WRF Workflow

This document is written with the -v setting ;-). The intention is to provide a step-by-step guide for getting a new WRF simulation running for someone starting from limited experience in both linux world and WRF. Skip steps and adapt as needed, especially if you are not on the OU OSCER system.

Some helpful info and links:

- WRF main webpage
- Tutorial materials: Doc/Pub drop-down menu → Tutorial Presentations
- Official WRF new users workshop, as stated on that tutorial page: "*The next WRF Tutorial will take place January 30 - February 3, 2023, and will be held virtually.*" I think these are free, so worth considering.
- User's Guide
- Technical Description (i.e., more of a formal publication describing how equations and physics are implemented)
- I highly recommend using a virtual SSH framework with VSCode (see #7 in our doc here) to access OSCER (or whatever system you're on), which is a nice terminal, visual code editor, and graphics viewer all rolled into one

Steps:

1. **Gain access to a supercomputer environment**
   - Whether OSCER, Cheyenne, etc.
   - Request a new account on OSCER here: https://www.ou.edu/oscer/support/accounts/new_account
   - Once approved, be sure you can log onto the system via SSH
   - I suggest checking out this info on setting up some shortcuts using your ".bashrc" or ".bash_profile" file

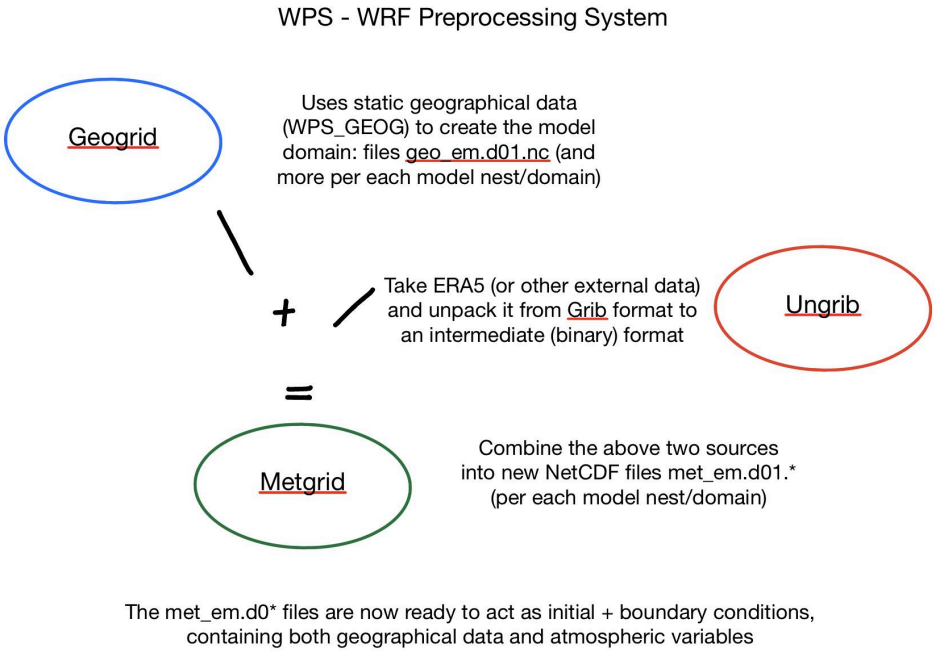
2. **Watch 25-min intro to the WRF model**
   - https://youtu.be/wzSu-343b-0
   - The PDF slideshow for this video is here
   - In that presentation, you'll find an incredibly overwhelming flow chart with tons of components; *we primarily only use three components*:
     - **WPS/** (tutorial slides): pre-processes the model/reanalysis data that we'll use as initial and boundary conditions
     - Within **WRF/**:
       - **Real** (tutorial slides): vertically interpolates the output from the above step onto the WRF vertical grid
       - **WRF-ARW** (tutorial slides): runs our simulation

3. **Download WRF and WPS as described here**
   - Log into the downloads page
   - From your home directory, create a new subdirectory (e.g., "wrf-stuff") where you will keep all WRF code and run simulations
   - From that new directory, use git to get both packages:
     - **git clone --recurse-submodules https://github.com/wrf-model/WRF**
     - **git clone https://github.com/wrf-model/WPS**
   - Change WRF from the default version (latest - v4.5) to the latest of v4.3 (all versions and their tags can be viewed using **git tag**):
     - Go down into the new directory **WRF/**, issue **git checkout v4.3.3**
     - And for **../WPS/:** run **git checkout v4.3.1**

4. **Create source file bashrc_wrf**
   - According to the examples **near the end** of this document
   - This file loads all necessary modules and sets environmental variables to paths based on what's available on the system you're using, including possibly linking to files you've installed, e.g., via Conda.
   - This file is therefore very much system-dependent and needs to be tailored for what's available.
   - The examples below are for OSCER and Derecho.
   - I suggest making this text file in your main ("wrf-stuff") directory, where it can be readily sourced for compiling & running both WRF and WPS
   - Use this file via "**source bashrc_wrf**" for every new terminal window instance whenever you need to do any compiling or running of simulations.

