Improving I/O Performance of the Weather Research and Forecast (WRF) Model

Tricia Balle
Cray Inc.
Benchmarking and Presales
Pukekawa, New Zealand
pburgess@cray.com

Pete Johnsen
Cray Inc.
Performance Engineering
St Paul, MN, USA
pjj@cray.com

Abstract— **As HPC resources continue to increase in size and availability, the complexity of numeric weather prediction models also rises. This increases demands on HPC I/O subsystems, which continue to cause bottlenecks in efficient production weather forecasting. In this paper we review the available I/O methodologies in the widely used NCAR Weather Research and Forecasting (WRF) model. We focus on the newer PNETCDF_QUILT technique that uses asynchronous I/O (quilt) servers alongside Parallel NetCDF.**

This paper looks at a high-resolution nested WRF case and compares the performance of various I/O techniques. The PNETCDF_QUILT technique is then described in detail. The Cray implementation of MPI-IO and useful diagnostic settings are discussed. The focus is on the performance of WRF on the Cray XC40 platform, with both Sonexion 2000 and Cray DataWarp storage. The DataWarp results are some of the first available and will be interesting to a wide range of Cray users.

Keywords-component; WRF, I/O, parallel I/O, MPI-IO, Cray DataWarp

I. INTRODUCTION

As supercomputing resources continue to increase in size and availability, the resolution and complexity of numeric weather prediction models also rise, markedly improving forecast accuracy. Unfortunately, this increases demands on HPC I/O subsystems, which continue to cause bottlenecks in efficient production weather forecasting. It is now common to see numeric weather simulations at grid resolutions of less than 2 kilometers covering the entire globe. Such runs can generate terabytes of weather information written frequently during a forecast cycle.

In this paper we review the available I/O methodologies in the widely used Weather Research and Forecasting (WRF) model [1], focusing on the newer quilt server + Parallel NetCDF (PNETCDF_QUILT) technique that uses asynchronous I/O (quilt) servers alongside Parallel NetCDF. NetCDF, a common data format used in the environmental sciences, is used throughout this study [2].

Performance on the Cray XC40 platform with Sonexion 2000 storage is discussed, and some early results using Cray DataWarp storage are also presented. Recent analysis of WRF benchmarks with larger domain sizes and output file sizes than have typically been seen prompted the discovery of two bugs in the WRF v3.6 and v3.7 source code, which suggests that the PNETCDF_QUILT technique is not very commonly used and deserves wider appreciation.

A. WRF Background

The WRF model was developed as a collaborative project by the National Center for Atmospheric Research (NCAR), the National Oceanic and Atmospheric Administration (NOAA), the National Centers for Environmental Prediction, the Air Force Weather Agency, the Naval Research Laboratory, the University of Oklahoma and the Federal Aviation Administration in the United States. It is a regional- to global-scale numerical weather prediction model intended for both research applications and operational weather-forecast systems. It is suitable for a broad spectrum of meteorological applications across scales ranging from meters to thousands of kilometers. A variation of WRF is used as the NOAA's primary regional forecast model for forecasts of 5 days ahead, and is used by weather agencies all over the world with thousands of registered users.

The WRF system incorporates two different dynamics solvers; the Advanced Research WRF (ARW) solver (developed by the Mesoscale and Microscale Meteorology Division of NCAR) and the Nonhydrostatic Mesoscale Model solver (developed by the National Centers for Environmental Prediction, US); in this paper, we discuss the former solver only.

The ARW solves the fully compressible, non-hydrostatic Euler equations using a finite-difference scheme on an Arakawa C-grid staggering in the horizontal plane and a terrain following, dry hydrostatic pressure vertical coordinate. There are 2nd- to 6th-order advection options for spatial discretization in both horizontal and vertical directions. Integration in time is performed using a time-split method with a 2nd- or 3rd-order Runge-Kutta scheme with a smaller time step for acoustic- and gravity-wave modes. The model supports periodic, open, symmetric and specified lateral boundary conditions and is capable of whole-globe simulations using polar Fourier filtering and periodic east-west boundary conditions [3].

The WRF model has, from the outset, been designed and written to perform well on massively parallel computers. It is written in Fortran90 and can be built in serial, parallel (MPI)

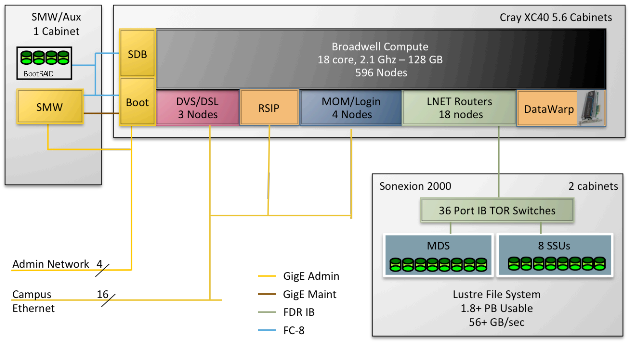

Figure 1: Cray XC40 System Diagram

and mixed-mode (OpenMP and MPI) forms, simply by choosing the appropriate option during the configure process.

All results presented in this paper were obtained with version 3.7.1 of WRF using the ARW core compiled using the Cray CCE compilation suite in mixed mode (though all runs used MPI only).

B. Benchmark System Configuration

All runs mentioned in the first part of this paper were made on a Cray XC40 system using Intel Broadwell processors with 18 cores per socket (36 cores per node) and 128GB of 2400mhz DDR4 memory per node (see Fig. 1).

