Compiling WRF 4.2.2 and Running CONUS Benchmarks on the Stampede2 Supercomputer

Final Project for EAS 5555

Peter Vaillancourt

5/25/2021

Abstract

When running the Weather Research and Forecasting Model (WRF), the user has a vast array of choices and options that ultimately can cause the numerical results of a simulation to vary. Even considering identical initial conditions, one may obtain different results on different systems. Additionally, performance is an important consideration for WRF users due to the high computational cost that can be incurred for larger simulations and forecasts. Benchmarking is a common practice in scientific computing and high-performance computing (HPC) used to evaluate or compare between different systems, software, compilers, or choice of dependencies and library options. This tutorial will cover how to use the WRF version 4.2.2 CONUS benchmarks provided by NCAR to evaluate a setup of WRF. In this walk-through, we demonstrate compiling WRF 4.2.2 on TACC's Stampede2 supercomputer with the Intel 18.0.2 for use on the Intel Skylake nodes, as well as how to run the CONUS 12km and CONUS 2.5km benchmarks on this system. All scripts and files used that were not already provided by NCAR have been made available in the GitHub repository federatedcloud/WRFv4-Benchmarking.

Contents

Scripts

The federatedcloud/WRFv4-Benchmarking GitHub repository we set up for running these benchmarks contains several useful scripts that we will take advantage of for configuring and compiling WRF, organizing and linking data to run the benchmarks, and submitting jobs to Slurm on Stampede2 to perform the runs. All of these scripts are free and open source with a BSD 3-Clause License. Please feel free to use, modify, and distribute these scripts within the loose terms of the license.

For this tutorial, we will explain when and how to use these scripts yourself, though they may require modification depending on your target system. First, change to the directory you would like to clone the repository (we went back to `$WORK2/WRF_Benchmarking`), clone the repository, and switch to the Stampede2 directory to begin work:

```
cd $WORK2/WRF_Benchmarking/
git clone https://github.com/federatedcloud/WRFv4-Benchmarking.git
cd WRFv4-Benchmarking/Stampede2
```

You will notice that there is already a `LIBRARIES` and a `results` directory, which we will use later.

CONUS Benchmark Data

Summary

The WRF Users Page has a large collection of useful resources for WRF users, including some useful benchmarks. The WRF v4.2.2 Benchmark Cases page is where you can obtain some background information on the benchmarks, links to the input data used to simulate the benchmarks, compressed data tarballs (archive of data compressed with the `tar` utility), and some information on the simulations performed to generate the benchmarks.

Get the Data

On most HPC machines, there is a recommended location to store large data files, as well as where it is or is not recommended to perform heavy I/O operations. In our case, we download the data to the `$WORK2` filesystem, after setting up a directory called `data` under our `WRF_Benchmarking` directory. The CONUS 12km tarball is 1.5 GB, and the CONUS 2.5km tarball is 14 GB, so allow appropriate time for downloading.

CONUS 12km

First, change to the directory you would like to contain your data (`cd $WORK2/WRF_Benchmarking/data` in this case), and run the `wget` command.

```
wget https://www2.mmm.ucar.edu/wrf/users/benchmark/v4_bench_conus12km.tar.gz
```

You can then unpack the archive with

```
tar -zxvf v4_bench_conus12km.tar.gz
```

This will create a directory called `v4_bench_conus12km`, which contains several useful files. The `README` file contains a description of each of the files, and basic instructions on how to run and compare the benchmark. A few important notes on the files, beyond what is in the `README`:

- `runwrf.csh` - The PBS submission script used to run the original simulation (likely on Cheyenne), while we will be using a Slurm submission script (the job scheduler on Stampede2).
- `namelist.input` - This should be used to run the benchmark, however, we found some issues with it that needed to be corrected before it would run properly. Included in our GitHub repository is a file called CONUS12km-namelist.input, which is the modified version that worked to run the benchmark on Stampede2.
- `diffwrf.py` - This comparison script requires you to have the python module for netcdf installed, and possibly a few others, depending on your system.