5. **Compile WRF**
   - From **WRF/**, set the current terminal environment by issuing **source ../bashrc_wrf**

- Next, **./configure** to set paths/shortcuts that the compiler will read
    - Assuming you are using a similar module setup to the OSCER version below (with Intel compilers): select the "intel" compiler option run in "dmpar" (distributed memory, parallel), which should be **option #15** (
        - **NOTE:** Option #15 assumes that you're using the Intel compiler; if you're using GNU or something else, choose accordingly (e.g., 34 for GNU distributed memory)
    - Select **option 1** for non-moving nest
    - Assuming the environment is properly set up, this should create **configure.wrf**, which sets paths/shortcuts/options for the WRF compiler
    - **For Derecho:**
        - **option #50, as recommended by NCAR ppl on WRF forum**
            - **This option works but is yielding an MPICH error with the very large domain**
        - **~~option #24?~~**
        - **Testing:**
            - **X 24, as recommended somewhere in the WRF forum**
            - **X 78, per this page describing new compiler flag option for ifx/icx**
                - **Failed, because didn't have the right modules loaded**
                - **Need to upload intel-oneapi/2023.0.0**
- Compile WRF: run "**./compile -j 2 em_real > compile.out 2>&1 &**" and pray (note: can take ~20 minutes and will run in the background)
    - **-j 2**: distributes compile process over 6 nodes to accelerate it
    - **em_real**: the ARW core (as opposed to NMM)
    - **compile.out** will be an extremely long text file that fills up with the compilation output, including many warnings that can be ignored ("Error" and "ERROR" are signs of problems)
    - The ending "**&**" returns your command line while the job runs in the background
- If WRF successfully compiled, **main/** will now have four new executables: **ndown.exe, real.exe, tc.exe, and wrf.exe**
- **If compiling fails or you must recompile for any reason:**
    - **./clean -a** to delete all binary files and executables
        - This creates a fresh start for compiling…you will likely have problems if you don't do this
        - **-a** cleans all compiled files
    - Rerun **./configure** and compile from there



6. **Compile WPS**
   - With WRF successfully compiled, WPS now can be compiled
   - Next, issue **source ../bashrc_wrf** assuming you're starting fresh.
   - **./configure** to create the file "configure.wps" which is sets compiler options when you run **compile**
   - [for OSCER] Select **option 17** for intel in serial
     - This again assumes you're using intel compilers
     - Serial is fine: WRF preprocessing is a small enough task that we don't need to run things in parallel
   - [for Derecho] Select **option 37** for the Cray XC CLE/Linux Intel compiler
   - **./compile > compile.out 2>&1 &** (takes up to ~10 minutes)
   - **Success** if there are now 3 executables in the WPS directory:
     - geogrid.exe
     - metgrid.exe
     - ungrib.exe
   - **If compiling fails or need to recompile for any reason:**
     - **./clean -a** will delete all binary files and executables
       - **-a** cleans all files (you always want this)
     - Rerun **./configure** and continue from there

7. **Use WPS to determine your model domain**
   - As in previous steps, be sure to rerun **source ./bashrc_wrf** if you're starting a new terminal window/session here
   - It helps to do this step first to decide the geographical bounds for your forcing data (which we download in the next step)
   - **namelist.wps** sets all of the grid parameters: the date/time range, geographical bounds, model grid specifications, etc.

- I'd suggest checking out the userguide's description of namelist variables to familiarize yourself with this (note: there are far more variables than need to be explicitly set). I'm showing an example of one of my namelists below.
    - Description of the Variables
- WPS_GEOG: this is the static geographical data (e.g., global topography, land/sea info, etc.) that geogrid.exe uses to set the model environment and terrain:
    - You will either need to download the dataset, available from here (highest resolution, mandatory),
      OR
      If on OSCER, since I've already downloaded this data, **vi** into **namelist.wps** and set **geog_data_path** to:
      '/ourdisk/hpc/radclouds/auto_archive_notyet/tape_2copies/WPS_GEOG/'
- Run **geogrid.exe**, which will produce geo_em.d01.nc (and others for each nested domain)
    - Type **./geogrid.exe** to run an executable.
- Check out the domain (and iterate):
    - Copy util/plotgrids_new.ncl to current directory (where namelist.wps lives)
        - **cp util/plotgrids_new.ncl .**
            - The period at the end of the command represents current directory
    - Try **module load NCL** to access available installed NCL packages
        - Note: when you load NCL, it will change many other packages loaded by bashrc_wrf, so re-source the latter once you're done in this step
    - Use plotgrids_new.ncl to create a plot of the model domain:
        - As default, running **ncl plotgrids_new.ncl** will produce an x-window showing the model grid(s)
            - If you need to install it, you can add x-window support using XQuartz
        - You can also go into script and add a line to create a PDF or PNG ("wps_show_dom.png")
            - Comment out (add ; in front of it): **type = "x11"**
            - Insert a new line: **type = "png"**
            - Now when you run **ncl plotgrids_new.ncl**, you will get a .png file named "wps_show_dom.png"
            - Highly recommended: use a virtual SSH framework with VSCode (see #7 in our doc here) to access and modify code, where you can also directly visualize the figure
- Once you're happy with your model domain viewed in the figure produced by NCL, you're ready to grab the domain bounds to inform domain for grabbing ERA5 data
- At this point, re-run geogrid.exe to produce the geo_em files on your tuned domains
- To print the domain bounds (for the largest domain), use **ncdump -h geo_em.d01.nc** and check the *corner_lats* and *corner_lons* variables:
    - There are many slightly different "versions" of these corners; this is because there are actually different grids used for the u-wind, v-wind, mass variables, etc.
- Use a buffer of at least 5º lat and lon beyond the corner variables, and this will guide how you specify your domain in the ERA5 download script

