WRF Scaling and Performance Assessment

Akira Kyle[1], Davide Del Vento [2], Dixit Patel [3], Brian Vanderwende [2], Negin Sobhani [2]

[1]Carnegie Mellon University [2]National Center For Atmospheric Research [3]University of Colorado Boulder

Introduction

The Weather Research and Forecast (WRF) model is a parallel mesoscale numerical weather forecasting application used in both operational and research environments. WRF is among the more commonly run codes by atmospheric scientists on NCAR's Cheyenne supercomputer. Hence it is very important for WRF's users to know how to obtain the best performance of WRF on Cheyenne, especially as users scale their runs to larger core counts.

Benchmark Cases

We found that the official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior. Since we were interested in benchmarking the most recent version of WRF (4.0), we updated the old CONUS benchmarks and created several new benchmark cases. These benchmark cases cover commonly used physics parameterizations. The CONUS benchmarks use the CONUS physics suite, however the 2.5 km resolution case disables `cu_physics`. The Hurricane Maria benchmarks use the TROPICAL physics suite but with `cu_physics` disabled and `sf_sfclay_physics = 1` for both resolutions.

Region	Resoultion	Horizontal Gridpoints	Vertical Gridpoints	Total Gridpoints	Time step	Run Hours
CONUS	12 km	425	300	127,500	72	6
CONUS	2.5 km	1901	1301	2,473,201	15	6
Maria	3 km	1396	1384	1,932,064	9	3
Maria	1 km	3665	2894	10,606,510	3	1

Compilers and Message Passing Interface Libraries

We compared the performance of the Intel and GNU compilers with various compilation flags summarized below. Note that Cheyenne has 4,032 dual-socket nodes each with an 18 core, 2.3-GHz Intel Xeon E5-2697V4 Broadwell processor.

- GNU Compiler Collection (GCC) versions 6.3.0, 8.1.0
  - WRF compiles with `-O2` by default
  - `-O3` : enables all `-O2` optimization along with optimizations such as function inlining and loop vectorization
  - `-Ofast` : enables all `-O3` optimizations along with disregarding strict standards compliance (such is for floating point operations)
  - `-mfma` : enables Fused Multiply-Add instruction set
  - `-march=native` : enables target instruction set to be everything supported by the compiling machine
- Intel Compiler versions 17.0.1, 18.0.1
  - WRF compiles with `-O3` by default
  - `-Xhost` : similar to GNU's `-march=native`
  - `-fp-model fast=2` : similar to GNU's `-Ofast` optimization

We see from Figure 1 that the Intel compiler is consistently faster than the Gnu compiler across all flags tried. We also see that for both Intel and Gnu, the `-Ofast` (for Gnu) or `-fp-model fast=2` (for Intel) are the only flags that make a significant difference in speed. Other flags tried such as `-mfma` or `-march=native -Xhost` made little to no difference in WRF's speed.

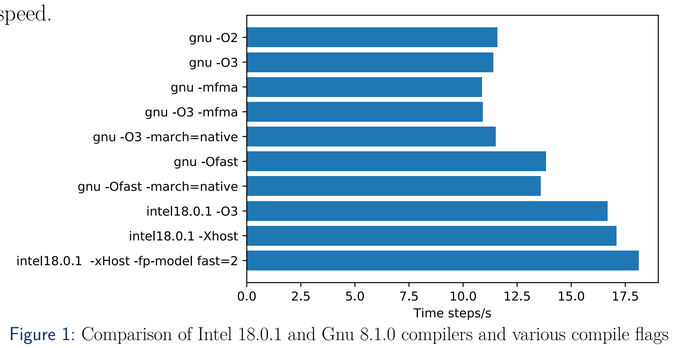

Figure 1: Comparison of Intel 18.0.1 and Gnu 8.1.0 compilers and various compile flags

Message Passing Interface Libraries

We also tested the scaling performance of several MPI implementations. Note that Cheyenne uses a Mellanox EDR InfiniBand high-speed interconnect with a Partial 9D Enhanced Hypercube single-plane interconnect topology.

- MPT version 2.18 (Default MPI for Cheyenne)
- MVAPICH version 2.2
- OpenMPI version 3.1.0
- Intel MPI version 2018.1.163
- MPICH version 3.2

We see from Figures 2 and 3 that MPT, MVAPICH and OpenMPI all have similar performance, while MPICH has overall poor performance and the performance Intel MPI does not scale well to large node counts.

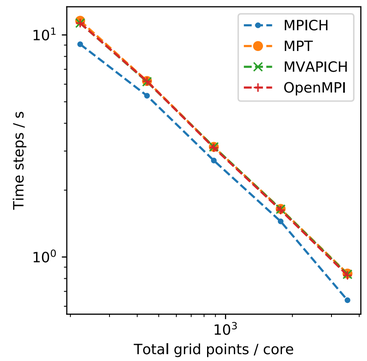

Figure 2:MPI comparison with Gnu 8.1.0

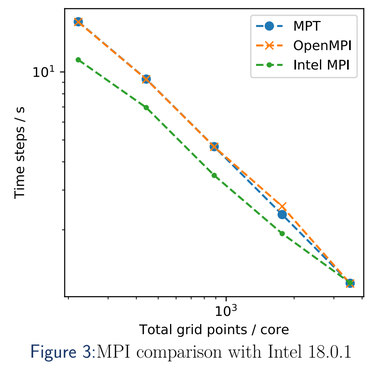

Figure 3:MPI comparison with Intel 18.0.1

Run Time Scaling Results

Figures 4 and 5 show the total run time for WRF using increasing numbers of cores with the total run time broken down into the initialization time, computation time, and writing time. These results use the 1 km Hurricane Maria Benchmark case. We see that on Cheyenne the initialization and writing output times remain relatively fixed, only increasing slightly as you move to larger core counts. However on Yellowstone, the initialization time scaled much poorer at large node counts, eventually leading to unfeasible long jobs. This improvement in the scaling of the initialization time is likely due to the improvements made to WRF's initialization code with how the MPI calls are performed along with improvements in the MPI used in Cheyenne versus Yellowstone.