The system was configured with Sonexion 2000 storage hardware [4] providing a Lustre parallel file system with 16 OSTs (stripes). Applications, like WRF, that take advantage of parallel I/O techniques can drastically lower the overhead associated with file I/O. Files can be striped across these OSTs to improve I/O performance. Cray DataWarp storage hardware [5] was also configured for the runs in the latter part of the paper.

C. WRF Benchmark Configuraton

The configuration for the WRF runs reported in this paper covers a region of Southeast Asia and is comprised of over 350 million grid points (see Fig. 2). Since WRF uses single precision (4 byte), each weather state variable takes approximately 1.5 Gbytes storage for both domains, in memory and on disk. Specifically:

- Two nested domains, with 3km and 1km resolutions.
- 5 second timestep
- 28 vertical levels
- Domain 1: 1770 (EW) x 1986 (NS)
- Domain 2: 2974 (EW) x 3118 (NS)
- 30 minute simulation
- History files written by both domains every 15 minutes (timesteps 0,180,360)

The domains are large enough to allow for scaling to thousands of MPI ranks. The simulation length is kept short to improve test turnaround time.

II. WRF I/O PERFORMANCE

WRF's well-defined I/O API provides several different implementations of its I/O layer:

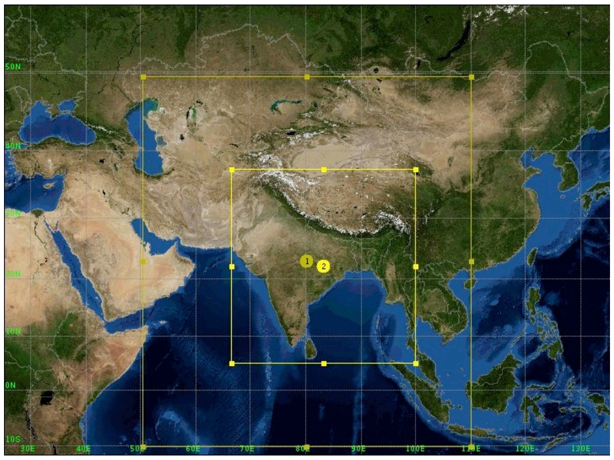

Figure 2: Benchmark Forecast Domain Configuration

- Serial NetCDF [2]: Default layer.
- Parallel NetCDF: A good alternative layer that works well at lower core counts built on the Parallel NetCDF (PNetCDF) library [6],[7], which supports parallel I/O.
- Quilt Servers: A third technique for writes that uses I/O (or quilt) servers that deal exclusively with I/O, enabling the compute PEs to continue with their work without waiting for data to be written to disk before proceeding.
- Quilt Servers with PNetCDF: An additional technique that combines the I/O server concept with PNetCDF to enable parallel asynchronous writes of WRF history and restart files. This technique proves to be highly advantageous on the Cray XC40 under certain circumstances.

On the Cray XC40, the NetCDF and PNetCDF libraries are available as standard modules and are supported with the standard programming environment releases. While the serial NetCDF and PNetCDF techniques can be used for both input files and history/restart output files, the I/O server techniques are only applicable when writing files and so only output I/O is considered in most of what follows.

A. WRF Output Files

During a typical WRF simulation, both forecast history files and restart files are written periodically. While history files tend to be retained, typically all but the most recent restart file will be discarded.

In the configuration considered here, a separate NetCDF output history file is created by WRF for each of the two domains at specified output intervals (frames_per_outfile=1 in the input namelist). The larger Domain 2 creates the largest history file (19.8G per frame or output step); Domain 1 creates 7.5G per frame or output step. All output files are uncompressed to enable comparison of the I/O techniques, and all reported times are in seconds.

B. Serial NetCDF

To output a distributed array in WRF, the default I/O layer mentioned above (compile option –DNETCDF) gathers all of the data onto the master MPI rank 0 using a call to MPI

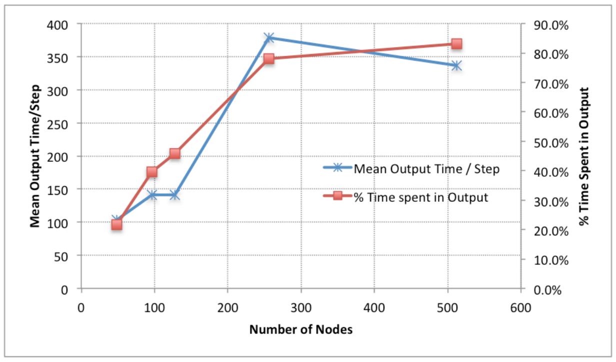

Figure 3: Serial NetCDF: Mean time per output step and % time spent in output (MPI only)

Gatherv, reconstructs the array and then writes it to disk using the standard, serial NetCDF library. All other MPI ranks block until the master has completed the write. The time taken to do all of this for each domain is written by WRF to standard output rsl.out.0000:

Timing for Writing wrfout_d01_2015-03-10_00_00_00 for domain 1: 5.53356 elapsed seconds

This metric is referred to as "effective" I/O time in what follows. Clearly, since it includes the time taken by the call to MPI Gatherv plus NetCDF formatting, it is not strictly the time taken to write the data to disk but it does represent the amount of wall-clock time attributable to producing the output.

