Code Structure & Compilation

Christina Holt

Developmental Testbed Center

Slides adapted from HWRF Tutorial presentation by Timothy Brown

Overview

- HWRF Components
- System requirements
- Obtaining HWRF
- HWRF Settings
- Building HWRF

HWRF Components

- WRF (Weather Research and Forecasting)
- WPS (WRF Pre-processor)
- UPP (Unified Post Processor)
- HWRF Utilities
- GFDL Vortex Tracker
- NCEP Coupler
- MPIPOM-TC (Princeton Ocean Model)
- GSI (Gridpoint Statistical Interpolation)
- HWRF Run Component

System Requirements: Utilities

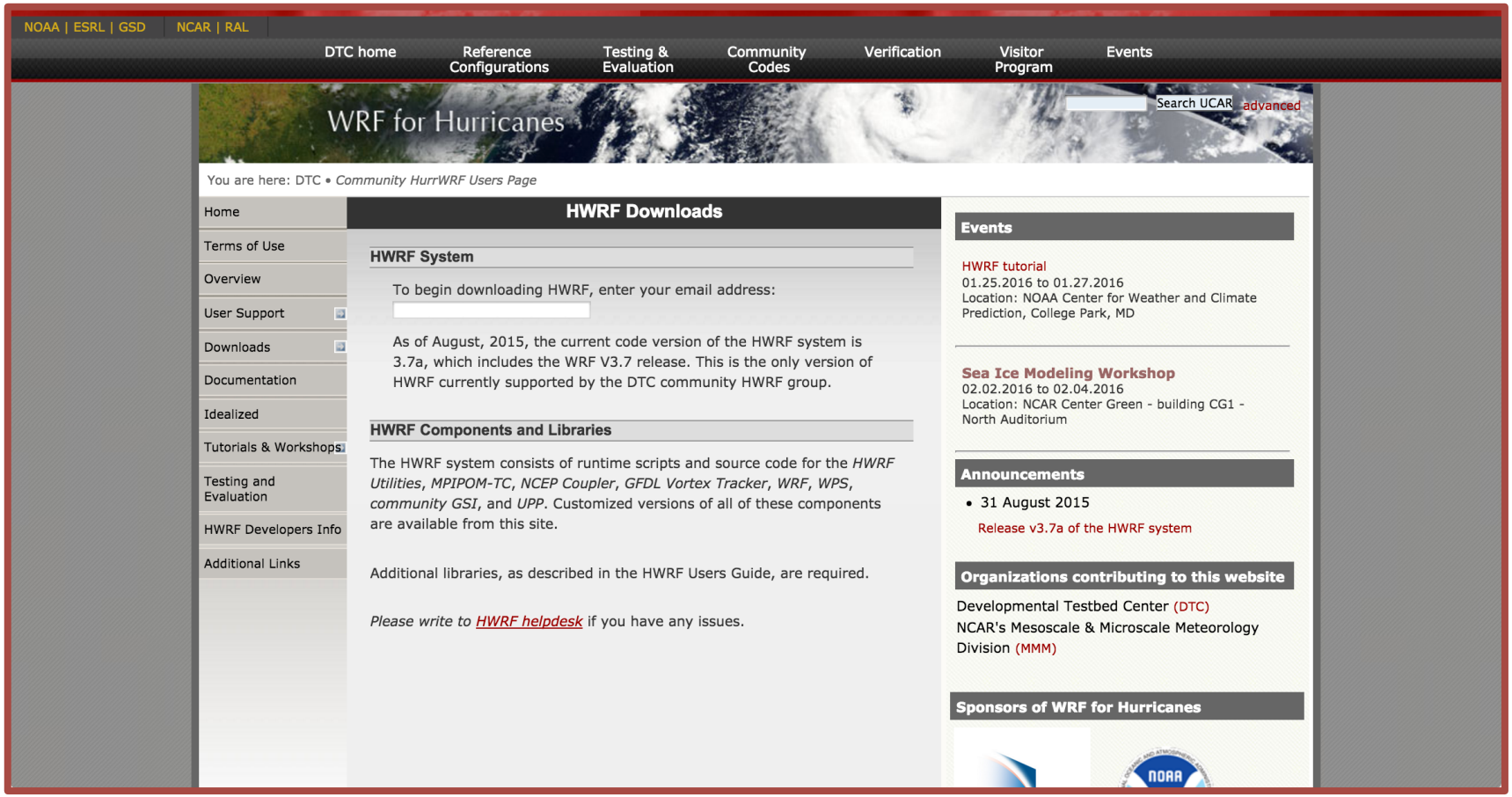
- FORTRAN 90+ compiler
- C compiler
- MPI v1.2+
- Perl
- netCDF v3.6+
  - if netCDF v4.1+ is used, HDF5 and SZIP libs may also be required
- LAPACK and BLAS (Linear algebra library)
- Python
- Parallel-netCDF
- PNG
- JasPer, Jpeg, zlib (GRIB 2 support)

