Science & Technology Facilities Council

Technical Report
DL-TR-2011-003

Configuring WRF for Maximum Performance on HECToR

AR Porter

July 2011

Enquiries about copyright, reproduction and requests for additional copies of this report should be addressed to:

Chadwick Library
Science and Technology Facilities Council
Daresbury Laboratory
Daresbury Science and Innovation Campus
Warrington
WA4 4AD

Tel: +44(0)1925 603397
Fax: +44(0)1925 603779
email: librarydl@stfc.ac.uk

Science and Technology Facilities Council reports are available online at: http://epubs.stfc.ac.uk

ISSN 1362-0207

Configuring WRF for Maximum Performance on HECToR

A. R. Porter,
STFC Daresbury Laboratory, UK
andrew.porter@stfc.ac.uk

June 2010

Contents

1 How to Use this Document

The heart of this document consists of three sections on how to improve the performance of WRF; section 4 contains options that can be applied at run time and are therefore easiest to try, section 5 contains options that can be applied when WRF is compiled and section 6 contains options related to input/output.

Section 2.3 and Appendix B give information on how to check the performance of WRF for yourself and section 3 gives information on checking that your changes haven't broken anything.

Essentially, we recommend you try the following:

1. Generate a version of your production job that completes in under 20 minutes on the minimum number of cores that you'd consider running on (*i.e.* edit the *run_hours* and *run_minutes* fields of the namelist.input file);

2. Run this short version of your job on varying numbers of cores and select the best core count — the number of cores on which your job runs fastest while still being cost effective (there's little point doubling the number of cores your job uses if that only makes it run a fraction faster);

3. On that core count, experiment with using > 1 tile per MPI patch (section 4.1);

4. Try changing the way in which the model domain is decomposed across cores (section 4.2);

5. Reduce the time lost to writing history data by using I/O quilting (section 6) — experiment with the number of IO servers to find the optimimum;

6. Compile WRF for mixed-mode with optimising flags on (section 5). Run with one MPI process per node and four OpenMP threads per process on Phase IIa. On Phase IIb use two MPI processes per node with 12 OpenMP threads each (see Appendix A);

7. If your model contains nested domains, add the *-DSGIALTIX* flag when compiling WRF (section 5.1);

8. CHECK that your results are sufficiently close to those produced by a default build/run of WRF before going into production.

Note that WRF v.3.1.1 binaries compiled on Phase IIa of HECToR for both MPI-only and mixed-mode execution are available in the */work/n02/n02/wrf/bin/* directory on HECToR.

2 Introduction

This document is intended to provide a WRF user on HECToR with information on how to make their simulations run as quickly as possible while still giving sufficiently correct results. It is one of the outputs of the dCSE-funded project to optimise the performance of the Weather Research and Forecast model (WRF) on the Cray XT component of HECToR. More details may be found in either the project report [5] or Cray User Group paper [6].

2.1 WRF

WRF versions 3.0.1.1 and 3.1.1 were used for the work on which these guidelines are based. It is expected that the conclusions will carry through to future releases although caution may have to be exercised with specific details such as namelist options.

2.2 Computers Used

HECToR is the current (2010) incarnation of the UK's national academic supercomputing service. When this project began, the scalar component of HECToR was at Phase I with each 'node' of the Cray XT comprising a single, dual-core AMD 2.6 GHz Opteron chip. However, HECToR was upgraded to Phase IIa in 2009 with each node comprising a single, quad-core AMD 2.3 GHz Barcelona chip. (The node interconnect is unchanged from Phase I). Just as this project was ending, Phase IIb of HECToR became available. This has compute nodes built from two, 12-core AMD 2.1 GHz Magny-Cours chips while initially retaining the same interconnect as Phases I and IIa.

For some of the results mentioned in this document we have also used 'Monte Rosa' which is a Cray XT5 at the Swiss National Supercomputing centre [1]. Rosa has compute nodes built from two, six-core AMD Opteron 2.4 GHz Istanbul chips giving 12 cores per node compared to HECToR Phase IIa's four. The compute-node interconnect is the same as that on HECToR. Unless stated otherwise, any results presented in this document were obtained on Phase IIa of the HECToR Cray XT.

2.3 Performance Measurement

When examining the performance of WRF it is not sufficient simply to examine the total wall-clock time used by a job. This is because model initialisation can account for a significant fraction of the time taken by a short (*e.g.* 10 model minutes) benchmarking run. It is best therefore to look at the mean time taken to step the whole model forwards in time. This information is printed to the *rsl.out.0000* file by MPI process zero. If using nested domains, it is the time taken to step the outermost one. A script to extract this information and exclude the effects of input/output is available on HECToR (*/work/n02/n02/wrf/bin/analyse_times.pl*) and is reproduced in Appendix B. That Appendix also contains information on extracting the time spent by a WRF job on doing