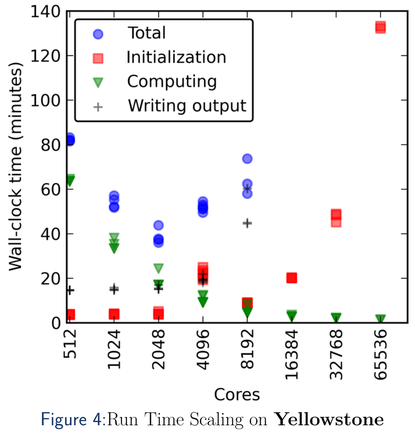

Figure 4:Run Time Scaling on **Yellowstone**

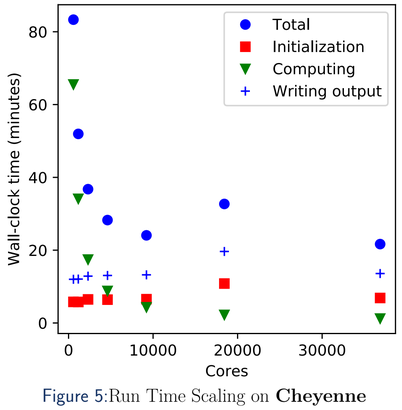

Figure 5:Run Time Scaling on **Cheyenne**

Computation Time Scaling Results

When expressing the scaling as a function of WRF gridpoints per core, we see that all the cases scale similarly in both Figure 7 on Cheyenne and Figure 6 on Yellowstone. Note that both axes are logarithmic, so a small distance between points corresponds to a large difference in values. On both Cheyenne and Yellowstone, in the high relative gridpoints per core region, we see that WRF has linear strong scaling. This means that WRF is making effective use of the parallel computational resources available to it. So increasing the number of cores a run uses, will proportionately decrease WRF's computation time (however initialization and I/O time may increase) while about same number of total core-hours will be used for computation.

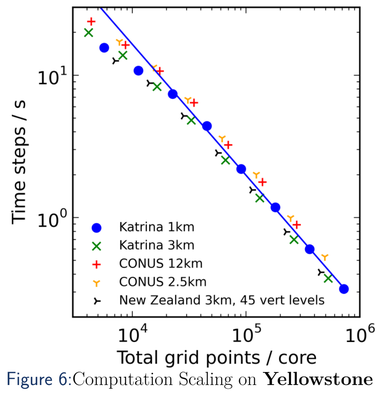

Figure 6:Computation Scaling on **Yellowstone**

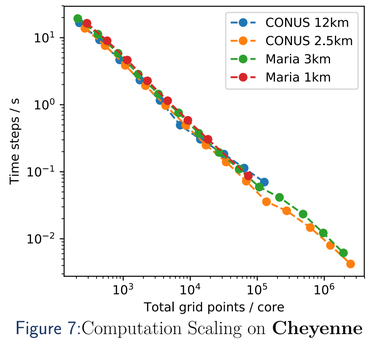

Figure 7:Computation Scaling on **Cheyenne**

Running WRF on Cheyenne versus Yellowstone differs in the the low relative gridpoints per core region. On Yellowstone we see WRF depart from the linear strong scaling relationship. User's running WRF in this low gridpoints per core region on Yellowstone would effectively be using more core-hours to run the same simulation than if they had run it on fewer cores. In this low gridpoints per core region, MPI communication starts to dominate the actual time spent in computation. However on Cheyenne, we see that WRF does not significantly depart from this linear strong scaling at any amount of gridpoints per core. Likely this is due to improvements in the WRF's MPI code along with a better network interconnect on Cheyenne than Yellowstone along with a MPI library. Furthermore WRFV4.0 will refuse to run if there is a minimum patch size of less than 10 grid points in either direction. This limits users from running with fewer than 100 gridpoints per core, which would likely be a very MPI communication bound region where WRF would depart from its linear strong scaling.

The interesting feature in Figure 7 in the high gridpoints per core region where the time steps per second seem to jump slightly is an artifact of the memory constraints on Cheyenne. The normal nodes on Cheyenne have only 64 GB of memory, which is less than on Yellowstone. WRF runs with too many gridpoints per node will run out of memory and be killed. The maximum number of gridpoints per node that will fit into the 64 GB of memory of that node depends on the physics parameterizations, however, we observed that typically the maximum is between 10^5 and 10^6 total gridpoints. Thus to obtain the results in the very large gridpoints per core region, we utilized Cheyenne's 128 GB memory nodes for an additional point or two, then we undersubscribed the cores on each node. This undersubscription of cores is likely responsible for the small bump in speed observed. However we do not recommend that users undersubscribe cores as core-hours are charged for all cores on the node so undersubscription of cores will be an inefficient use of a user's core-hour allocation. Finally it's worth noting that the vertical axis between Figures 6 and 7 is shifted due to the difference in clock speeds between the processors used in Yellowstone versus those used in Cheyenne.