WRF Tools: A Software Package for Dynamical Downscaling using WRF

Andre R. Erler

August 28, 2017

1 The Dynamical Downscaling Pipeline

Downscaling is the process of obtaining high-resolution information from low-resolution fields, such as the output produced by a Global Climate Model (GCM). The most common form is statistical downscaling, where a statistical relationship between large-scale fields or weather patterns and local weather is extracted from observations, and then applied to the output of GCMs. This approach implicitly assumes the stationarity of the statistical relationship and its application is problematic when the large-scale circulation shifts into a different regime (or climate zone). To overcome these limitations, the approach employed here involves the use of a regional weather/climate model to generate the high-resolution information in a dynamically consistent way that can accommodate major shifts in climate regimes. This approach is called dynamical downscaling (*Giorgi*, 2006; *Maraun et al.*, 2010). It is a physics based method that relies less on calibration than statistical methods, but is computationally much more expensive. The regional model, running at high resolution,

is generally computationally much more expensive than the global model (see *Erler*, 2015, §2.1.5), so that the role of the global model in dynamical downscaling can be seen as "merely" providing initial and boundary conditions for the regional model.

Because the global and the regional model are two separate models, only one-way offline coupling is possible. That means that the global model runs first, and then the data are reprojected and pre-processed, before they can be used as boundary conditions for the regional model. The variables used at the lateral boundaries are horizontal winds, temperature, pressure/geopotential, and water vapor; at the lower boundary, sea-surface temperature and sea ice fraction are provided by the global model; land surface variables are only used during initialization. The regional model generates its own weather and climate inside its domain; however, it still inherits much of the large-scale weather patterns from the global model through the lateral boundary conditions, SSTs and sea ice, and spectral nudging in the upper atmosphere (this is more so the case in winter than in summer, due to the typical size and propagation of weather systems in mid-latitudes).

In order to orchestrate the timely execution of the pre-processing pipeline, the continuous operation of the regional model, as well as automatic archiving and post-processing of the data in a High Performance Computing (HPC) environment, a considerable amount of software had to be developed. The software stack designed to handle this task is the WRF Tools package. The package also contains tools to set up and manage multiple concurrent simulations. WRF Tools is also largely platform independent. Furthermore, a general purpose software package for post-processing and analysis in a workstation environment has been developed, which is called GeoPy; it is entirely written in Python. The GeoPy package forms the backbone of the analysis presented in this study; its main features are dataset abstraction, regridding, advanced statistical functions, high-level plotting functionality, and parallelization. In addition, specialized analysis code was developed for hydrological / basin-scale analysis and Extreme Value Analysis (EVA); these packages are based

Software Packages and Analysis Code Developed Specifically for This Dissertation

	Python	*Comments*	Fortran	*Comments*	Shell	*Comments*
WRF Tools	4300	1500	2300	800	6200	2800
GeoPy	20100	6900	-	-	-	-
Hydro	2400	1100	-	-	-	-
EVA	2000	900	-	-	-	-
Total	28800	10400	2300	800	6200	2800

Table 1: The Lines of Code (LoC) written for this dissertation, listed by programming language and software package, as well as totals by language; comment lines are listed separately for each language and not included in the LoC count.

on GeoPy, but are too specific to be included in the general analysis package that GeoPy is meant to be.

The packages and the Lines of Code (LoC) for each major programming language are listed in Tab. 1. The WRF Tools package also contains a large amount of C-Shell and NCL[1] code, however, this code was largely adapted from existing code written by Dr. J. Gula and Adam Philips from NCAR with modifications by the author of this work; it is thus not listed. Most of the Fortran code has also been adapted from work by Dr. J. Gula, however, the code refactoring and improvement was substantial. It is further noted that the listed Shell script code does not include automatically generated code, queue submission scripts ("run scripts"), or setup/configuration files for individual experiments/simulations. Overall, the author feels that the code listed in Tab. 1 is a fair representation of the authors software development effort for this project.

[1]NCL (NCAR Command Language) is a scripting language specifically developed by NCAR for data analysis in atmospheric science. For example the CVDP package is primarily written in NCL.