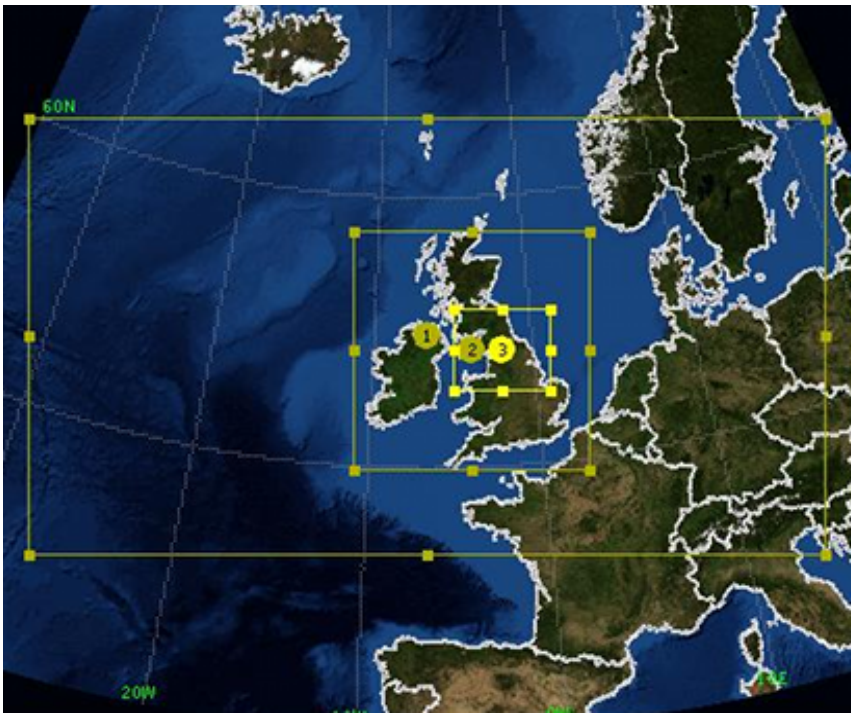

Figure 1: The three domains for the 'Great North Run' benchmarking configuration.

input/output (I/O).

Performance variability is a fact of life on large, shared machines and can be significant on HECToR (typically $\sim$ 10% but can be much larger), especially when doing I/O or running on a large number ($\geq$ 1024) of cores (when a job is less likely to be mapped to a contiguous section of physical compute nodes). Therefore, as well as using the script mentioned above which performs averaging over all of the time-steps executed in a job, we recommend running the same job at least three times and looking at the mean step times from all three.

3 Results Verification and Visualisation

When making modifications to the way in which WRF is compiled and/or run, it is crucial to check that no scientifically-significant changes occur in the simulation results that WRF produces. It is down to you, the user, and your knowledge of the type of simulation to be performed to decide which aspects of the results need to be checked.

The effects of the changes described in this document were tested on a six model-hour run of the 'Great North Run' configuration shown in figure 1. The surface-pressure and surface-temperature fields were used to compare the results produced by binaries compiled with different compilers and differing degrees of optimisation. In particular, checks were performed between results from the most optimised binaries and those compiled with flags that force the compiler to only use math operations that are IEEE compliant.

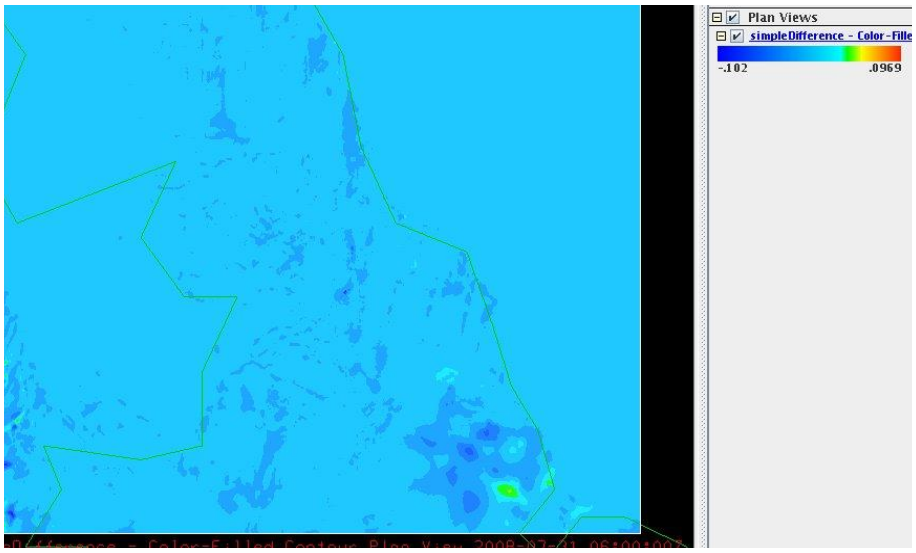

Figure 2: Differences between the 'Temperature at 2m' fields for the inner GNR domain produced by two different builds of WRF doing a six-hour simulation.

Results were compared using the Integrated Data Viewer (IDV) [4] with its ability to display the difference between two fields. An example of the small differences typically found is shown in Figure 2 which displays the Temperature at an elevation of two metres across the inner-most domain of the GNR configuration. Note the scale at the top-right of the image which shows that all differences are ~ 0.2 Kelvin.

4 Run-time Options

This section details the options most readily-available to the user since they only involve changes to the *namelist.input* file prior to job execution.