System Requirements: Hardware

- Disk Space
  - Compressed source code: 3.3 Gb
  - Fully built system: 3.5 Gb
  - 126 hour forecast input data (GRIB only): 51Gb
  - 126 hour forecast output data:
    - $WORKhwrf: 516 Gb
    - $COMhwrf: 26 Gb
- Computation
  - Coupled atmosphere/ocean task uses 477 processors by default and takes about 3 hours
  - Many initialization tasks run with 48 processors
  - Others run with only one or two processors

Downloading HWRF

Source code is available on the DTC HWRF webpage



Setting Up HWRF

1. Extract the contents of the hwrfrun tar file

```
tar –xvf HWRF_v3.7a_hwrfrun.tar.gz
```

2. Move to the source directory of the unpacked hwrfrun directory `cd hwrfrun/sorc`

3. Unpack the remaining tar files

```
for i in $(ls *tar.gz)
> do
> tar –xzvf $i
> done
```

4. Check that all 8 source directories exist

```
$ ls
gfdl-vortextracker  GSI        hwrf-utilities
ncep-coupler        pomtc      UPP
WPSV3               WRFV3
```

Setting Up HWRF

5. Check for patches to the code

http://www.dtcenter.org/HurrWRF/users/support/known_issues/index_v3.7a.php

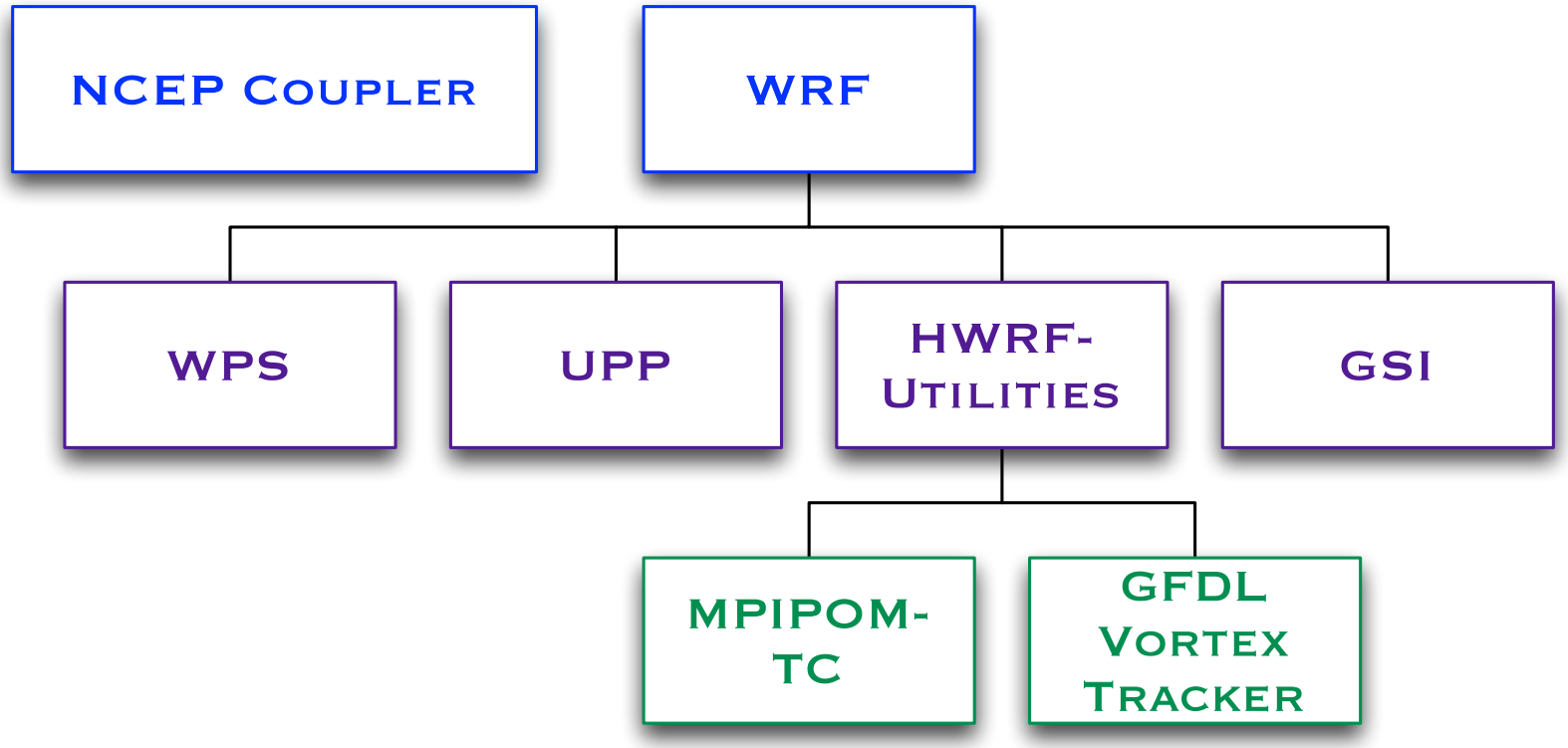
6. Load the required modules

```
module load
```

7. Set external library paths

```
export LIB_Z_PATH=/usr/lib64
export LIB_PNG_PATH=/usr/lib64
export LIB_JASPER_PATH=/usr/lib64
export INC_JASPER_PATH=/usr/include
export JASPERLIB=/usr/lib64
export JASPERINC=/usr/include