- I.e., use a Lon0 that is less than the minimum longitude in corner_lon, and so forth

8. **Getting ERA5 boundary condition data**
    - While these processes will differ if using other model forcing data, some of the info in here should still apply
    - This step will yield new code and forcing data, so I suggest making a new subdirectory where this code and forcing data will live
    - We will use the python API to download ERA5 from the CDS climate data store
    - There could be multiple ways to install and load this API, but I recommend using Conda (see this doc on our drive for background on Conda)
        - If you have your own local installation of Conda, you can likely install it directly to your base environment (i.e., run **conda deactivate** or **conda activate base**)
        - If you are using conda as an available module, you will likely need to create your own new environment first before you can install new packages (see our conda cheat sheet)
        - Then run: **conda install -c conda-forge cdsapi**
    - We will also use some nice bash & API scripts borrowed from this blogger
    - Go to the CDS store and register for an account, then go to your user profile
        - ***Make sure you also accept terms and conditions (need to check a box), or otherwise the download won't work. (link to T&C)***
        - ***If the above link doesn't work,*** just wait until the last step of this section where you run the shell script. An error message will pop up and at the end of the error message will be a link to the T&C.
    - Once you are logged in, go to and follow these instructions to install the CDS API key using your unique code (the file $HOME/.cdsapirc is read when you download data)
        - To create a hidden file called **.cdsapirc** (the dot in front of it makes it hidden), simply type **vi .cdsapirc** to create the file while in your personal home directory. Then copy and paste the 'url' and 'key' info from that link ^^^ into this file.
    - To get the data, create and use the three download scripts listed below, which have been slightly adapted from the blog post linked above
        - The first two scripts (*GetERA5-sl.py* and *GetERA5-pl.py*):
            - These contain a list of variables and levels that WPS will look for ("pl" for pressure level and "sl" for single level, i.e., surface data)
            - These scripts should not need to be directly modified; the run script (below) will use their contents, modified via the "sed" commands, to create new scripts that will do the actual downloading
        - The third script is the run script (*run_getera5.sh*)
            - Use this script to set time/date and geographical bounds of the ERA5 data you want to download, i.e., the variables ***DATE1, DATE2, Nort, West, Sout, East***
                - ERA5 will read in data from North -> South, and West -> East. Examples:

World region	North (degrees latitude)	West (degrees longitude)	South (degrees latitude)	East (degrees longitude)	Retrieve bounds in degrees latitude / longitude [North, West, South, East]
Australia, New Zealand and South East Asia	40	60	-60	180	`[40, 60, -60, 180]`
Africa	40	-20	-40	60	`[40, -20, -40, 60]`
North and South America	50	-135	-60	-35	`[50, -135, -60, 35]`
Europe	75	-15	30	42.5	`[75, -15, 30, 42.5]`

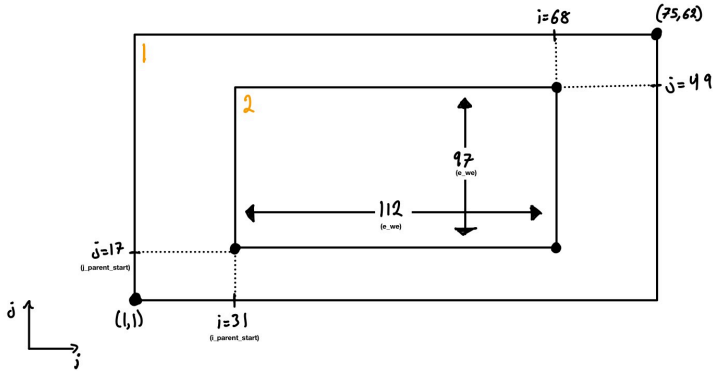
        - Note that if you want 20°S or 20°W, you must put a negative in front, meaning -20==20°S or 20°W. The Africa example helped me understand.
    - Be sure that these variables in *run_getera5.sh* are set such that WPS will find the data it's looking for, based on the settings of *namelist.wps* (e.g., ***DATE1*** here must be ≥ ***start_date*** in namelist.wps, and so forth)
    - Change *run_getera5.sh* to be executable by running: **chmod a+x run_getera5.sh**
    - Then, run it by issuing the command **./run_getera5.sh**
    - The "sed" commands in this script issue *search/replace* commands on the GetERA5-* files then write out the result of those to new script copies, which are named using your specific dates
    - Lastly, it feeds those new scripts to python
- Assuming the CDSAPI has been set up correctly, and python is able to see that, then when the run script is called it should produce write-out indicating that it is accessing the Climate Data Store, with the order first *Queued* and then *Downloading*,
- If you now see new ***.grib*** files from this process, you have the golden ticket to move onto running WPS to prepare your model boundary and initial conditions