For small WRF domain sizes and low rank counts, the serial NetCDF approach can be a reasonable option. However, given the dependence of this method on MPI_Gatherv and its serial nature, as MPI rank counts increase and domain sizes get larger, it becomes a huge bottleneck to performance. It is also the case that rank 0 can quickly run out of memory as global arrays are collected though using the Cray ALPS MPMD multiple-binary launch capability to place rank 0 on its own node can mitigate this issue. Fig. 3 is an extreme illustration of how much time writing output files can take, showing both mean time per output step and percentage of time spent in write. Note that although the effective output time is significant, largely due to the MPI_Gatherv overhead, the time taken to actually write the output files in all cases is constant and much smaller (at a rate of about 0.9GB/s).

One way to reduce the MPI_Gatherv bottleneck is to use MPI/OpenMP hybrid mode to reduce the number of MPI ranks per node, which pushes the bottleneck out to higher node counts. However, adding OpenMP threads only delays rather than removes the point at which serial NetCDF becomes unworkable, and we now consider the alternative I/O strategies available in the WRF software. The most straightforward alternative is parallel NetCDF.

C. *Parallel NetCDF*

As mentioned above, WRF contains an I/O layer (compile option –DNETCDF –DPNETCDF) implemented with PNetCDF, which is an extension of the NetCDF library that supports parallel I/O. PNetCDF is a collaborative effort between Argonne Labs and Northwestern University [6],[7]. On the Cray XC40, PNetCDF is implemented using Cray MPI-IO, which works closely with the Lustre parallel file system to align read and write operations to Lustre stripes. The Cray MPI-IO layer uses aggregators to write data in groups of PEs [8].

To use PNetCDF, the user must set *io_form_{history/restart/input}=11* in the input namelist file. It is also often necessary to set *nocolons=.true.*.

The combination of PNetCDF with the Cray MPI-IO layer means that the MPI ranks are aggregated into groups (the number of groups can be based on MPI-IO hints if provided by the user, but defaults to the Lustre stripe count of the output file) and then one aggregator from each group performs the write to file. This reduces both gather times and contention considerably, particularly since MPI-IO collective buffering attempts to assign one aggregator per OST and to make sure that each one is on a different node.

The Cray MPI-IO library has benefitted from extensive work over the past few years and is fully integrated with the Cray MPICH library and the Lustre file system structure (OST striping). It also provides some useful diagnostic tools to quickly get a handle on what the MPI-IO layer is doing on a per-file basis.

To use PNetCDF on the Cray XC40, the input and history (and/or restart) files are striped over as many OSTs as desired, up to the maximum on the filesystem. For smaller problem sizes, 4 to 8 stripes can be sufficient, though this increases for larger-sized problems. A rule of thumb is to use somewhere between sqrt(nprocs) and 0.5*sqrt(nprocs) OSTs up to the max configured, where nprocs is the number of MPI ranks being used; a number of OSTTs evenly divisible into nprocs is ideal. For the runs in this paper, 16 OSTs were available and 16 stripes were used. Various Cray MPI-IO environment variables can then be set to check whether and how the striping has worked.

At runtime, the MPI-IO library checks the Lustre striping of the output file (inherited from the run directory or individually specified as shown below) and assigns the same number of MPI-IO aggregators to that file. It automatically spreads the aggregators out across the nodes containing compute ranks as evenly as it can. The compute ranks are then shared among the aggregators so the I/O is performed collectively (i.e., in parallel).

The Cray MPIIO layer provides the environment variable MPICH_MPIIO_AGGREGATOR_PLACEMENT_DISPLAY, which when set writes to stdout the detail of how the aggregators have been placed among the compute nodes (NIDS):

```
Aggregator Placement for wrfinput_d01
RankReorderMethod=3
AggPlacementStride=-1
  AGG    Rank       nid
  ----  ------  --------
     0       0  nid00016
```

```
    1      434  nid00041
    2      868  nid00125
    3     1302  nid00221
    4     1736  nid00245
    5     2170  nid00269
    6     2604  nid00293
    7     3038  nid00317
    8       18  nid00349
    9      452  nid00373
   10      886  nid00773
   11     1320  nid00797
   12     1754  nid00821
   13     2188  nid00845
   14     2622  nid00869
   15     3056  nid00893
```

Instead of striping the output file before the run (via the lfs setstripe command), MPI-IO hints can alternatively be used to set the number of aggregators (or cb_nodes) used for each file. For example, the following setting ensures that any file starting with the string "wrfout" will be created and striped over 16 OSTs and thus that 16 aggregators will be used for writing it:

MPICH_MPIIO_HINTS= "wrfout:striping_factor=16"*

Subsequently, the runtime environment variable MPICH_MPIIO_HINTS_DISPLAY can be set to display whether or not a file has been opened via the MPI-IO layer and how it was striped (look for cb_nodes=16 to see that we are indeed using all the OSTs we requested):

```
PE 0: MPIIO hints for wrfoutput_d01:
cb_buffer_size         = 16777216
romio_cb_read          = automatic
romio_cb_write         = automatic
cb_nodes               = 16
cb_align               = 2
romio_no_indep_rw      = false
romio_cb_pfr           = disable
romio_cb_fr_types      = aa
romio_cb_fr_alignment = 1
romio_cb_ds_threshold = 0
romio_cb_alltoall      = automatic
ind_rd_buffer_size     = 4194304
ind_wr_buffer_size     = 524288
romio_ds_read          = disable
romio_ds_write         = disable
striping_factor        = 16
striping_unit          = 1048576
romio_lustre_start_iodevice = 0
direct_io              = false
aggregator_placement_stride = -1
abort_on_rw_error      = disable
cb_config_list         = *:*
romio_filesystem_type = CRAY ADIO:
```