CONUS 2.5km

Again, change to the directory you would like to contain your data, run the `wget` command, and unpack.

```
cd ..
https://www2.mmm.ucar.edu/wrf/users/benchmark/v4_bench_conus2.5km.tar.gz
tar -zxvf v4_bench_conus2.5km.tar.gz
```

This archive contains almost the exact same list of files as the 12km benchmark, including another useful README. In this case, the included `namelist.input` did not need any modifications, but I would warn agains using the `runwrf.csh` to submit your job as it will cause you to request many more resources than necessary to run the restart simulation.

Configuring and Compiling WRF 4.2.2 with Intel 18.0.2

Now that you have your data, you need to download, configure, and compile WRF. A full compilation tutorial for WRF is provided by NCAR, so we will only touch on the relevant parts of the process for this case. We are going to compile WRF 4.2.2 for real cases using the Intel 18.0.2 compiler, Intel MPI 18.0.2, NetCDF 4.6.2, PNetCDF 1.11.0, and PHDF5 1.10.4. Before we begin, there are a few choices of note to consider.

Compiler Choices

The Intel compiler with Intel MPI was selected to best make use of Stampede2's Omni-Path interconnect. The `dmpar` option was selected to enable Distributed-Memory Parallelism, and the Intel Xeon with Advanced Vector Extensions (AVX) option was selected to achieve the best performance possible for the Intel Skylake processors. All of these choices are built into the scripts we use below. This is one example of hardware configuration determining the ideal choice of compiler and options, so it is important to be aware of what choices are best for the system you run on in order to achieve the best performance.

Dependencies

All of these dependencies were available as modules in a variety of versions on Stampede2, so choosing versions may not an obvious or easy process. A good way to skip the pain of trial and error when selecting dependencies for compiling WRF on a new HPC system is to reach out to system administrators for advice on suggested versions, and if there are any special configuration steps they recommend. Additionally, it is important to select the right dependencies to match the configuration. All dependencies are loaded as part of the scripts used below.

Download WRF

Now that you are ready to download WRF, there are a few places you can obtain the source code from. We chose to download the release tarball from the official GitHub repository. Make sure you are in the Stampede2 directory (after cloning the repository containing our scripts above), and download the WRF 4.2.2 tarball:

```
wget https://github.com/wrf-model/WRF/archive/refs/tags/v4.2.2.tar.gz
tar -zxvf v4.2.2.tar.gz
mv WRF-4.2.2/ WRF
```

You might notice that we have also changed the name of the directory to just `WRF` for simplicity. The WRF directory contains everything you need to compile WRF, excluding the aforementioned dependencies, compilers, and some libraries you will need to install.

Installing Libraries

After downloading and unpacking WRF, we can go into the `LIBRARIES` directory, and used a bash script to install the libraries needed:

```
cd LIBRARIES
./install_libs.sh
```

This script contains some commands to load the appropriate modules and sets some environment variables before it downloads, unpacks, and configures the libraries. In this case, we are setting up:

- zlib 1.2.7
- libpng 1.2.50
- Jasper 1.900.1

These versions can be modified and source links changed without much hassle to customize as you see fit. This script also configures the environment to use TACC's NetCDF module, with more configurations included in the configuration/compile script.

Configure & Compile WRF

We have combined the configure and compile steps into a single step using another bash script. Before running this script, there are some important features to understand. Each time the script is run it will clean the current configuration of WRF (if one has already been set up), ensure the appropriate modules are loaded and environment variables defined, run `configure`, make some modifications to the `configure.wrf` file, and run `compile em_real`. The compilation is also timed, and the output will show up in the Slurm output file if you submit a job (as recommended below).

The script first sets the `WRF_SRC_ROOT_DIR` environment variable to the location of the WRF directory, so this will need to be modified to your location if different. The other paths and environment variables set are relative to this directory, or determined by TACC module configurations, so they should not need to be modified if running on Stampede2. If running on a system other than Stampede2, you should verify every path is set up correctly for your system. You should also verify that the configuration choices are correct as well. While preparing your first compile using this script, it is recommended that you comment out the `compile` step before running, run it, and then check the `configure.wrf` file for correctness. You can then uncomment the compile step and save the script.