9. Use WPS to prepare model initial and boundary conditions

- All of the below commands should be executed from within WPS/
- **Update your namelist.wps**: make sure that your date range is consistent with (or at least doesn't exceed the bounds of) the date range of the downloaded data (ERA5 or otherwise)
    - Check the date bounds for each domain you're running
    - Example of geogrid bounds when translated into a gridbox

```
&share
 wrf_core = 'ARW',
 max_dom = 2,
 start_date = '2008-03-24_12:00:00','2008-03-24_12:00:00',
 end_date   = '2008-03-24_18:00:00','2008-03-24_12:00:00',
 interval_seconds = 21600,
 io_form_geogrid = 2
/

&geogrid
 parent_id         =   1,   1,
 parent_grid_ratio =   1,   3,
 i_parent_start    =   1,  31,
 j_parent_start    =   1,  17,
 e_we              =  74, 112,
 e_sn              =  61,  97,
 geog_data_res     = 'default','default',
 dx = 30000,
 dy = 30000,
 map_proj = 'lambert',
 ref_lat   = 34.83,
 ref_lon   = -81.03,
 truelat1  =  30.0,
 truelat2  =  60.0,
 stand_lon = -98.
 geog_data_path = '/mmm/users/wrfhelp/WPS_GEOG/'
/
```



- Also check *interval_seconds*, which is the interval it expects between data files for the external ERA5 dataset
- Fantastic resource to further help understand namelist.wps

- ungrib.exe needs a guide (table) to read the ERA5 data; we will set this up as a shortcut from the table to the current (WPS/) directory using
  ln -sf ungrib/Variable_Tables/Vtable.ERA-interim.pl ./Vtable
  - "ln -sf" creates a "symbolic link" of **Vtable.ERA…** as the target **./Vtable**
- Run **./link_grib.csh path/ERA5*** where you *replace "path"* with the path to the data
  - The asterisk* is a wildcard that references all files starting with ERA5
- Run **./ungrib.exe**, which puts the ERA5 data into an intermediate file format
- To run metgrid, we'll need another table to tell Metgrid which WRF version we're using:
  - **ln -sf metgrid/METGRID.TBL.ARW ./METGRID.TBL**
- Run **./metgrid.exe**, which puts the intermediate data into NetCDF format on the model grid, i.e., merging it with the output from Geogrid; there will now be individual *met_em** files for each forcing-data time step * n-domains
  - If you get an error message when trying to run metgrid.exe that looks like `WARNING: Field PRES has missing values`, this is related to the domain of the ERA5 data not being far enough beyond the bounds metgrid is looking for.
  - If you get an error `./metgrid.exe: error while loading shared libraries:...`, make sure you **source ../bashrc_wrf** since you deactivated it earlier
  - If this step succeeded, you can now run **ncview** on your new met*.nc files to see how your data looks, now on your model

10. A pause on namelists: save them!

- You now have a version of **namelist.wps** that you're happy with and are about to start developing a new namelist (**namelist.input**) to be used for running your simulations (that one is a much longer text file)
- ***Save copies:*** To preserve these precious (albeit small) files in case of a loss for whatever reason, make a directory called e.g. "namelists" in your top-level "wrf-stuff" directory and keep up-to-date versions of each of your namelists in there.
- You can/should include the different namelists you'll use for different experiments/simulations, and you can append helpful tags at the end to indicate what they correspond to

- You can/should also include any other text files you'd like to retain, e.g., your submission scripts (the **.job* files), and maybe even your *configure* files
- Case study #1: note that when you recompile WRF, at the end of the job the compiler moves any existing *namelist.input* within your /run directory to a new name with a date stamp on it and creates a new default namelist.input in its place. This is a great way to lose your namelist, and I'm speaking from experience here. ;-)

11. Running WRF: You are now ready…congratulations!! 🎆

- **The WRF/run directory**
  - Contains a ton of stuff, *some* of which will be used by REAL and WRF
  - Should also contain symbolic links to all the executables
  - This is therefore where we run the next steps, but…
  - Since the model output is likely going to far exceed your available home-directory storage space, in practice it is best to link the WRF/run directory (+ all contents) somewhere else, such as a workspace of your own created on our shared OURDisk drive:
    - /ourdisk/hpc/radclouds/auto_archive_notyet/tape_2copies
  - I would suggest using symbolic links to the entire run directory and its full contents. From the directory where you plan run your simulation, execute the command
    - ***"ln -sf {wrfdir}/run/* ."***
  - where ***${wrfdir}*** represents where you have compiled WRF and the "." is the pointer to the current directory
  - The "run" directory referenced below will now refer to this new location
- ***met_em* files***
  - These files generated by WPS need to be accessible to ***real.exe*** in your run directory
  - You can therefore either move these files to the directory where you'll run the simulations or place them somewhere for long-term storage and create symbolic links to them from your run directory:
    - I.e., while in your **run** directory, type ***ln -sf PATH/met_em* .***
    - (note the dot at the end, which refers to ***current directory***)
    - where -***sf*** tell it "symbolic" and "force," i.e., it will remove preexisting files in the destination directory
- ****.exe files***
  - If your executable files (ndown.exe, real.exe, tc.exe, wrf.exe) aren't linked, apply the same step as above…
    - While in your run directory, type ***ln -sf PATH/*.exe .***
- ***namelist.input*** provides all the key settings for both REAL and WRF, which prepare the initial/boundary conditions and then run the actual model
  - I placed an example namelist from one of my large TC simulations (Maria) on our shared drive
  - There's a lot to learn and tweak in this namelist, so I suggest keeping the user guide open for this stage and doing some studying up as you start
  - For starters, all of the spatial domain settings must match exactly between namelist.wps from the WPS steps and this namelist, since REAL will look for met_em* files with those settings in the current directory
- If you need special output variables (**the radiative tendencies are not included by default!**), also do the following:

