Benchmarking and Evaluation of the Weather Research and Forecasting (WRF) Model on the Cray XT5

Don Morton, Oralee Nudson, *and* **Craig Stephenson**,
Arctic Region Supercomputing Center, University of Alaska, Fairbanks, Alaska

ABSTRACT: *The Weather Research and Forecasting (WRF) model is utilized extensively at ARSC for operational and research purposes. ARSC has developed an ambitious suite of benchmark cases and, for this work, we present results of scaling evaluations on the Cray XT5 for distributed (MPI) and hybrid (MPI/OpenMP) modes of computation. Additionally, we report on our attempts to run a 1km-resolution case study with over one billion grid points.*

KEYWORDS: XT5, WRF, benchmarking

1. Introduction

The Arctic Region Supercomputing Center (ARSC) has supported the Weather Research and Forecasting (WRF) model for a number of applications since 2005. These applications include quasi-operational real-time forecasts for the Fairbanks Weather Forecast Office of NOAA's National Weather Service, research support for the Alaska Volcano Observatory's volcanic ash prediction model, a wildfire smoke dispersion model, a study of Beaufort Sea wind regimes for petroleum recovery operations, and a broad range of research and education activities.

Given its wide use at ARSC, a benchmarking suite is being constructed for testing WRF's performance on architectures ranging from small, single-CPU systems, novel architectures such as GPU and PS3 systems, to the largest supercomputing systems with tens of thousands of processing elements. This paper describes the benchmarking suite, its initial application on ARSC's Cray XT5, and efforts to push the limits with very large-scale simulations that tax the available computational resources.

2. The benchmarking suite

The benchmarking suite is inspired by John Michalakes' [Michalakes2008] benchmark site, which maintains a collection of ready-to-run input data through the provision of a restart file (a complete dump of the model state at a specified forecast time), lateral boundary condition file, and a namelist (run time parameters) file. With these files, a user with a properly installed version of WRF is able to run the benchmark case to evaluate performance, producing a native WRF output file of all forecast variables at the simulation's end, which can be compared with the benchmark file available on the site. In short, the user simply needs to compile WRF locally, download the pre-processed input data, and run the case study.

We borrowed heavily from this paradigm and adapted to our unique needs and interests in supporting a broad array of architectures with a simulation domain centered on Alaska. Our site [ARSC2009] currently consists of a collection of test cases based on a common domain (Figure 1), but with different grid resolutions (see Table 1). The common domain is a 6075km x 6075km area on a polar projection, centered on Frank Williams' (director of ARSC) office, or as close as model resolution will allow. All cases use 28 vertical levels.

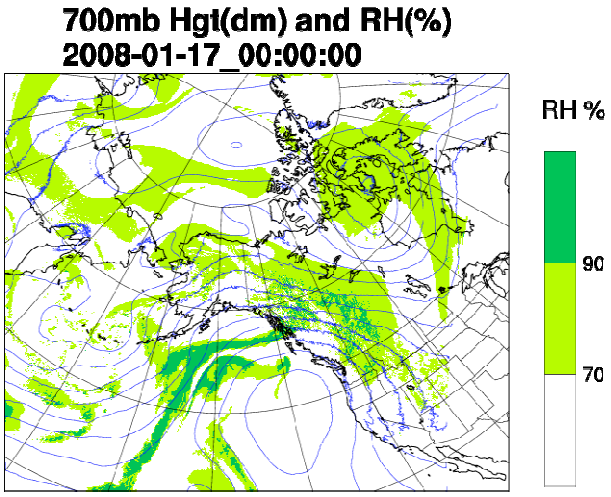

Figure 1. 6075x6075 km benchmark test domain.

Table 1. Various resolutions of the 6075x6075km benchmark suite

Horiz Res	Grid points
81km	75x75x28 = 157,500
27km	225x225x28 = 1,417,500
9km	675x675x28 = 12,757,500
3km	2025x2025x28 = 114,817,500
1km	6075x6075x28 = 1,033,357,500

The benchmark case is derived from a 72-hour forecast for a weather event that surprised Fairbanks residents with an unexpected major snow fall, captured well by the WRF model. To create the benchmark files the weather model is executed for 48 forecast hours (to a point with interesting weather), and a restart file containing a dump of the full state of the model at that time, is created. In addition a file with lateral boundary conditions (LBC's), and a modified namelist file are created, filled with the runtime parameters needed to start the simulation with the restart file and LBC's, and execute for three forecast hours. The benchmark simulation terminates in the production of an output file formatted in WRF-native NetCDF. The intent of producing this output file is to allow for comparison with a "standard" WRF output file from our own simulations, though we caution users to realize that there is no guarantee that our WRF output file has "more correct" values than ones that might be produced at other sites. It merely provides a means to see if the benchmark simulations produce reasonable output. It is envisioned that comparisons will be performed by using a graphics package to display the standard and the benchmark outputs, or the *diffwrf* utility to obtain a numerical comparison of desired fields.