```

Build Dependencies



Environment Variables Needed

WRF

```
HWRF=1
WRF_NMM_CORE=1
PNETCDF_QUILT=1
WRF_NMM_NEST=1
WRFIO_NCD_LARGE_FILE_SUPPORT=1
```

POM-TC & Tracker

```
WRF_DIR=${HWRFsorc}/WRFV3
LIB_W3_PATH=${HWRFsorc}/hwrf-utilities/libs
LIB_SP_PATH=${HWRFsorc}/hwrf-utilities/libs
LIB_BACIO_PATH=${HWRFsorc}/hwrf-utilities/libs
LIB_BLAS_PATH=${HWRFsorc}/hwrf-utilities/libs
LIB_SFCIO_PATH=${HWRFsorc}/hwrf-utilities/libs
LIB_G2_PATH=${HWRFsorc}/hwrf-utilities/libs

LAPACK_PATH=${MKLROOT}
```



```
${HWRFsorc}=${HWRFhome}/sorc
```

Component Directories

- Each component directory contains its own set of scripts and source code
- There are variations on the naming convention of each component, but they all contain the following basics
  - See the HWRF Users Guide v3.7a for specific naming

`arch/`	architecture compiling options
`clean`	script for cleaning compiled files and exectutables
`compile`	script for compiling
`configure`	script for creating configure file
`exec/`	compiled executables (also main/ and bin/)
`src/`	source code

Building a Component

- Set the necessary environment variables
- Clean:

```
./clean
```

- Configure:

```
./configure
```

```
1. Linux x86_64, PGI compiler (dmpar)
2. Linux x86_64, PGI compiler, SGI MPT (dmpar)
3. Linux x86_64, Intel compiler (dmpar)
4. Linux x86_64, Intel compiler, SGI MPT (dmpar)
5. Linux x86_64, Intel compiler, IBM POE (dmpar)
```

