Geosci. Model Dev., 4, 611–623, 2011
www.geosci-model-dev.net/4/611/2011/
doi:10.5194/gmd-4-611-2011

Geoscientific Model Development

Influence of the compiler on multi-CPU performance of WRFv3

T. Langkamp and J. Böhner

Institute of Geography, University of Hamburg, Germany

Received: 14 February 2011 – Published in Geosci. Model Dev. Discuss.: 21 March 2011
Revised: 27 June 2011 – Accepted: 4 July 2011 – Published: 13 July 2011

Abstract. The Weather Research and Forecasting system version 3 (WRFv3) is an open source and state of the art numerical Regional Climate Model used in climate related sciences. These years the model has been successfully optimized on a wide variety of clustered compute nodes connected with high speed interconnects. This is currently the most used hardware architecture for high-performance computing (Shainer et al., 2009). As such, understanding the influence of hardware like the CPU, its interconnects, or the software on WRFs performance is crucial for saving computing time. This is important because computing time in general is rare, resource intensive, and hence very expensive.

This paper evaluates the influence of different compilers on WRFs performance, which was found to differ up to 26 %. The paper also evaluates the performance of different Message Passing Interface library versions, a software which is needed for multi CPU runs, and of different WRF versions. Both showed no significant influence on the performance for this test case on the used High Performance Cluster (HPC) hardware.

Emphasis is also laid on the applied non-standard method of performance measuring, which was required because of performance fluctuations between identical runs on the used HPC. Those are caused by contention for network resources, a phenomenon examined for many HPCs (Wright et al., 2009).

1 Introduction: the benchmark goal

In numerical weather modelling limited computing capacity is a crucial problem.Especially for climate relevant time spans of 30 yr simulation time (or even more), for high model resolutions, and/or big domains (global) every percent of gained model performance counts. Thus, the performance of the Regional Climate Model (RCM) Weather Research and Forecast (WRF) system is evaluated in this paper. More precisely the benchmark was done for one of the two available numerical solvers of the WRF: the Advanced Research WRF (ARW) version 3.

Since access was restricted to Tornado, a High Performance Cluster (HPC) of the *Deutsche Klima-Rechenzentrum* (DKRZ), an inter-comparison study between different HPC hardware was not possible. Either way such a study would not be as interesting at all, because many hardware-related studies are already available for WRF. Most of them are collected by WRF developer John Michalakes of the National Center for Atmospheric Research (NCAR), and made publicly available on the central web page www.mmm.ucar.edu/wrf/WG2/benchv3. The page is updated at least yearly with new benchmark measurements since 2008.

Instead of such a hardware-related study different software influencing WRFs performance was evaluated. This includes different versions of WRF, the Message Passing Interface (MPI) library, and the compiler. This kind of benchmark yet has been published in conference papers, but not in peer reviewed journals. For example, the search results on the *ISI Web of Science* http://apps.isiknowledge.com on 24 May including the following words in the title were:

- 0 results for "compiler* benchmark* WRF*",
- 16 results for "compiler* benchmark*"

3 of the 16 results were published after the year 2000. But they lack a real-world benchmark situation, which would give an idea on what is left of compiler optimizations and their performance gains on HPCs. Also other databases and search words yield no relevant peer reviewed literature. The non-existence of a journal paper on this matter documents the need and the motivation for such a paper.

Correspondence to: T. Langkamp
(thomas.langkamp@uni-hamburg.de)

Published by Copernicus Publications on behalf of the European Geosciences Union.

But as mentioned, at least some conference papers dealt with a compiler comparison benchmark on WRF. For example Shainer et al. (2009), Roe and Stevens (2010), or Newby and Morton (2008), made benchmarks with a focus on MPI and network issues. Similar work was published lately in two presentations with an additional focus on Intel versus the GNU Compiler Collection (GCC) and Intel versus PGI compilers by the HPC Advisory Council (December 2010) and Sankaran (October 2010) respectively. Both benchmarks were done on HPCs with Intel CPUs, the latter additionally on AMD CPUs.

For this paper a benchmark suite was implemented on the Tornado HPC which operates AMD CPUs of another series, thus the results allow a relative comparison with the results of Sankaran and the HPC Advisory Council only. As an extension to their presentations this paper compares three major compilers instead of two on a single machine running WRFv3, and documents the whole implementation process in such a detailed way that other users should be able to conduct comparable benchmarks on their systems.

1.1 Introduction: important software influencing WRFs performance

The compiler is the most important performance-related software for any program which the user can compile. The compilation process translates the source code written in a programming language like C or Fortran into a machine-readable binary format. This process enables the performance optimization of the software for different hardware.

In case of software that can take advantage of HPCs due to the time-parallel use of many CPUs (like WRF) the MPI library is a second important software influencing WRFs performance. MPI is responsible for the efficient communication between the CPUs, which often is a bottleneck due to a limited transfer bandwidth of their connection. Hence, four different MPI versions available on Tornado were compared in this performance evaluation.

The third important software is WRF itself. It is not always necessary to use the most recent version in regard of new features or model accuracy because they may not be needed in specific research questions. In this case one should always just use the fastest version. One can expect older, less complex versions to perform better, however, one may also expect newer versions to perform better because of source code optimizations. Therefore the two ARW versions 3.1.1 and 3.2.1 were evaluated.

The fourth parameter influencing the performance is set in the configuration file configure.wrf, which will be generated right before the compilation. One can set in the configure.wrf the option to compile WRF for single or multi-CPU hardware. Since practical application cases of WRF running on a single-CPU are rare – especially if an HPC is available – only a multi-CPU option was of interest. WRF offers three of them: The first multi-CPU option is named smpar, working with an OpenMP shared memory thread paradigm, not to be confused with MPI. Second option is dmpar and works with an MPI task distributed memory paradigm. The third approach is a hybrid one, that combines MPI and OpenMP, where each MPI task spawns a number of threads to utilize shared memory in a node. But as already shown by Morton et al. (2009), by the HPC Advisory Council (December 2010) and others for WRF the dmpar case often outperforms the hybrid and the smpar cases clearly. However, referring to Morton et al. (2009) there were groups that found opposing results on other architectures and with other compilers, but only with “slightly” and not clearly better performance of the hybrid approach. So no emphasis is laid on hybrid versus dmpar comparison within this paper.

In a nutshell, this software benchmark suite will consist of an inter-comparison of compiler, MPI library, and WRF version influencing WRFs performance. However, the focus is on the compiler as the most important tool in performance optimization of software.

1.2 Introduction: about the WRF model

WRF is a mesoscale numerical weather prediction system and also a regional to global (experimental stage) climate model allowing simulations reflecting either real data or idealized configurations. It features a 3-dimensional variational data assimilation system, and an open source software architecture allowing for computational parallelism and system extensibility due to the modularity of its components (see Fig. 1).

