WRF Data Assimilation System

Michael Kavulich, Jr.

Special thanks to:

Xin Zhang, Xiang-Yu Huang

WRFDA Tutorial, July 2014, NCAR

Many slides are borrowed from WRF software lectures
WRFDA System – Outline

- *Introduction*
- WRFDA Software Overview
- Computing Overview
Introduction – What is WRFDA?

• A data assimilation system for the WRF Model (ARW core)
  – 3D- and 4D-VAR, FGAT, Ensemble, and Hybrid methods
• Designed to be flexible, portable and easily installed and modified
  – Open-source and public domain
  – Can be compiled on a variety of platforms
  – Part of the WRF Software Framework
• Designed to handle a wide variety of data
  – Conventional observations
  – Radar velocity and reflectivity
  – Satellite (radiance and derived data)
  – Accumulated precipitation
The following data are not processed by OBSPROC:

- Radar, Precipitation data in ASCII format (require separate pre-processing)
- Conventional obs in PREPBUFR format
- Radiance, GPSRO in BUFR format

Blue: Supported by WRFDA team
WRFDA System – Outline

• Introduction
• WRFDA Software
• Computing Overview
program da_wrfvar_main
!----------------------------------------------------------------------------
! Purpose: Main program of WRF-Var. Responsible for starting up, reading 
in (and broadcasting for distributed memory) configuration data, defining 
and initializing the top-level domain, either from initial or restart 
data, setting up time-keeping, and then calling the da_solve 
routine assimilation. After the assimilation is completed, 
the model is properly shut down.
!----------------------------------------------------------------------------
use module_symbols_util, only : wrfu_finalize
use da_control, only : trace_use, var4d
use da_tracing, only : da_trace_init, da_trace_report, da_trace_entry, &
da_trace_exit
use da_wrf_interfaces, only : wrf_shutdown, wrf_message, disable_quilting
use da_wrfvar_top, only : da_wrfvar_init1,da_wrfvar_init2,da_wrfvar_run, &
da_wrfvar_finalize
#ifdef VAR4D
use da_4dvar, only : clean_4dvar, da_finalize_model
#endif
implicit none
! Split initialisation into 2 parts so we can start and stop trace here
    call disable_quilting
call da_wrfvar_init1
    if (trace_use) call da_trace_init
    if (trace_use) call da_trace_entry("da_wrfvar_main")
call da_wrfvar_init2
call da_wrfvar_run
call da_wrfvar_finalize
#endif VAR4D
    if (var4d) then
        call clean_4dvar
        call da_finalize_model
    end if
#endif
    call wrf_message("*** WRF-Var completed successfully ***")
    if (trace_use) call da_trace_exit("da_wrfvar_main")
    if (trace_use) call da_trace_report
call wrfu_finalize
call wrf_shutdown
end program da_wrfvar_main
WRFDA Directory structure

```
arch
clean
compile
configure
dyn_em
dyn_exp
external
frame
inc
main
Makefile
phys
README.DA
Registry
run
share
test
tools
var
```

**Legend:**
- Blue – directory
- Green – script file
- Gray – other text file

**Build scripts**

- `registry.var`
- `WRFDA source code directory`
- `README file with information about WRFDA`
WRFDA/var Directory structure

- **build** → Executables built here
- **convertor**
- **da** → WRFDA main source code contained here
- **external** → Source code for external libraries (CRTM, BUFR, etc.)
- **gen_be** → GEN_BE source code
- **graphics**
- **Makefile**
- **obsproc** → OBSPROC source code
- **README.basics**
- **README.namelist**
- **README.radiance** → More README files with useful information
- **run** → Useful runtime files (mostly for radiance)
- **scripts**
- **test** → Data for tutorial cases

Legend:
- Blue – directory
- Green – script file
- Gray – other text file
Hierarchical software architecture

- **Insulate** scientists' code from parallelism and other architecture/implementation-specific details
- **Well-defined interfaces** between layers, and external packages for communications, I/O.
• **Registry**: an “Active” data dictionary
  - Tabular listing of model state and attributes
  - Large sections of interface code generated automatically
  - Scientists manipulate model state simply by modifying Registry, without further knowledge of code mechanics
  - `registry.var` is the dictionary for WRFDA
### WRFDA Software – Architecture