    - Add the following line under &time_control:
        - ***iofields_filename = "var_extra_output", "var_extra_output", "var_extra_output"***
            - ^ need to specify this per domain
        - ***ignore_iofields_warning = .true.,***
            - ^ suppresses nuisance messaging
    - The following line of text is what I have in my file var_extra_output, which is simply a single line of this text:
        - **+:h:0:RTHRATSW,RTHRATLW,RTHRATSWC,RTHRATLWC,OLR,GLW,GSW,refl_10cm,RTHBLTEN,RTHCUTEN,H_DIABATIC,RTHFTEN**
        - See text on "Run-Time I/O Option" in the WRF user's guide for more info about syntax here
    - Have that file sitting in your /run directory where ***wrf.exe*** is called.
- Once you are happy with your namelist, onto running the model…
- **Batch jobs:** you will conduct the next steps using batch scripts, i.e., passing the work onto compute nodes, whether our own group's nodes (the *-p radclouds*) or the *normal* partition (*-p normal*) that is available to all (see OSCER guide here)
- Some info about these batch scripts and how SLURM handles them:
    - Example batch scripts to run both REAL and WRF are provided below; you can create and keep these scripts in your directory "nameslists" to avoid any accidental losses but will run these scripts from the directory where you're actually running the simulation jobs
    - When you issue the commands ***sbatch batch_real.job*** and later ***sbatch batch_wrf.job***, the SLURM workload management system passes the **.job** script to the compute nodes, and actually issues the command once you reach the top of the queue, if there is one, subject to availability.
        - SBATCH first reads all of the "#SBATCH" commands, which set job parameters; see here for more info on these SBATCH variables
        - It then executes the commands below those lines
        - NOTE: that **bashrc_wrf** is always sourced first in these scripts, so make sure to create a copy of that file into the **run** directory. This also means these batch* files should be located in the **run** directory as well.
        - Notice also that REAL is run on one node; it is a relatively small and quick job; WRF, however, is distributed over multiple nodes
            - 3 nodes may ultimately not be enough for your main WRF job, so it is ideal to start with a small model domain and short run time for testing purposes
    - Once submitted, you can check the status of your job via ***sacct***
        - If you want sacct to provide more information, run the below line to add important info like start time, end time, and elapsed time, and remove columns that aren't useful. See this link for more sacct info
        - **export SACCT_FORMAT=JobID,JobName,Partition,AllocCPUS,Start,End,Elapsed,State,ExitCode**
    - Once the job is actually running (from status PENDING → RUNNING or COMPLETED), text output should appear in the outbatch* scripts (output is being directed to these as per the #SBATCH settings)

- Run REAL with the command ***sbatch batch_real.job***, which will execute ***./real.exe***: this step is the first stage of running WRF; it only needs to be run once to generate the below list of boundary & initial condition files in your current /run directory based on all the *met_em** files:
    - *wrfinput_d** – model initial conditions that allow the model integration to begin (i.e., a single time step)
    - *wrflowinput_d** – lower boundary condition file if you are using it (e.g., SST; command+F for "sst_update" in the user guide) (also single time step)
    - *wrfbdy_d** – boundary conditions that will update the model lateral edges every *interval_seconds* (this should match namelist.wps) during the integration (multiple time steps as per the time settings in the namelist)
    - **NOTE**: To avoid any errors, you should always rerun batch_real.job if you end up changing your start time, even if it is within the bounds of your original run. (i.e., I successfully run a cntl run between 10/1 - 10/14, and would like to run sensitivity experiments from a restart file at 10/5 - 10/6. If you don't rerun batch_real.job, you will be met with errors.)
- Run WRF with the command ***sbatch batch_wrf.job***, which will execute ***./wrf.exe***: the main step of running WRF 🙂
    - If real.exe has completed successfully, this step will run the actual model integration
    - Now that you have your initial and boundary condition files out to x-hours or days that you ultimately plan to run, you can roll back the simulation end time in the namelist just for testing purposes, i.e., to see that the model will initialize properly and produce the initial time step (or two)
    - When on a shared public queue (e.g., *-p normal* on OSCER), you can submit a job with a very short job time (-t) and play around with a very short model integration time just for testing phase
    - Once you know things are working as expected, you are ready to increase the total model integration time, increase the -t setting, and let it rip
    - Note that when WRF fails for some reason, you likely do not need to rerun real.exe if it ran successfully before; you can rerun WRF as many times as you please, provided that the initial & boundary condition files are in the same /run directory

12. Post processing of WRF output

- WRF produces output as individual single-time-step files with aaalll variables in each single file
- It is often more handy to have the output processed as follows:
    - #1 to have variables on either an isobaric or basic height-level grid instead of the model vertical coordinate
    - #2 to have all time steps combined into a single file, with 1 file per variable

#1 Can be done in wrf-python using vertcross method

#2 Making one .nc file containing all wrf output files:

1. Move your wrf output files from your WRF/run/ directory into a separate folder where you will contain all wrf output files.