As WRF is executing it prints to *stdout* the wall-time in seconds required for each timestep. At the end of the simulation this list of execution times for each time step can be processed with a Python script that we provide, to accumulate the total time spent over the integration of timesteps, filtering out the time required for initial and terminal I/O and setup operations.

To date, the cases with horizontal grid resolution of 81km, 27km, 9km and 3km have been fully implemented, but the 1km case, with over 1 billion grid points, is problematic. These issues will be discussed later in this paper.

3. Our own preliminary benchmarking

Our preliminary benchmarking activities are meant to provide an overview of performance issues on ARSC systems, stimulating more targeted studies in the future. We made no effort to optimize our compilations; rather, we used default options provided by the WRF distribution. In the following tests, we used WRF V3.0.1.1, compiled with PGI 7.2-3, using the following compile options from the WRF configuration script to produce a distributed memory (MPI) version:

Cray XT CNL/Linux x86_64, PGI compiler with gcc (dmpar)

and a hybrid MPI/OpenMP version:

Cray XT CNL/Linux x86_64, PGI compiler with gcc (dm+sm)

All tests were performed on ARSC's Cray XT5, *pingo*, comprised of 432 compute nodes, each with 32 GB of shared memory, 2 quad core 2.3 GHz AMD Opteron processors, connected through the Cray Seastar 2+ interconnect interface, giving a total of 3,456 compute cores. The ambitious large-scale test cases are supported by a 150 TB Lustre scalable file system.

The first tests performed were an assessment of basic scalability using MPI distributed memory as shown in Figures 3 - 6. In the 81km and 27km cases we assigned MPI tasks sequentially to a node, up to eight tasks (see Figure 2 for XT5 node layout, essentially comprising two processors with four cores each). In the 9km and 3km cases we assigned eight MPI tasks to each node, one task per core.

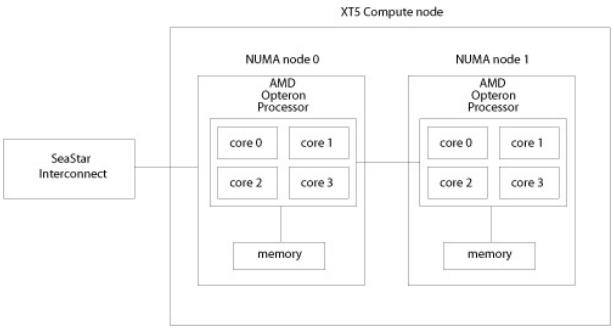

Figure 2. Layout of an XT5 compute node (from *Louhi User's Guide*, © CSC – IT Center for Science Ltd.)

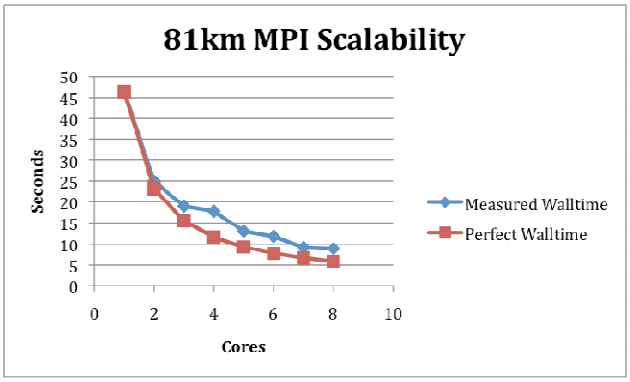

Figure 3. Scalability for 81km test case, 1-8 PEs.

