WRF Scaling and Performance Assessment

Comparison of Compilers and MPI Libraries on Cheyenne

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

August 2, 2018

[1] Carnegie Mellon University

[2] NCAR NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

[3] University of Colorado Boulder

Outline

- Background
  - WRF
  - Cheyenne
  - Benchmark Cases
- Compilers
- Message Passing Interface Libraries
- Run Time Scaling
- Computation Time Scaling
- MVAPICH scaling

Background

- The Weather Research and Forecast (WRF) model is a parallel mesoscale numerical weather forecasting application used in both operational and research environments.

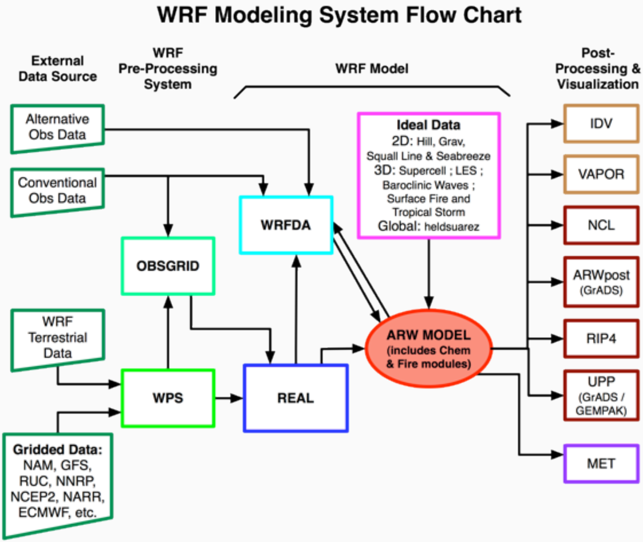
The Weather Research and Forecast Model

- 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.

The Weather Research and Forecast Model

- 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.
    - Thus 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.

WRF System Flowchart



Cheyenne

- 4,032 computation nodes

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops
- 313 TB total system memory

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops
- 313 TB total system memory
  - 64 GB/node on 3,168 nodes, DDR4-2400

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops
- 313 TB total system memory
  - 64 GB/node on 3,168 nodes, DDR4-2400
  - 128 GB/node on 864 nodes, DDR4-2400

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops
- 313 TB total system memory
  - 64 GB/node on 3,168 nodes, DDR4-2400
  - 128 GB/node on 864 nodes, DDR4-2400
- Mellanox EDR InfiniBand high-speed interconnect

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops
- 313 TB total system memory
  - 64 GB/node on 3,168 nodes, DDR4-2400
  - 128 GB/node on 864 nodes, DDR4-2400
- Mellanox EDR InfiniBand high-speed interconnect
  - Partial 9D Enhanced Hypercube single-plane interconnect topology

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops
- 313 TB total system memory
  - 64 GB/node on 3,168 nodes, DDR4-2400
  - 128 GB/node on 864 nodes, DDR4-2400
- Mellanox EDR InfiniBand high-speed interconnect
  - Partial 9D Enhanced Hypercube single-plane interconnect topology
  - Bandwidth: 25 GBps bidirectional per link

Cheyenne

- 4,032 computation nodes
  - Dual-socket nodes, 18 cores per socket
    - 145,152 total processor cores
  - 2.3-GHz Intel Xeon E5-2697V4 (Broadwell) processors
    - 16 flops per clock
  - 5.34 peak petaflops
- 313 TB total system memory
  - 64 GB/node on 3,168 nodes, DDR4-2400
  - 128 GB/node on 864 nodes, DDR4-2400
- Mellanox EDR InfiniBand high-speed interconnect
  - Partial 9D Enhanced Hypercube single-plane interconnect topology
  - Bandwidth: 25 GBps bidirectional per link
  - Latency: MPI ping-pong < 1 μs; hardware link 130 ns

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.
- Wanted to benchmark most recent version of WRF (4.0), so we had to update the old CONUS benchmarks.

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.
- Wanted to benchmark most recent version of WRF (4.0), so we had to update the old CONUS benchmarks.
    - Also created several new benchmark cases.

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.
- Wanted to benchmark most recent version of WRF (4.0), so we had to update the old CONUS benchmarks.
    - Also created several new benchmark cases.
- Benchmark cases cover commonly used physics parameterizations.

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.
- Wanted to benchmark most recent version of WRF (4.0), so we had to update the old CONUS benchmarks.
  - Also created several new benchmark cases.
- Benchmark cases cover commonly used physics parameterizations.
  - CONUS benchmarks use the CONUS physics suite.

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.
- Wanted to benchmark most recent version of WRF (4.0), so we had to update the old CONUS benchmarks.
  - Also created several new benchmark cases.
- Benchmark cases cover commonly used physics parameterizations.
  - CONUS benchmarks use the CONUS physics suite.
    - But 2.5 km resolution case disables `cu_physics`.

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.
- Wanted to benchmark most recent version of WRF (4.0), so we had to update the old CONUS benchmarks.
  - Also created several new benchmark cases.
- Benchmark cases cover commonly used physics parameterizations.
  - CONUS benchmarks use the CONUS physics suite.
    - But 2.5 km resolution case disables `cu_physics`.
  - Hurricane Maria benchmarks use the TROPICAL physics suite

Benchmark cases

- Official CONUS (Contiguous United States) benchmarks for WRF only ran on WRF versions 3.8.1 or prior.
- Wanted to benchmark most recent version of WRF (4.0), so we had to update the old CONUS benchmarks.
  - Also created several new benchmark cases.
- Benchmark cases cover commonly used physics parameterizations.
  - CONUS benchmarks use the CONUS physics suite.
    - But 2.5 km resolution case disables `cu_physics`.
  - Hurricane Maria benchmarks use the TROPICAL physics suite
    - But `cu_physics` disabled and `sf_sfclay_physics = 1` for both resolutions.