Submit a Job to Compile

The configure step is quick, but the compilation step will likely take over a half hour to complete. For example, the output of `time` for this compile on Stampede2 was:

```
real    40m3.981s
user    53m33.782s
sys 2m3.533s
```

You can submit a job to Slurm to compile WRF, freeing up your shell to continue other work while you wait. If you are on Stampede2 and using the Skylake nodes, then you could submit a job to the `skx-dev` queue. Depending on how many other people are using the system, you may want to verify that resources are available. You can see a list of queue statuses on Stampede2 using:

```
sinfo -S+P -o "%18P %8a %20F"
```

It is recommended that you use `-N 2` because WRF will default to compiling on 2 cores, but it is not recommended that you use more than this because it can cause compilation errors instead of speeding up the wait. You can submit the job using:

```
sbatch -p skx-dev -n 1 -N 2 -t 02:00:00 -A ADD_PROJECT_NAME_HERE ./config_compile_18.sh
```

Verifying WRF Compiled

After submitting the job, you can check the status using:

```
squeue -u $USER
```

This will list the status of all active or recent jobs for your user. The job ID provided by the Slurm output when you submitted should be listed. Once the status is completed, or the job is not listed anymore (it will disappear off the list a little while after completing), you can take a look at the `compile_em_real.log` in the `WRF` directory to see the result of the compile.

If the compile completed, the file might be thousands of lines long, so do not try to read the whole file. Regardless of status, near the very end of the file there will be a status that either looks like success:

```
--->                  Executables successfully built                  <---
```

Or errors:

```
---> Problems building executables, look for errors in the build log  <---
```

A successful compile will also contain the following executables in or under the `WRF` directory:

- ndown.exe
- main/real.exe
- main/tc.exe
- main/wrf.exe

If your compile is successful, you can move forward with running the benchmarks. If there are errors, you can look through the compile log in more detail looking for error messages. You may have to adjust paths to dependencies, edit the configuration file, or search the WRF User's Forum for solutions to your errors. If you get stuck, you can also reach out to system administrators for your system for help.

CONUS 12km Benchmark

Set Up the Run

At this point, if you've been following along exactly, you should have the CONUS 12km data located in `$WORK2/WRF_Benchmarking/data/v4_bench_conus12km`, the scripts for running the benchmarks located in `$WORK2/WRF_Benchmarking/WRFv4-Benchmarking/Stampede2`, and WRF 4.2.2 that has been compiled with Intel 18.0.2 located in `$WORK2/WRF_Benchmarking/WRFv4-Benchmarking/Stampede2/WRF` (or perhaps different paths as you chose). Make sure that you are in the `Stampede2` directory for the following commands to ensure they work correctly (until Results Comparison). If your paths are different, then you will have to update the scripts accordingly.

The `README` file in for the CONUS 12km data explains which files you will need for the run, which can be copied or moved around if you like. However, we've provided a convenient bash script called `make_CONUS12km_links.sh` to link the data files from the CONUS 12km data location (`DATA_LOCATION`) to the `WRF_RUN_LOCATION`. The script will read the contents of the file called `CONUS12km_files` and link each file that is listed, so if you would like to change which files are linked you only need to edit `CONUS12km_files`. The script to make the links can be run with:

```
./make_CONUS12km_links.sh
```

For the CONUS 12km case, we do not use the provided `namelist.input` file due to the aforementioned errors encountered. We instead copy our version into the correct location using:

```
cp CONUS12km-namelist.input WRF/run/namelist.input
```

Submit Job

Once the data files are linked and the namelist is copied into the `run` directory, you can submit the job to run the benchmark. The `runwrf-CONUS12km.sh` can be used to submit to the `skx-dev` queue on Stampede2 to run the job (after you have entered the name of your project allocation in place of `ADD_PROJECT_NAME_HERE`):

```
./runwrf-CONUS12km.sh
```

This Slurm submission script will request a single node, using 36 cores (which is not a full node since Skylake nodes have 48 cores on Stampede2), with a time limit of 1 hour and a job name of `wrf4-CONUS12-singlenode`. If there are `rsl` files leftover from previous runs, it will remove them (save them elsewhere before the run if you would like to keep them). The script also uses `ibrun` instead of `mpirun` or `mpiexec` (or other variants) based on TACC's recommendations for MPI jobs.