WRF can be run with two dynamical cores, the Nonhydrostatic Mesoscale Model (NMM) and ARW. ARW-only features are regional climate and seasonal time-scale research, coupled-chemistry applications, global and idealized simulations – while both NMM and ARW are suitable to do research on atmospheric physics/parameterizations, case-studies, real-time NWPs and forecast systems, and data assimilation (Dudhia, 2011).

Both are freely available via the same source code packages at www.mmm.ucar.edu/wrf/src for versions 2.0 up to 3.3. Precompiled binaries are only available for version 3.1 and x86 CPUs via Robert Rozumalski of the US National Weather Service at http://strc.comet.ucar.edu/wrfems/index.htm.

WRFs development is organized and promoted by the National Oceanic and Atmospheric Administration consisting of the National Centres for Environmental Prediction (NCEP) and the Forecast Systems Laboratory, by the Air Force Weather Agency (AFWA), the Naval Research Laboratory, the University of Oklahoma, the Federal Aviation Administration, and NCAR. WRFs advances in physics, numerics, and data assimilation are contributions by a broad and rapidly growing research community. It is in operational use at NCEP, AFWA and other centres (WRF Homepage, 2011). Technical and physical details can be found in the

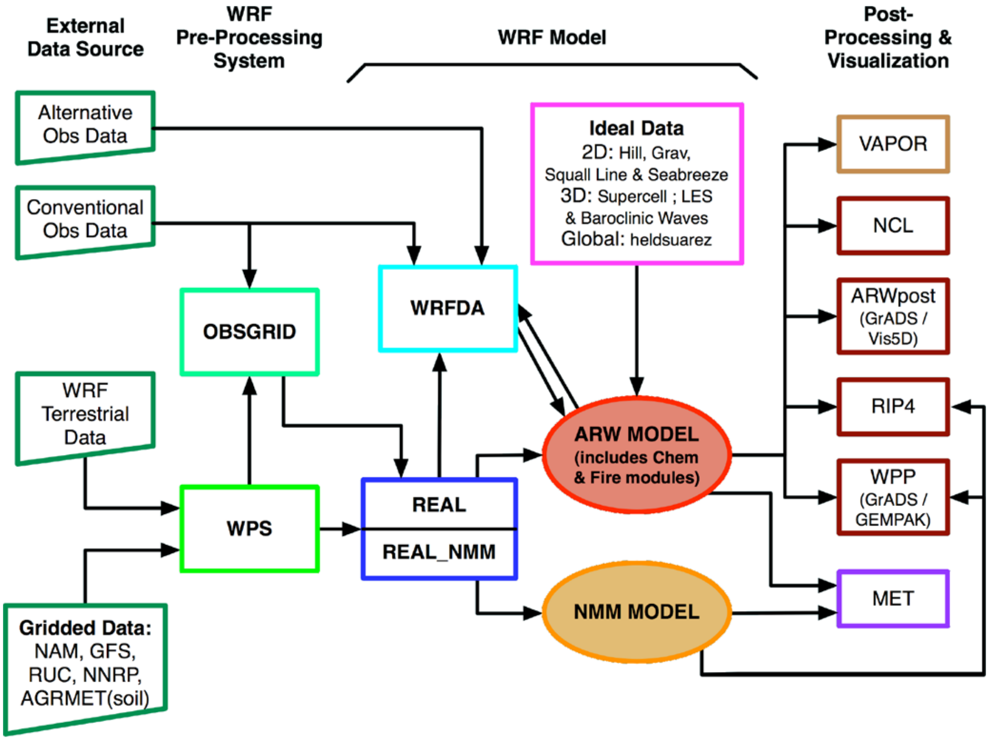
External Data Source
WRF Pre-Processing System
WRF Model
Post-Processing & Visualization
Alternative Obs Data
Conventional Obs Data
WRF Terrestrial Data
Gridded Data: NAM, GFS, RUC, NNRP, AGRMET(soil)
OBSGRID
WRFDA
WPS
REAL
REAL_NMM
Ideal Data
2D: Hill, Grav, Squall Line & Seabreeze
3D: Supercell ; LES & Baroclinic Waves
Global: heldsuarez
ARW MODEL (includes Chem & Fire modules)
NMM MODEL
VAPOR
NCL
ARWpost (GrADS / Vis5D)
RIP4
WPP (GrADS / GEMPAK)
MET

Fig. 1. A schematic of the used model environment. Mainly the WRF Modelling System consists of the WRF Preprocessing System (WPS) and the physical numerical core Advanced Research WRF (ARW). The NMM core is an alternative used in operational weather forecast (from Wang et al., 2011).

user guide by Wang et al. (2011) and in the technical physical documentation by Skamarock et al. (2008).

2 The benchmark setup

After describing the benchmark goal, the environmental software components, and the model to benchmark in the previous section, this section will document the detailed setup of the benchmark suite, consisting of specifications of hardware, environmental software, model software, and model domain.

2.1 Hardware: specifications of Tornado

The Tornado HPC of the DKRZ is a 2048 core Linux cluster. Core describes a subunit of today's multi-core CPUs, which is capable of processing one task at a time each. The 256 compute servers of Tornado consist of two quad-core CPUs each (model AMD Opteron 2384, 2.7 GHz), including 4 gigabyte RAM per core. Furthermore a single of the 256 compute servers is referred to as a node (model Sun X2200 M2). This multiplies up to 4 cores times 2 CPUs times 256 nodes equals 2048 cores; and 4 gigabyte RAM times 8 cores times 256 nodes equals 8 terabyte RAM.

The nodes are interconnected via Gigabit Ethernet and a low latency Infiniband DDR network. More details like on storage and login nodes – which are in this case non-crucial components concerning performance – see Fig. 2 and https://tornado-wiki.dkrz.de/farm/HardwareOverview. (Note: In other scenarios with e.g. heavy write loads the storage nodes might even be more important than the compute nodes.)