Summary of Benchmark Cases

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

Compilers

- GNU Compiler Collection (GCC) versions 6.3.0, 8.1.0

Compiler Options

- GNU Compiler Collection (GCC) versions 6.3.0, 8.1.0
  - WRF compiles with `-O2` by default

Compiler Options

- 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 and more aggressive loop unrolling

Compiler Options

- 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 and more aggressive loop unrolling
  - `-Ofast` : enables all `-O3` optimizations along with disregarding strict standards compliance (such is for floating point operations)

Compiler Options

- 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 and more aggressive loop unrolling
  - `-Ofast` : enables all `-O3` optimizations along with disregarding strict standards compliance (such is for floating point operations)
  - `-mfma` : enables Fused Multiply-Add instruction set

Compiler Options

- 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 and more aggressive loop unrolling
  - `-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

Compiler Options

- 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 and more aggressive loop unrolling
  - `-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 (Default compiler for Cheyenne)

Compiler Options

- 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 and more aggressive loop unrolling
    - `-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 (Default compiler for Cheyenne)
    - WRF compiles with `-O3` by default

Compiler Options

- 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 and more aggressive loop unrolling
  - `-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 (Default compiler for Cheyenne)
  - WRF compiles with `-O3` by default
  - `-Xhost` : similar to GNU's `-march=native`

Compiler Options

- 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 and more aggressive loop unrolling
  - `-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 (Default compiler for Cheyenne)
  - WRF compiles with `-O3` by default
  - `-Xhost` : similar to GNU's `-march=native`
  - `-fp-model fast=2` : similar to GNU's `-Ofast` optimization

Compiler Performance Results

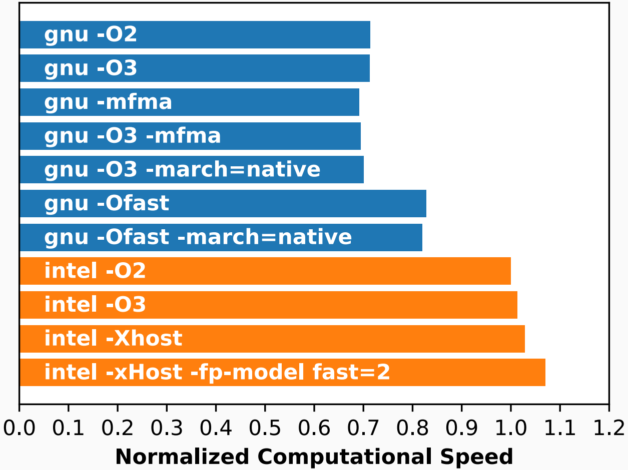
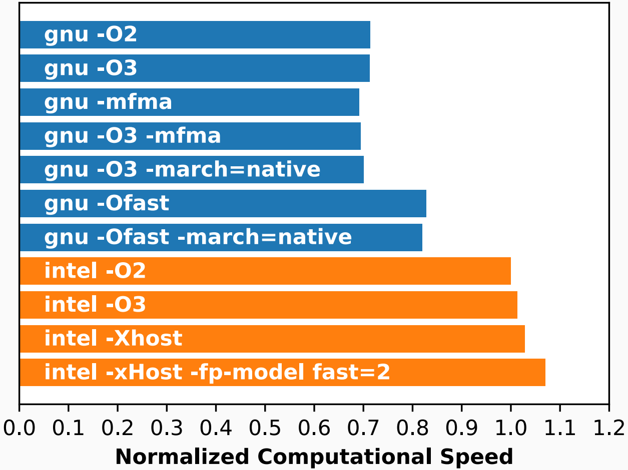


Fig. 1: Comparison of Intel 18.0.1 and Gnu 8.1.0 compilers with various compilation flags normalized to default Intel WRF compilation

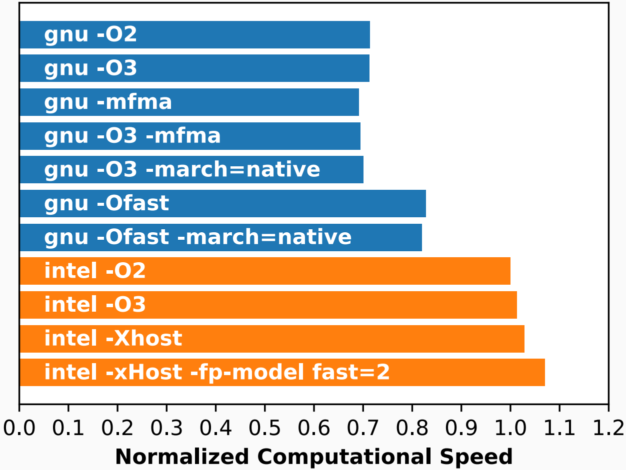
Runs made using CONUS 12 km Benchmark Case on 2 Nodes

Compiler Performance Results



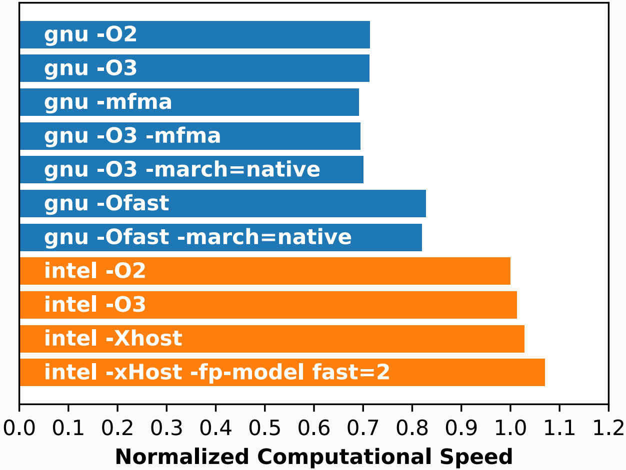
Intel compiler is consistently 25-30% faster than the Gnu compiler across all flags tried.