<table>
<thead>
<tr>
<th>Variable type</th>
<th>Variable name</th>
<th>Namelist name</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rconfig</td>
<td>integer</td>
<td>rttov_emis_atlas_ir</td>
<td>1 0 - &quot;rttov_emis_atlas_ir&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>integer</td>
<td>rttov_emis_atlas_mw</td>
<td>1 0 - &quot;rttov_emis_atlas_mw&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>integer</td>
<td>rtminit_print</td>
<td>1 1 - &quot;rtminit_print&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>integer</td>
<td>rtminit_nsensor</td>
<td>max_instruments -1 - &quot;rtminit_platform&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>integer</td>
<td>rtminit_platform</td>
<td>max_instruments -1.0 - &quot;rtminit_satid&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>integer</td>
<td>rtminit_satid</td>
<td>max_instruments -1.0 - &quot;rtminit_sensor&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>integer</td>
<td>rad_monitoring</td>
<td>max_instruments -1.0 - &quot;rad_monitoring&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>real</td>
<td>thinning_mesh</td>
<td>max_instruments 60.0 - &quot;thinning_mesh&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>thinning</td>
<td>namelist,wrfvar14 1 .true. - &quot;thinning&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>read_biascoef</td>
<td>namelist,wrfvar14 1 .false. - &quot;read_biascoef&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>biascorr</td>
<td>namelist,wrfvar14 1 .false. - &quot;biascorr&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>biasprep</td>
<td>namelist,wrfvar14 1 .false. - &quot;biasprep&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>rttov_scatt</td>
<td>namelist,wrfvar14 1 .false. - &quot;rttov_scatt&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>write_profile</td>
<td>namelist,wrfvar14 1 .false. - &quot;write_profile&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>write_jacobian</td>
<td>namelist,wrfvar14 1 .false. - &quot;write_jacobian&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>qc_rad</td>
<td>namelist,wrfvar14 1 .true. - &quot;qc_rad&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>write_iv_rad_ascii</td>
<td>namelist,wrfvar14 1 .false. - &quot;write_iv_rad_ascii&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>write_oa_rad_ascii</td>
<td>namelist,wrfvar14 1 .false. - &quot;write_oa_rad_ascii&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>write_filtered_rad</td>
<td>namelist,wrfvar14 1 .false. - &quot;write_filtered_rad&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>use_error_factor_rad</td>
<td>namelist,wrfvar14 1 .false. - &quot;use_error_factor_rad&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>use_landem</td>
<td>namelist,wrfvar14 1 .false. - &quot;use_landem&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>use_antcorr</td>
<td>namelist,wrfvar14 max_instruments .false. - &quot;use_antcorr&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>use_mspps_emis</td>
<td>namelist,wrfvar14 max_instruments .false. - &quot;use_mspps_emis&quot;</td>
</tr>
<tr>
<td>rconfig</td>
<td>logical</td>
<td>use_mspps_ts</td>
<td>namelist,wrfvar14 max_instruments .false. - &quot;use_mspps_ts&quot;</td>
</tr>
</tbody>
</table>
## WRFDA Software – Architecture

<table>
<thead>
<tr>
<th>Driver Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domains</strong>: Allocates, stores, decomposes, represents abstractly as single data objects</td>
</tr>
</tbody>
</table>
Minimization/Solver Layer

- Minimization/Solver routine, choose the function based on the namelist variable, 3DVAR, 4DVAR, FSO or Verification, and choose the minimization algorithm.
• **Observation Layer**
  – **Observation interfaces**: contains the gradient and cost function calculation subroutines for each type of observations.
Call Structure Superimposed on Architecture

\[ \text{da\_sound\_f90(da\_sound)} \]
WRFDA and J

\[ J(x) = \frac{1}{2} (x - x^b)^T B_0^{-1} (x - x^b) + \frac{1}{2} (y_0 - H(x))^T R^{-1} (y_0 - H(x)) \]

- Model background \(x^b\)
- Background error \(B_0\)
- Observations \(y_0\) and their associated error statistics \(R\)
- Minimize this cost function \(J(x)\) to find the analysis \(x^a\)
- Run forecast, repeat for cycling mode
WRFDA broken down by process

Read namelist

Set up framework

Set up background

Set up observations and error

Set up background error

Minimize cost function

Calculate \( y - H(x) \)

Calculate diagnostics

Formulate analysis

Compute analysis

Diagnostics

Clean up

Outer loop

\( x_b \, y, R \)

\( X_a \)
WRFDA broken down by process

**Input files**

- Read namelist
- Set up framework
- Set up background
- Set up observations and error
- Set up background error