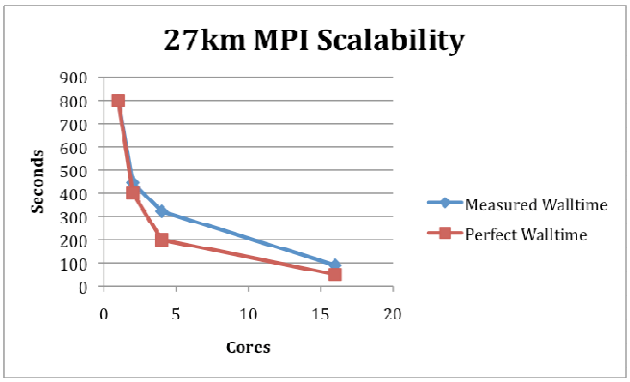

Figure 4. Scalability for 27km test case, 1-16 PEs.

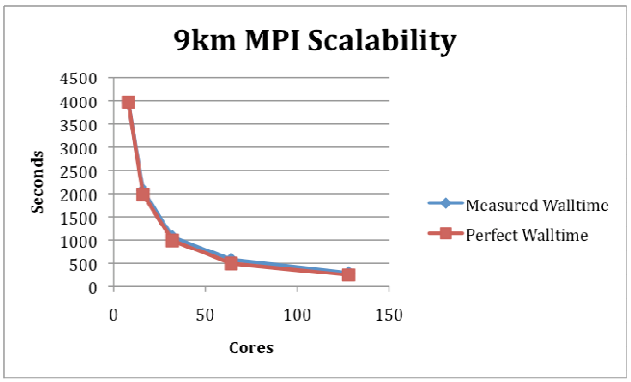

Figure 5. Scalability for 9km test case, 8-128 PEs.

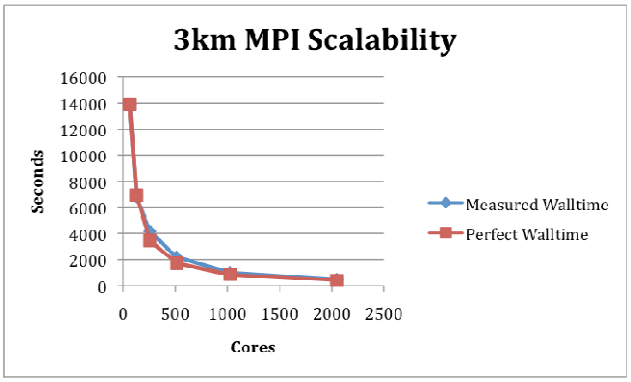

Figure 6. Scalability for 3km test case, 64-2048 PEs.

In the 9km and 3km cases, nodes were always fully loaded with one task per core, and measured speedup was near-optimal, relative to a fully-loaded node base case of 8 PEs. In the 81km and 27km cases, the measured speedup is not nearly as optimal as we load up a node from 1 to 8 tasks. Although some of this degradation might be a result of the fine granularity of computation to communication, it seems likely that scalability within a node will show performance loss. This motivates the next set of tests in which we compare the performance of cases in which nodes are fully populated with eight tasks, against cases where we distribute tasks so that they don't use all of a node's cores.

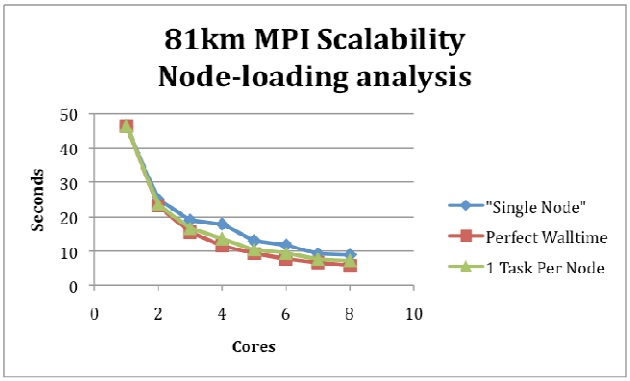

Figure 7. Scalability comparison - 1-8 MPI tasks assigned to a single node versus 1-8 MPI tasks, one task per node.