Similar to what was discussed in the Submit a Job to Compile section, you can check on the status of queues and your job. Once completed, check your Slurm output file for the job to make sure there were no MPI errors, and to check your the time. For example, our run of the 12km benchmark time output was:

```
real    1m54.621s
user    63m24.756s
sys 2m31.459s
```

You can also check the `rsl` files for relevant output and error messages (located in `WRF/run`) as applicable.

Results Comparison

Because we ran the script to link the relevant data into the `WRF/run` directory, we will now go there (`cd WRF/run`) to run the comparison using the `diffwrf.py` script that was included with the benchmark data. For Stampede2, we had to install a few relevant python libraries for our user before the script would run successfully:

```
pip install --user netcdf4
pip install --user pathlib
```

You can then run the comparison using the command from the `README`:

```
python diffwrf.py wrfout_d01_2019-11-27_00:00:00.orig wrfout_d01_2019-11-27_00:00:00
```

Which will produce output that looks like this:

```
Found wrfout_d01_2019-11-27_00:00:00.orig
Found wrfout_d01_2019-11-27_00:00:00

Differences of the output from two WRF model simulations for a few important fields
Variable Name                Minimum      Maximum      Average      Std Dev         Skew
=====================================================================================
U                           -4.16084      5.93886 -2.99952e-05    0.0102766      11.8325
V                           -3.41641      2.29156  8.73981e-06    0.0114926      -1.3646
W                          -0.274579     0.211226 -6.55367e-06   0.00122614    -0.281197
T                           -1.05276      1.01389 -0.000684021   0.00470221     -17.3978
PH                          -19.5206      16.5378    -0.108796     0.201202     -1.65342
QVAPOR                   -0.00202087   0.00280839 -1.49706e-09   5.2856e-06      91.7791
```

```
TSLB                         -0.160492     0.138885 -6.54742e-06   0.00160533     -4.95462
MU                            -17.2239      20.7786    0.0285752     0.280453      3.19572
TSK                           -2.86723      2.43886 -0.000130465    0.0682393     -1.21115
RAINC                        -0.418266      0.84219  2.52772e-06   0.00583878       21.832
RAINNC                       -0.162371     0.162565 -3.46978e-06   0.00161308     -3.45321
```

These are our results comparing the NCAR-provided original with our output run on Stampede2. Based on the guidance in the `README`, these numbers appear to be good. The tolerance of the results you are looking for may vary depending on the differences between the 2 configurations you are comparing outputs for. If you would like to compare 2 results of your own making – such as an Intel compile and a GNU compile of the same version of WRF on the same system – then you can simply rename your first output to `wrfout_d01_2019-11-27_00:00:00.orig` instead of linking their original (simply remove it from the `CONUS12km_files` list before running `make_CONUS12km_links.sh`) and make sure it is located in the `WRF/run` folder before running `diffwrf.py`.

Cleanup

Before starting the CONUS 2.5km benchmark (or some other run), we need to cleanup from the 12km run. If you do not do this step, the the linking script for the 2.5km benchmark will not overwrite the existing files, and you will not be able to properly run the next benchmark. First, move an files you would like keep elsewhere for storage, then run the following from the `WRF/run` directory:

```
cat ../../CONUS12km_files | xargs rm
rm namelist.input
```

The first command will remove all of the links that were created by the `make_CONUS12km_links.sh` script, and the second will remove the `namelist.input` since we manually copied that in. The `rsl` files will be removed as part of the runscript, so we can safely ignore those for now.

CONUS 2.5km Benchmark

Set Up the Run

The process for running the CONUS 2.5km benchmark is very similar to that of the CONUS 12km one, starting with moving to the `Stampede2` directory (`cd ../..`). The `README` file in for the CONUS 2.5km data is nearly identical as well. We've provided another convenient bash script called `make_CONUS2.5km_links.sh` to link the data files from the CONUS 2.5km data location (`DATA_LOCATION`) to the `WRF_RUN_LOCATION`. The script will read the contents of the file called `CONUS2.5km_files` and link each file that is listed (a slightly different list), as before:

```
./make_CONUS2.5km_links.sh
```

For the CONUS 2.5km case, we do not need to edit the provided `namelist.input`, so it is included in the list of files that are linked from the provided data.