**Namelist**

- \( x_b \)

**Variables**

- \( y, R \)
- \( B \)

**Outer loop**

- Minimize cost function
- Calculate \( y - H(x) \)
- Formulate analysis

**Diagnostics**

- Compute analysis
- Calculate diagnostics

**Clean up**

- Clean up

**Symbols**

- \( x_a \)
Input files: Namelist

- **File name:** `namelist.input`
- Specifies Input/Output options, domain details, types of observations to assimilate and how to assimilate them
- Allows user great flexibility to change the usage of WRFDA without having to recompile
- A large number (>1000) of namelist options govern the running of WRFDA; however, users will typically only be concerned with setting a few dozen of these
- More details can be found in the User’s Guide
Input files: $x_b$ (background)

- File name: $fg$
- Can be either a WRF input file created by WPS and real.exe, or a WRF output file from a forecast.
Input files: $y$ (observations) and $R$ (observation errors)

- **File name:** `ob.ascii`, `amsua.bufr`, `ob01.rain`, etc.
- **WRFDA accepts a wide variety of observations in several different formats**
  - OBSPROC ASCII format (surface, sounding, GPS, etc.)
  - PREPBUFR format (surface, sounding, etc.)
  - BUFR format (radiance)
  - Other ASCII format (radar, precipitation)
- **Observation errors are either provided in the observation file, or standard errors (file name: obserr.txt) are used.**
Input files: \textbf{B (background error)}

- **File name:** \texttt{be.dat}
- **This is a binary file containing background error information**
  - \texttt{cv\_options=3} NCEP background error formulation
    - File provided with WRFDA code
    - Not recommended: should be used with caution
  - \texttt{cv\_options=5} NCAR background error formulation
    - File created using \texttt{gen\_be} utility
    - Recommended option
  - \texttt{cv\_options=6;} Multivariate Background Error (MBE) statistics
    - Still experimental: not officially supported
WRFDA broken down by process

- **Namelist**
  - Read namelist
  - Set up framework
  - Set up background
  - Set up background error

- **Y, R**
  - Set up observations and error

- **B**
  - Set up background error

- **Minimize cost function**
  - Compute analysis
  - Minimize cost function
  - Calculate $y - H(x)$

- **Outer loop**
  - Formulate analysis
  - Calculate $y - H(x)$

- **WRFDA Processes**
  - Calculate diagnostics
  - Minimize cost function
  - Formulate analysis

- **Clean up**
  - Formulate analysis
  - Clean up

**Diagnostics**

- **Xa**
Read namelist

Set up framework

Set up background

Set up observations and error

Set up background error

Minimize cost function

Calculate \( y - H(x) \)

Calculate diagnostics

Formulate analysis

Clean up

Diagnostics

Outer loop
Read namelist

- Read user-specified options from `namelist.input`
- Set default values for options *not* specified in the namelist
- Perform consistency checks between namelist options

Calling order:
```
da_wrfvar_main ==> call da_wrfvar_init1, da_wrfvar_init2 ==> call initial_config
```

Calling subroutines:
```
da_wrfvar_main.f90 ==> da_wrfvar_init1.inc, da_wrfvar_init2.inc ==> module_configure.F
```
Set up framework

Read namelist → Set up framework → Set up background → Set up observations and error → Set up background error

Minimize cost function → Calculate $y - H(x)$ → Outer loop

Compute analysis → Minimize cost function

Calculate diagnostics → Formulate analysis

Diagnostics → Clean up

$\mathbf{x}_b$ → $\mathbf{y}, \mathbf{R}$ → $\mathbf{B}$
Set up framework

- Utilize WRF Software Framework distributed memory capability to allocate and configure the domain
- Allocate needed memory, initializes domain and tile dimensions, etc.
- Create output files

Calling order:
da_wrfvar_main ==> call da_wrfvar_init2 ==> call alloc_and_configure_domain
da_wrfvar_main ==> call da_wrfvar_run.inc ==> call da_wrfvar_interface ==> call da_solve ==> call da_solve_init

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_init2.inc ==> module_domain.F
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_solve_init.inc
Set up background

- **Namelist**
  - Read namelist
  - Set up framework

- **Set up background**
  - $X_b$
  - $y, R$
  - $B$

- **Set up observations and error**

- **Set up background error**

- **Minimize cost function**

- **Calculate $y - H(x)$**

- **Outer loop**

- **Formulate analysis**

- **Clean up**

- **Calculate diagnostics**

- **Diagnostics**

- **Clean up**

- $X_a$
Set up background

- Read the first-guess file
- Extract fields used by WRFDA
- Create background FORTRAN 90 derived data type $xb$, etc.