2 The WRF Pre-processing System

Before the automatic downscaling pipeline that has been developed, can be described, it is in order to briefly summarize the WRF Pre-processing System (WPS). Note however, that this summary is simply intended to aid the appreciation of the software stack described later; for the practitioner seeking to replicate this work, there is no substitute for reading the manual and/or attending a tutorial at NCAR. The WPS package, as described in the WRF User Guide, consists of three main programs, which have to be executed sequentially; a fourth program, which is usually considered to be part of the WRF package, is also part of the preprocessing pipeline and will be described here as well.

1. `geogrid.exe` This program computes the WRF projection and grid layout from namelist parameters and interpolates a large set of static geographic data (such as elevation or land cover data) onto the WRF grid. The static data is available globally at a resolution of 30 arcsec and can be downloaded from NCAR ($\approx$ 55 GB uncompressed). `geogrid.exe` produces NetCDF output.

2. `ungrib.exe`/`unccsm.exe` The `ungrib.exe` program generates WRF Intermediate Format (IM) files from output files of common meteorological reanalysis and forecast products in GRIB format; it relies on tables and namelist parameters to extract the necessary information and write the IM files (one per input file). The IM files are Fortran binary files that follow specific structure conventions outlined in the WRF User Guide. `ungrib.exe` cannot handle NetCDF data from climate models, so that a companion program, `unccsm.exe`, had to be developed, originally by Dr. J. Gula and revised by the author, in order to convert CCSM3 and CESM1 output to IM files. `unccsm.exe` relies on an additional pre-processing step, involving a NCL script that leverages the specialized regridding functionality provided by NCL to interpolate CESM output.

3. `metgrid.exe` This program reads the IM files as well as the `geogrid.exe` output to interpolate the forcing data to the WRF grid and compute the required quantities from the input. The interpolation is based on tabled rules that depend on the source dataset and the variable to be interpolated. The table originally provided for CESM data contained errors in the SST and sea ice interpolation section, so that some simulations suffer SST and sea ice interpolation errors (see *Erler*, 2015, §2.1.2). The CESM table, as well as the ERA-Interim table, have been successfully revised by the author. `metgrid.exe` outputs one NetCDF file per input time step and WRF domain.

4. `real.exe` In the final step, `real.exe` creates the initial and boundary condition files that are used by WRF. Unlike the other programs, `real.exe` is part of the WRF package. A companion program, `ideal.exe`, exists to generate initial and boundary conditions for idealized experiments. `real.exe` creates several NetCDF files that, collectively, are referred to as wrfinput files.

The `geogrid.exe` program only needs to be executed once per experiment, while the other programs run during every iteration of the pre-processing step. Note that each of the above programs writes its entire output to disk and the next program in the pipeline reads it as input from disk, i.e. all communication between steps in the pre-processing pipeline happens via the file system. This is extremely inefficient, because all of the above programs are I/O limited in their performance, and in an HPC environment the file system is the most unreliable resource, because it is shared. In Sect. 4.1 an alternate approach is outlined that avoids unnecessary disk access.

3 The Downscaling Cycle

The length of each experiment in this study is typically 15 model years; however, due to wallclock limitations and numerical instability it is not possible to run the regional model continuously for the two to three month of real time that would be required to complete the simulation. Partly because of the wallclock limit of 48 hours on the SciNet HPC facility and partly to mitigate the effects of numerical instability, a typical model time interval of one month was chosen, which translates to a worst-case runtime of about 2 days in the case of severe numerical instability (see *Erler*, 2015, §2.3.5 for details on runtime and performance). Each of these one month chunks will be referred to as a "run step" or simply a "step". Each 15 year experiment consists of 180 steps; for each step the model has to be restarted and a new set of input files with boundary conditions has to be provided (the initial conditions are only needed for the first step). This cycle is schematically displayed in Fig. 1. The starting point is 6-hourly forcing data; these data can either be generated by a concurrently running GCM (e.g. CESM) or be retrieved from archive. In the following we will refer to the 6-hourly CESM data as "high frequency" or "HF" output, because CESM typically only outputs monthly data. The first WPS pre-processing step is launched after `geogrid.exe` completes. During this step `ungrib.exe`/`unccsm.exe`, `metgrid.exe`, and `real.exe` run in a in-memory and fully parallelized pipeline that is efficiently orchestrated by a Python driver module (PyWPS, see Sect. 4.1); the wrfinput files are then written to disk, ready to be used by WRF. After the WPS step finishes, the first WRF step job is submitted to the queue, and the cycle begins. At the beginning of each WRF step, the WPS pre-processing job for the next step is launched, before the actual WRF simulations is started. After the WRF step is completed successfully, the next WRF job is submitted, or, in case of a numerical instability, the step is repeated with a smaller time step. In predetermined intervals the WRF job will also automatically launch archiving and post-processing jobs