Compiler Performance Results



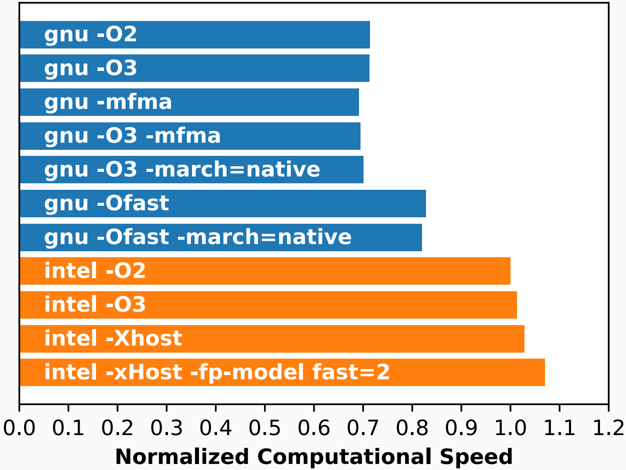
We also see that for both Intel and Gnu, the `-Ofast` (for Gnu) or `-fp-model fast=2` (for Intel) are the only flags that make a significant difference in speed.

Compiler Performance Results



Other flags tried such as `-mfma` or `-march=native -Xhost` made little to no difference in WRF's speed.

Compiler Performance Results



WRF has compilation option (66) which enables `-fp-model fast=2` and `-Xhost` and a few other flags.

Message Passing Interface Libraries

MPIs Tested on Cheyenne

- SGI's MPT version 2.18 (v2.15 is default MPI on Cheyenne)

MPIs Tested on Cheyenne

- SGI's MPT version 2.18 (v2.15 is default MPI on Cheyenne)
- Ohio State University's MVAPICH version 2.2

MPIs Tested on Cheyenne

- SGI's MPT version 2.18 (v2.15 is default MPI on Cheyenne)
- Ohio State University's MVAPICH version 2.2
- OpenMPI version 3.1.0

MPIs Tested on Cheyenne

- SGI's MPT version 2.18 (v2.15 is default MPI on Cheyenne)
- Ohio State University's MVAPICH version 2.2
- OpenMPI version 3.1.0
- Intel MPI version 2018.1.163

MPIs Tested on Cheyenne

- SGI's MPT version 2.18 (v2.15 is default MPI on Cheyenne)
- Ohio State University's MVAPICH version 2.2
- OpenMPI version 3.1.0
- Intel MPI version 2018.1.163
- MPICH version 3.2

MPI Comparison Results

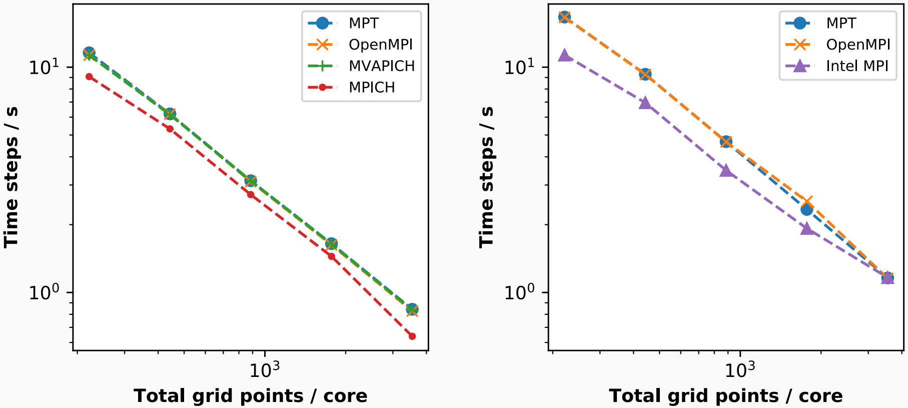
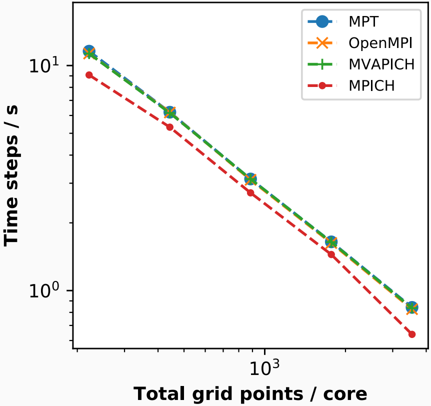


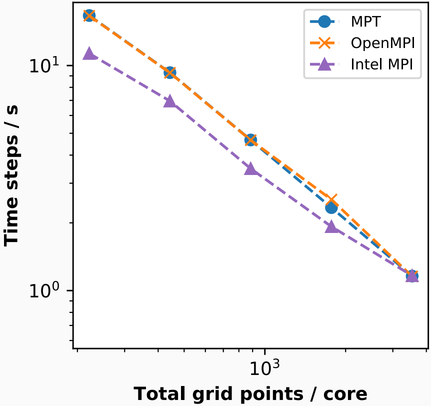
Fig. 2: MPI comparison using **Gnu 8.1.0**