Calling order:

da_wrfvar_main ==> call da_wrfvar_init2 ==> call da_med_initialdata_input
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==> call da_setup_firstguess

Calling subroutines:

da_wrfvar_main.f90 ==> da_wrfvar_init2.inc ==> da_med_initialdata_input.inc
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_setup_firstguess.inc
Set up observations and error

1. Read namelist
2. Set up framework
3. Set up background
4. Set up observations and error
5. Compute analysis
6. Minimize cost function
7. Calculate $y - H(x)$
8. Calculate diagnostics
9. Formulate analysis
10. Outer loop
11. Clean up
12. Namelist
13. $x_b$
14. $y, R$
15. $B$
16. Diagnostics
17. $x_a$
Set up observations and error

- Read in observations
- Assign observational error
- Create observation FORTRAN 90 derived data type \textit{ob}
- Domain and time check

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==> call da_setup_obs_structures

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_setup_obs_structures.inc
Set up background error

- Read namelist
- Set up framework
- Set up background
- Set up observations and error
- Set up background error

- Compute analysis
- Minimize cost function
- Calculate $y - H(x)$
- Outer loop
- Calculate diagnostics
- Formulate analysis
- Clean up

- Diagnostics
- $x_b$
- $y, R$
- $B$
Set up background error

- Reads in background error statistics from be.dat
- Extracts necessary quantities: eigenvectors, eigenvalues, lengthscales, regression coefficients, etc.
- Creates background error FORTRAN 90 derived data type \textit{be}
- Specifics of background error in WRFDA be covered in more detail in a later talk

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==> call da_setup_background_errors

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_setup_background_errors.inc
Calculate $y - H(x)$

- **Read namelist**
- **Set up framework**
- **Set up background**
- **Set up observations and error**
- **Set up background error**
- **Minimize cost function**
- **Calculate $y - H(x)$**
- **Compute analysis**
- **Outer loop**
- **Calculate diagnostics**
- **Formulate analysis**
- **Clean up**
Calculate $y - H(x)$ (Innovation)

- Calculate model equivalent of observations through interpolation and variable transformations
- Compute observation minus first guess ($y - H(x)$) value
- Create innovation vector FORTRAN 90 derived data type $iv$

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==>
call da_solve ==> call da_get_innov_vector, da_allocate_y

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==>
da_solve.inc ==> da_get_innov_vector.inc, da_allocate_y.inc
Minimize cost function

- Read namelist
- Set up framework
- Set up background
- Set up observations and error
- Set up background error
- Compute analysis
- Calculate diagnostics
- Calculate $y - H(x)$
- Formulate analysis
- Clean up

Outer loop

$X_a$
Minimize cost function

- Use conjugate gradient method
  - Initializes analysis increments to zero
  - Computes cost function (if desired)
  - Computes gradient of cost function
  - Uses gradient of the cost function to calculate new value of analysis control variable
- Increment this process until specified minimization is achieved

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==> call da_minimise_cd

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_minimise_cd.inc

Compute analysis

**Namelist**
- Read namelist
- Set up framework

**Outer loop**

**Diagnostics**
- Compute diagnostics

**Set up background**
- Set up background

**Set up observations and error**
- Set up background error

**Set up framework**
- \( x_b \)

**Calculate**
- \( y - H(x) \)

**Formulate analysis**
- \( R \)

**Minimize cost function**
- Minimize cost function

**Clean up**
- \( x_a \)
Compute analysis

- Convert control variables to model space analysis increments
- Calculate analysis = first-guess + analysis increment
- Perform consistency checks (e.g., remove negative humidity)

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==> call da_transfer_xatoanalysis

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_transfer_xatoanalysis.inc
Calculate diagnostics

Read namelist -> Set up framework -> Set up background -> Set up observations and error -> Set up background error

Compute analysis -> Minimize cost function -> Calculate $y - H(x)$

Outer loop

Calculate diagnostics

Formulate analysis -> Clean up

Diagnostics
Calculate diagnostics

- Output $y - H(x_b), y - H(x_a)$ statistics for all observation types and variables
- Compute $x_a - x_b$ (analysis increment) statistics for all model variables and levels
- Statistics include minimum, maximum (and their locations), mean and standard deviation.