2.2 Environmental software: operating system, grid engine, MPI, and compiler

The operating system of Tornado is Debian GNU/Linux compiled with Kernel 2.6.16.60–0.31 from March 2006. Security updates and bugfixes are as current as January 2008. The performance of WRF is definitely influenced by the old Kernel version and its' included hardware-drivers. Phoronix.com (2010), the biggest linux benchmark site, tested the linux-kernels of the last 5 yr with a broad benchmark-suite. E.g. one of the benchmarks (NASA's NAS Parallel Benchmarks – IS.C test) evaluates the performance of parallel supercomputers. Phoronix shows a 4 % performance degradation of 2.6.16 over the recent 2.6.3x versions and also discusses the numerous kernel-changes which might explain those. The other benchmarks differ more or less or even not at all. However, because it is considered a running system the administrators will not touch the kernel, thus no version-comparison was possible to quantifiy the

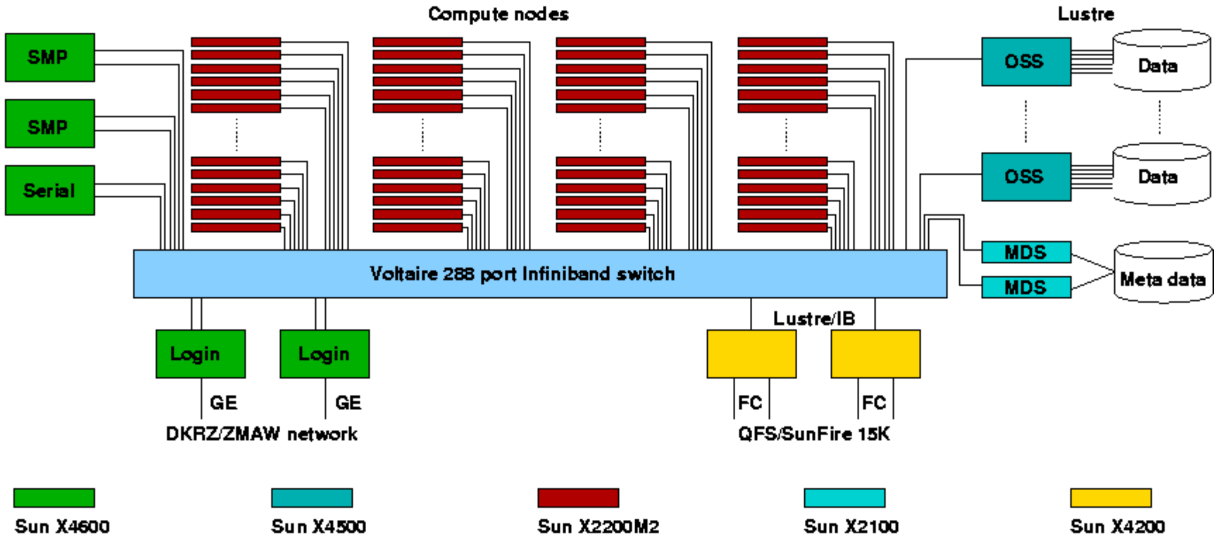

Fig. 2. A schematic of the used cluster hardware also known as "Tornado" of the *Deutsche Klima Rechenzentrum* (from the Tornado Wiki Hardware Overview).

probably negative influence on WRF. The installed Sun Grid Engine (SGE) software version 12.1 is responsible for submission and scheduling of jobs from Tornado's login nodes to its compute node queue. After a job submission to the queue MPI starts distributing the job to the requested amount of cores.

MPI libraries are available from different proprietary software vendors like Intel. The preinstalled MPI on Tornado is an open source implementation of MPI, named Open MPI. The installation and benchmarking of another MPI, even of only another version of Open MPI was successful for Open MPI 1.5.1, but worked only for 8 cores. Altogether the Open MPI versions 1.3.3, 1.4.0, 1.4.3, and 1.5.1 were evaluated. (Note: Open MPI should not in any way be confused with OpenMP needed for smpar shared memory runs, see Sect. 1.1)

The most straightforward compiler for WRF is the GCC. It consists of the C compiler gcc, the Fortran compiler Gfortran, and several more. Gfortran and gcc are both needed, because WRFs framework connecting all parts of WRF is written in C while all physical modules are written in Fortran. However, only the latter significantly influences the application's performance (see Sect. 2.3).

Besides the open source GCC, proprietary Fortran and C compilers of the manufacturers Intel and PGI were used. The Sun compiler came without a proper Open MPI installation. (Note: Open MPI should be compiled with the same compiler as WRF to avoid complications; Until version 11 the Intel compilers were free of charge for non-commercial use, since the actual version 12 they are chargeable as those from PGI.)

There are more compilers supported by WRF (see Wang et al., 2011), but they are not compatible with Tornado's hardware, with exception of the PathScale compiler, which was not preinstalled. It would be worth testing it in future research.

2.3 Model software: compiling of netCDF, WRF and WPS

An important prerequisite to compile WRF is an installation of the Network Common Data Format (netCDF) used as input and output format by WRF. Further prerequisites for netCDF and WRF are numerous and usually shipped with the Linux Distribution. Only the most important packages shall be noted here: zlib, perl, a shell like bash or csh, make, M4.

A specialized version of netCDF that affects the performance in combination with large domains is parallel netCDF (pNetCDF, http://trac.mcs.anl.gov/projects/parallel-netcdf). As discussed in Morton et al. (2010), in its most common usage WRF decomposes its domain in a number of patches or tasks equal to the number of used cores. In this mode all tasks will hold a roughly equal sized sub-domain, and "Task 0 will have the additional responsibility to perform I/O operations and coordinate scatter/gather operations" (Morton et al., 2010, p. 4). This works until domain sizes of several hundred million cells. Because a much smaller domain was used in this case pNetCDF or one of its alternatives also described in Morton et al. (2010) were not needed. As a consequence the common netCDF version 4.01 was used and compiled – once for every compiler manufacturer to avoid software conflicts. The preinstalled Open MPI actually was available in versions for every different main release of a compiler to avoid software conflicts. (Note: to compile with a certain combination of compiler, MPI and netCDF on a system where many of those are available, one first manually has to set all the environmental paths to their installation directories. It is also wise to compile netCDF with "large file support" to access bigger than 2 GB files. A detailed online tutorial on those prerequisites and finally the compiling of WRF can be found at www.mmm.ucar.edu/wrf/OnLineTutorial/index.htm.)

The two WRF ARW versions 3.1.1 and 3.2.1 had to be compiled with special MPI- and compiler-options. Those are specific to this case and therefore not documented within the WRF user's guide (Wang et al., 2011) or in the online tutorial just mentioned. The general compile-options, also known as flags, can be altered in the `configure.wrf`, just like the option that tells WRF to compile for single- or multi-CPU usage (see Sect. 1.1). In most cases all flags are automatically chosen correctly by the configure script. But because Open MPI did not recognize the flags `-f90=$(SFC)` and `-cc=$(SCC)`, those had to be erased from the lines `DM_FC` and `DM_CC`, on the one hand. On the other hand, the flag `-DMPI2_SUPPORT` had to be added to the line `DM_CC`. Additionally the flag `-ip` had to be erased from the line `CFLAGS_LOCAL` for versions using an Intel compiler. For PGI and GCC compilers the auto-generated `configure.wrf` was sufficient (see Appendix A).

There are also compiler-flags that influence the performance of WRF-runs, but only for WRFs Fortran coded part. This was verified by turning off all optimizations for the C code compilation in the `configure.wrf` line `CFLAGS_LOCAL` by setting `-O3` to `-O0`. As expected this did not change the benchmark results. The Fortran code instead is considerably faster with full optimizations (e.g. `-O3`) or flags like -qhot, but this might crash the model while running or compiling (WRF Users Page 2008). In most cases sticking to `-O2` is sufficient, only one user reported in the WRF User's Forum (2009) that he had to fall back to `-O0` to compile WRF.