Fig. 3: MPI comparison using **Intel 18.0.1**

Runs made using CONUS 12 km Benchmark Case

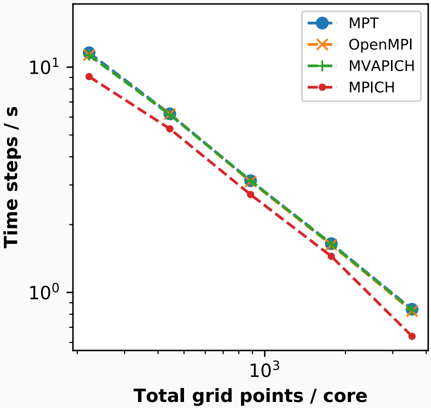
MPI Comparison Results

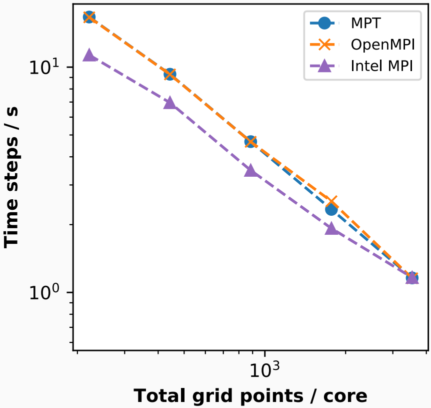




- MPT, MVAPICH and OpenMPI all have similar performance.

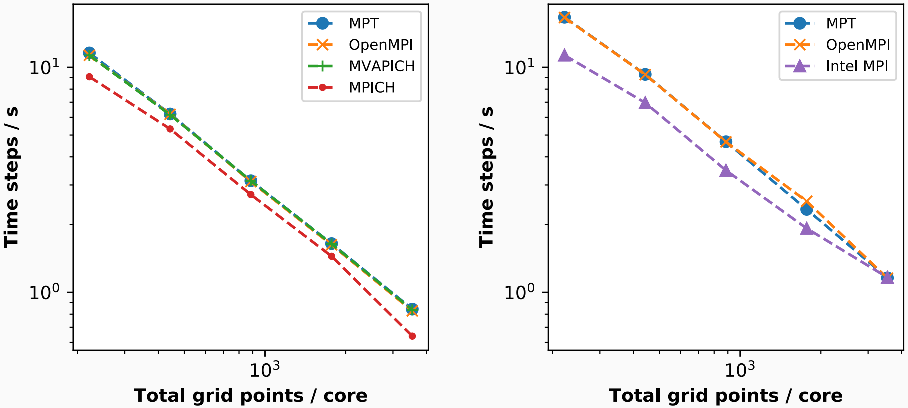
MPI Comparison Results





- MPT, MVAPICH and OpenMPI all have similar performance.
- MPICH has overall poor performance and the performance.

MPI Comparison Results



- MPT, MVAPICH and OpenMPI all have similar performance.
- MPICH has overall poor performance and the performance.
- Intel MPI does not scale well to large node counts.

Total Run Time Scaling

Run Time Scaling Comparison

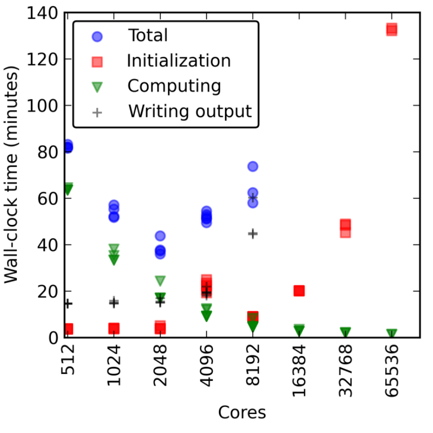


Fig. 4: WRF V3.3 Run Time Scaling on **Yellowstone**

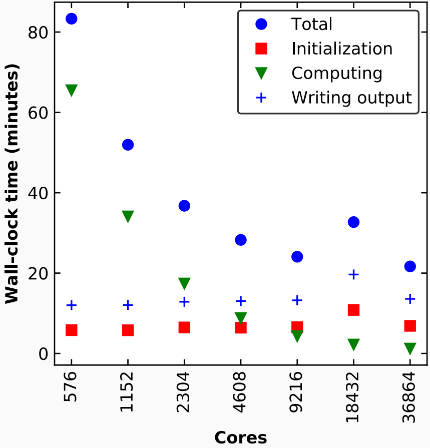
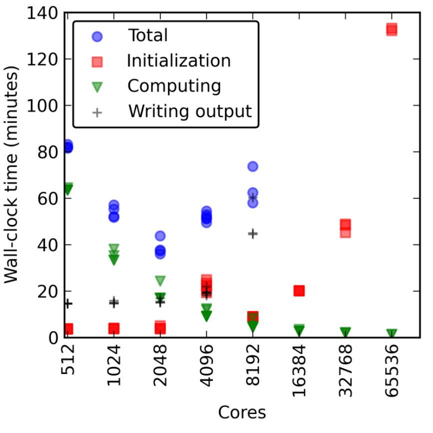
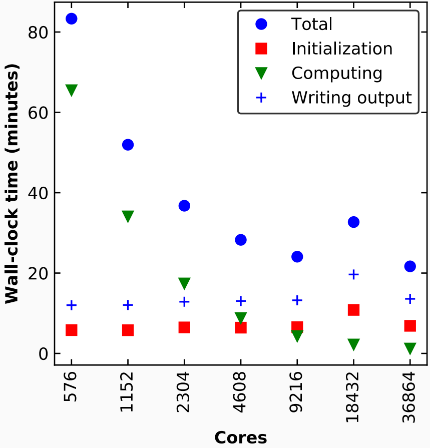


Fig. 5: WRF V4.0 Run Time Scaling on **Cheyenne**

Runs made using Hurricane Maria 1 km Benchmark case.