(typically every model year). At the end of each step WRF writes the entire model state to disk, so that no information is lost between steps. In order to facilitate easy handling of the data, the WRF output is split into several file (streams); some are written 6-hourly and some are written daily. A Python utility (`ioconfig.py`) is provided in the WRF Tools package to rewrite the output stream Registry of WRF in a reproducible manner based on a output configuration file. By default, WRF output is stored in the `wrfout/` subdirectory. The post-processing module (WRFavg, see Sect. 4.2 for details) computes monthly means and certain indices for climatic extremes from these data.

4 The WRF Tools Package

The WRF Tools package consists of two larger Python modules, one for pre-processing and one for post-processing (see below), as well as a large array of tools (mostly shell script) and templates to automate repetitive tasks involved in stetting up experiments and running the automated downscaling cycle.

In order to set up an experiment, a new folder has to be created, which will be the root folder of the experiment. A configuration script `xconfig.sh` is then placed in the root folder and the setup script `setupExperiment.sh` is executed, which sources the configuration script. In the configuration script environment variables are set, which control everything from the domain setup, namelist group templates, individual namelist parameters, the forcing dataset, the machine to run on, and the executables to be used. The setup script creates the namelist files, links the appropriate tables, input data, and executables and automatically generates job submission scripts for the appropriate queue system / machine. The begin and end dates for each job step are also defined in a file (`stepfile`), which is defined in the configuration script (a Python utility is provided to generate stepfiles).

The script `startCycle.sh` is used to run `geogrid.exe` and launch the initial WPS and

WRF jobs; it also creates a compressed archive of the experiment configuration (including all tables and auto-generated scripts) that is archived with the first automatic archiving job.

Every job step runs in its own subdirectory, which is created before the WPS job is submitted. This is necessary to prevent interference with the pre-processing job for the next step. Furthermore, every step requires a new set of namelist, where the start and end dates, as well as the runtime and restart interval, are set to the appropriate values for the step. The `cycling.py` utility handles this task, as well as parsing the step file for the next step definition.

There are two major components of the software stack that deserve a more detailed description: the PyWPS pre-processing module and the WRFavg post-processing module; both are built around a core Python package (`wrfrun` and `wrfavg`) and several helper scripts and libraries, which are described below. Both modules are invoked as part of the automated downscaling pipeline and machine-specific run scripts are automatically created by the setup script.

The package is continuously being developed and improved; a recent version is available on GitHub: `https://github.com/aerler/WRF-Tools`

4.1 The PyWPS Pre-processing Module

The PyWPS module is the driver module for the pre-processing system as it was developed for and used in this study. It is fully parallelized and makes extensive use of a so-called RAM-disk (an in-memory file system), so that no unnecessary temporary files are written to disk; currently the system is only implemented for a shared memory environment. The Python program `pywps.py` itself only organizes the data pipeline between the three WPS programs. At the beginning of the process, the start and end dates of the step are extracted from the WPS namelist and all available input files that match this date range are

identified and distributed to the available processors for parallel time step-by-time step processing. `pywps.py` then sets up the environment and runs `ungrib.exe`/`unccsm.exe` and `metgrid.exe` for each input time step, such that all output is written to the RAM-disk. `pywps.py` itself is launched by a PyWPS driver script, which initializes the RAM-disk and launches `real.exe` after `pywps.py` completes; `real.exe` also reads the `metgrid.exe` output directly from RAM-disk.[2]

The architecture of `pywps.py` is object oriented, with classes representing dataset types. Two generic classes are currently available: one for datasets using `ungrib.exe`, and one for those using `unccsm.exe` (and the NCL preprocessing script). Currently child classes are available for ERA-Interim, NARR, and CFSR (using `ungrib.exe`) and CESM1 and CCSM3 (using `unccsm.exe`). The framework is easily extensible to accommodate new datasets.

`pywps.py` supports comprehensive logging, and in case of an exception/error the current program state and logs from all sub-programs are written to disk for debugging.