Further performance-flags were altered within this paper to compare GCC with the unoptimized PGI. Thus the five PGI optimization flags `fastsse`, `Mvect=noaltcode`, `Msmartalloc`, `Mprefetch=distance:8`, and `Mfprelaxed` were deleted consecutively from `configure.wrf`, line `FCOPTIM`, before the compilation. This left six differently optimized WRF/PGI configurations (full to no optimization).

There are also non value-safe optimizations that lead to differing model outputs. Those are e.g. adding the flag `-IPF_fp_relaxed` (Cisneros et al., 2008; Semenov et al., 2005), or removing the `-fp-model precise` flag from the line `FCBASEOPTS` when using Intel compilers (for reference and explanatory notes see the Intel-note in your `configure.wrf`). Both optimizations were not considered worth testing. They should be used careful, only after comparing the differences between a representative model-output of the value-safe compiled model and the non value-safe compiled one. If the errors are not relevant for ones research target, one can use them.

2.4 Model domain: size, resolution, decomposition, and duration

WRF developer John Michalakes (2010) offers two standard benchmark domains for WRF version 3.0 at www.mmm.ucar.edu/wrf/WG2/benchv3, named CONUS, with resolutions of 2.5 and 12 km. It is crucial to run those benchmark cases exactly according to the instructions on that page if one wants to compare different HPC hardware. Since CONUS lays no emphasis on benchmarking software like compilers, it was decided to evaluate the more up to date WRF versions 3.1.1 and 3.2.1 with an easily available domain of a size somewhat between CONUS 2.5 and 12 km. Therefore the already available Default January 2000 case[1] of the WRF online tutorial case studies[2] was used. Only little modifications were made solely to resolution and duration via the namelist configuration files (see Appendix B). This has two advantages. The first one is that other users can implement the same benchmark easily by following the detailed instructions and download locations of the online tutorial case studies. The second advantage is that in consequence there is no new documentation needed.

The Default January 2000 case is a winter storm of 24 to 25 January over the east coast of North America with a horizontal resolution of 30 km and 28 vertical levels. At this resolution the domain comprises $74 \cdot 61 \cdot 28 = 126\,392$ grid points. On Tornado this relatively small grid turned out to slow down the computation when using more than 8 cores. This can be explained due to Tornado's limitations in network resources or the so called computations to communications ratio, which was to small for more than 8 cores. To elaborate, the computations to communications ratio describes the phenomenon, that the more cores share the computation of a domain, the more they have to communicate about the physical fluxes between the sub-domains: "Computation is related to the local domain's volume, and communication is related to the local domain's boundary area" (Salapura et al., 2006).

Even though Tornado's nodes have low latency Infiniband interconnects, the time to communicate those processes is longer than the time needed to process the sub-domain. (Note: the intra-node communication between the 8 cores of a node is much faster than the inter-node communication using Infiniband and thus represents no bottleneck.)

The problem with the computations to communications ratio can be circumvented by raising the horizontal resolution of the Default January 2000 case from 30.0 km to 3.333 km or 10 237 752 grid points, and lowering the time step accordingly to 20 s. Consequently, computation of the resulting domain scaled up to 32 cores, but 64 cores were sometimes as slow as 8 cores. This performance hit may be unique to Tornado. According to the administrators the cause may be

[1] www.mmm.ucar.edu/wrf_tmp/WRF_OnLineTutorial/SOURCE_DATA/JAN00.tar.gz

[2] www.mmm.ucar.edu/wrf/OnLineTutorial/CASES/index.html

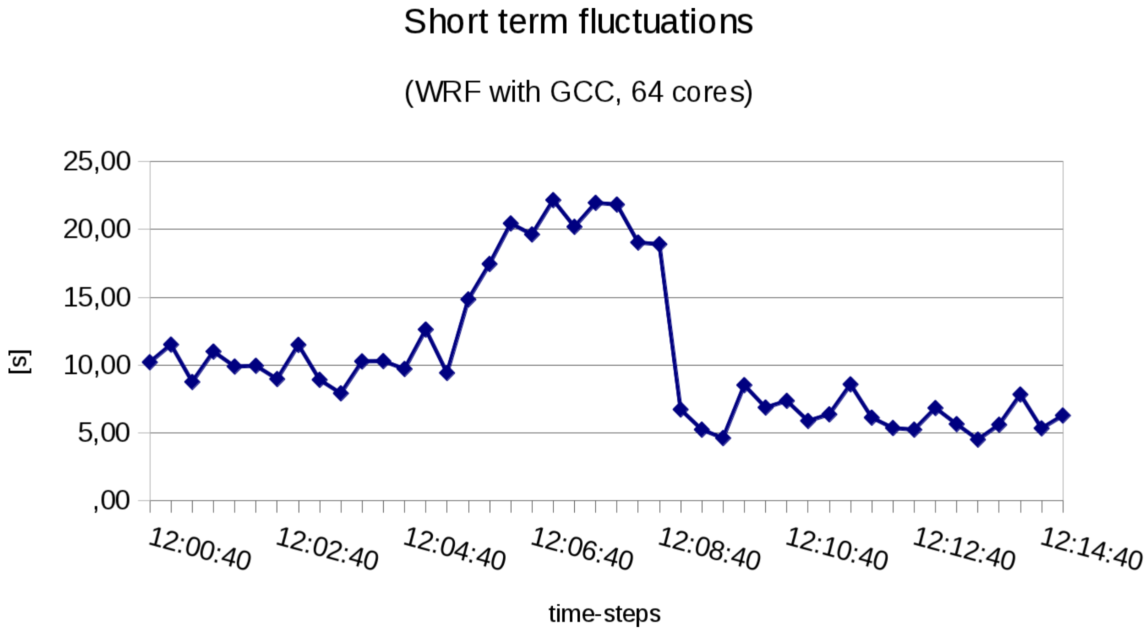

Fig. 3. Short term fluctuations of a WRF run at 64 cores with GCC. One clearly can see the short term fluctuations due to overall high network load on Tornado where the time needed for the computation of one time step is 22.162 s maximum compared to 4.508 s minimum what equals a span of 492 % where no span should exist at all.

that at high network loads Tornado falsely switches the use of the lower latency Infiniband to the higher latency Gigabit Ethernet. A new MPI version did not fix the problem. Thus, the problem was circumvented by evaluating results from 32 cores at most.