Another very useful setting to use at runtime is MPICH_MPIIO_STATS=1:

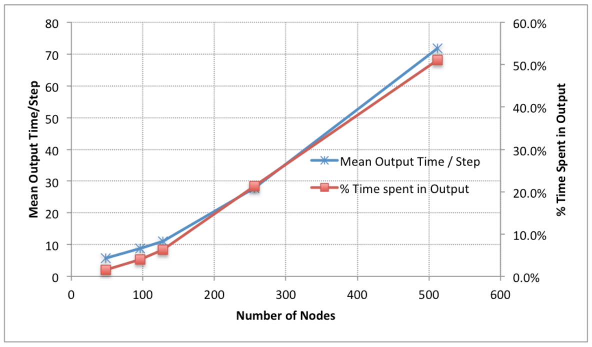

Figure 4: Parallel NetCDF: Mean time per output step and % time spent in output (MPI only)

```
+-------------------------------------+
| MPIIO read access patterns for
| wrfoutput_d01
|   independent reads        = 1
|   collective reads         = 457452
|   independent readers      = 1
|   aggregators              = 16
|   stripe count             = 16
|   stripe size              = 1048576
|   system reads             = 7727
|   stripe sized reads       = 7512
|  total bytes for reads = 7964753971
|               = 7595 MiB = 7 GiB
|   ave system read size     = 1030769
|   number of read gaps      = 1
|   ave read gap size        = 1
| See "Optimizing MPI I/O on Cray XE
| Systems" S-0013-20 for explanations.
+-------------------------------------+
```

To see even more performance information, try the setting MPICH_MPIIO_STATS=2, which provides a timeline viewable in the Cray Apprentice2 tool and data from which bandwidth charts can be generated, among other features.

Fig. 4 illustrates the effect on write times and percentage when the serial NetCDF in Fig. 3 is replaced by pNetCDF. Again, no OpenMP threads were used. Each aggregator writes its data to file at close to 0.9GB/s.

PNetCDF is a great technique to use as an alternative to serial NetCDF and can give good performance. However, as the number of MPI ranks continues to increase, it too starts to take an unacceptable percentage of total run time and we again look for a better alternative. As a rule of thumb, once the time taken to write a single output frame increases beyond two or three seconds, it becomes advantageous to investigate the use of quilt servers.

D. *Asynchronous I/O (Quilt Servers)*

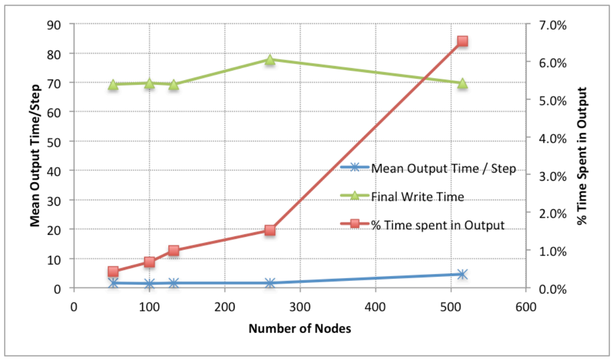

Figure 5: Quilt Servers: Mean time per output step and % time spent in output (MPI only, 4 extra nodes assigned to quilt servers)

As described above, whether using serial NetCDF or PNetCDF, all MPI ranks have to wait for the master rank to finish writing history data to disk before proceeding with their computation. When this wait time starts to place a heavier overhead on the total runtime than is acceptable, it makes sense to set aside one or more ranks (known as quilt or I/O servers) to deal exclusively with the I/O, so that once the compute (client) ranks have sent their data to these I/O servers, they can continue with their work while the data is formatted and written to disk in the background (asynchronously). Whether or not this technique is appropriate depends on the amount of output time taken by PNetCDF and the number of compute ranks being used, since it can be inefficient to dedicate too high a proportion of ranks to I/O only.

WRF provides such I/O server functionality (compile with –DNETCDF only), enabling the user to select at runtime via the input namelist_quilt the number of groups of I/O servers to allocate (nio_groups) and the number of I/O ranks per group (nio_tasks_per_group). More than one group of I/O servers is necessary if there is more than one domain, or if the time between subsequent output steps is short, since each group can work on only one output frame at any one time, and if too few groups are available the WRF simulation will stall waiting for the next group to become free.

The compute tasks are numbered from 0 upwards in MPI_COMM_WORLD and their total number nprocs = nproc_x * nproc_y, where the processor decomposition (nproc_x, nproc_y) gives the number of compute processors along the x and y axes, respectively.

The total number of I/O servers (whose first rank number immediately follows the final compute rank number in MPI_COMM_WORLD) is given by nio_groups * nio_tasks_per_group, where nio_tasks_per_group cannot exceed nproc_y. WRF attempts to match each I/O server with compute tasks in the east-west rows, and ideally (though this is not mandatory) nproc_y should be an exact multiple of nio_tasks_per_groups. This ensures that each I/O server rank handles the same number of compute ranks, which should lead to the best balance, but it also has important implications for the parallel quilt method in the following section. Together these groups of compute ranks and I/O server are mapped onto an MPI communicator; the gather of data from compute ranks onto the I/O server is implemented using an MPI_Gatherv across that communicator. See [9] for a nice illustration of how ranks are assigned to I/O servers.