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==> call da_transfer_xatoanalysis

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_transfer_xatoanalysis.inc
Outer loop

- **Read namelist**
- **Set up framework**
- **Set up background**
- **Set up observations and error**
- **Set up background error**

**Namelist**

- **Compute analysis**
- **Minimize cost function**
- **Calculate y − H (x)**

**Outer loop**

- **Calculate diagnostics**
- **Formulate analysis**
- **Clean up**

**Diagnostics**
Outer loop

• An outer loop is a method of iterative assimilation to maximize contributions from observations non-linearly related to the control variables (e.g., GPS refractivity, Doppler radial velocity)
  – After the previous steps, the analysis $x_a$ is used as the new first guess
  – The cost function minimization and diagnostic steps are repeated
  – This can be repeated up to ten times

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc

Further reading: Rizvi et al., 2008 (http://www.mmm.ucar.edu/wrf/users/workshops/WS2008/abstracts/P5-03.pdf)
Write analysis

- **Namelist**
  - Read namelist
  - Set up framework
  - Set up background
  - Set up background error

- **x_b**
- **y, R**
- **B**

- **Minimize cost function**
  - Minimize cost function
  - Compute analysis
  - Calculate diagnostics
  - Set up observations and error

- **Outer loop**
- **Calculate y - H(x)**

- **Formulate analysis**
  - Formulate analysis

- **Clean up**
  - Clean up
Write analysis

- Write analysis file in native WRF format.

Calling order:
da_wrfvar_main ==> call da_wrfvar_run ==> call da_wrfvar_interface ==> call da_solve ==> call da_transfer_xatoanalysis

Calling subroutines:
da_wrfvar_main.f90 ==> da_wrfvar_run.inc ==> da_wrfvar_interface.inc ==> da_solve.inc ==> da_transfer_xatoanalysis.inc
Clean up

- Read namelist
- Set up framework
- Set up background
- Set up observations and error
- Set up background error
- Compute analysis
- Minimize cost function
- Calculate $y - H(x)$
- Formulate analysis
- Outer loop
- Calculate diagnostics
- Diagnostics
- $x_b$
- $y, R$
- $B$
- Clean up
Clean up

- Deallocate dynamically-allocated arrays, structures, etc.
- Timing information
- Clean end to WRFDA

Calling order:
d_a_wrfvar_main ==> call d_a_wrfvar_run ==> call d_a_wrfvar_interface ==> call d_a_solve
d_a_wrfvar_main ==> call d_a_wrfvar_finalize

Calling subroutines:
d_a_wrfvar_main.f90 ==> d_a_wrfvar_run.inc ==> d_a_wrfvar_interface.inc ==> d_a_solve.inc
d_a_wrfvar_main.f90 ==> d_a_wrfvar_finalize.inc
WRFDA broken down by process

Read namelist → Set up framework → Set up background → Set up observations and error → Set up background error

Calculate $y - H(x)$ → Calculate diagnostics → Formulate analysis

Minimize cost function → Compute analysis

Outer loop

Clean up

Diagnostics

Output files

$x_a$
Output files: Diagnostics

- **File names:** `grad_fn, jo, qcstat_conv*, statistics, etc.`
- There will be a number of diagnostics files output by WRFDA
  - Many will end in `.0000, .0001, etc.;` these are diagnostics specific to each processor used
  - Many will also contain a `_01;` these files will appear for each outer loop as `_02, _03, etc.`
- More or fewer output files can be specified by certain namelist options
Output files: $x_a$ (analysis)

- **File name:** `wrfvar_output`
- This is the model output in WRF native format. This file can be used directly for research purposes, or used to initialize a WRF forecast.
Cycling mode

- Because WRFDA takes WRF forecast files as input, the system can naturally be run in cycling mode
- WRFDA initializes a WRF forecast, the output of which is fed back into WRFDA to initialize another WRF forecast
- Requires boundary condition updating
Cycling mode

WRFDA in the WRF Modeling System

Background Preprocessing (WPS, real)

\( x^b \)

Cold-Start Background

Update Low BC (UPDATE_BC)

Update Lateral BC (UPDATE_BC)

Forecast (WRF)

Observation Preprocessing (OBSPROC)

\( y^o, R \)

WRFDA

\( x^q \)

Background Error (gen_be)

\( B_0 \)