An optional tuning parameter within the `namelist.input` – not dependent on environmental software – is `numtile`. Like elaborated in Sect. 2.3, WRF decomposes the domain into tasks or patches each assigned to a core via a MPI process. Each patch can be further decomposed into tiles that are processed separately, but by default there is only one tile (`numtile` not set or = 1). If the single tile is too large to fit into the cache of the CPU and/or core it slows down computation due to WRF's memory bandwidth sensitivity (Roman, 2009). In order to reduce the size of the tiles, it is possible to increase their number via `numtile` = x (see Appendix B). However the optimal value x heavily depends on the cache sizes of core and CPU, the number of assigned cores, and the size of the domain. Thus there is no other way than experimentation to find what value gives the best performance. For reference purposes, the best values found here were 64/32/16 for 8/16/32 cores, respectively. This lead to a speed up of 22 % for 8 cores rising to 26 % for 32 cores. Altogether, a benchmark setup should not be expensive, speaking of workload for the user and computing time for the CPU. By minimizing the duration of a run instead of the default 12 h only 15 min of model time were computed equal to 45 time steps, 20 s each. Further, an intelligent matrix of compiler/WRF/MPI-combinations instead of testing every possible combination (see Table 1 and Sect. 4) leads to a small test sample with a maximum of explanatory power.

3 Method of performance measurement

In most model benchmarks the performance metric is the average time per model time step over a representative period of model integration, ignoring the additional time needed for the model initialization. Alternatively the metric is just the total runtime including initialization averaged over three or more runs (Note: initialization of WRF needs longer the more cores used).

Here instead, the performance metric used is the minimum computing time needed for one of the 45 model time steps. This was done because of large performance fluctuations on Tornado between time steps within and across identical runs like others have encountered before (Wright et al., 2009). They sum up to large differences in computation time for whole runs, as for averaged model time step performance.

Those fluctuations are supposed to be caused also by the contention for network resources. But here they are not limited in regard to many cores communicating about the subdomains of a single job, what resulted in fluctuations for core counts >32. Here they were limited in regard to many users computing many jobs especially during rush-hours creating short term performance fluctuations even for runs using only 2 nodes. Ideally a HPCs network should be able to handle parallel computation on all its nodes without performance

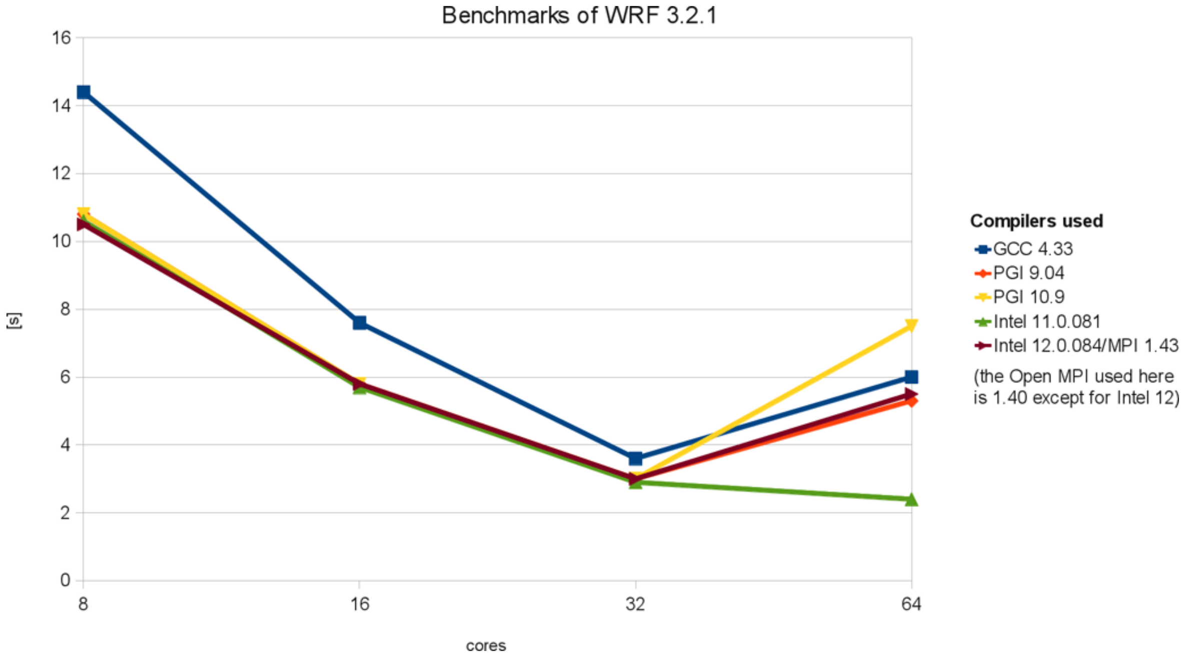

Fig. 4. Data of Table 1, but for Open MPI 1.40/1.43 only. All results are minimal seconds needed for one of 45 model time steps of the Default January 2000 case at 3.33 km horizontal resolution. The runs with 64 cores are strongly varying and were sometimes not reproducible (>5 runs).

degradation. But in practice the processor within the Infiniband network switch, handling the communication requests, often is the bottleneck. Thus the network load due to users leads to random performance fluctuations of up to 500 % for 64 cores (see Fig. 3), decreasing with less cores, due to increasing computations to communications.

The only non-random picture of the fluctuations was that especially before lunch-break and at Friday afternoons a rush-hour of job-computations appeared, where the fluctuation reached its peak.

Hence, on the one hand, with the random fluctuations in mind, measuring the average time over many time steps or whole runs would mask the optimal performance gain. This optimal gain is possible with the optimal compiler on HPCs, where the network is not the limiting factor, as shown by Shainer et al. (2009) for 192 cores Or Newby et al. (2008) for 16 384 cores, with WRFs' performance scaling up almost linear. Salapura et al. (2006) used 2048 cores with an efficiency "a little higher than 50 %".

On the other hand, regarding the rush-hour fluctuations, one must run the model several times spread over a day, first to detect if and when rush-hours appear on the system, and second, to sort out rush-hour influenced measurements. As conclusion model runs with an identical setup (of core count, compiler, MPI, and WRF) had to be repeated at least twice on different days with some distance to the rush-hours. They had to be repeated more often, if the minimum time needed for one time step was not reproducible.

Section 4 consequently shows only the reproducible minimum time needed for a time step. This method overcomes the influence of the hardware (Infiniband switch) and shows only the possible performance gain through the software (compiler, WRF, MPI) as it was intended by this work in the first place.