2. Use an NCO command that stitches .nc files together by using the **ncrcat** function:
    a. On the command line, run: **ncrcat wrfout_d01* stitchd01_wrfout &**
    b. As it runs, you can check the size of the file and watch it grow by running (wait time: minutes): **ls -lrth**
3. This will take all the files starting with **wrfout_d01** and stitch them together to output it as one .nc file named **stitchd01_wrfout**
   I like to be more specific with my output file name i.e.
   d01_2015-11-24_12:00:00--26_06:00:00
4. After creating this one .nc file, remove all the individual wrf output files to avoid duplicates in order to use storage sparingly.
5. You can take it a step further and if you know you will be looking at a couple variables within the stitched wrfout file, run the command:
   cdo -selname,[VARIABLE] [FILENAME] [OUTFILE]
   To get a .nc file with just that variable.
   Insert the variable name ([VARIABLE]), the stitched wrfout file [FILENAME], and what you'd like the .nc file with one variable to be named [OUTFILE].

- The python package wrf-python is super handy, with a lot of built-in functionality for handling WRF model output, and with Xarray functionality
- The stand-alone (i.e., non-python) packages CDO and NCO are other tools that people commonly use for easy command-line netCDF file manipulation (e.g., can combine time steps, run time-averaging, print statistics on a given variable, etc.). Both tools are usually available via ***module load*** on supercomputers
- You'll want to think ahead about a sensible file organization approach to keep track of different simulation output, where you'll have repeating outputs that you want to keep safe and separate.
- Understanding where each variable is measured is important because not all variables are measured in the same place. This is a good source that explains the differences

Bashrc_wrf for OSCER

File to be sourced anytime WRF or WPS will be compiled or run

```
#!/bin/bash
#
# **ENVIRONMENTAL PACKAGES SHOULD REMAIN STATIC FOR A GIVEN MODEL EXPERIMENT**
#
# This shell script loads the necessary environmental packages and sets
# environmental variables/paths that WRF looks for during compile and
# when running.
#
# Setup using modules on OSCER.
#

module purge
module load netCDF-Fortran/4.5.3-iimpi-2021a
module load JasPer/2.0.33-GCCcore-11.3.0
module load libpng/1.6.37-GCCcore-11.3.0

export I_MPI_CC=icc
export I_MPI_CXX=icpc
export I_MPI_F90=ifort
export I_MPI_F77=ifort
export I_MPI_FC=ifort

# Necessary if remotely working from a Mac
export LC_ALL="en_US.UTF-8"

#FOR WRF

#WRF
export WRF_EM_CORE=1
export WRFIO_NCD_LARGE_FILE_SUPPORT=1

#NETCDF
export LCLSRC=$HOME/localsrc
export NETCDF=$LCLSRC/netcdf
export NETCDF_INC=$NETCDF/include
export NETCDF_LIB=$NETCDF/lib
export NETCDF_BIN=$NETCDF/bin
# Copy over NETCDF files to common directory since netcdf C and FORTRAN modules are separate
rm -rf $LCLSRC
mkdir -p $LCLSRC
mkdir -p $NETCDF
mkdir $NETCDF/include
mkdir $NETCDF/lib
mkdir $NETCDF/bin
cp -raf $EBROOTNETCDF/include/*  $NETCDF/include
```

```
cp -raf $EBROOTNETCDF/lib64/* $NETCDF/lib
cp -raf $EBROOTNETCDF/bin/* $NETCDF/bin
cp -raf $EBROOTNETCDFMINFORTRAN/include/* $NETCDF/include
cp -raf $EBROOTNETCDFMINFORTRAN/lib/* $NETCDF/lib
cp -raf $EBROOTNETCDFMINFORTRAN/bin/* $NETCDF/bin

#HDF5
export HDF5=$EBROOTHDF5

# FOR WPS

#jasper
export JASPER=$EBROOTJASPER
export JASPERLIB=$JASPER/lib
export JASPERINC=$JASPER/include

#libpng
export LIBPNG=$EBROOTLIBPNG
export LIBPNGLIB=$LIBPNG/lib
export LIBPNGINC=$LIBPNG/include
```