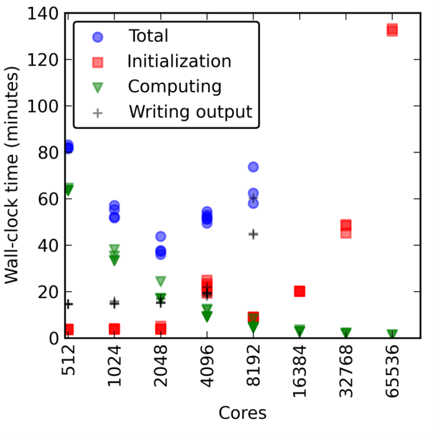
Run Time Scaling Comparison

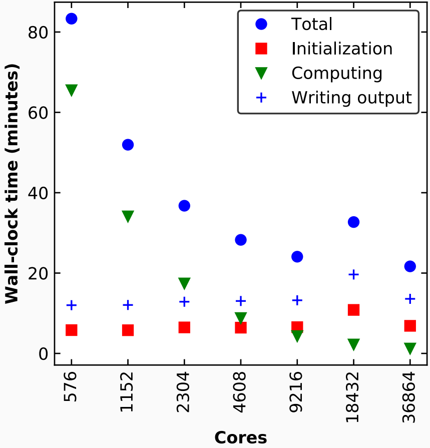




On Yellowstone (Fig 4), the initialization time scaled much poorer at large node counts, eventually leading to unfeasibly long jobs.

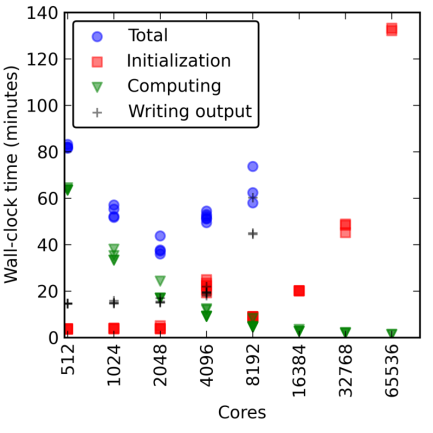
Run Time Scaling Comparison

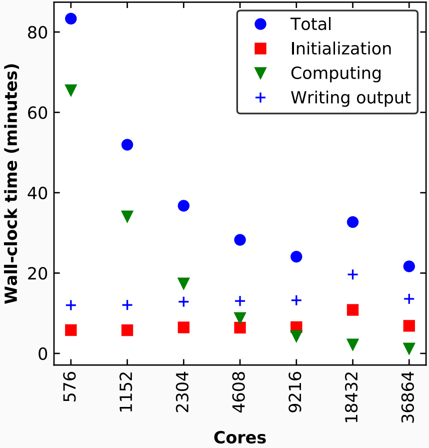




On Cheyenne (Fig 5), the initialization and writing output times remain relatively fixed, only increasing slightly as you move to larger core counts.

Run Time Scaling Comparison





This improvement in the scaling of the initialization time is likely due to improvements made in the MPI collectives in WRF's initialization and writing output code along with improvements to the MPI used on Cheyenne versus Yellowstone.

Computation Time Scaling

Computation Time Scaling Results

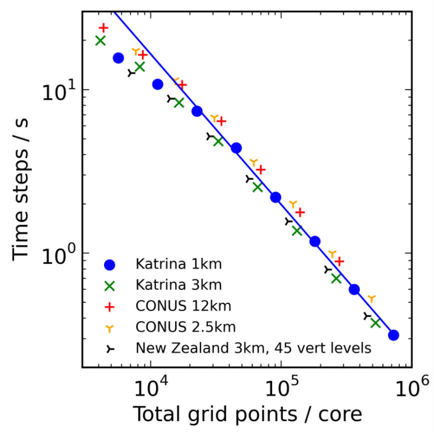


Fig. 6: WRF V3.3 Computation Scaling on **Yellowstone**

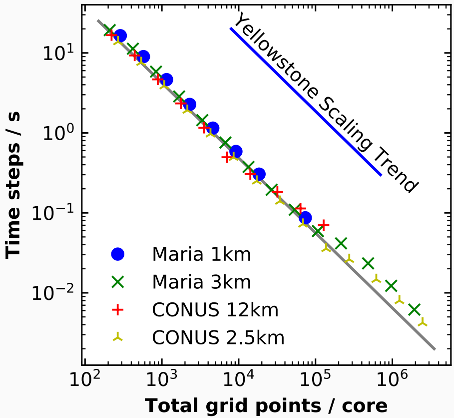
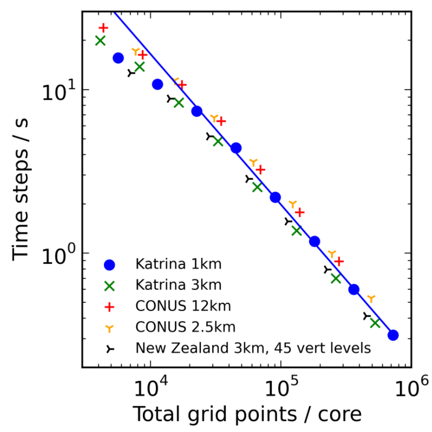
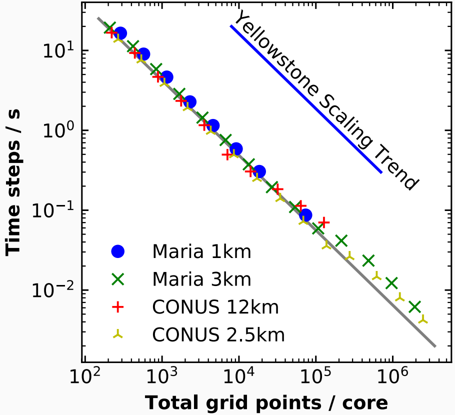