4 Results and conclusions

The benchmark results for WRF 3.2.1 are shown in Table 1 and Fig. 4. The main finding is that the Intel compilers gain up to 26 % performance compared to GCC on Tornado, an AMD CPU (model Opteron 2384) based system. In a consequence, testing and then choosing the right compiler before doing a big run is absolutely worth the work. While this trend will be valid for different hardware, the percentage will vary however. For example, if a HPC is equipped with Intel instead of AMD CPUs the Intel compilers are likely to gain even more performance compared to GCC. This was shown by the HPC Advisory Council (2010), which found up to 92 % performance gain for Intel 12.0 versus GCC 4.4 on their Intel CPU (model Xeon 5670) based system; or by Sankaran (2010), who found up to 37 % performance gain for PGI 9 versus Intel 11.1 on an Intel CPU (Xeon 5500-series) based system and 34 % on a AMD CPU (Opteron 2400-series) based system. But if a newer GCC version (current 4.5.1) is available on your HPC, this already might reduce the performance gap a little bit. In the long run, even

Table 1. Results for WRF 3.2.1. All results are minimal seconds needed for one of 45 model time steps of the Default January 2000 case at 3.33 km horizontal resolution. The strongly varying and sometimes not reproducible (>5 runs) results of runs with 64 cores are marked *. (To compare the results with other benchmarks like CONUS one may calculate the "simulation speed" (Michalakes, 2010). As example for the Intel 12 compiler at 8 cores it is 20 s/10.5 s = 1.9, where 20 is the always identical simulation time step.)

Compiler	GCC	PGI		Intel	
Version	4.33	9.04	10.9	11.0.081	12.0.084 (2011)
8 cores					
Open MPI 1.33				10.6	
Open MPI 1.40	14.4	10.8	10.8	10.6	
Open MPI 1.43 (1.51)	14.6 (14.3)				10.5
16 cores					
Open MPI 1.33				5.7	
Open MPI 1.40	7.6	5.8	5.8	5.7	
Open MPI 1.43	7.8				5.8
32 cores					
Open MPI 1.33				3.2	
Open MPI 1.40	3.6	3	3	2.9	
Open MPI 1.43	4				3
64 cores					
Open MPI 1.33				4.4*	
Open MPI 1.40	6*	5.3*	7.5*	2.4*	
Open MPI 1.43	4.5*				5.5*

the trend shown by the results in Table 1 may change with the advancements of hard- and software. Therefore it should be considered testing available software combinations influencing WRFs performance before submitting the main job, if no comparable benchmark result not older than one or two years is available.

Another major result is that there is almost no performance difference between the costly Intel and PGI compilers on this platform, or between different compiler release versions of the same manufacturer. There is also almost no difference between the different MPI release versions. (Note: while the benchmarking with Open MPI 1.3.3, 1.4.0, 1.4.3 worked seamlessly, the beta version 1.5.1 was able to compile and run WRF only with GCC and worked only on a single node.)

Furthermore the gap between GCC and Intel/PGI decreases with increasing node numbers down to 20 % for four nodes or 32 cores on this system. The cause might again be the decreasing computation to communication ratio as explained in Sect. 2.4. (With more cores there is more communication needed at the same amount of computation, thus the network limitations drain the potential performance gain.) Table 2 shows the results for WRF 3.1.1. A maximum performance gain over WRF 3.2.1 of 2.7 % was found with PGI 9.04 on two nodes, what almost lies within the range of measuring accuracy[3]. It is unlikely that the performance gap would increase with other compiler/MPI-combinations, thus no further tests in case of different compilers for WRF 3.1.1 were done. But Table 2 additionally shows results between WRF 3.1.1 versions compiled with different PGI compiler flags, set within the `configure.wrf`. This shows which flags are responsible for which percentage of performance gain. Asked to the cause of the performance discrepancy between GCC and PGI compilers Mathew Colgrove of PGI wrote (e-mail of 12 January 2011) "It's possible that it's our auto-vectorizer (SSE) but more likely a combination of many optimizations".

PGI and Intel are able to better optimize their compilers because they are reduced to function on the x86 hardware platform. GCC instead aims to support a broad range of hardware and operating systems and thus has to focus more on compatibility as on performance optimizations (GCC platforms, 2011).

To compare GCC with PGI without optimizations, the five PGI optimization flags `fastsse`, `Mvect=noaltcode`, `Msmartalloc`, `Mprefetch=distance:8`, and

[3] The main changes between WRF 3.2.1 and 3.1.1 which may have caused this performance gap are listed here: www.mmm.ucar.edu/wrf/users/wrfv3.2/updates-3.2.html.

Table 2. Results for WRF 3.1.1. All results are minimal seconds needed for one of 45 model time steps of the Default January 2000 case at 3.33 km horizontal resolution. The strongly varying and sometimes not reproducible (>5 runs) results of runs with 64 cores are marked *.

Compiler	GCC	PGI	(Fortran optimazation flags turned off)	
Version	4.33	9.04	9.04 (fastsse)	9.04 (fastsse+Mvect)
8 cores Open MPI 1.40	14.2	10.6	12.7	17.5
16 cores Open MPI 1.40	7.4	5.7	6.6	9
32 cores Open MPI 1.40	3.6	3	3.4	4.7
64 cores Open MPI 1.40	3.3*	3.8*	2.7*	5.4*

`Mfprelaxed` were deleted consecutively from `configure.wrf`, line `FCOPTIM`, before the compilation. This left six differently optimized WRF/PGI configurations (full to no optimization), but of which only three left different results (see Table 2). The flags `fastsse` and `Mvect` were responsible for the biggest performance gain, and without them PGI is slower than GCC. The other flags had no effect. For information on them see reference PGI flags (2011). The flag `fastsse` enables the use of special instruction sets common in today's x86-CPUs of AMD and Intel and adds 17 % speed up at 8 cores. The flag `Mvect` instructs the vectorizer to generate alternate code for vectorized loops when appropriate (PGI flags, 2011). This feature does not work with WRF since by default it is set to `noaltcode`, what adds performance. To verify this Mvect was set to altcode, what resulted in a performance degradation of 27 %. Since it is not the focus of this paper to deal with the numerous other non default and non value-safe optimizations of GCC, Intel and PGI, those benchmarks were left out here.

Another further noteworthy result of the whole work was the detection of three different performance fluctuations, discussed in Sects. 2.4 and 3. The first fluctuation appeared with a too small computations to communications ratio, suspected to be caused by the switching from Infiniband to the slower Gigabit Ethernet under high network loads. It was temporarily circumvented by rising the domains' resolution and limiting the study to 32 cores. The second fluctuation appeared randomly due to the user-induced random load of the network. The third and largest fluctuation appeared periodically at specific rush hours before lunch and on Friday afternoons. All three phenomena appeared in a different type of time-fluctuation and were not easy to identify, but are important to know about for accurate measurements. These findings may help others occupied with benchmarking tasks in the future.

Summarizing, first testing and then choosing a compiler is worth the work for WRF on a HPC, thinking of the many WRF users which just stick to GCC compilers as the easiest "out of the box" solution. In addition a careful monitoring for exceptional performance fluctuations is also needed for accurate benchmarking. Those are tasks which can't be accomplished by every WRF user. Hence, more and continuous publications of the professionals on this matter are important to provide at least a rough overview on the performance of the countless possible combinations of hardware and software like the compiler.