One I/O group is selected at runtime to handle the output to a particular file, and within that I/O group the servers forward their data to a designated root server, which performs the actual write.

The quilt server technique essentially reduces significantly the output overhead seen by the compute ranks in waiting for a write to complete (for all output steps aside from the final frame). The standard rsl.out.0000 output mentioned above now should show an effective output time of well under a second, since that is what a typical compute rank now sees:

Timing for Writing wrfout_d01_2015-03-10_00_00_00 for domain 1: 0.53356 elapsed seconds

The actual time taken for the gather and write is of course well above this effective metric. See Appendix A for some practical hints on using quilt servers on the Cray XC system.

Once I/O servers have been introduced, the performance bottleneck associated with I/O is no longer the sending of data to disk (the time to do this remains unchanged, but is asynchronous), but the gather of data from compute PEs onto the I/O servers.

The disadvantage of this technique can be that the quilt server writes are still slow (somewhat equivalent to the speeds seen in serial NetCDF though with less time spent in gather due to fewer ranks participating), though overlapped, and the write of the output at the final step can overwhelm a short simulation. Fig. 5 shows the case in the previous two figures where now four nodes of I/O servers (two groups of 16 servers, 8 allocated per node) have been added in each instance. Also illustrated here is the time to write the final output. The percentage of time spent in I/O is now well below 10%, ignoring the final output step. Each root I/O server writes to disk at a rate of around 0.9GB/s.

As mentioned above, the write time is now overlapped with the computation meaning that (as is the case here) if the integration time between output steps is longer than the time taken to write a file, only one group of I/O servers per domain will be required and the only apparent overhead will be in the time to write out the files at the final step. However, if the simulation is short this final output time could well overwhelm the total runtime and perform less well than PNetCDF. If the integration time between output steps is much shorter than the time to write a file (which can happen as more and more MPI ranks are assigned to a computation, or if MPI/OpenMP hybrid mode is used, or if the output files themselves are much larger than in this particular case), more and more groups of I/O servers can be required, meaning increasing numbers of I/O nodes need to be assigned to the simulation. One scenario where this might be a problem is in severe thunderstorm forecasting, where output might be required every simulation minute and multiple ensembles are needed.

However, what if we could reduce the amount of time taken to write an output file as well as overlapping that write

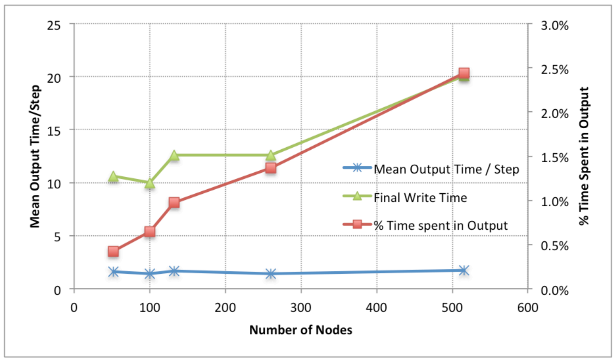

Figure 6: Mean time per output step and % time spent in output (MPI only, 4 extra nodes assigned for quilt servers)

with the computation? That would then minimize the number of groups of I/O servers necessary to perform the output, and also reduce the final output step time. Fortunately, Andrew Porter from the STFC Daresbury Laboratory has implemented in WRF just such a technique, which we describe in the following section.

E. *Asynchronous I/O with PNetCDF*

If the WRF code is compiled using the predefines –DNETCDF –DPNETCDF –DPNETCDF_QUILT and the asynchronous I/O servers are requested in the namelist input file with *io_form_history=11*, instead of serial I/O, the writes to disk are performed in parallel using MPI-IO so we gain the combined benefits of both techniques.

As mentioned in the previous section, it is ideal if the number of I/O servers per group is evenly divisible into nproc_y. This clearly leads to the most balanced scenario, but it turns out it is not functionally necessary in all cases, particularly when the domain sizes are small and the number of I/O servers is also small (say, 8 or below). However, this becomes much more important when the number of servers is 16. See Appendix B for further discussion.

Fig. 6 shows the improvement seen when combining PNetCDF with the asynchronous quilt servers. As in the previous section, four extra nodes are assigned for two groups of 16 I/O servers. The percentage of time spent in I/O is now well below 3% and the final step overhead has dropped significantly. This means that we could write output much more frequently or scale up the MPI rank counts much higher while still only requiring two groups of I/O servers. We can also better handle short runs that might otherwise be overwhelmed by the final output writes.

The actual write rate of each I/O server group, though performed in parallel via a collective MPI-IO call, is somewhat slower than in the PNetCDF-only case, since the aggregator ranks are now packed onto very few I/O-only nodes rather than spread out among the compute nodes, so their I/O is then also concentrated through very few OSTs.

