion, run "make –j N" to build in parallel
  - Unlike ordinary make, all objects and module files are contained in the build directory, not in the src directory.
  - Modules are moved to build/include
  - Binaries are in build/bin
  - Libraries are in build/lib
  - Object files are in build/src/CMakeFiles/*.dir

Sample output

mpotts@yslogin6:build — ssh -Y -K zin — 125×39

```
emc-lw-mpotts:~/data/comgsi/build 226; cmake -DBUILD_CORELIBS=ON ..
-- The C compiler identification is GNU 5.2.0
-- The CXX compiler identification is GNU 5.2.0
-- Check for working C compiler: /export/emc-lw-mpotts/mpotts/data/bin/gcc
-- Check for working C compiler: /export/emc-lw-mpotts/mpotts/data/bin/gcc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /export/emc-lw-mpotts/mpotts/data/bin/g++
-- Check for working CXX compiler: /export/emc-lw-mpotts/mpotts/data/bin/g++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- The Fortran compiler identification is GNU 5.2.0
-- Check for working Fortran compiler: /export/emc-lw-mpotts/mpotts/data/bin/gfortran
-- Check for working Fortran compiler: /export/emc-lw-mpotts/mpotts/data/bin/gfortran  -- works
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Checking whether /export/emc-lw-mpotts/mpotts/data/bin/gfortran supports Fortran 90
-- Checking whether /export/emc-lw-mpotts/mpotts/data/bin/gfortran supports Fortran 90 -- yes
Setting paths for Generic System
/export/emc-lw-mpotts/mpotts/data/comgsi
Setting GNU flags
Setting GNU Compiler Flags
-- Found MPI_C: /export/emc-lw-mpotts/mpotts/lib/libmpi.so
-- Found MPI_CXX: /export/emc-lw-mpotts/mpotts/lib/libmpi_cxx.so;/export/emc-lw-mpotts/mpotts/lib/libmpi.so
-- Found MPI_Fortran: /export/emc-lw-mpotts/mpotts/lib/libmpi_usempif08.so;/export/emc-lw-mpotts/mpotts/lib/libmpi_usempi_ign
ore_tkr.so;/export/emc-lw-mpotts/mpotts/lib/libmpi_mpifh.so;/export/emc-lw-mpotts/mpotts/lib/libmpi.so
-- Found NetCDF: /export/emc-lw-mpotts/mpotts/data/lib64/libnetcdff.a;/export/emc-lw-mpotts/mpotts/data/lib/libnetcdf.a
-- HDF5: Using hdf5 compiler wrapper to determine C configuration
-- HDF5: Using hdf5 compiler wrapper to determine Fortran configuration
-- Found HDF5: /export/emc-lw-mpotts/mpotts/data/lib/libhdf5_fortran.so;/export/emc-lw-mpotts/mpotts/data/lib/libhdf5.so;/usr
/lib64/librt.so;/export/emc-lw-mpotts/mpotts/data/lib/libsz.so;/export/emc-lw-mpotts/mpotts/data/lib/libz.so;/usr/lib64/libdl
.so;/usr/lib64/libm.so (found version "1.8.15.1") found components:  C HL
-- Could NOT find CURL (missing:  CURL_LIBRARY CURL_INCLUDE_DIR)
 trying to find lapack
-- Looking for Fortran sgemm
```

CMake options

- Build in debug mode—

```
cmake -DCMAKE_BUILD_TYPE=DEBUG Path-to-GSI
```

- Build core libraries

```
cmake -DBUILD_CORELIBS=ON Path-to-GSI
```

- Build Global without WRF dependencies (regional with WRF is default)

```
cmake -DBUILD_GLOBAL=ON -DUSE_WRF=OFF Path-to-GSI
```

Current configurations

- CMake is currently configured to fully support the following machines—
  - WCOSS
  - Theia
  - S4
  - *Cheyenne*

```
export HDF5_ROOT=/glade/u/apps/ch/opt/netcdf/4.4.1.1/intel//16.0.3
export WRFPATH=/glade/p/work/wrfhelp/PRE_COMPILED_CODE_CHEYENNE/WRFV3.9_intel_dmpar_large-file
module load intel/16.0.3 netcdf/4.4.1.1 impi/5.1.3.210 mkl/11.3.3 cmake/3.7.2

mkdir build; cd build
cmake -DHDF5_dl_LIBRARY_RELEASE:FILEPATH=/usr/lib64/libdl.so  -DBUILD_CORELIBS=ON \
-DHDF5_m_LIBRARY_RELEASE:FILEPATH=/usr/lib64/libm.so  ..
```