- Compile:

```
./compile 2>&1 | tee build.log
```

- Check the executables

Cleaning a Component

- Clean script usage:
  - No argument – removes object files, pre-processed files, and executables

```
./clean
```

  - With –a argument – removes ALL built files, including the configuration file
    - Use this option if the compilation fails or if re-configuration is required

```
./clean –a
```

Configuring a Component

- Run the configure script (no arguments)

```
./configure
```

- The list for each component is slightly different, but reflects the options available for your system based on the compiler and architecture.

```
1.  Linux x86_64, PGI compiler (dmpar)
2.  Linux x86_64, PGI compiler, SGI MPT (dmpar)
3.  Linux x86_64, Intel compiler (dmpar)
4.  Linux x86_64, Intel compiler, SGI MPT (dmpar)
5.  Linux x86_64, Intel compiler, IBM POE (dmpar)
```

- Choose an appropriate option
  - All components must use the same compiler
- A configure file (configure.###) will be created
  - configure.wps, configure.wrf, etc.

Configure File

- File generated by running the configure script
- A list of variables controlling the compiler and flags for the architecture choice in the configure menu option list
- The configure.### files may be edited after running the respective configure scripts
- To permanently change options, change the settings for your platform in arch/configure.defaults before running clean and configure again

Compiling a Component

- Run the compile script (additional argument needed for WRF)
- Capture the standard error and standard out in a file while sending it to the screen, as well

Most components:

```
./compile |& tee build.log
```

WRF component:

```
./compile nmm_real |& tee build.log
```

Compile Log

- The standard error and output of each component should be redirected to a text file
- The log file will reflect each file that is compiled for the component, along with any errors, warnings, etc.
- A successful build of a component should not contain an instance of the string “Error”

Check the Executables

- Each component generates executables indifferent directories

```
sorc/WRFV3 $ ls main
convert_em.F   ideal_nmm.F   module_wrf_top.F   module_wrf_top.o
real_em.F      real_nmm.f90  wrf_ESMFMod.F      wrf.f90
depend.common  libwrflib.a   module_wrf_top.f90 ndown_em.F
real_nmm.exe   real_nmm.o    wrf.exe            wrf.o
ideal_em.F     Makefile      module_wrf_top.mod nup_em.F
real_nmm.F     tc_em.F       wrf.F              wrf_SST_ESMF.F
```

- Refer to the HWRF Users Guide v3.7 for a complete list of executables for each component

Building the Idealized

- The Idealized TC WRF creates two unique executables that cannot be interchanged with the standard WRF ones

```
$ tar xzf HWRF_v3.7a_WRFV3.tar.gz
$ mv WRFV3 WRFV3_idealized
$ cd WRFV3_idealized
[WRFV3_idealized]$ setenv IDEAL_NMM_TC=1
[WRFV3_idealized]$ ./clean –a
[WRFV3_idealized]$ ./configure
[WRFV3_idealized]$ ./compile nmm_tropical_cyclone |& tee compile.ideal
```

- Successful compilation results in `ideal.exe` and `wrf.exe`

```
sorc/WRFV3 $ ls main
convert_em.F   ideal_nmm.F   module_wrf_top.F   module_wrf_top.o
real_em.F      real_nmm.f90  wrf_ESMFMod.F      wrf.f90
depend.common  libwrflib.a   module_wrf_top.f90 ndown_em.F
ideal_em.F     real_nmm.o    wrf.exe            wrf.o
ideal.exe      Makefile      module_wrf_top.mod nup_em.F
real_nmm.F     tc_em.F       wrf.F              wrf_SST_ESMF.F
```

Summary

- Download tar files from the website
- Only use components from a single release
- Set the necessary environment variables and load modules
- Build each component with ./configure and ./compile in the proper order
- Keep the HWRF Users' Guide v3.7a handy

http://www.dtcenter.org/HurrWRF/users/docs/users_guide/HWRF_v3.7a_UG.pdf

For more help, email:

hwrf-help@ucar.edu