4.2 The WRF Post-processing Module WRFavg

A post-processing system has been developed specifically to aggregate WRF output into monthly datasets. The module is called WRFavg (pronounced "WRF average"). It is parallelized (shared memory) in the sense that files from different output streams and domains can be processed in parallel (depending on the number of available processors), but each file type is processed strictly sequentially. This is necessary to take advantage of accumulated flux variables and for the computation of some variables (e.g. consecutive dry-days), but is also useful to ensure that no data is missing.The data are aggregated into a monthly time series for each output stream and domain, but it is not necessary for the WRF output

[2] `real.exe` is already fully parallelized, using MPI.

files to be saved in monthly intervals. Output files can contain an arbitrary number of time steps as long as they are in order and contain the time stamp variable (`Times`). For reference, the available output streams are listed in Tab. 2.

The main module of WRFavg is `wrfavg`; derived variables are defined as classes in the library module `derived_variables.py`. Due to its object oriented structure, WRFavg can easily be extended to compute virtually any derived quantity, by adding a new child class to the library module. Instances of the class can then be added to the list of derived variables for the desired output stream type, and the variable will be computed and stored in the averaged output. Base classes are provided for regular variables, simple extremes, averaged extremes (e.g. pentad precipitation maxima), and consecutive threshold exceedance (e.g. consecutive dry-days). The computation takes full advantage of accumulated output variables and linearity in derived variables.

The WRFavg module contains many options to control its behavior. Typically it is used in one of three modes: it either creates a new output file and processes all available WRF output, it appends all new WRF output to an already existing average file, or it recomputes one or several variables in an existing average file. The monthly output files are usually stored in the `wrfavg/` subdirectory; the naming scheme is the same as the WRF output naming scheme, except that the time stamp is replaced by the string `monthly`. `wrfavg` also has comprehensive logging capabilities and exception/error handling for debugging.

WRFavg is invoked during the automatic post-processing stage (usually once per model year), but a batch processing script based on GNUparallel (*Tange et al.*, 2011) is also provided to efficiently recompute averages for several experiments at once. A similar batch post-processing script is also provided for CESM output, which concatenates monthly output files (and converts them to NetCDF-4), and runs the AMWG and CVDP analysis packages on the CESM output (see *Erler*, 2015, §2.1.5).

Archiving The automatic archiving script `ar_wrfout_fineIO.pbs` is launched alongside WRFavg and archives the WRF output on tape; it is only implemented for the HPSS system and specific to SciNet. `ar_wrfout_fineIO.pbs` can also check/verify the state of the archive, safely remove files from disk (after the archive has been checked), and retrieve archived files.[3] In order to avoid small archive files, the output streams are combined into three archive groups (listed in Tab. 2), which are archived in a single file (DIAGS) or one file per domain (MISC3D and DYN3D). A much less sophisticated archiving script is also available for CESM, but it has to be configured and submitted manually.[4]

References

Erler, A. R. (2015), High Resolution Hydro-climatological Projections for Western Canada, Ph.D. thesis, University of Toronto.

Giorgi, F. (2006), Regional climate modeling: Status and perspectives, in *Journal de Physique IV (Proceedings)*, vol. 139, pp. 101–118, EDP sciences.

Maraun, D., et al. (2010), Precipitation downscaling under climate change: Recent developments to bridge the gap between dynamical models and the end user, *Reviews of Geophysics*, *48*(3).

Tange, O., et al. (2011), Gnu parallel — the command-line power tool, *The USENIX Magazine*, *36*(1), 42–47.

[3] Archiving is currently implemented as a shell script, but given its current complexity, it would be desirable to re-implement the archiving system in Python.

[4] It would be highly beneficial to developed an automated archiving system for CESM as well; possibly also an automated post-processing system.

Currently Supported WRF Output Streams

Name	Interval	Description	Archive
`const`	once	constant fields, e.g. topography, land use, etc.	DIAGS
`srfc`	6-hourly	general meteorological 2 D surface variables	DIAGS
`xtrm`	daily	daily maxima and minima of selected variables	DIAGS
`hydro`	daily	variables for hydrological analysis	MISC3D
`lsm`	daily	land surface variables	MISC3D
`rad`	daily	radiation variables	MISC3D
`plev3d`	6-hourly	meteorological variables on pressure levels (850, 700, 500, 250, and 100 hPa)	DIAGS
`drydyn3d`	6-hourly	dry dynamical variables on all model levels	DYN3D
`moist3d`	6-hourly	moist variables on all model levels	MISC3D
`fdda`	daily	nudging increments for the outer dommain	MISC3D
`snow`	hourly	surface variables, mass and energy fluxes	-

Table 2: The available output streams, their output intervals and archive group. All except the `snow` output stream, which is intended for surface mass balance studies over ice, were used in the simulations presented here.