Further reading: User’s Guide, Chapter 6, section “Updating WRF Boundary Conditions”

The following data are not processed by OBSPROC:

- Radar in ASCII (requires separate pre-processing)
- PREPBUFR (+ GPSRO in BUFR)
- Radiance in BUFR format
WRFDA System – Outline

- Introduction
- WRFDA Software Overview
- Computing Overview
WRFDA Parallelism

- WRFDA can be run serially or as a parallel job
- WRFDA uses *domain decomposition* to divide total amount of work over parallel processes
- The *decomposition* of the application over processes has two levels:
  - The *domain* is broken up into rectangular pieces that are assigned to MPI (distributed memory) processes. These pieces are called *patches*
  - The *patches* may be further subdivided into smaller rectangular pieces that are called *tiles*, and these are assigned to *shared-memory threads* within the process.
- However, *WRFDA does not support shared memory parallelism!* So distributed memory is what I will cover here.
Parallelism in WRFDA: Multi-level Decomposition

Inter-processor communication
Distributed Memory Communications

When Needed?

Communication is required between patches when a horizontal index is incremented or decremented on the right-hand-side of an assignment.

Why?

On a patch boundary, the index may refer to a value that is on a different patch.

Following is an example code fragment that requires communication between patches:

Note the tell-tale \( +1 \) and \( -1 \) expressions in indices for \( \text{rr} \), \( \text{H1} \), and \( \text{H2} \) arrays on right-hand side of assignment.

These are \textit{horizontal data dependencies} because the indexed operands may lie in the patch of a neighboring processor. That neighbor’s updates to that element of the array won’t be seen on this processor.
subroutine da_transfer_xatowrf(grid)

   do k=kts,kte
      do j=jts,jte+1
         do i=its,ite+1
            u_cgrid(i,j,k)=0.5*(grid%xa%u(i-1,j,k)+grid%xa%u(i,j,k))
            v_cgrid(i,j,k)=0.5*(grid%xa%v(i,j-1,k)+grid%xa%v(i,j,k))
         end do
      end do
   end do
.
.

subroutine da_transfer_xatowrf(grid)
.
  do k=kts,kte
    do j=jts,jte+1
      do i=its,ite+1
        u_cgrid(i,j,k) = 0.5*(grid%xa%u(i-1,j ,k)+grid%xa%u(i,j,k))
        v_cgrid(i,j,k) = 0.5*(grid%xa%v(i ,j-1,k)+grid%xa%v(i,j,k))
      end do
    end do
  end do
.

Distributed Memory Communications

- Halo updates

memory on one processor

memory on neighboring processor
Distributed Memory Communications

Halo (contains information about adjacent patch)
Halo (contains information about adjacent patch)

Inter-processor communication
(Halos update from adjacent patch after each minimization step)
Grid Representation in Arrays

- Increasing indices in WRFDA arrays run
  - West to East (X, or I-dimension)
  - South to North (Y, or J-dimension)
  - Bottom to Top (Z, or K-dimension)

- Storage order in WRFDA is IJK, but for WRF, it is IKJ (ARW) and IJK (NMM)

- Output data has grid ordering independent of the ordering inside the WRFDA model
Grid Representation in Arrays

• The extent of the logical or *domain* dimensions is always the "staggered" grid dimension. That is, from the point of view of a non-staggered dimension (also referred to as the ARW “mass points”), there is always an extra cell on the end of the domain dimension

• In WRFDA, the minimization is on A-grid (non-staggered grid). The wind components will be interpolated from A-grid to C-grid (staggered grid) before they are output, to conform with standard WRF format
WRFDA I/O

• Streams: pathways into and out of model
  – Input
    • $fg$ is the name of the input
    • $wrfvar_output$ is the name of output
  – Boundary
    • Only needed for 4DVAR.
Summary

- WRFDA is designed to be an easy-to-use data assimilation system for use with the WRF model
- WRFDA is designed within the WRF Software Framework for rapid development and ease of modification
- WRFDA can be run in parallel for quick assimilation of large amounts of data
Appendix – WRFDA Resources

• WRFDA users page
  – Download WRFDA source code, test data, related packages and documentation
  – Lists WRFDA news and developments

• Online documentation
  – Chapter 6 of the WRF Users’ Guide; documents installation of WRFDA and running of various WRFDA methods

• WRFDA user services and help desk
  – wrfhelp@ucar.edu
Appendix – Derived Data Structures

• Driver layer
  – All data for a domain is an object, a domain **derived data type** (DDT)
  – The domain DDT is dynamically allocated/deallocated
  – Only one DDT is allowed in WRFDA; it is **head_grid**, defined in frame/module_domain.F
  – WRFDA doesn’t support nested domains.