F. *Summary Comparison of all Four Methods*

Figs. 7--10 summarize the detail of the previous four sections, showing how the proportion of time spent in I/O is

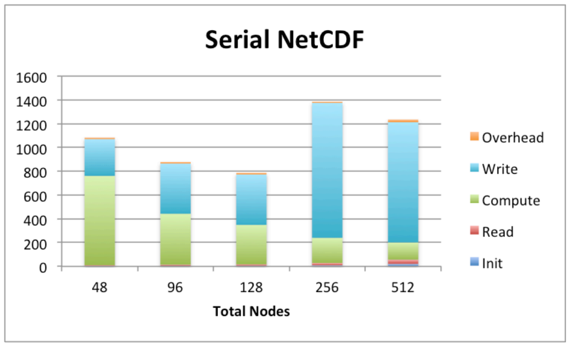

Figure 7: Serial NetCDF: Proportion of time spent in compute and I/O

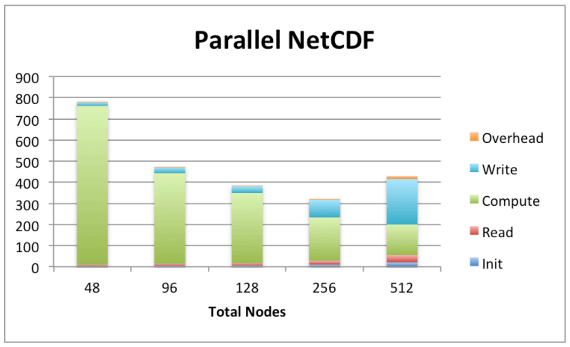

Figure 8: Parallel NetCDF: Proportion of time spent in compute and I/O.

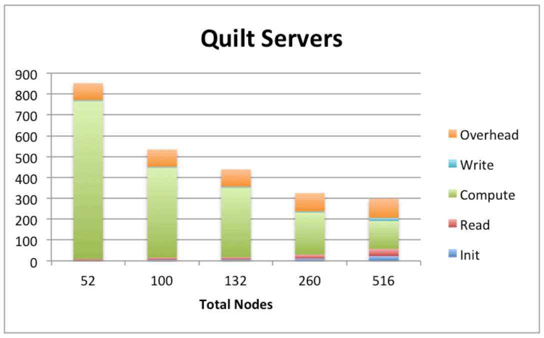

Figure 9: Quilt Servers: Proportion of time spent in compute and I/O

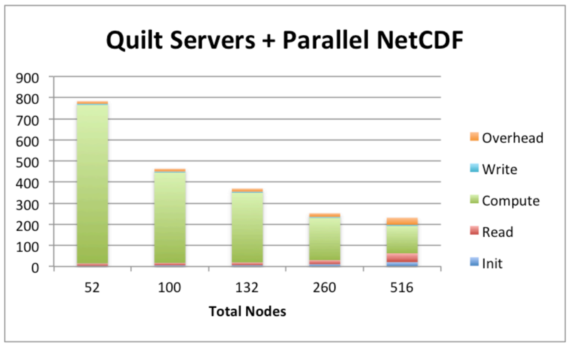

Figure 10: Parallel NetCDF + Quilt Servers: Proportion of time spent in compute and I/O

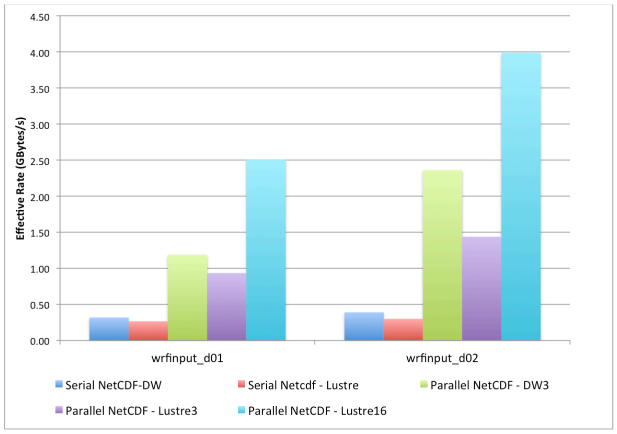

Figure 11: Effective input I/O rates, in GBytes/sec, using 3 DataWarp SSDs and Lustre (3 and 16 OSTs). Includes data collection and formatting times for serial and parallel NetCDF

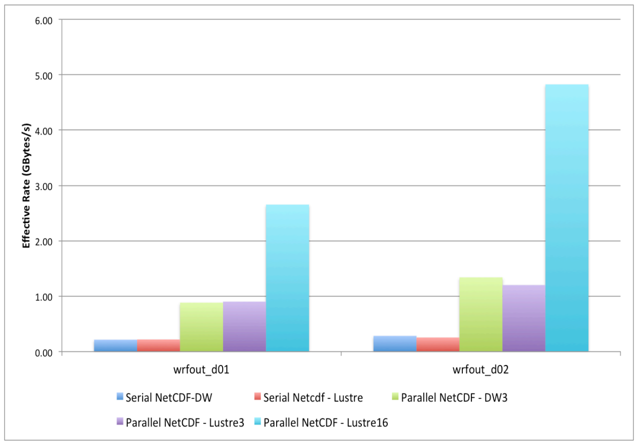

Figure 12: Effective output I/O rates in GBytes/sec, using 3 DataWarp SSDs and Lustre (3 and 16 OSTs). Includes data collection and formatting times for serial and parallel NetCDF

reduced as the four I/O methods are implemented in turn. The read and write times shown are effective times as seen by compute rank 0, and the actual time for the final step write is included in the "overhead."

III. CRAY DATAWARP

A recently introduced data storage offering, Cray DataWarp [5], provides a new, cost effective file-based storage option. It is comprised of commercial SSD hardware and Cray-developed software and is installed on Cray XC service nodes connected directly to the Cray XC system's Aries high-speed network (see Fig. 1). This section discusses the use and performance of DataWarp using the same WRF benchmark case as in previous sections.