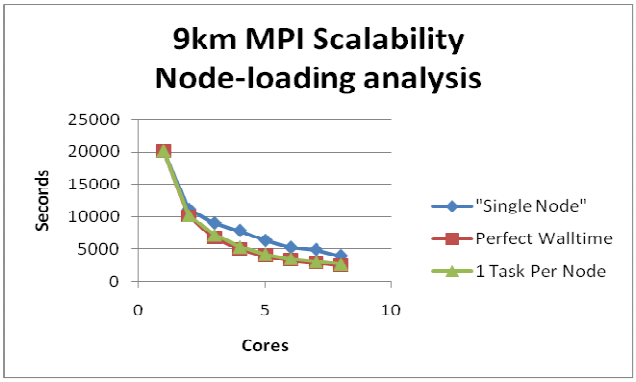

Figure 8. 1-8 MPI tasks assigned to a single node versus 1-8 MPI tasks, one task per node.

In both cases (Figures 7 and 8), it's clear that assigning one MPI task per node (essentially wasting the other seven cores on a node) gives us close-to-optimal speedup.

In Figure 9 we look at this another way, considering eight MPI tasks for the 9km case study, and observing how performance varies as we distribute the tasks 1 per node, 2 per node, and 4 per node. Again, it is clear that we will see our best performance by assigning one task per node, at the cost of wasting the other seven cores on a node. We observe that performance degrades only slightly when increasing the task load from 4 tasks per node to 8 tasks per node, and note this as an area to be further analyzed.

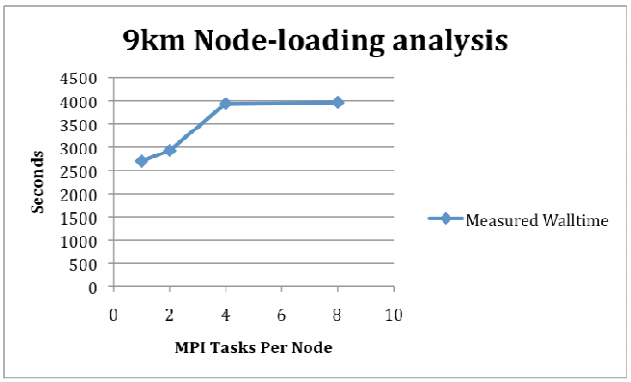

Figure 9. Performance comparison from distribution of WRF MPI tasks 1 per node, 2 per node, 4 per node, and 8 per node.

WRF supports a hybrid distributed memory (MPI) and shared memory (OpenMP) model whereby coarse domain partitioning produces *patches* of grid points for the distributed memory paradigm, and patches may be further decomposed into *tiles* of grid points for threaded shared memory computations. Although some groups have noted slightly better performance with this paradigm, our tests – again, using default compile options – show the distributed memory MPI paradigm to perform better.

The first hybrid test is meant to observe the scalability of threads, assigned to a single MPI task, increasing from one to eight on a single node (Figure 10). The *Perfect Walltime* curve is relative to a single thread and, as in the distributed memory mode we observe that scalability degrades as we load a node's cores.

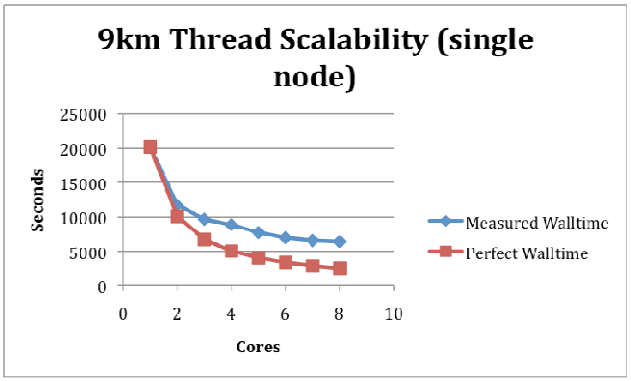

Figure 10. General scalability of 1-8 OpenMP threads (assigned to a single MPI task) on a single node.

Taking this a step further, in Figure 11 we compare thread performance on a single node (1-8 threads running under a single MPI task) to MPI task performance, whereby we assign 1-8 MPI tasks to a single node. In this case, we see that the hybrid and distributed memory modes produce comparable performance for a small

number of cores per node, but as we more fully utilize the cores in a node, MPI shows significantly better performance than the hybrid mode. This is a finding that we discuss in more detail shortly.