• Every Registry defined **state, I1**, and **namelist** variable is contained inside the DDT (locally known as a **grid** of type **domain**), where each node in the tree represents a separate and complete 3D model domain/nest.
Appendix – Derived Data Structures

- **cvt**
  - Real type array to store the control variables
  - It is an all-ZERO array during the first outer loop and will be updated at the end of each outer loop

- **xhat**
  - Real type array to store the control variables
  - It stores the control variables for each inner loop.

- **be**
  - It is used to store the background error covariance.
Appendix – Derived Data Structures

- **iv**
  - Stores the innovations for each observational type $y - Hx$
- **ob**
  - Stores the observations $y$
- **re**
  - Store the residual $y - H(x + \Delta x)$
Appendix – WRFDA structure

- Primarily written in Fortran and C
- Part of the WRF Software Framework
  - Hierarchical organization
  - Multiple functions (3DVAR, 4DVAR, etc.) use the same framework to simplify development
  - Plug observation type interface
Appendix – Parallel Computing Terms (Hardware)

• **Processor:**
  - A device that reads and executes instructions in sequence from a memory device, producing results that are written back to a memory device.

• **Node:** One memory device connected to one or more processors.
  - Multiple processors in a node are said to share-memory and this is “shared memory parallelism”
  - They can work together because they can see each other’s memory
  - The latency and bandwidth to memory affect performance
Appendix – Parallel Computing Terms (Hardware)

- **Cluster**: Multiple nodes connected by a network
  - The processors attached to the memory in one node cannot see the memory for processors on another node
  - For processors on different nodes to work together they must send messages between the nodes. This is “distributed memory parallelism”

- **Network**: 
  - Devices and wires for sending messages between nodes
  - Bandwidth – a measure of the number of bytes that can be moved in a second
  - Latency – the amount of time it takes before the first byte of a message arrives at its destination
Appendix – Parallel Computing Terms (Software)

- **Process:**
  - A set of instructions to be executed on a processor
  - Enough state information to allow process execution to stop on a processor and be picked up again later, possibly by another processor

- Processes may be lightweight or heavyweight
  - **Lightweight processes**, e.g. shared-memory threads, store very little state; just enough to stop and then start the process
  - **Heavyweight processes**, e.g. UNIX processes, store a lot more (basically the memory image of the job)
Appendix – Parallel Computing Terms (Software)

- Every job has at least one heavy-weight *process*.
  - A job with more than one heavy-weight process is a distributed-memory parallel job
  - Even on the same node, heavyweight processes do not share memory
- Within a heavyweight process you may have some number of lightweight processes, called *threads*.
  - Threads are shared-memory parallel; only threads in the same memory space can work together.
  - A thread never exists by itself; it is always inside a heavy-weight process.
- Heavy-weight processes are the vehicles for distributed memory parallelism
- Threads (light-weight processes) are the vehicles for shared-memory parallelism
Appendix – Parallel Computing in WRFDA context

• Since the process model has two levels (heavy-weight and light-weight = MPI and OpenMP), the decomposition of the application over processes has two levels:
  – The **domain** is first broken up into rectangular pieces that are assigned to heavy-weight processes. These pieces are called **patches**
  – The **patches** may be further subdivided into smaller rectangular pieces that are called **tiles**, and these are assigned to **threads** within the process.
Appendix – Parallel Computing APIs

• Message Passing Interface – MPI, referred to as the communication layer

• MPI is used to start up and pass messages between multiple heavyweight processes
  – The **mpirun** command controls the number of processes and how they are mapped onto nodes of the parallel machine
  – Calls to MPI routines send and receive messages and control other interactions between processes
Appendix – Parallel Computing APIs

- OpenMP is used to start up and control threads within each process
  - Directives specify which parts of the program are multi-threaded
  - OpenMP environment variables determine the number of threads in each process
  - [http://www.openmp.org](http://www.openmp.org)
- OpenMP is usually activated via a compiler option
- MPI is usually activated via the compiler name
- The number of processes (number of MPI processes times the number of threads in each process) usually corresponds to the number of processors
- In general, WRFDA should not be run with shared memory!
Distributed Memory (MPI) Communications