Fig. 7: WRF V4.0 Computation Scaling on **Cheyenne**

Computation Time Scaling Results

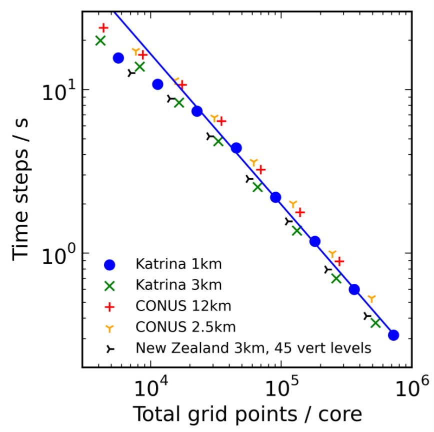


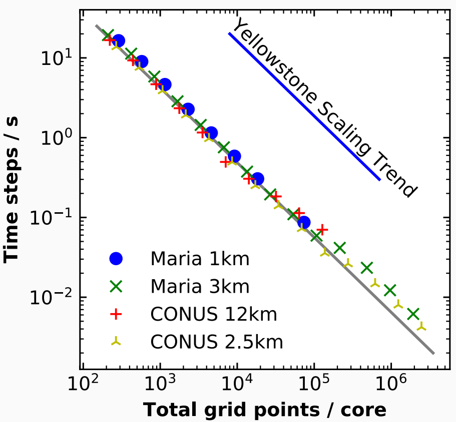


Large number of gridpoints per core region:

- On both Yellowstone (Fig 6) and Cheyenne (Fig 7) WRF experiences linear strong scaling
- Increasing number of cores will proportionately decrease computation time while the same number of total core-hours will be used for computation

Computation Time Scaling Results

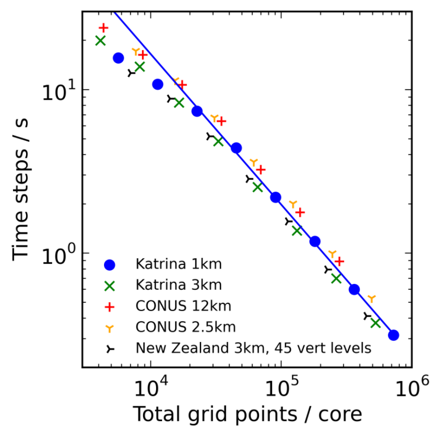


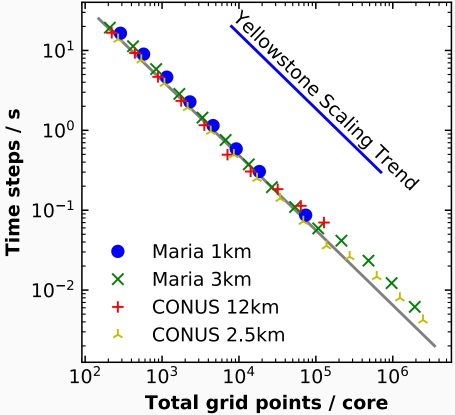


Small number of gridpoints per core region:

- On Yellowstone (Fig 6), WRF departs from linear strong scaling
  - Runs in this region would use more core-hours to run the same simulation than if they had been run on fewer cores
  - MPI communication dominates the actual time spent in computation

Computation Time Scaling Results

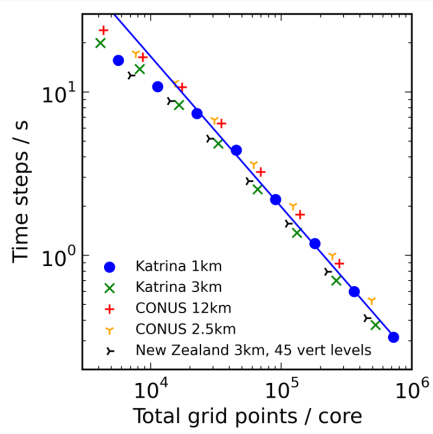


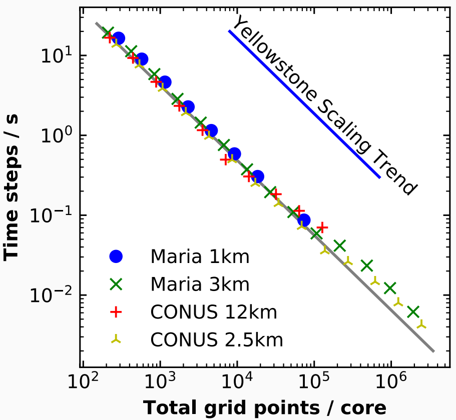


Small number of gridpoints per core region:

- On Cheyenne (Fig 7), WRF doesn't significantly depart from linear strong scaling
  - Likely due to improvements in WRF's MPI code along and a better network interconnect on Cheyenne than Yellowstone

Computation Time Scaling Results

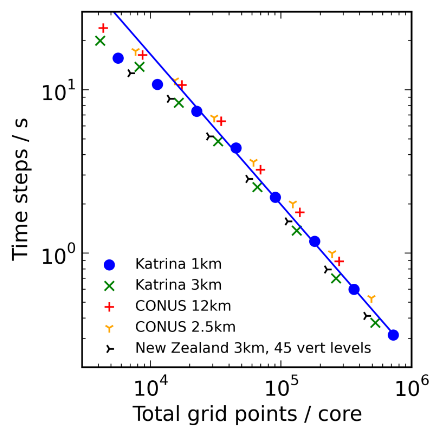


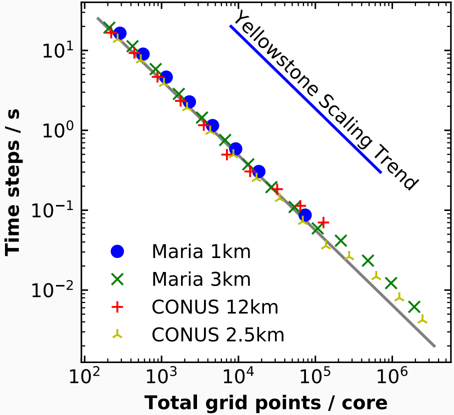


Starting with V4.0, WRF refuses to run with a minimum patch size of less than 10 grid points in either direction

- Prevents users from running with fewer than 100 gridpoints per core where WRF computation would be very MPI bound

Computation Time Scaling Results

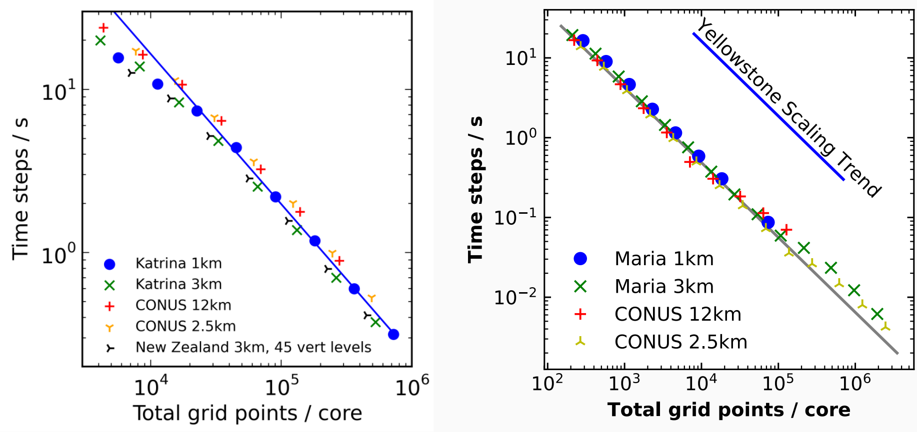




Cheyenne has ~1.78 GB of memory/core which is ~12% less than Yellowstone

- Runs with too many gridpoints/node will run out of memory and be killed
- Typically the max gridpoints/node that will fit into memory the is between 10^5 and 10^6 total gridpoints but it depends on the physics parameterizations

Computation Time Scaling Results



Runs in the very large gridpoints per core region on Cheyenne (Fig 7) used the 128 GB memory nodes and/or undersubscribed the cores on each node

- This causes the small bump in speed observed starting around 10^5 gridpoints/core
- Undersubscribing cores is an inefficient use of a user's core-hour allocation

MVAPICH Scaling

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks
- Tried setting some runtime environment variables:

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks
- Tried setting some runtime environment variables:
  - BIND

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks
- Tried setting some runtime environment variables:
  - BIND
    - `MV2_CPU_BINDING_POLICY=hybrid`

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks
- Tried setting some runtime environment variables:
  - BIND
    - `MV2_CPU_BINDING_POLICY=hybrid`
    - `MV2_HYBRID_BINDING_POLICY=bunch`

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks
- Tried setting some runtime environment variables:
  - BIND
    - `MV2_CPU_BINDING_POLICY=hybrid`
    - `MV2_HYBRID_BINDING_POLICY=bunch`
  - HW

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks
- Tried setting some runtime environment variables:
  - BIND
    - `MV2_CPU_BINDING_POLICY=hybrid`
    - `MV2_HYBRID_BINDING_POLICY=bunch`
  - HW
    - `MV2_USE_MCAST=1`

MVAPICH Runtime Options

- Interested in MVAPICH as a potential default MPI for the next NCAR supercomputing system
- MVAPICH developed for InfiniBand networks
- Tried setting some runtime environment variables:
  - BIND
    - `MV2_CPU_BINDING_POLICY=hybrid`
    - `MV2_HYBRID_BINDING_POLICY=bunch`
  - HW
    - `MV2_USE_MCAST=1`
    - `MV2_ENABLE_SHARP=1`