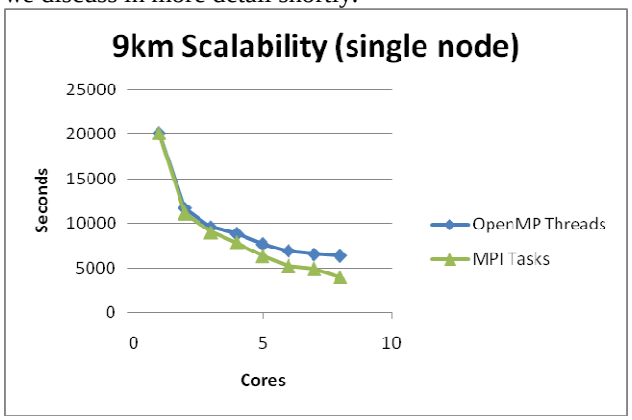

Figure 11. Single node performance of 1-8 OpenMP threads (assigned to a single MPI task) versus 1-8 MPI tasks.

Looking at a more real-world scenario (Figure 12), we compare the hybrid performance with MPI performance on a larger number of fully-loaded nodes. In the hybrid scheme we again allocated one MPI task to each node, and each MPI task managed eight OpenMP threads – one thread per core. In the distributed memory scheme we assigned eight MPI tasks to each node – one task per core.

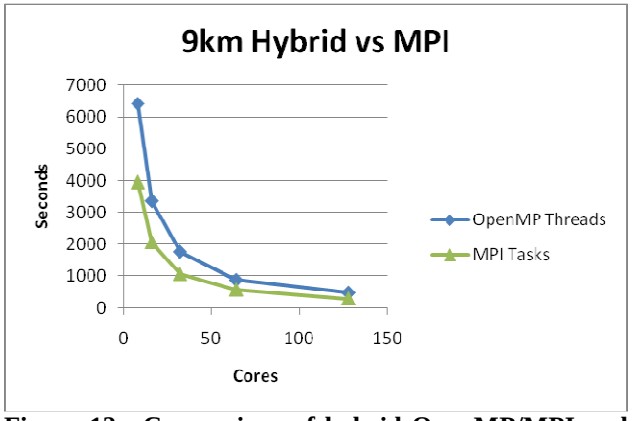

Figure 12. Comparison of hybrid OpenMP/MPI and MPI on 8-128 PEs. OpenMP threads are executed eight to a node, assigned to a single MPI task. In MPI distributed memory mode, MPI tasks are assigned eight to a node.

Although the scaling of the graph suggests a possible improvement of hybrid performance as we increase the number of PEs, the reality is that in all of these cases the performance of the hybrid scheme was worse by almost a factor of two – in the 128 PE case, the hybrid scheme took 490 sec, while the distributed memory scheme took 294 sec.

Finally, we were curious about how different MPI/OpenMP decompositions might affect performance, so we initiated a series of scalability tests comparing the following hybrid scenarios:

- 1 MPI task per node, managing eight threads
- 2 MPI tasks per node, each managing four threads
- 4 MPI tasks per node, each managing two threads
- Pure MPI – eight MPI tasks per node

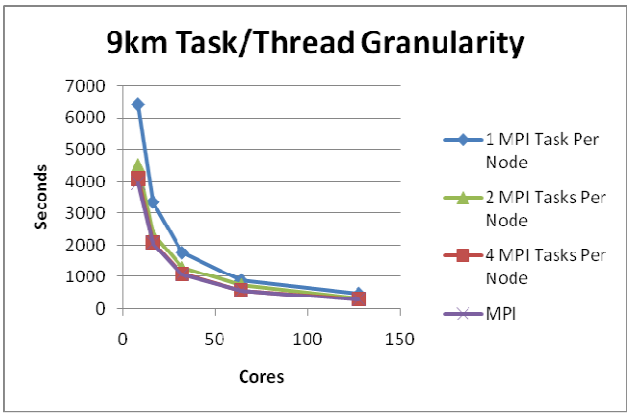

Figure 13. Comparison of different hybrid OpenMP/MPI allocations.

Although it's hard to tell from the graph (Figure 13), hybrid performance improved as we increased the number of MPI tasks on each node, decreasing the number of threads managed by an MPI task. The pure MPI mode still exhibited the best performance, though the case where we used four MPI tasks per node, each managing only two threads, exhibited performance very close to that of the MPI distributed memory mode. We believe this may indicate that MPI processes have a greater affinity for a core, exhibiting much less context switching than a set of OpenMP threads. In short, the use of MPI distributed memory mode has consistently given us better performance than a hybrid approach, but we cautiously note that we have seen other presentations where the hybrid approach was marginally better than MPI (though we are not aware of any similar studies performed on the Cray XT5 with a PGI-compiled WRF).