Bashrc_wrf for Derecho

File to be sourced anytime WRF or WPS will be compiled or run

```
#!/bin/bash
#
# **ENVIRONMENTAL PACKAGES SHOULD REMAIN STATIC FOR A GIVEN MODEL EXPERIMENT**
#
# This shell script loads the necessary environmental packages and sets
# environmental variables/paths that WRF looks for during compile and
# when running.
#
# Setup using modules on DERECHO.
#

module purge
module load ncarenv/23.09
module load intel-classic/2023.2.1
module load ncarcompilers/1.0.0
module load cray-mpich/8.1.27
module load craype/2.7.23
module load netcdf-mpi/4.9.2

# Necessary if remotely working from a Mac
export LC_ALL="en_US.UTF-8"

#FOR WRF

#WRF
export WRF_EM_CORE=1
export WRFIO_NCD_LARGE_FILE_SUPPORT=1

# FOR WPS

export LIBINC=/glade/u/apps/derecho/23.09/opt/view/lib64
export JASPERLIB=$LIBINC
export JASPERINC=$LIBINC
export LIBPNGLIB=$LIBINC
export LIBPNGINC=$LIBINC
```

namelist.wps (just as an example)

namelist.wps

```
&share
 wrf_core = 'ARW',
 max_dom = 2,
 start_date = '2017-09-14_00:00:00','2017-09-14_00:00:00',
 end_date   = '2017-09-14_12:00:00','2017-09-14_12:00:00',
 interval_seconds = 3600
 io_form_geogrid = 2,
/

&geogrid
 parent_id         =   1,   1, 2,
 parent_grid_ratio =   1,   5, 3,
 i_parent_start    =   1,  80, 82,
 j_parent_start    =   1,  46, 75,
 e_we              =  390, 1201, 1249,
 e_sn              =  240, 741, 751,
 geog_data_res = '30s','30s','30s',
 dx = 15000,
 dy = 15000,
 map_proj = 'mercator',
 ref_lat   = 13.0,
 ref_lon   = -45.0,
 truelat1  = 30.0,
! truelat2  = 60.0,
! stand_lon = -65.0,
 geog_data_path = '/ourdisk/hpc/radclouds/auto_archive_notyet/tape_2copies/WPS_GEOG/'
 opt_geogrid_tbl_path = '/home/jamesrup/ensemble-tropical-cyclone-cases/WPS/geogrid/'
/

&ungrib
 out_format = 'WPS',
 prefix = 'FILE',
/

&metgrid
 fg_name = 'FILE'
 io_form_metgrid = 2,
 opt_metgrid_tbl_path = '/home/jamesrup/ensemble-tropical-cyclone-cases/WPS/',
/
```

Scripts for downloading ERA5 reanalysis with all variables necessary to initialize WRF

GetERA5-sl.py

```
import cdsapi

c = cdsapi.Client()

c.retrieve(
    'reanalysis-era5-single-levels',
    {
        'product_type':'reanalysis',
        'format':'grib',
        'variable':[
            '10m_u_component_of_wind','10m_v_component_of_wind','2m_dewpoint_temperature',
            '2m_temperature','land_sea_mask','mean_sea_level_pressure',
            'sea_ice_cover','sea_surface_temperature','skin_temperature',
            'snow_depth','soil_temperature_level_1','soil_temperature_level_2',
            'soil_temperature_level_3','soil_temperature_level_4','surface_pressure',
            'volumetric_soil_water_layer_1','volumetric_soil_water_layer_2','volumetric_soil_water_layer_3',
            'volumetric_soil_water_layer_4'
        ],
        'date':'DATE1/DATE2',
        'area':'Nort/West/Sout/East',
        'time':'00/to/23/by/1',
    },
    'ERA5-DATE1-DATE2-sl.grib')
```

GetERA5-pl.py

```
import cdsapi

c = cdsapi.Client()

c.retrieve(
    'reanalysis-era5-pressure-levels',
    {
        'product_type':'reanalysis',
        'format':'grib',
        'pressure_level':[
            '1','2','3',
            '5','7','10',
            '20','30','50',
            '70','100','125',
            '150','175','200',
            '225','250','300',
```

```
            '350','400','450',
            '500','550','600',
            '650','700','750',
            '775','800','825',
            '850','875','900',
            '925','950','975',
            '1000'
        ],
        'date':'DATE1/DATE2',
        'area':'Nort/West/Sout/East',
        'time':'00/to/23/by/1',
        'variable':[
            'geopotential','relative_humidity','specific_humidity',
            'temperature','u_component_of_wind','v_component_of_wind'
        ]
    },
    'ERA5-DATE1-DATE2-pl.grib')
```

run_getera5.sh

```
#!/bin/bash -l

DATE1=20170414
DATE2=20170415
Nort=60
West=80
Sout=15
East=150

YY1=`echo $DATE1 | cut -c1-4`
MM1=`echo $DATE1 | cut -c5-6`
DD1=`echo $DATE1 | cut -c7-8`
YY2=`echo $DATE2 | cut -c1-4`
MM2=`echo $DATE2 | cut -c5-6`
DD2=`echo $DATE2 | cut -c7-8`

sed -e "s/DATE1/${DATE1}/g;s/DATE2/${DATE2}/g;s/Nort/${Nort}/g;s/West/${West}/g;s/Sout/${Sout}/g;s/East/${East}/g;" GetERA5-sl.py > GetERA5-${DATE1}-${DATE2}-sl.py

python GetERA5-${DATE1}-${DATE2}-sl.py

sed -e "s/DATE1/${DATE1}/g;s/DATE2/${DATE2}/g;s/Nort/${Nort}/g;s/West/${West}/g;s/Sout/${Sout}/g;s/East/${East}/g;" GetERA5-pl.py > GetERA5-${DATE1}-${DATE2}-pl.py

python GetERA5-${DATE1}-${DATE2}-pl.py
```