4.1 Optimisation of Cache Usage

WRF was written to support mixed-mode execution where each MPI process is further parallelised across a number of OpenMP threads. This has implications for the way in which WRF decomposes the model domain. When running in distributed-memory (dm)/MPI-only mode, the model domain is decomposed into as many rectangular 'patches' as there are PEs (*i.e.* MPI processes) and each patch is assigned to a PE. When running in mixed (dm+sm)-mode those patches are further decomposed into 'tiles' which are then shared amongst the available OpenMP threads, see figure 3 for an illustration.

Although this decomposition is generated automatically by WRF, the user can control the number of tiles per patch, *even when no OpenMP is being used.* Changing the number of tiles/patch can be used to change the size of the arrays within the computation in order to make more efficient use of the cache architecture of the processors.

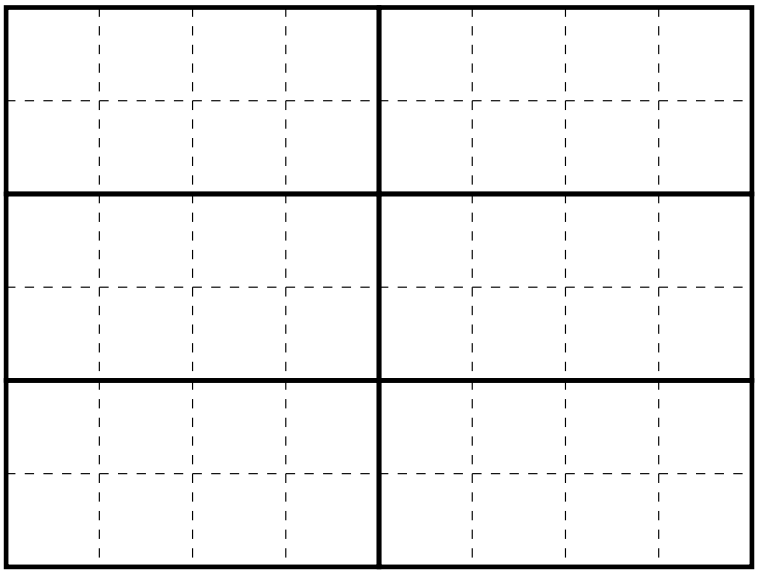

Figure 3: Illustration of a domain decomposition within WRF suitable for a mixed-mode job with six MPI processes, each with eight OpenMP threads. Patches are drawn with bold, solid lines and tiles with dashed lines.

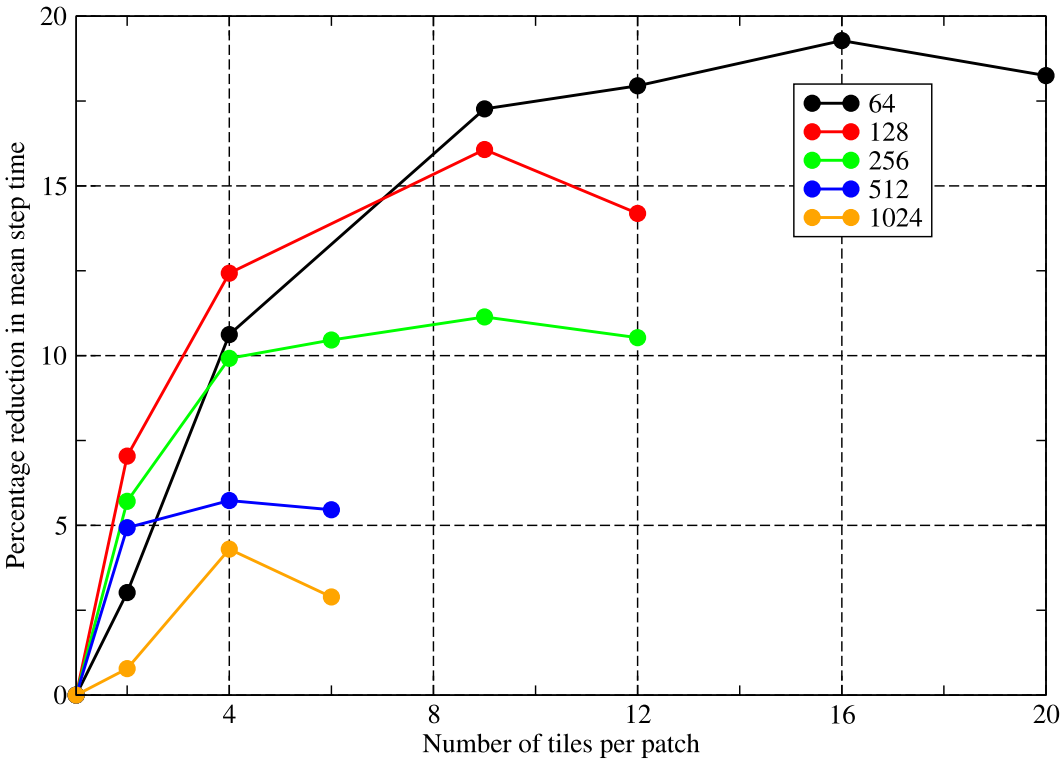

Figure 4: The percentage reduction in the mean wall-clock time taken to step the model as a function of the number of tiles used per patch. Results are shown for a variety of PE counts for the GNR configuration. Lines are guides to the eye.