Submit Job

Linking the data files is all that is required to be ready to submit the job. Before you submit the job, however, it is worth noting that this is a larger and longer run. From the `Stampede2` directory, the `runwrf-CONUS2.5km.sh` can be used to submit a job to the `skx-dev` queue on Stampede2 (again entering the name of your project allocation in place of `ADD_PROJECT_NAME_HERE`):

```
./runwrf-CONUS2.5km.sh
```

This time our submission script requests 4 full nodes (48 cores each), but still with a time limit of 1 hour and a job name of `wrf4-CONUS2.5-multinode`. Because this benchmark is multinode, the runtime results will provide a more effective evaluation of the performance for larger-scale runs. For example, our run of the 2.5km benchmark time output was:

```
real    46m23.317s
user    545m24.336s
sys 9m16.091s
```

As with the 12km script, the `rsl` files leftover from previous runs are removed, and `ibrun` is used to initiate the MPI job. You will still want to check your Slurm output file and `rsl` files to ensure there were no errors.

Results Comparison

If you would like to compare 2 results of your own making, first move or link the original as `wrfout_d01_2019-11-27_00:00:00.orig` in the `WRF/run` directory before running `diffwrf.py`. Next, return to the `run` directory to run the comparison using the `diffwrf.py` script that was included with the benchmark data:

```
python diffwrf.py wrfout_d01_2019-11-27_00:00:00.orig wrfout_d01_2019-11-27_00:00:00
```

The provided script may hit a python divide by zero error when performing the calculations for `RAINC`, resulting in zeros for that row. It appears this way in the `README` as well, so it is likely a bug in the script, not in the results.

The output from our results comparing the NCAR-provided original with our output run on Stampede2 (with the divide by zero error removed):

```
Found wrfout_d01_2019-11-27_00:00:00.orig
Found wrfout_d01_2019-11-27_00:00:00

Differences of the output from two WRF model simulations for a few important fields
Variable Name               Minimum      Maximum      Average      Std Dev        Skew
=================================================================================
U                          -0.721531      1.11926 -0.000135359   0.00375231   -0.0989335
V                           -1.24143      1.25165 -7.83834e-06   0.00404853    0.0165563
W                           -0.38914     0.382936 -2.66358e-05   0.00115425     -1.71199
T                          -0.577423     0.598206  -0.00325452   0.00357311     0.977274
PH                          -29.4688       33.134     -0.44161     0.542068     -1.26133
QVAPOR                   -0.00335396    0.0035795 -2.88285e-08  4.73663e-06      -101.21
TSLB                       -0.125702      0.11319 -8.26345e-06   0.00114675    -0.421092
MU                          -41.8051      35.8218     0.145044     0.178355      6.66657
TSK                         -3.91763      4.23738 -0.000325014    0.0543067     0.327253
RAINC                              0            0            0            0            0
RAINNC                     -0.407825     0.168114   2.7888e-05   0.00113085     -129.431
```

Again, these numbers appear to be good based on the guidance in the `README`.

Cleanup

This step is not strictly necessary if you are not planning another run from this configuration, but useful if you are. First, move an files you would like keep elsewhere for storage, then run the following from the `WRF/run` directory:

```
cat ../../CONUS2.5km_files | xargs rm
rm rsl.*
```

Again, this will remove all of the links that were created by the `make_CONUS2.5km_links.sh` script. The second command will remove the `rsl` files.

Next Steps

Now that you know how to compile WRF and run the CONUS benchmarks, you can use these benchmarks to evaluate WRF compiled with different compilers or on different systems. Using the `time` command output to compare runs on various systems can also be useful for gauging performance, especially for multinode runs such as the CONUS 2.5km benchmark. Personally, I intend to use these benchmarks to test, verify functionality of, and compare WRF compiled bare metal on Stampede2 to containerized (Docker and Singularity) and virtualized WRF compiled with the same (or similar) Intel compiler and dependencies. This will be a first step in exploring whether containerization or virtualization play a significant role in performance and/or numerical stability of results for WRF.