- Halo updates
- Parallel transposes
Distributed Memory (MPI) Communications

• Halo updates
• Parallel transposes

all y on patch

all z on patch

all x on patch
Review – Computing Overview

APPLICATION (WRF)

SYSTEM (UNIX, MPI, OpenMP)

HARDWARE (Processors, Memories, Wires)

- Domain contains Patches contain Tiles
- Job contains Processes contain Threads
- Cluster contains Nodes contain Processors

Distributed Memory Parallel

Shared Memory Parallel
## Appendix – WRFDA/var/da

### Directory structure

<table>
<thead>
<tr>
<th>Main WRFDA Program (driver):</th>
<th>da_main</th>
</tr>
</thead>
</table>

### WRFDA Subroutines (mediation layer)

- da\_4dvar
- da\_control
- da\_etkf
- da\_define\_structures
- da\_dynamics
- da\_grid\_definitions
- da\_interpolation
- da\_minimisation
- da\_physics
- da\_setup\_structures
- da\_varbc
- da\_vtox\_transforms

### OBSERVATION TYPES

- da\_airep
- da\_airsr
- da\_bogus
- da\_buoy
- da\_geoamv
- da\_gpspw
- da\_gpsref
- da\_metar
- da\_mtgirs
- da\_pilot
- da\_polaramv
- da\_profiler
- da\_pseudo
- da\_qscat
- da\_radar
- da\_radiance
- da\_rain
- da\_satem
- da\_ships
- da\_sound
- da\_ssmi
- da\_synop
- da\_tamdar
Appendix – WRFDA History

• Developed from MM5 3DVar beginning around 2002, first version (2.0) released December 2003
• 4DVAR capability added in 2008, made practical with parallelism starting with Version 3.4 (April 2012)
• Developed and supported by WRFDA group of MMM, part of NESL
• Requirements emphasize flexibility over a range of platforms, applications, users, performance
• Current release WRFDA v3.6 (April 2014)
• Shares the WRF Software Framework
Appendix – WRFDA and J

\[ J(x) = \frac{1}{2} (x - x^b)^T B_0^{-1} (x - x^b) + \frac{1}{2} (y_0 - H(x))^T R^{-1} (y_0 - H(x)) \]

- Model background \((x^b)\)
- Background error \((B_0)\)
- Observations \((y_0)\) and their associated error statistics \((R)\)
- Minimize this cost function \((J(x))\) to find the analysis \((x^a)\)
- Run forecast, repeat for cycling mode
Appendix – WRFDA and J

\[ J(x) = \frac{1}{2} (x - x^b)^T B_0^{-1} (x - x^b) + \frac{1}{2} (y_0 - H(x))^T R^{-1} (y_0 - H(x)) \]

- Model background \((x^b)\) is the “first guess” of the atmospheric state before data assimilation
- The background can be provided by WPS and real.exe, or a full WRF forecast
Appendix – WRFDA and J

\[ J(x) = \frac{1}{2} (x - x^b)^T B_0^{-1} (x - x^b) + \frac{1}{2} (y_0 - H(x))^T R^{-1} (y_0 - H(x)) \]

- Background error covariance \((B_0)\) describes the relationship between different errors in the background
- Arguably the most important ingredient to a successful forecast
- Several options available:
  - \(cv\_options=3;\) generic NCEP Background Error model
    - This is recommended for testing and debugging **only**
  - \(cv\_options=5;\) the NCAR Background Error model
    - The default and recommended BE formulation
    - Requires \(gen\_be\) (learned in later presentation)
  - \(cv\_options=6;\) Multivariate Background Error (MBE) statistics
    - Not officially supported
Appendix – WRFDA and J

\[
J(x) = \frac{1}{2} (x - x^b)^T B_0^{-1} (x - x^b) + \frac{1}{2} (y_0 - H(x))^T R^{-1} (y_0 - H(x))
\]

- Observations (\(y_0\)) and their associated errors (\(R\)) are essential to any data assimilation process
- WRFDA can assimilate a wide variety of observations
  - Conventional observations
    - Includes radiosonde, ships, surface, etc.
    - Should be in LITTLE_R format for ingest into OBSPROC
  - Satellite radiance data
    - Can assimilate data from dozens of instruments
    - Assimilated directly in BUFR format
    - Requires radiative transfer model (CRTM or RTTOV)
- Radar velocity and reflectivity, accumulated precipitation, others