Figure 4 shows the effect on the performance of WRF built in dm mode running the GNR configuration when the number of tiles per patch is increased. Increasing the number of tiles has the greatest effect on the lower processing-element (PE) counts when the patches and thus arrays are at their largest. So we see that using 16 tiles/patch on a 64-PE job achieves a speed-up of almost 20% whilst the best achieved for the 1024-PE job is approximately 5% with just four tiles/patch. Note that the optimum number of tiles for any given job will depend upon the size and shape of the domains used in the configuration that WRF is running as well as the number of MPI processes: the user must test with their own configuration in order to determine the optimum number of tiles to use.

Altering the number of tiles can be done at run time by setting the *numtiles* member of the *&domains* section of the *namelist.input* file. By default numtiles is set to one when running in dm mode and to the number of OpenMP threads when running in dm+sm mode. If the model domains are sufficiently large, the performance of WRF in dm+sm mode might also be improved by increasing the number of tiles from the default value.

4.2 Choice of Domain Decomposition

Given a number of PEs, n, on which to execute in dm mode WRF will, by default, select a factorisation $n_x.n_y = n$ where n_x and n_y are as close as possible to $\sqrt{n}$. Here, n_x and n_y are the horizontal and vertical dimensions of the processor grid on which the domain is decomposed. The choice of this decomposition can affect the performance of the simulation since it changes the dimensions of the rectangular patches (and hence of the arrays) that each PE works on and also changes the length of the boundaries across which PEs must exchange halo data. Users may explicitly select a decomposition rather than allowing WRF to set it. This is achieved by setting the *nproc_x* and *nproc_y* fields in the *&domains* section of the *namelist.input* file.

Predicting the effect on performance is complex but it is straightforward to run a few experiments and select the best domain decomposition for the model to be run. Figure 5 shows the results of doing this for the GNR configuration on 256 and 504 PEs on HECToR . Although the number of factorisations of 256 is limited, we can still find a case (8×32) that improves on the performance of the default decomposition by approximately 10%. On 504 PEs where the patches are much smaller, the effects are much less dramatic.

4.2.1 Effect of Domain Size/Shape

Since WRF uses the same decomposition for each domain of a nested run, it will scale most efficiently to higher numbers of cores when all of the nested domains in a configuration are of the same shape. Otherwise scaling will be limited by PEs having insufficient work to do on the domain with the shortest extent in a given dimension. Cray staff have found that a minimum of seven grid points is required in each direction per PE. Beyond this point, no futher performance improvement is obtained through running WRF on more cores.

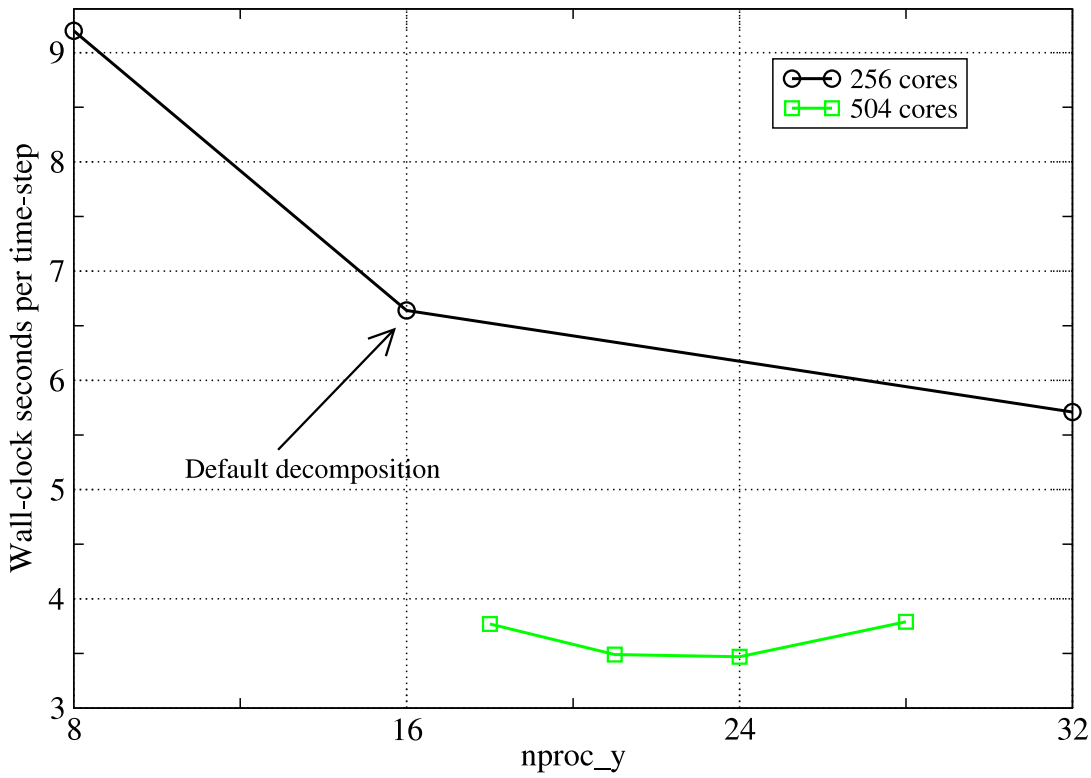

Figure 5: The variation in performance of WRF running the GNR configuration on 256 and 504 PEs as the x-y decomposition of the PE grid is varied. Lines are guides to the eye.