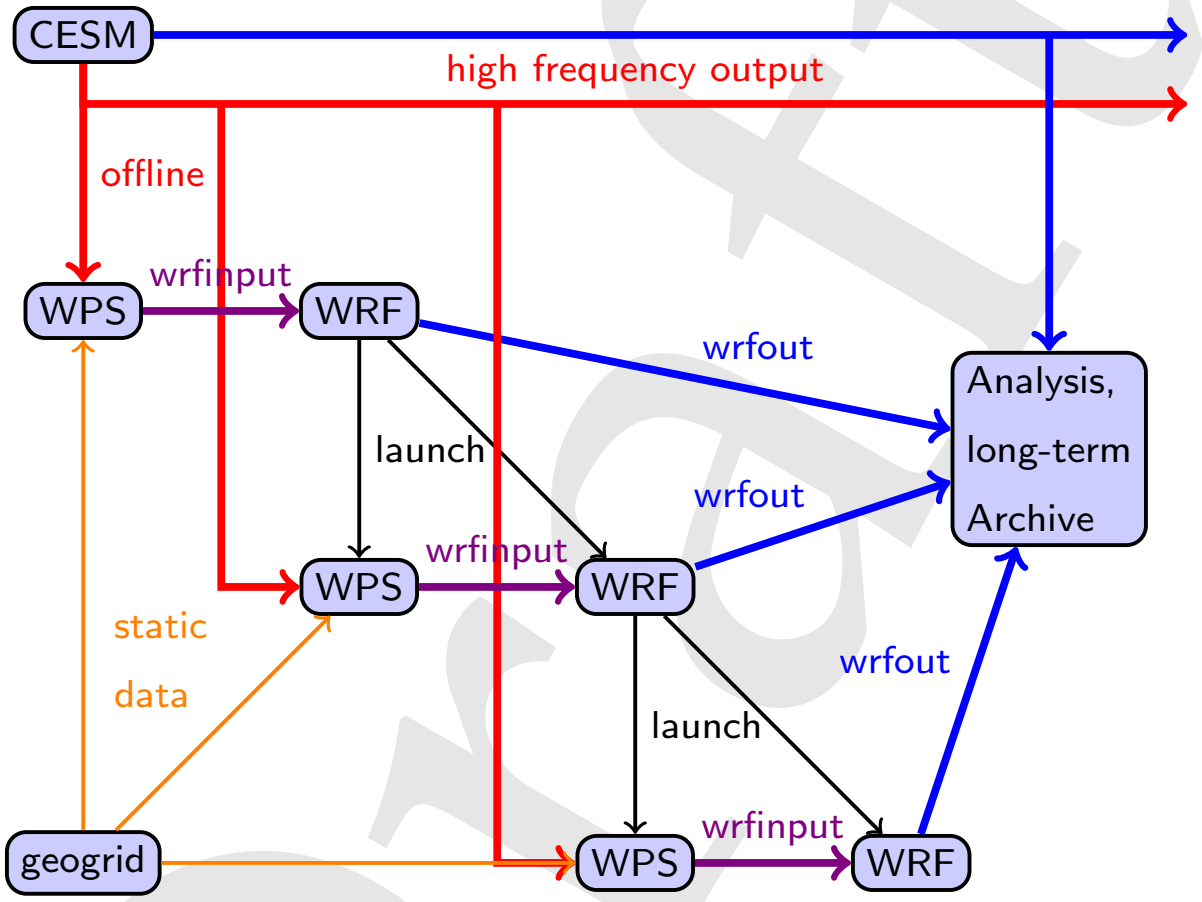

Figure 1: Data-flow schematic for continuous operation of WRF with CESM boundary data and automatic archiving and post-processing. ERA-Interim, NARR, and CFSR data can be downscaled in the same way; hooks are provided to add additional datasets.