3. Setting up a large-scale problem

Our original intent was to include a 1km resolution case in the benchmark suite, consisting of over one billion grid points. As we pursued this direction we ran into memory limitations, and consultations with Peter Johnsen

and John Michalakes provided some insight in other ways to go about this.

Table 2. Memory requirements for the billion grid point case.

Tasks	Per Task (GBytes)	Global Buffer (GBytes)
128	4.4	4.1
160	3.5	4.1
256	2.2	4.1
512	1.1	4.1
1024	0.58	4.1

An estimate of memory requirements for this billion point case is outlined in Table 2 (estimates come from computations provided by John Michalakes). Although great effort has gone into making WRF decompose well amongst available processors, there is the necessity of maintaining some global buffer space for necessary scatter/gather operations. This additional memory is assigned to PE0, and in large-scale cases can pollute an otherwise well-balanced decomposition. For example, consider that the XT5 nodes each have 32 GBytes available (some of which is used by the OS). Table 2 suggests that with 160 PEs WRF would use 3.5 GBytes per task, or 28 GBytes per node. However, PE0 (and only PE0) would additionally need to store the global buffer of 4.1 GBytes, so we have to allocate enough cores to fit this additional requirement into PE0. One way around this is to use some clever PBS Pro wizardry and assign PE0 to a node by itself, and then assign the other tasks eight to a node (except for the last node, which will only have seven tasks assigned to it). In this way, PE0 only needs to satisfy the memory requirements of a single task plus the global buffer.

Although we made a number of attempts, we were never able to get this billion grid point case running. Despite numerous attempts, we constantly ran into a roadblock during the pre-processing stage, in which the model was complaining that we were requesting a vertical level higher than was possible given the input data. For those who might be interested, the specific error message was

p_top_requested < grid%p_top_possible from data

Since we were using the same input data that we had used in the other benchmarking cases, our initial suspicion (based on previous experience with fine resolution domains) was that the 1km resolution constituted too much downscaling, and that ultimately our errors were numerical in origin.

To test this hypothesis that our problem was caused by the fine resolution, we went back to the 3km resolution case, which we knew was successful, and increased the horizontal domain from 2025x2025 grid points to 6075x6075, thus giving us a billion grid points at a coarser resolution (with a resulting domain that extended into the southern hemisphere). However, we ran into exactly the same problem that we encountered with the 1km resolution case, so then began to suspect that somehow our data was becoming corrupted, possibly a result of a software defect that manifested itself in large-scale memory situations.

To test the hypothesis that the memory usage (as opposed to high resolution) was somehow causing a problem, we went back to our 1km case, but reduced the number of horizontal grid points from 6075x6075 to 3038x3038, giving us about 250 million grid points. This case set up well for us, strengthening our theory that the sheer size of the problem (in terms of number of grid points) is the real issue that needs to be addressed. This is where we stand now on the implementation of a 1km benchmark case – making this work is a worthy pursuit, given that there are no practical limitations that should be stopping us, and achieving success with this case helps us push the limits of WRF and the Cray XT5.

In the meantime, given the problems we were facing with the 1km case, we decided to create a 2km benchmark case that covered the same region as our other successful cases, using 3038x3038x28 grid points (approximately 250 million). The intent was to produce this benchmark in the same way that we produced the others – run the simulation, generate restart files (complete dump of the state of the model), and provide the user with the restart file, namelist of parameters, and lateral boundary condition file so that they can run a benchmark case with no pre-processing required. Although our simulations proceeded well, our scheme fell apart due to memory allocation issues when we tried to write the restart file.

To gain a feel for the restart file sizes, we note that the restart file for the 9km case is 4.2 GBytes, in size, and about 37.7 GBytes for the 3km case. This file is, by default, written solely by PE0, so it is easy to see that memory limitations may come into play. WRF provides a capability to perform I/O operations on a set of processors separate from the computations, and Peter Johnsen was very helpful in pointing out how to facilitate this in an XT5 / PBS Pro environment (along with use of parallel NetCDF), but we ran into too many roadblocks and temporarily abandoned this path.