5 Compile-time Optimisations

WRF is supplied with its own, interactive 'configure' script which seeks to identify the platform and offer appropriate options. On HECToR which has the Portland Group (PGI), Pathscale (PS), Gnu and Cray compiler suites installed, this results in approximately 20 different ways of building WRF. However, the relatively-young Cray compiler was unable to build WRF and therefore was not an option for this project. This is likely to change in the future, especially since Cray now own the PS compiler.

Running the configure script generates a *configure.wrf* text file which stores all of the configurable compilation options. Once a good configuration has been generated, a copy of this text file may be taken in order to keep a backup of it. It can also be used as a starting point for compiling WRF on other computers of similar architecture.

By default, the WRF configure script sets safe compiler optimising flags — *i.e.* those that the WRF development team are confident will not significantly alter the results produced by the code. However, if one is prepared to do some testing, it is possible to generate a faster-running WRF binary.

Which compiler is best to use depends on what sort of WRF binary you want to produce; MPI only (dm) or mixed MPI/OpenMP (dm+sm). For dm-mode, the PGI compiler was found to produce a slightly faster WRF binary than the PS compiler. The default *configure.wrf* file produced when configuring to use PGI contains commented-out suggestions for a better set of flags. Taking those as a starting point, we have found the best set of flags for PGI to be: -O3 -fastsse -Mvect=noaltcode -Msmartalloc -Mprefetch=distance:8 -Mfprel. These did not significantly alter the results obtained for the GNR configuration but *you need to test for yourself* as changes of compiler, WRF version, and different configurations and physics options could all affect this.

For dm+sm mode, the PS compiler was found to be best and again, performance can be improved by editing the *configure.wrf* file to use additional compiler flags:
-O3 -OPT:Ofast:ro=1:malloc_algorithm=1:early_intrinsics=ON -LNO:prefetch_ahead=8:full_unroll=10.

For machines such as the Cray XT6 with large (12 or more) cores per compute node, we strongly recommend building WRF in dm+sm mode as it performs significantly better than the dm-mode version on large core counts (512+). In our tests we have found the PS compiler to produce the fastest mixed-mode version (using the same optimising flags as given above). An example job script for running a mixed-mode job on HECToR is given in Appendix A.

Note that WRF v.3.1.1 binaries built on Phase IIa with the various configurations described in this section are all available in the */work/n02/n02/wrf/bin* directory on HECToR.

5.1 Nesting Domains

When running configurations involving one or more nested domains we have found that the process of inter-domain interpolation can become surprisingly time consuming. The

culprit is the initialisation of the temporary arrays used in the interpolation. In many cases this initialisation is unnecessary and can be switched-off at compile time by editing *configure.wrf* and adding *-DSGIALTIX* to the line that sets the *ARCH_LOCAL* variable. Doing so reveals a tiny bug in the code which can be fixed by editing the *patch_domain_rsl_lite* routine in *WRFV3/external/RSL_LITE/module_dm.F*. You must add 'alloc_space_field' to the line specifying which routines are USE'd from the *module_domain* module:

```
USE module_domain, ONLY : domain, head_grid, &
find_grid_by_id, alloc_space_field
```

and then re-build. As with the compiler flags, it is essential to check that compiling WRF with *SGIALTIX* defined does not alter the results produced for the configuration of interest.

We found this fix to give a 20–25% improvement in the performance of dm+sm-mode WRF when running the GNR configuration on 512–2048 cores of HECToR. See the project report [5] for a plot.

6 Input/Output Performance

Writing (and sometimes reading) data often becomes a bottleneck for high-performance applications and WRF is no exception The default approach to outputting history data in WRF is to gather all of the data onto the master PE, reconstruct the whole field and then write it to disk using the standard, serial netCDF library. The number of bytes written to disk is then broadcast back to all PEs meaning that all PEs must wait until the master has completed the write. The time taken to do all of this for each domain is written to standard output and can be found in the *rsl.out.0000* file.

WRF has several different mechanisms available for doing I/O (see the WOMPS project report for more details [5]) and we recommend that WRF's so-called 'I/O quilting' functionality be used in place of the default. In this case, the user chooses to reserve a number of PEs purely for doing I/O. These 'IO servers' receive data from the remaining (compute) PEs and deal with writing it to disk while the compute PEs continue number crunching. The bottleneck now is the sending of data from the compute PEs to the IO servers and how long this takes depends on how many compute PEs each IO server must receive data from.

We tested the performance of this functionality on HECToR for the GNR configuration run on 1024 compute PEs and the results are shown in figure 6. Use of the IO servers reduces the time 'lost' by the compute PEs in writing a single frame of history from 30 to 19 seconds. The optimum number of IO servers in this case is 12 so that there are approximately 85 compute PEs per IO server.