  - *Yellowstone*

```
setenv HDF5_ROOT /glade/apps/opt/hdf5/1.8.16/intel/16.0.2
setenv WRFPATH  /glade/p/work/wrfhelp/PRE_COMPILED_CODE/WRFV3.9_intel_dmpar_large-file
module load intel/16.0.3 netcdf/4.4.1 mkl/10.3.11 cmake/3.3.1

mkdir build; cd build
cmake -DBUILD_CORELIBS=ON -DHDF5_m_LIBRARY_RELEASE:FILEPATH=/usr/lib64/libm.so \
-DHDF5_rt_LIBRARY_RELEASE:FILEPATH=/usr/lib64/librt.so \
-DHDF5_dl_LIBRARY_RELEASE:FILEPATH=/usr/lib64/libdl.so -DCMAKE_EXE_LINKER_FLAGS:STRING="-parallel" ..
```

Building on a new system

- Specify desired compiler using Environment variables CC, CXX, FC
- Specify WRF location with environment variable WRFPATH
- Specify –DBUILD_CORELIBS=ON if you do not have pre-compiled libraries (bacio, sp, nemsio, etc.)
  - Using git --recursive to clone the repository from VLab will automatically pull all required library source code
- Manually specify location of NetCDF and HDF5 files if they are not in standard install locations

Tweaking CMake settings and dealing with errors

- Cmake creates a CMakeCache.txt file in the build directory
  - can be edited with any text editor (e.g. vim, emacs, etc.)
  - Fill in missing/NOT FOUND variables with correct paths/filenames
- Use the ccmake gui to edit and test changes on the fly
  - run "ccmake ." in the build directory
  - Type "t" to toggle advanced setting "on"
  - use "/" to search for variable names
  - Hit enter and then edit
  - Type "c" to run cmake
  - If no errors are encountered, type "g" to generate a new CMakeCache.txt file