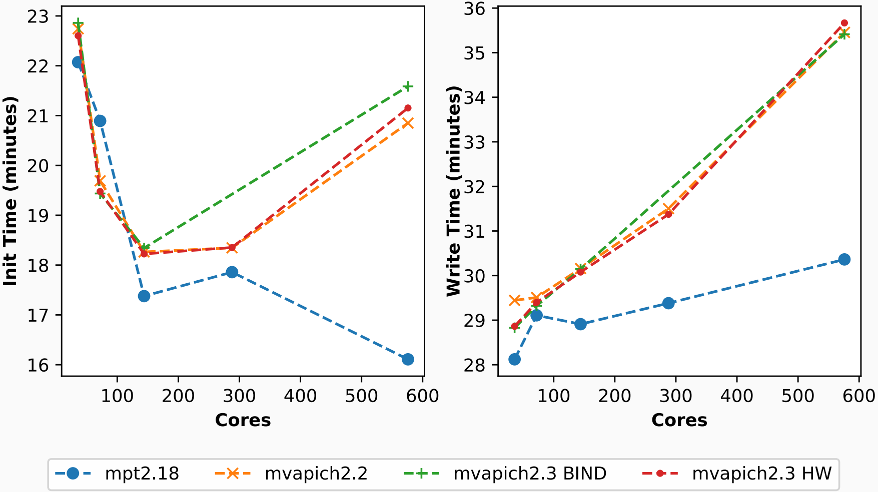


Fig. 8: MVAPICH CONUS 12 km Init and Write Scaling

MVAPICH Scaling Results

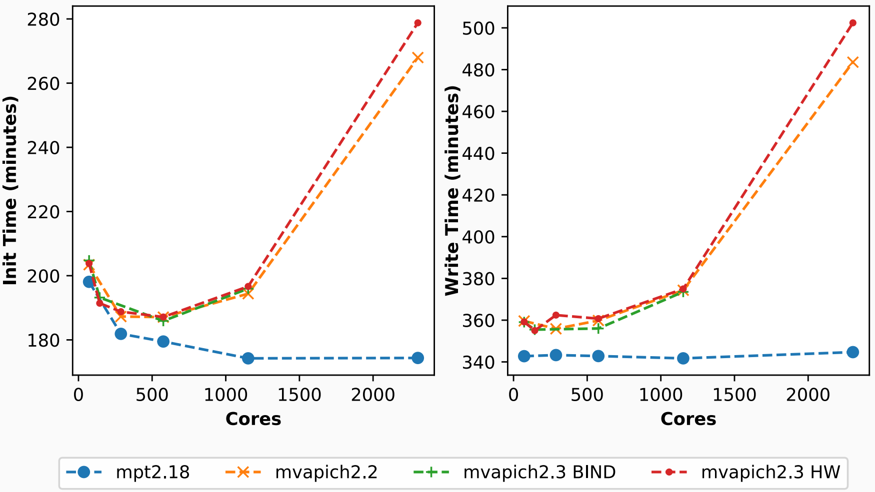


Fig. 9: MVAPICH Maria 3km Init and Write Scaling

Conclusion

Summary

- Intel compiler consistently faster than Gnu compiler

Summary

- Intel compiler consistently faster than Gnu compiler
  - Users should use `-fp-model fast=2` or `-Ofast` for a modest performance increase

Summary

- Intel compiler consistently faster than Gnu compiler
  - Users should use `-fp-model fast=2` or `-Ofast` for a modest performance increase
- MPT, OpenMPI, and MVAPICH show similar performance while Intel MPI and MPICH have poorer performance

Summary

- Intel compiler consistently faster than Gnu compiler
  - Users should use `-fp-model fast=2` or `-Ofast` for a modest performance increase
- MPT, OpenMPI, and MVAPICH show similar performance while Intel MPI and MPICH have poorer performance
- WRF's initialization and writing time show improvements compared to previous results on Yellowstone with a previous WRF version due to better MPI collectives.

Summary

- Intel compiler consistently faster than Gnu compiler
  - Users should use `-fp-model fast=2` or `-Ofast` for a modest performance increase
- MPT, OpenMPI, and MVAPICH show similar performance while Intel MPI and MPICH have poorer performance
- WRF's initialization and writing time show improvements compared to previous results on Yellowstone with a previous WRF version due to better MPI collectives.
- WRF V4.0 scales well across entire run-able region

Summary

- Intel compiler consistently faster than Gnu compiler
  - Users should use `-fp-model fast=2` or `-Ofast` for a modest performance increase
- MPT, OpenMPI, and MVAPICH show similar performance while Intel MPI and MPICH have poorer performance
- WRF's initialization and writing time show improvements compared to previous results on Yellowstone with a previous WRF version due to better MPI collectives.
- WRF V4.0 scales well across entire run-able region
  - Will run out of memory on runs with too many of gridpoints per core

Summary

- Intel compiler consistently faster than Gnu compiler
  - Users should use `-fp-model fast=2` or `-Ofast` for a modest performance increase
- MPT, OpenMPI, and MVAPICH show similar performance while Intel MPI and MPICH have poorer performance
- WRF's initialization and writing time show improvements compared to previous results on Yellowstone with a previous WRF version due to better MPI collectives.
- WRF V4.0 scales well across entire run-able region
  - Will run out of memory on runs with too many of gridpoints per core
  - WRF will prevent runs with too few of gridpoints per core

Acknowledgments

- Mentors
  - Davide Del Vento
  - Brian Vanderwende
  - Alessandro Fanfarillo
  - Negin Sobhani
- Project Partner
  - Dixit Patel
- The SIParCS Program and Admins
  - Rich Loft
  - AJ Lauer
  - Jenna Preston
  - Eliott Foust
  - Valerie Sloan
  - Shilo Hall

Project Git Repository

All the results presented here along with the benchmarking scripts, WRF namelists, analysis code, and more can be found in the git repository for this project:

`https://github.com/akirakyle/WRF_benchmarks`