The number of IO servers to use is set at runtime via the *nio_tasks_per_group* member of the *&namelist_quilt* section of the *namelist.input* file. (The *nio_groups* member must be left set equal to one.) If the user wishes to preserve the number of PEs actually used for computation then they will need to add the value of *nio_tasks_per_group* to the number

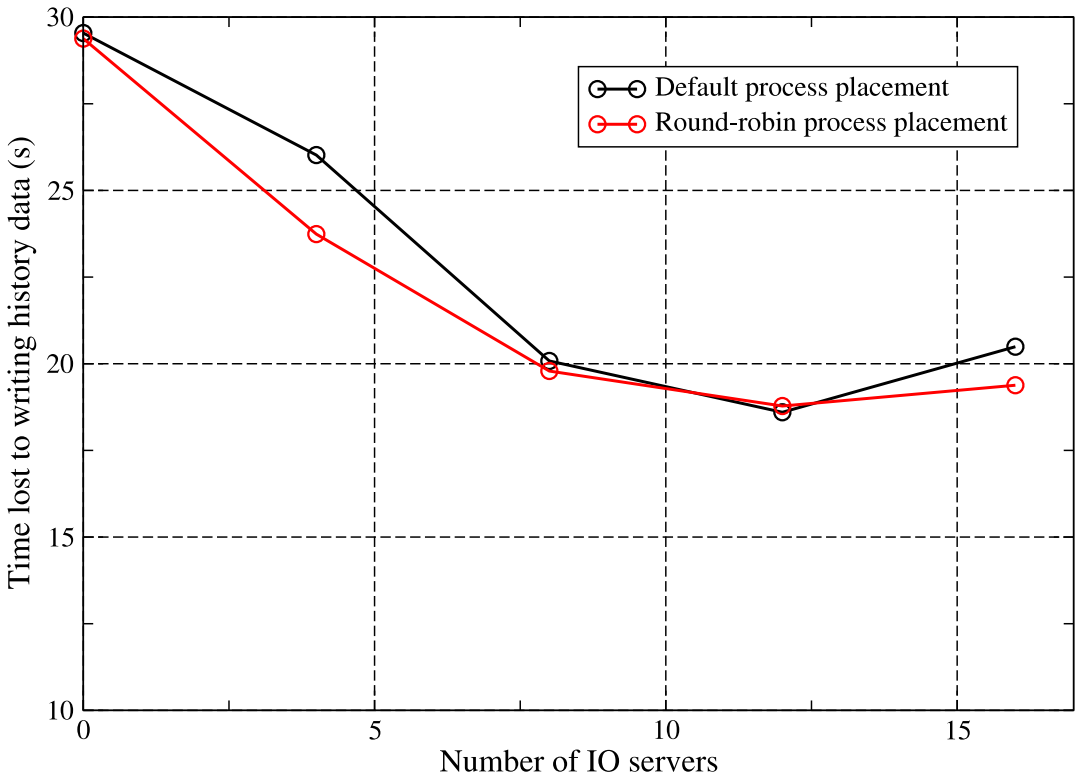

Figure 6: The mean time 'lost' by the compute nodes due to writing a single history frame for all three domains in the GNR model. WRF was run with 1024 compute PEs plus the specified number of IO servers.

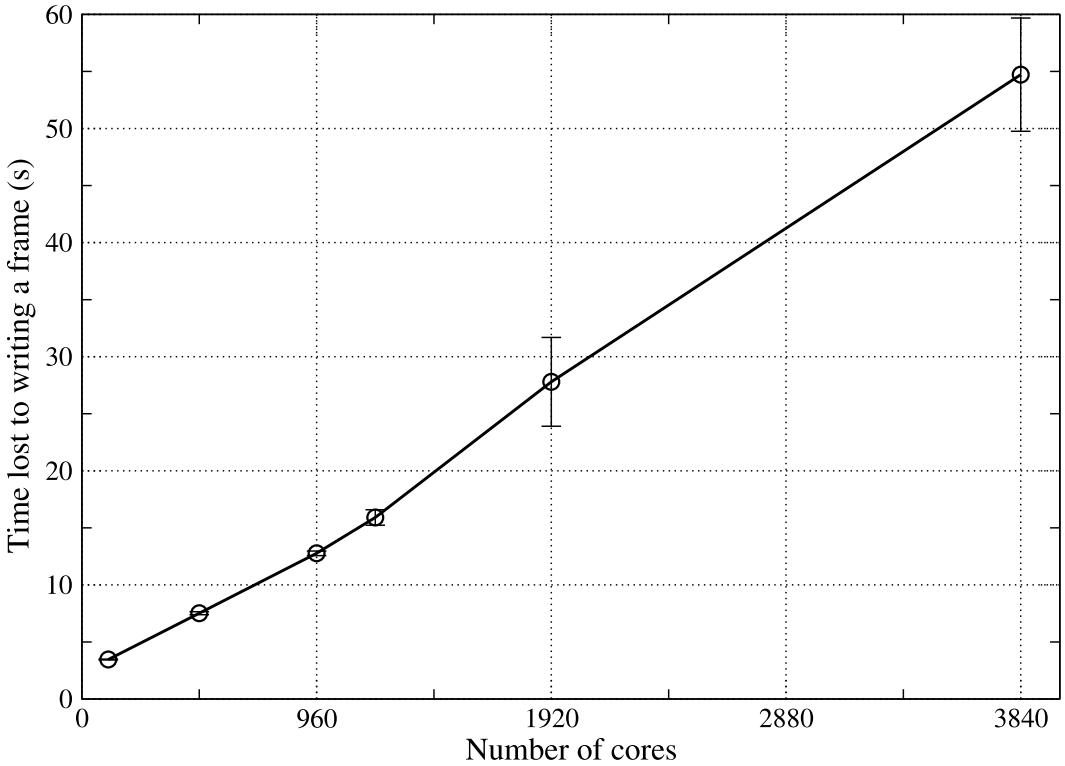

Figure 7: The time 'lost' to writing a single frame of history data as the number of cores used in the WRF job is increased. Each data point represents the mean value obtained from three separate runs and error bars show the associated standard error. Lines are guides to the eye.