ccmake sample

```
mpotts@yslogin2:build-y2 — ssh -Y -K -X yellowstone.ucar.edu — 131×40
                                                  Page 4 of 6
 FRAMEMODULE                      /glade/p/work/wrfhelp/PRE_COMPILED_CODE_CHEYENNE/WRFV3.9_intel_dmpar_large-file/frame/module_mac
 FRAMEPACK                        /glade/p/work/wrfhelp/PRE_COMPILED_CODE_CHEYENNE/WRFV3.9_intel_dmpar_large-file/frame/pack_utils
 GITCOMMAND                       /usr/bin/git
 GSICONTROL                       CONTROL_EXE-NOTFOUND
 HDF5_CXX_COMPILER_EXECUTABLE     HDF5_CXX_COMPILER_EXECUTABLE-NOTFOUND
 HDF5_C_COMPILER_EXECUTABLE       /glade/apps/opt/netcdf/4.4.1/intel/16.0.3/bin/h5cc
 HDF5_C_INCLUDE_DIR               /glade/apps/opt/netcdf/4.4.1/intel/16.0.3/include
 HDF5_DIFF_EXECUTABLE             /glade/apps/opt/netcdf/4.4.1/intel/16.0.3/bin/h5diff
 HDF5_DIR                         HDF5_DIR-NOTFOUND
 HDF5_Fortran_COMPILER_EXECUTAB   HDF5_Fortran_COMPILER_EXECUTABLE-NOTFOUND
 HDF5_Fortran_HL_INCLUDE_DIR      /glade/apps/opt/hdf5/1.8.16/intel/16.0.2/include
 HDF5_HL_INCLUDE_DIR              /glade/apps/opt/netcdf/4.4.1/intel/16.0.3/include
 HDF5_IS_PARALLEL                 OFF
 HDF5_dl_LIBRARY_DEBUG            HDF5_dl_LIBRARY_DEBUG-NOTFOUND
 HDF5_dl_LIBRARY_RELEASE          /usr/lib64/libdl.so
 HDF5_hdf5_LIBRARY_DEBUG          HDF5_hdf5_LIBRARY_DEBUG-NOTFOUND
 HDF5_hdf5_LIBRARY_RELEASE        /glade/apps/opt/netcdf/4.4.1/intel/16.0.3/lib/libhdf5.so
 HDF5_hdf5_fortran_LIBRARY_DEBU   HDF5_hdf5_fortran_LIBRARY_DEBUG-NOTFOUND
 HDF5_hdf5_fortran_LIBRARY_RELE   /glade/apps/opt/hdf5/1.8.16/intel/16.0.2/lib/libhdf5_fortran.so
 HDF5_hdf5_hl_LIBRARY_DEBUG       HDF5_hdf5_hl_LIBRARY_DEBUG-NOTFOUND
 HDF5_hdf5_hl_LIBRARY_RELEASE     /glade/apps/opt/hdf5/1.8.16/intel/16.0.2/lib/libhdf5_hl.so
 HDF5_hdf5hl_fortran_LIBRARY_DE   HDF5_hdf5hl_fortran_LIBRARY_DEBUG-NOTFOUND
 HDF5_hdf5hl_fortran_LIBRARY_RE   /glade/apps/opt/hdf5/1.8.16/intel/16.0.2/lib/libhdf5hl_fortran.so
 HDF5_m_LIBRARY_DEBUG             HDF5_m_LIBRARY_DEBUG-NOTFOUND
 HDF5_m_LIBRARY_RELEASE           /usr/lib64/libm.so
 HDF5_rt_LIBRARY_DEBUG            HDF5_rt_LIBRARY_DEBUG-NOTFOUND
 HDF5_rt_LIBRARY_RELEASE          /usr/lib64/librt.so
 HDF5_sz_LIBRARY_DEBUG            HDF5_sz_LIBRARY_DEBUG-NOTFOUND
 HDF5_sz_LIBRARY_RELEASE          /glade/apps/opt/netcdf/4.4.1/intel/16.0.3/lib/libsz.so
 HDF5_z_LIBRARY_DEBUG             HDF5_z_LIBRARY_DEBUG-NOTFOUND
 HDF5_z_LIBRARY_RELEASE           /glade/apps/opt/netcdf/4.4.1/intel/16.0.3/lib/libz.so
 HGCOMMAND                        /usr/bin/hg
 HOSTNAME                         yslogin2

HDF5_CXX_COMPILER_EXECUTABLE: HDF5 C++ Wrapper compiler.  Used only to detect HDF5 compile flags.
Press [enter] to edit option                                                    CMake Version 3.3.1
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently On)
```

Regression testing with CMAke

- CMake includes the CTest module by default
- GSI has been configured to run the EMC regression testing suite via CTest
  - Currently only supported on WCOSS, Theia, and S4 (Univ. Wisconsin)
  - Regression test cases need to be downloaded
  - Control experiment using trunk/master may need to be built/run
- Run using either ctest directly or "make test" with or without arguments
  - "make test ARGS="…." is the same as "ctest ARGS…."
  - To launch all 15 tests at once, run in parallel with "ctest –j 15"
  - To run individual tests, use "ctest –I start,stop,stride,testX,testY,….

Regression testing results

- Results will be run in a scratch directory where available
  - /ptmpp1/$LOGNAME on WCOSS
  - /scratch/NCEPDEV/stmp3/$LOGNAME on Theia
  - /scratch/short/$LOGNAME on S4
- Test directory will be based on the location of your build directory (e.g. _global_save_$LOGNAME_comgsi_build)
  - 15 subdirectories containing 4 directories each (loproc-exp, hiproc-exp, loproc-control, hiproc-control) + one more directory with comparison of results

Conclusions

- CMake is designed to make codes more portable
- Should run out of the box on EMC platforms and with slight tweaks on NCAR machines
- If libraries are not available, use in conjuction with git clone –recursive to pull src for library dependencies and then build libraries along with GSI
- Use ccmake or edit CMakeCache.txt file directly to identify any missing pieces to build (e.g. HDF5 libraries)
- Configure global option without WRF dependencies with
  `cmake –DBUILD_CORELIBS=ON –DUSE_WRF=OFF –DBUILD_GLOBAL=ON path-to-GSI`