However, since the only problem was a failure to generate restart files, we were still able to run the 2km simulation with 250 million grid points just to get a rough feel for scalability. Hence, Figure 14, although not generated from our standard benchmark restart file, depicts the performance (for three forecast hours – just like the standard benchmark cases) of a quarter-billion grid point case, using MPI distributed memory mode

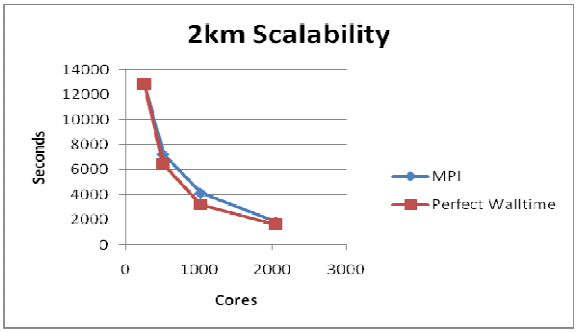

Figure 14. 2km scalability, MPI distributed memory mode.

We additionally ran two hybrid cases and compared with an MPI case, all running on 1024 cores:

- 8 MPI tasks assigned to each of 128 nodes – **4,109 sec.**
- 1 MPI task assigned to each of 128 nodes, each task running 8 threads – **5,511 sec.**
- 1 MPI task assigned to each of 256 nodes, each running 4 threads (in other words, half the cores on the 256 nodes were left idle) – **3,910 sec.**

Summary

Our primary motivation at this point is to create a versatile suite of WRF benchmarks that we hope will facilitate testing the entire spectrum of available architectures, and that this will be useful to the HPCMP community and others seeking to understand WRF performance on new systems. Borrowing heavily from [Michalakes2008] we have created the initial infrastructure [ARSC2009] to support this venture. Our current challenge is to continue pushing the limits by creating a one-billion grid point benchmark, and that is where we will direct our near-term efforts. In pushing the limits on large-scale WRF simulations, we note that Johnsen, Nyberg and Michalakes have successfully run a 2-billion grid point case, so we're not on untrodden ground.

By creating the benchmark suite we had the opportunity to perform some initial testing of ARSC's, Cray XT5, *pingo*, finding that in all cases, use of the distributed memory mode yielded better performance than the hybrid MPI/OpenMP mode. Other groups, however, have found slightly better results with the hybrid approach (albeit on other architectures and with other compilers), so this is an area worthy of further analysis.

Acknowledgments

The authors gratefully acknowledge the support of the Arctic Region Supercomputing Center in its provision of computational resources and top-notch user support. Peter Johnsen of Cray, Inc. and John Michalakes of NCAR provided invaluable guidance and inspiration for this work.

About the Authors

Don Morton has been associated with the Arctic Region Supercomputing Center (ARSC) since 1994, while maintaining full time academic positions in Oklahoma and Montana and he has been a sporadic contributor to CUG meetings (1999 and 2005). Don is currently wrapping up his final semester after twelve years at The University of Montana Missoula for a full-time position at ARSC to finally pursue the research and development activities which he has always been passionate about, particularly in the area of weather modelling.

Craig Stephenson began working at ARSC as an Undergraduate Student Assistant in May of 2004. Following the completion of his B.S. in Computer Science from the University of Alaska Fairbanks, Craig began working full time as a User Consultant for ARSC. In addition to assisting users on a daily basis, Craig's greatest work related contributions include developing a tool for Project Gutenberg which allows users to create a customized cd of Project Gutenberg electronic books, programming the backend job submission and results reporting for the Community Portal for Collaborative Research on Tsunamis, and recently becoming an editor for the popular ARSC HPC Newsletter.

Oralee Nudson's first introduction to parallel computing began as an Undergraduate at Boise State University writing PVM for a twelve processor Beowulf cluster. This experience paved the way for her employment at ARSC as a User Consultant. Oralee has contributed to ARSC by developing helpful system tools for use on the ARSC supercomputers, is a member of the system support and usability teams, and has recently taken on the supervisory responsibilities for the ARSC Undergraduate Research Project Assistants. Oralee will graduate with her M.S. in Computer Science from the

University of Alaska Fairbanks in May 2009 and plans to continue working full time as a User Consultant for ARSC.

References

[ARSC2009] ARSC WRF Benchmarking Suite
http://weather.arsc.edu/BenchmarkSuite/

[Michalakes2008] WRF V3 Parallel Benchmark Page
http://www.mmm.ucar.edu/wrf/WG2/bench/