There are several ways to use DataWarp storage, currently controlled per job, through a supported workload manager (WLM) such as SLURM or Moab/Torque. A typical use is as a scratch file space where any data located on DataWarp exists only for the length of the job that allocated the space. Any files needed as input to the job or which are written by the job and need to be permanently saved are staged by the WLM via user script directives. This is the method used for this WRF benchmark. WRF initial conditions, wrfinput*, are prestaged to DataWarp and forecast history files, wrfout*, written to DataWarp during the forecast run are staged to permanent Lustre storage at the end of the job. Note that there are cases in many weather forecast applications, including WRF, where many temporary files, such as preprocessing output or periodic restart files, will not need staging to other storage. The following directives to control DataWarp allocation and staging were added to WRF run script. These directives are parsed by the WLM when the batch job is submitted and take care of allocation of the proper amount and type of storage requested as well as staging of any requested files:

#DW jobdw type=scratch access_mode=striped capacity=1150GiB
#DW stage_in type=file source=INPUT/wrfinput_d01 destination=$DW_JOB_STRIPED/wrfinput_d01
#DW stage_out type=file destination=OUTPUT/wrfout_d01_2015-03-10_00_00_00 source=$DW_JOB_STRIPED/wrfout_d01_2015-03-10_00_00_00

In the first directive above we allocate 1.15 TB of scratch disk storage striped over three DataWarp nodes, 383 GB per node (the allocation chunk size is defined at installation time). The next two directives specify files to be staged in before the job runs and staged out when it completes.

For this study, three DataWarp nodes, each comprised of two Intel P3608 SSDs, were used for reading WRF initial conditions and writing forecast history. Each Intel P3608 is capable of over 5 GB/s when reading and over 3 GB/s writing, giving each DataWarp node a peak I/O bandwidth of approximately 8 GB/s read and 6 GB/s write. With multiple DataWarp nodes, the parallel I/O techniques and WRF features described in earlier sections also apply.

Figs. 11 and 12 show effective input and output I/O rates as seen from the WRF computation processes. As mentioned earlier, effective rates are lower than actual rates as they include time needed for data collection and formatting. The serial NetCDF and PNetCDF methods (see sections B and C above) were evaluated using the three available DataWarp nodes and then compared to using Lustre with a striping factor of three to ensure a consistent number of parallel writers/readers. A comparison with 16 Lustre stripes is also given.

For serial NetCDF, the actual Posix read and write rates for the DataWarp SSDs were in the range 1.25 to 1.75 GB/s, compared to just under 1GB/s for the Lustre hardware. Note that under ideal conditions, maximum write rates are 6GB/s and 4GB/s for one DataWarp node and one Sonexion 2000 OST, respectively.

As can be seen in the figures, using the parallel I/O features of WRF with Cray DataWarp gives better performance than Lustre for the same number of aggregators. Using more DataWarp nodes (similar to adding more Lustre stripes) will decrease the I/O overhead, as seen by the application, even further.

IV. Conclusion

In this paper we have addressed the fact that, for many applications, I/O overhead can limit scaling to higher numbers of cores and place an unnecessary cap on potential throughput. Cray XC systems have the capability to nearly eliminate these restrictions when used in conjunction with parallel I/O features implemented within an application.

We have considered the various I/O methods available in the widely-used WRF software applied to a realistic forecast case and have demonstrated how to use the parallel I/O features of the Cray XC40 system to optimize I/O performance; these features become ever more critical as forecast domain sizes continue to increase, leading to the need for higher numbers of MPI ranks for faster computation and to larger input and output files. While the more familiar PNetCDF and quilt server techniques are good options in a range of WRF scenarios, the advantages of combining the two methods along with the efficient implementation of Cray MPI-IO enable WRF users to scale their forecast simulations to higher node counts and larger problem sizes than ever before. While the default single-writer NetCDF method limits the MPI-only scaling of this particular forecast case to approximately 128 nodes, by switching to the superior parallel I/O options the forecast case scales to at least 512 nodes, cutting the time-to-forecast by over 75%.

While the scaling of the Lustre filesystem is clearly demonstrated, it can be seen from these initial results that even very few Cray DataWarp SSDs can provide equivalent or superior effective I/O performance compared to the same number of Lustre OSTs, at a much reduced cost and with higher reliability. Early results from WRF experiments made at the King Abdullah University of Science and Technology (KAUST) in Saudi Arabia, where the new Cray XC40 system Shaheen II has both a Sonexion 2000 Lustre filesystem with 144 OSTs and also 268 DataWarp nodes, agree with the initial impression that the comparison between equal numbers of OSTs and DataWarp nodes shows roughly equivalent performance, with DataWarp in general performing better [10]. The KAUST analysis has shown good WRF I/O scaling so far on up to 100 DataWarp nodes, with investigations still ongoing.

V. Appendices

A. *Quilt Servers on the Cray XC40*

In this section we give a few tips on using quilt servers on the Cray XC system. Initially, try setting the number of I/O server groups to the number of output files per output step (so if there are four nested domains and one history file each per step, use four groups). If the “Timing for Writing” metric in the standard rank 0 output file is still well over a second, this indicates that a file had to wait for an I/O group to become available before proceeding, in which case try more groups until all the writes are overlapped according to the rank 0 output.