Scripts to submit WRF jobs

batch_real.job

```
#!/bin/bash
#SBATCH -p radclouds
#SBATCH -N 1
#SBATCH -n 56
#SBATCH --exclusive
#SBATCH --output=outbatch.%j.txt
#SBATCH --error=outbatch.%j.txt
#SBATCH -t 00:20:00 # max job run time HH:MM:SS

module purge
source bashrc_wrf

mpirun ./real.exe
```

batch_wrf.job

```
#!/bin/bash
#SBATCH -p radclouds
#SBATCH -N 3
#SBATCH -n 168
#SBATCH --exclusive
#SBATCH --output=outbatch.%j.txt
#SBATCH --error=outbatch.%j.txt
#SBATCH -t 03:00:00 # max job run time HH:MM:SS

module purge
source bashrc_wrf

mpirun ./wrf.exe
```

Results of WRF testing with different compiler settings

After failures to compile WRFV4 using Conda to set up necessary libraries, we found success using OSCER modules:

```
module purge
module load parallel-netcdf
module load netCDF-Fortran/4.4.4-intel-2016a
module load libtirpc/1.2.6-GCCcore-9.3.0
module load JasPer/2.0.14-GCCcore-9.3.0
module load libpng/1.6.26-intel-2016a
```

The catch: parallel-netcdf has only been installed for intel-2016, so loading this module was shifting all compilers back to 2016 versions. WRF would run for a short time then crash after, e.g., ~10 min of wall-clock time.

New approach: avoid parallel-netcdf and retain more up-to-date compilers. The following upload

```
module purge
module load netCDF-Fortran/4.5.3-iimpi-2021a
module load JasPer/2.0.33-GCCcore-11.3.0
module load libpng/1.6.37-GCCcore-11.3.0
```

has so far been successful, keeping all compilers at 2021 versions.

The question now is whether we can speed up model run time via parallel I/O (input/output) while taking advantage of NetCDF4 format compression, which leverages HDF5.

File size comparison:

> Tests with the above module list and using NetCDF4/HDF5 compression (i.e., NETCDF_classic = 0) yield high-res domain output files of ~1.8 GB, vs. ~4.8 GB when compression is turned off by setting NETCDF_classic = 1.
>
> **I.e., a 267% increase in file size when using NetCDF-classic.**

Output timing comparison:

> It took ~95 sec per high-resolution output file write-out when using compression (NETCDF_classic = 0), compared to 20 sec with NETCDF_classic = 1.
>
> **I.e., a 475% increase in required write-out time when using NetCDF4/HDF5 compression. This amounts to ~0.5 h required additional wall-clock time per 24 output time steps, or ~2 h additional for 100 output time steps.**

So what wins out? The increase in storage space when using the non-compressed NetCDF-classic approach is honestly quite forbidding. While it could take modestly more wall-clock time (= more core hours on other systems) to complete a given model experiment, usually computing time is not a bottleneck, while storage is, as it is expensive.

Next steps moving forward…

As of WRFv4.4, there is now the capability to use parallel I/O with compression (NetCDF4/HDF5), which would be the best of both worlds. According to the pertinent WRF Git commit, all we need is NetCDF v4.7.4 or later (our modules offer v4.8) and HDF5 v1.10.3 or later (our modules get us v1.10.7). Also requires setting the environmental variable NETCDFPAR instead of just the usual NETCDF, referencing the necessary include/lib/etc.

More info provided in WRF/docs/README.netcdf4par and the associated git commit.

When configure is run with those settings, configure seems to work with the expected settings:

```
This installation of NetCDF is 64-bit
                 C compiler is 64-bit
           Fortran compiler is 64-bit
              It will build in 64-bit

NetCDF version: 4.8.0
Enabled NetCDF-4/HDF-5: yes
NetCDF built with PnetCDF: no
Using parallel NetCDF via NETCDFPAR option
```

But then when running, after setting io_form_history = 13 and io_form_restart = 13, it never gets past the output step. It doesn't trip and error and crash, it just sits there and never moves, with the output file appearing and never growing from ~15 mb.

We have now gotten this to work, thanks to support from OSCER's David Akin, who pointed out that the number of processors given to WRF via the batch script should make it "easier" for WRF to auto-divide up the model domain.

The namelist settings should include

```
 nproc_x = -1
 nproc_y = -1
```

under `&domains`, which tells WRF to auto-tile the domain based on number of available processors (i.e., based on the batch script).

Next, the example setup that Dave got to successfully run was using the following SLURM settings:

```
#SBATCH -N 2
#SBATCH --ntasks-per-node 50
#SBATCH -n 100
```

The kicker is that this job ran slightly ***slower*** than a comparison test without parallel I/O, so for now, let's set this approach aside.