of PEs being requested in their job script. In order to avoid being charged for un-used cores, this total number of PEs must remain a multiple of the number of cores per node (four on Phase IIa of HECToR but 24 on Phase IIb).

6.1 What if a domain won't fit into available memory?

The current trend in high-performance computing is to seek better performance by adding more cores to a node. Unfortunately, the amount of memory available per node is not increasing as rapidly and therefore the net effect is to reduce the amount of memory available to a single computational core. This, combined with the requirement to run ever larger models (resulting in increased memory usage), means that increasingly it will not be possible to gather a whole domain onto a single core for writing to disk.

There are two possible solutions to this problem. The first is to run on under-populated nodes, thus increasing the amount of memory available per core (although the job will still be charged for all of the nodes it occupies, irrespective of how many cores are actually used). The second solution is to use WRF's Parallel netCDF (pNetCDF) I/O layer in which every PE writes its portion of the data to disk. This is not an approach which will

work well for very large jobs (upwards of 5000 cores) but it is workable — figure 7 shows the performance obtained with this method for the GNR model on Rosa.

WRF must be built with pNetCDF support included in order to make this functionality available. The pNetCDF library is not available as a module on HECToR and therefore must be built manually, *e.g.*:

```
cd parallel-netcdf-1.1.1
export CC=cc
export FC=ftn
export MPIF77=$FC
export MPICC=$CC
./configure --enable-fortran
  --prefix=/home/n02/<MY_HOME_DIRECTORY>/parallel-netcdf-1.1.1/PGI
make
make install
```

Once that is done, set the *PNETCDF* environment variable to the location of the pNetCDF library (as specified with the *–prefix* argument to the configure script) before running the WRF *configure* script.

Once a WRF binary with pNetCDF support has been built, use of pNetCDF must be switched on in the *namelist.input* file. This is achieved by seting the *io_form_** elements of the *&time_control* section to 11:

io_form_history = 11
io_form_restart = 11
io_form_input = 11
io_form_boundary = 11

Note that IDV (version 2.3) was found to have trouble opening the CDF2 NetCDF files produced by pNetCDF. (Version 3.6.2 of the NetCDF library produces CDF1 NetCDF files.) The ncview [7] tool was used instead.

7 Conclusions

This document has described the main ways in which the performance of the WRF code may be both measured and improved by the user. These findings are a result of the dCSE WOMPS project; more detail on all of the issues discussed here may be found in the project report [5].

A Example job scripts

Below is an example script for running a mixed-mode WRF binary on Phase IIa of HECToR:

```
#!/bin/bash
# Script for running WRF in mixed mode (MPI + OMP) on 32 compute nodes
# with one MPI process and 4 OpenMP threads on each.
#PBS -N wrfmixedexe
#PBS -l mppwidth=32
#PBS -l mppnppn=1
#PBS -l mppdepth=4
#PBS -l walltime=00:20:00
#PBS -j oe
#PBS -A n02-weat

# This variable should be set to allow the ALPS and the OS scheduler to
# assign the task affinity rather than the compiler. If this is not set
# then you may see a large negative effect on performance.
export PSC_OMP_AFFINITY=FALSE

cd $PBS_O_WORKDIR

export MPICH_UNEX_BUFFER_SIZE=256M

# Set the number of MPI tasks
export NPROC=`qstat -f $PBS_JOBID | awk '/mppwidth/ {print $3}'`

# Set the number of MPI tasks per node
export NTASK=`qstat -f $PBS_JOBID | awk '/mppnppn/  {print $3}'`

# Set the number of OpenMP threads per node
export OMP_NUM_THREADS=`qstat -f $PBS_JOBID | awk '/mppdepth/  {print $3}'`

aprun -n $NPROC -N $NTASK -d $OMP_NUM_THREADS ./wrf.exe.mixed
```

On Phase IIb of HECToR there are 24 cores available per node and therefore *mppdepth* should be set to 24 in the above script. Experiments on Rosa have indicated that when the compute node has more than one socket (processor) it is preferable to run a single MPI process on each. To do this on HECToR IIb set $mppnppn = 2$ and $mppdepth = 12$.

B Scripts for performance analysis

B.1 Mean time per model time step

Below is a Perl script that uses AWK to calculate the mean time taken to step each domain in a WRF run. It takes as input one or more WRF stdout files (*e.g. rsl.out.0000*). Two passes are made of the file for each domain; the first to calculate an initial mean and

standard deviation and the second to calculate a revised mean, excluding any values that differ from the initial estimate of the mean by more than twice the standard deviation. This is done so as to exclude time steps which involve I/O as these take significantly longer.