Aspects particularly missing in this paper are performance measurements with a sophisticated tool like *The Vampir Performance Analysis Tool-Set* (Knüpfer et al., 2008), measurements with larger core-counts, with a broader range of compilers (PathScale, Sun/Oracle) and more MPI implementations (MPICH/MVAPICH/IntelMPI), and with the just released WRF version 3.3, April 2011.

Another important topic, especially of interest for compiler-developers, turned out to be the effect of different optimization flags. As a start PGIs value-safe flags were benchmarked. Later work should include the general optimization levels `O0` to `O3`, the *Profile-Guided Optimization*, and the *Interprocedural Optimization* (IPO) available for GCC and Intel compilers.

Appendix A

configure.wrf architecture specific settings

Open MPI-specific lines

```
DMPARALLEL        =  1 # 1 for dmpar and hybrid mode, 0 for smpar and serial (single-CPU)
OMPCPP            =  # -D_OPENMP # like the next two only for smpar mode
OMP               =  # -mp -Minfo=mp -Mrecursive
OMPCC             =  # -mp
```

[...]

```
DM_FC             =  mpif90 -f90=$(SFC)
DM_CC             =  mpicc -cc=$(SCC) -DMPI2-SUPPORT
```

[...]

Compiler-specific lines

```
# Settings for Linux x86_64, PGI compiler with gcc (dmpar)
```

[...]

```
SFC               =  pgf90
SCC               =  gcc
CCOMP             =  pgcc
```

[...]

```
PROMOTION         =  -r$(RWORDSIZE) -i4
ARCH_LOCAL        =  -DNONSTANDARD_SYSTEM_SUBR
CFLAGS_LOCAL      =  -w -O3
```

[...]

```
FCOPTIM           =  -fastsse -Mvect=noaltcode -Msmartalloc -Mprefetch=distance:8 -Mfprelaxed
                     # -Minfo=all=Mneginfo=all
FCREDUCEDOPT      =  $(FCOPTIM)
FCNOOPT           =  -O0
FCDEBUG           =  # -g $(FCNOOPT)
FORMAT_FIXED      =  -Mfixed
FORMAT_FREE       =  -Mfree
FCSUFFIX          =
BYTESWAPIO        =  -byteswapio
FCBASEOPTS_NO_G   =  -w $(FORMAT_FREE) $(BYTESWAPIO) $(OMP)
FCBASEOPTS        =  $(FCBASEOPTS_NO_G) $(FCDEBUG)
MODULE_SRCH_FLAG  =  -module $(WRF_SRC_ROOT_DIR)/main
```

[...]

```
# Settings for Linux x86_64 ifort compiler with icc (dmpar)
```

[...]

```
SFC               =  ifort
SCC               =  icc
CCOMP             =  icc
```

[...]

```
PROMOTION         =  -i4
ARCH_LOCAL        =  -DNONSTANDARD_SYSTEM_FUNC
CFLAGS_LOCAL      =  -w -O3
```

[...]

```
FCOPTIM           =  -O3
FCREDUCEDOPT      =  $(FCOPTIM)
FCNOOPT           =  -O0 -fno-inline
                     -fno-ip
FCDEBUG           =  # -g $(FCNOOPT)
                     -traceback
FORMAT_FIXED      =  -FI
FORMAT_FREE       =  -FR
FCSUFFIX          =
BYTESWAPIO        =  -convert big_endian
FCBASEOPTS_NO_G   =  -w -ftz -align
                     all -fno-alias
                     -fp-model precise
                     $(FORMAT_FREE)
                     $(BYTESWAPIO)
FCBASEOPTS        =  $(FCBASEOPTS_NO_G)
                     $(FCDEBUG)
MODULE_SRCH_FLAG  =
```

[…]

```
# Settings for x86_64 Linux, gFortran compiler with gcc (dmpar)
```

[...]

```
SFC               =  gFortran
SCC               =  gcc
CCOMP             =  gcc
```

[...]

```
PROMOTION         =  # -fdefault-real-8 #
                     uncomment manually
ARCH_LOCAL        =  -DNONSTANDARD_SYSTEM_SUBR
CFLAGS_LOCAL      =  -w -O3 -c
                     -DLANDREAD_STUB
```

[...]

```
FCOPTIM           =  -O3 -ftree-vectorize
                     -ftree-loop-linear
                     -funroll-loops
FCREDUCEDOPT      =  $(FCOPTIM)
FCNOOPT           =  -O0
FCDEBUG           =  # -g $(FCNOOPT)
FORMAT_FIXED      =  -ffixed-form
FORMAT_FREE       =  -ffree-form
                     -ffree-line-length-none
FCSUFFIX          =
BYTESWAPIO        =  -fconvert=big-endian
                     -frecord-marker=4
FCBASEOPTS_NO_G   =  -w $(FORMAT_FREE)
                     $(BYTESWAPIO)
FCBASEOPTS        =  $(FCBASEOPTS_NO_G)
                     $(FCDEBUG)
MODULE_SRCH_FLAG  =
```

[…]

Appendix B

Changes in the namelist configuration files (bold)

Within the namelist.wps:

```
&share
start_date          =  '2000-01-24_12:00:00',
end_date            =  '2000-01-25_12:00:00',
interval_seconds    =  21600,
prefix              =  'FILE',
wrf_core            =  'ARW',
max_dom             =  1,
io_form_geogrid     =  2,
/
&geogrid

parent_id           =  1, 1,
parent_grid_ratio   =  1, 3,
i_parent_start      =  1, 31,
j_parent_start      =  1, 17,
e_we                =  666,
e_sn                =  549,
geog_data_res       =  '30s',
dx                  =  3333,
dy                  =  3333,
map_proj            =  'lambert',
ref_lat             =  34.83,
ref_lon             =  -81.03,
truelat1            =  30.0,
truelat2            =  60.0,
stand_lon           =  -98.0,
geog_data_path      =  'Your WPS_GEOG data loca-
                       tion'
/
[...]
```

```
Within the namelist.input:
```

```
&time_control
run_days            =  0,
run_hours           =  0,
run_minutes         =  15,
[...]
&domains
numtile             =  X # set a value for X, see Sect.
                       2.4
time_step           =  20,
max_dom             =  1,
s_we                =  1, 1, 1,
e_we                =  666,
s_sn                =  1, 1, 1,
e_sn                =  549,
s_vert              =  1, 1, 1,
e_vert              =  28, 28, 28,
num_metgrid_levels  =  27
dx                  =  3333,
dy                  =  3333,
[...]
```

Supplementary material related to this article is available online at: http://www.geosci-model-dev.net/4/611/2011/gmd-4-611-2011-supplement.zip.

Acknowledgements. The author thanks his colleagues for continuing support and discussion around the coffee breaks. I especially and greatly thank the whole DKRZ team pointing out Birgit Schüen for the outstanding and sustained support during the compilation process of WRF!