The number of I/O servers required per I/O server group depends on the domain sizes of the problem under consideration. Eight I/O servers per group is a good starting point, but for the problem under consideration, eight was not enough and so 16 were used. For reference, the error message observed when eight servers were tried was

```
---------- FATAL CALLED -----------
FATAL CALLED FROM FILE:<stdin> LINE: 685
Possible 32-bit overflow on output
server. Try larger nio_tasks_per_group
in namelist.
```

If a WRF run using quilt servers does fail, be sure to look at the specific rsl.error files for the I/O server ranks, as their error messages will not show up in rsl.error.0000.

Since the I/O servers gather data from many compute ranks, they require more memory than the compute ranks (generally a similar requirement to the serial rank 0 collector) and so cannot be fully packed onto nodes with large numbers of cores. For the runs in this paper we used 8 ranks per node, spread out evenly across both numa nodes (sockets). In order to ensure that the compute ranks could be differently packed on the nodes than the I/O server ranks, the MPMD (multiple program multiple data) facility of the Cray ALPS application launcher was used; for example,

aprun -j1 -ss -n 9216 –N36 -d 1 ./wrf.exe :
-j1 -n128 -N8 -d 4 ./wrf.exe

Previous investigations [3] into spreading the I/O ranks among the compute ranks have shown no real advantage to the write performance and this also messes up the careful compute rank ordering that can be done to minimize the halo communication times. However, it is advantageous to spread each I/O group out across the nodes used for the I/O servers rather than concentrating each group on its own node/s. This can be achieved via rank reordering within the I/O servers themselves.

To achieve good quilt server performance on the Cray XC40, set the environment variable MPICH_COLL_SYNC = MPI_Gather.

Note that if the WRF code has been compiled with NETCDF4 options, the quilt server method (and, in fact, the serial NetCDF method) will write a compressed output file, which can take a very long time. The namelist setting *use_netcdf_classic=.true.* avoids the use of the NETCDF4 compression.

Other useful hints on running WRF can be found in [11] and [12].

B. *Quilt Servers with PNetCDF on the Cray XC40*

This section describes some observations made when experimenting with PNetCDF and I/O servers. When PNetCDF quilting is enabled, each of the compute ranks assigned to an I/O group holds one “patch,” and an attempt is made to stitch those patches together before the data is finally written to disk. If each I/O server in the group contains the same number of stitched patches, then all servers do the same number of writes and collective MPI-IO is used to write the data. However, if the I/O servers contain a different number of stitched patches to one another, any leftover patches will have to be written in independent mode (i.e., serially). This is immediately detectable during a WRF simulation from the rank 0 standard output.

If nproc_y exactly divides the nio_tasks_per_group number, then each server will have an identical number of compute PEs and we are guaranteed that when their patches are stitched together, we end up with the same number of stitched patches on each I/O server and the writes are performed collectively. However, if this is not the case, there is a chance that different servers will end up with a different number of stitched patches. When there are fewer (8 or less) I/O servers per group, it appears that even with nproc_y not evenly divisible into nio_tasks_per_group, we are still very likely to end up with only one patch on each server and thus the write will still be performed collectively. However, with 16 I/O servers per group, we very often end up with two or three patches on some servers, and one on the rest, and the write speeds drop down to serial. This was very easily diagnosed on the Cray XC40 by setting the environment variable MPICH_MPIIO_HINTS_DISPLAY, which showed the output file being first opened in collective mode (cb_nodes=16) but then subsequently opened again in independent mode (cb_nodes=1).

This information enabled us to find a bug in the parallel NetCDF quilt code that used the default domain decomposition rather than the user-specified domain decomposition when assessing whether or not to use collective I/O. The fact that this had not been detected before in versions 3.6.1 or 3.7.1 of WRF indicates that perhaps this technique is not very widely used in the community; we believe that it deserves much more attention. The source code has been fixed in the recent WRF v3.8 release.

Acknowledgment

We thank George S. Markomanolis at KAUST for generously sharing the details of his early work with DataWarp.

References

[1] WRF website, wrf-model.org.

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

[3] A. Porter and M. Ashworth, “Configuring and Optimizing the Weather Research and Forecast Model on the Cray XT,” Cray User Group Proceedings, 2010.

[4] Cray Sonexion Storage, www.cray.com/products/storage/sonexion.

[5] “DataWarp User Guide S-2558-5204,” from Cray Inc. http://docs.cray.com/books/S-2558-5204/S-2558-5204.pdf.

[6] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher and M. Zingale, “Parallel netCDF: A High-Performance Scientific I/O Interface.” in Proceedings of Super Computing '03, November 2003.

[7] Parallel NetCDF, http://cucis.ece.northwestern.edu/projects/PnetCDF.

[8] “Getting Started on MPI I/O,” http://docs.cray.com/books/S-2490-40/, S-2490-40.

[9] E. Kemp, “WRF Quilting and Decomposition Notes,” from NASA Climate Downscaling Project Meeting, 2015, https://modelingguru.nasa.gov/servlet/JiveServlet/downloadBody/2560-102-1-6323/nuwrf_quilting_20150302.pdf

[10] George S. Markomanolis, KAUST, personal communication, March 2016.

[11] J. Michalakes and A. Porter, “Opportunities for WRF Model Acceleration,” 13th Annual WRF Users Workshop, 2012.

[12] P. Johnsen, “Tuning WRF Forecasts on the Cray XT (Getting the most out of your XT),” 2010 Alaska Weather Symposium at the University of Alaska in Fairbanks.