Note that the text has been formatted slightly for display here, principally through the addition of line breaks.

```
#!/usr/bin/perl

use strict;

if(@ARGV < 1){
    print "Usage: analyse_times.pl [-v]
           <space-delimited list of wrf stdout files>\n";
    exit;
}

my $verbose = 0;

if($ARGV[0] eq "-v"){
    $verbose = 1;
    shift @ARGV;
}

my $string="";
my $mean  =0.0;
my $stdev =0.0;

FILE: foreach my $file (@ARGV){

    next FILE unless (-f $file);

    if($verbose){
        print "====== Processing file ".$file.":\n\n";
    }

#   For models containing up to 3 domains
    DOMAIN: for (my $idomain=1;$idomain<4;$idomain++){

#       Sum the time (and the time squared ) taken for each timestep
#       of this domain
        $string=`awk 'BEGIN{first1=0;};
                      /on domain   $idomain:/{
```

```
                        # Skip the first time-step as it includes
                        # initialisation
                        if(first1==0){
                               first1=1;
                        }else{
                               tot=tot+\$9;totsq+=\$9*\$9;
                               cnt++;
                        }
                    };
                    END{if(cnt>0){
                               avg=tot/cnt;
                               stdev=sqrt(totsq/cnt - avg*avg);
                               print avg, stdev;
                         }else{
                               print "ERROR ERROR";
                         }
                    };' $file`;

        next DOMAIN if( index($string, "ERROR") > -1 );

# Extract the initial mean and standard deviation from the string
# produced by the awk script
        ($mean, $stdev) = split(/\s+/, $string);

        if($verbose){
            print "Domain $idomain: original mean = ".$mean.
                  ", original stdev = ".$stdev."\n";
        }

#       This version only rejects timings with values that are 2*stdev
#       GREATER than the mean (because doing IO takes longer)
        $string=`awk 'BEGIN{maxDiff=2.0*$stdev;oldavg=$mean;};
                    /on domain   $idomain:/{
                         diff=\$9 - oldavg;
                         if( diff < maxDiff){
                                 tot=tot+\$9;
                                 totsq+=\$9*\$9;
                                 cnt++;
                         }else{
                                 print "Ignoring: ",\$0;
                         }
                    };
                    END{avg=tot/cnt;
```

```
                        stdev=sqrt(totsq/cnt - avg*avg);
                        print "Domain $idomain, avg = ",avg,
                              "Std. dev. = ",stdev,"Std. err = ",
                              stdev/sqrt(cnt),"Total = ",tot;
                    }' $file`;

        print $string;

        if($verbose){print "\n";}
    }

}
```

B.2 Time lost to doing I/O

The time lost to doing I/O can be obtained directly from the stdout files by *e.g.*:

```
$ awk '/Timing for Writing/{sum += $8;count++;print;
                            if(count == 3){
                                           print "Sum = ",sum;
                                           count=0;sum = 0;
                                          }
                           }' rsl.out.0000
Timing for Writing wrfout_d01_2008-07-31_00_00_00 for dom 1: 3.04047 elapsed s.
Timing for Writing wrfout_d02_2008-07-31_00_00_00 for dom 2: 4.05419 elapsed s.
Timing for Writing wrfout_d03_2008-07-31_00_00_00 for dom 3: 4.78711 elapsed s.
Sum = 11.8818
Timing for Writing wrfout_d03_2008-07-31_00_05_00 for dom 3: 3.01881 elapsed s.
Timing for Writing wrfout_d01_2008-07-31_00_05_06 for dom 1: 2.36254 elapsed s.
Timing for Writing wrfout_d02_2008-07-31_00_05_06 for dom 2: 9.32022 elapsed s.
Sum = 14.7016
Timing for Writing wrfout_d03_2008-07-31_00_10_00 for dom 3: 3.00149 elapsed s.
Timing for Writing wrfout_d02_2008-07-31_00_10_03 for dom 2: 3.88293 elapsed s.
Timing for Writing wrfout_d01_2008-07-31_00_10_12 for dom 1: 2.34952 elapsed s.
Sum = 9.23394
```

References

[1] *The Swiss National Supercomputing Centre*, `http://www-users.cscs.ch/`

[2] *HECToR - The UK Supercomputing Service*, `http://www.hector.ac.uk/`

[3] *NetCDF*, http://www.unidata.ucar.edu/software/netcdf/

[4] *IDV - the Integrated Data Viewer*, http://www.unidata.ucar.edu/software/idv/

[5] A. R. Porter, M. Ashworth, A. Gadian, R. Burton and M. Bane, 'WRF code Optimisation for Meso-scale Process Studies (WOMPS) dCSE Project Report,' http://epubs.stfc.ac.uk/work-details?w=53226.

[6] A. R. Porter and M. Ashworth, 'Configuring and Optimizing the Weather Research and Forecast Model On the Cray XT' in *Proceedings of the Cray User Group 2010*, May 2010.

[7] *ncview - a visual browser for netCDF files*, http://meteora.ucsd.edu/~pierce/ncview_home_page.html