Edited by: P. Jöckel

References

Cisneros, G., Dembek, S. R., Dudhia, J., and Kain, J.: Effect of non-IEEE-compliant optimizations on WRF numerical results, 9th WRF Users' Workshop, Boulder, CO, available at: www.mmm.ucar.edu/wrf/users/workshops/WS2008/abstracts/P11-06.pdf (last access: 25 May 2011), 2008.

Dudhia, J.: WRF Modeling System Overview, available at: www.mmm.ucar.edu/wrf/users/tutorial/201104_Korea/01_WRF_Overview_Dudhia.pdf, last access: 25 May 2011.

GCC platforms: available at: http://gcc.gnu.org/install/specific.html, last access: 1 February 2011.

HPC Advisory Council: Weather Research and Forecasting (WRF) Performance Benchmark and Profiling, Best Practices of the HPC Advisory Council, available at: www.hpcadvisorycouncil.com/best_practices.php, www.hpcadvisorycouncil.com/pdf/WRF_Analysis_and_Profiling_Intel.pdf (last access: 25 May 2011), 2010.

Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M. S., and Nagel, W. E.: The Vampir Performance Analysis Tool-Set, in Tools for High Performance Computing Part 4, 139-155 – Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart, available at: doi:10.1007/978-3-540-68564-79, 2008.

Michalakes, J.: WRF V3 Parallel Benchmark Page, available at: www.mmm.ucar.edu/wrf/WG2/benchv3 (last access: 21 January 2011), 2010.

Morton, D., Nudson, O., and Stephenson, C.: Benchmarking and Evaluation of the Weather Research and Forecasting (WRF) Model on the Cray XT5, Cray User Group Proceedings, Atlanta, GA, 04–07, 2009.

Morton, D., Nudson, O., Bahls, D., and Newby, G.: Use of the Cray XT5 Architecture to Push the Limits of WRF Beyond One Billion Grid Points, PDF send by Morton via email dating 4 January 2011, in Cray User Group Proceedings, Edinburgh, Scotland, 24–27 May 2010.

Newby, G. B. and Morton, D.: Performance Evaluation of Emerging High Performance Computing Technologies using WRF, AGU Fall Meeting Poster, available at: www.petascale.org/AGUPoster2008.pdf (last access: 17 March 2011), 2008.

PGI flags: available at: www.pgroup.com/benchmark/speccpu/woodcrest/pgi701pre-cint2006.flags.html (last access 3 February 2011), 2006.

Phoronix.com: Five Years Of Linux Kernel Benchmarks: 2.6.12 Through 2.6.37, available at: www.phoronix.com/scan.php?page=article&item=linux_2612_2637&num=3 (last access: 25 May 2011), 2010.

Roe, P. K. and Stevens, D.: Maximizing Multi-Core Performance of the Weather Research and Forecast Model Over the Hawaiian Islands, 2010 AMOS Conference Technical Papers, Maui, Hawaii, available at: www.amostech.com/TechnicalPapers/2010/Posters/Roe.pdf (last access: 5 July 2011), 2010.

Roman, D.: Performance hints for WRF on Intel® architecture, available at: http://software.intel.com/en-us/articles/performance-hints-for-wrf-on-intel-architecture (last access: 9 February 2011), 2009.

Rozumalski, R.: NOAA/NWS SOO STRC WRF Environmental Modeling System, available at: http://strc.comet.ucar.edu/wrfems/index.htm (last access: 21 January 2011), 2010.

Salapura, V., Walkup, R., Gara, A.: Exploiting Workload Parallelism for Performance and Power Optimization in Blue Gene, in Micro, IEEE Sept.-Oct. 2006, 26(5), 67–81, doi:10.1109/MM.2006.89, 2006.

Sankaran, L.: System Tuning and Application Optimization for HPC, Hewlett-Packard Company, available at: www.cbtechinc.com/?LinkServID=68BA8DCC-FD89-E9E2-D3AEFD4E23C4A410&showMeta=0 (last access: 17 March 2011), 2010.

Semenov, A., Kashevarova, T., Mankevich, P., Shkurko, D., Arturov, K., and Panov, N.: WRF performance tuning for platforms based upon the Intel Itanium Processor, Joint WRF/MM5 Users' Workshop, Boulder, CO, June 2005, available at: http://www.mmm.ucar.edu/wrf/users/workshops/WS2005/abstracts/Session3/46-Mankevich.pdf (last access: 25 May 2011), 2005.

Shainer, G., Liu, T., Michalakes, J., Liberman, J., Layton, J., Celebioglu, O., Schultz, S. A., Mora, J., and Cownie, D.: Weather Research and Forecast (WRF) Model Performance and Profiling Analysis on Advanced Multi-core HPC Clusters, The 10th LCI International Conference on High-Performance Clustered Computing, Boulder, CO, available at: http://141.142.2.101/conferences/archive/2009/PDF/Shainer_64557.pdf (last access: 11 November 2010), 2009.

Skamarock, W. C., Klemp, J. B, Dudhia, J., Gill, D. O., Barker, D. M., Duda, M. G., Huang, X.-Y., Wang, W., and Powers, J. G.: A description of the advanced research WRF version 3, NCAR Technical Note, NCAR/TN-475+STR, 113 pp., 2008.

Tornado Wiki Hardware Overview: available at: https://tornado-wiki.dkrz.de/farm/HardwareOverview (last access: 24 May 2011), 2010.

Wang, W., Bruyère, C., Duda, M., Dudhia, J., Gill, D., Lin, H.-C., Michalakes, J., Rizvi, S., Zhang, X., Beezley, J. D., Coen, J. L., and Mandel, J.: ARW Version 3 Modeling System User's Guide 2011, available at: www.mmm.ucar.edu/wrf/users/docs/user_guide_V3/contents.html, January 2011.

WRF Homepage: available at: www.wrf-model.org, last access: 19 January 2011.

WRF Users Page: WRF Model Version 3.0.1: UPDATES, available at: www.mmm.ucar.edu/wrf/users/wrfv3/updates-3.0.1.html (last access 25 May 2011), 2008.

WRF User's Forum: Compilation failed for file da_wrfvar_top.f on IBM, available at: http://forum.wrfforum.com/viewtopic.php?f=5&t=1627 (last access 26 May 2011), 2009.

Wright, N. J., Smallen, S., Mills Olschanowsky, C., Hayes, J., and Snavely, A.: Measuring and Understanding Variation in Benchmark Performance, HPCMP Users Group Conference, 2009 DoD High Performance Computing Modernization Program Users Group Conference, available at: http://doi.ieeecomputersociety.org/10.1109/HPCMP-UGC.2009.72 (last access: 17 March 2011), 438–443, doi:10.1109/HPCMP-UGC.2009.72, 2009.