Performance Study of Parallel NetCDF in ROMS

MuQun Yang
Robert E. McGrath
Mike Folk
July 6, 2004 (revised August 27, 2004)

I. Introduction

This is the second report about Parallel NetCDF implemented by Argonne National Lab and Northwestern University [1-3] and the integration of Parallel NetCDF inside Regional Ocean Modeling System (ROMS) [4]. The first report, "Investigation of Parallel NetCDF with ROMS," presented an understanding of Parallel NetCDF and ROMS [5]. This report presents performance studies of Parallel NetCDF inside ROMS. This report may help ROMS users to choose whether to use Parallel IO in their own case. Readers should refer to the first report for more information about Parallel NetCDF.

II. Regional Ocean Modeling System (ROMS)

The Regional Ocean Modeling System (ROMS) is a free-surface; hydrostatic, primitive equation ocean model that uses stretched, terrain-following coordinates in the vertical and orthogonal curvilinear coordinates in the horizontal [4]. The ROMS model can be run in parallel using the MPI standard [6].

All input and output for the model is via NetCDF [7]. This paper reports investigations using parallel netCDF [1-3] to implement parallel IO in ROMS.

III. Why Parallel NetCDF?

In our earlier report, we discussed the possible advantages of using Parallel NetCDF in ROMS [3].

As computations scale up to use MPI on large multiprocessors, NetCDF IO in MPI applications becomes an IO bottleneck and may also exceed the memory capacity assigned to the current IO node. In this study, these limits were observed for ROMS.

The parallel NetCDF library addresses these problems by providing a version of NetCDF that uses MPI-I/O. In Parallel NetCDF, "[a]ll processes perform I/O operations cooperatively or collectively through the parallel NetCDF library to access a single netCDF file. This approach […] both frees the users from dealing with details of parallel I/O and provides more opportunities for employing various parallel I/O optimizations in order to obtain higher performance." [1] This is the obvious reason for all parallel I/O libraries.

IV. A Parallel NetCDF-ROMS History File Writer

This study used ROMS version 2.0. The model has an option for us to use MPI for computation and uses sequential IO with Unidata NetCDF 3.5. We added another option to output the ROMS history file with parallel NetCDF. To maintain ROMS better, we tried to isolate source files including parallel NetCDF function calls.

The changes made to ROMS are listed in the Appendix.

Note that the history file is only one of many files used by ROMS. A full implementation would need to make additional changes to ROMS to read the history file and to support parallel IO for other input and output files.

V. Performance Report

This paper reports some limited performance experiments using ROMS with parallel NetCDF.

The performance tests had been done on NCSA IBM P690 and NCAR IBM SP cluster. Both platforms are Symmetric Multiprocessor Systems, running IBM AIX, with the General Parallel File System. The NCSA IBM P690 has two nodes with sixteen IBM Power4 CPUs per node. Each application may use only one node.

The NCAR IBM SP cluster has 293 nodes, each node has four IBM Power3 CPUs. Multiple nodes can be used at NCAR IBM SP cluster. In this study, we used 4, 8, 16, 32 and 64 nodes of the cluster.

All the performance results are based on three experiments we made on these two platforms. Since we had to share computer resources with other users, we found that the data output time changed significantly under different model runs; especially on NCSA IBM P690. For each experiment, three runs were done and the best (lowest) wall-clock time was chosen.

These experiments compare the performance of model runs using sequential NetCDF and parallel NetCDF. In addition to the use of sequential or parallel IO, three other variables determine the IO performance of the model:

1. the size of the records that are written
2. the domain decomposition, which determines the contiguity of the writes to the disk
3. the number of processors used in the run

In addition, different machine architectures affect IO performance.

In each experiment, we set the timestep to 2, 4, 6, 8, 10 and made 5 model runs on each platform. For each timestep, the model wrote the NetCDF output into the history file and several comparatively small files. In the parallel component of both experiments, we used parallel NetCDF to generate the history file but serial NetCDF to generate the other files.

The history file include several variables that represent the physical model in a four dimensional array. The grid is organized in a FORTRAN array with dimensions:

1. latitude
2. longitude
3. depth
4. time

The computation uses a three dimensional array in memory (latitude, longitude, depth). This grid is written at a selected time steps.

When the model is run in parallel, the three dimensional grid is divided among the processors, so that each processor has part of the grid in its memory. The distribution of data is controlled by the *domain decomposition*.

The domain decomposition partitions the data according to latitude and longitude: each of the N processors is assigned a three dimensional slice with 1/N of the data. **Figure 1** shows an example of how the grid can be decomposed, showing the area assigned to two of the processors, P_1 and P_2. In **Figure 1**a, the processors are assigned a 1xN slice of the space, in **Figure 1**b, each is assigned a 2xN/2 slice, and in **Figure 1**c, the processor is assigned a 4xN/4 slice. Each of the different domain decompositions have equivalent amounts of data, but the data is from different regions of the array so the I/O patterns are substantially different. Figure 2 illustrates how the different decompositions affect where the data is written on disk.

In the case of sequential IO, the processors must pass the data to and from a single IO process. When data is written, the IO process assembles data from all the processors to create the whole three-dimensional array in the local memory of the single IO processor, and then writes the array to disk.

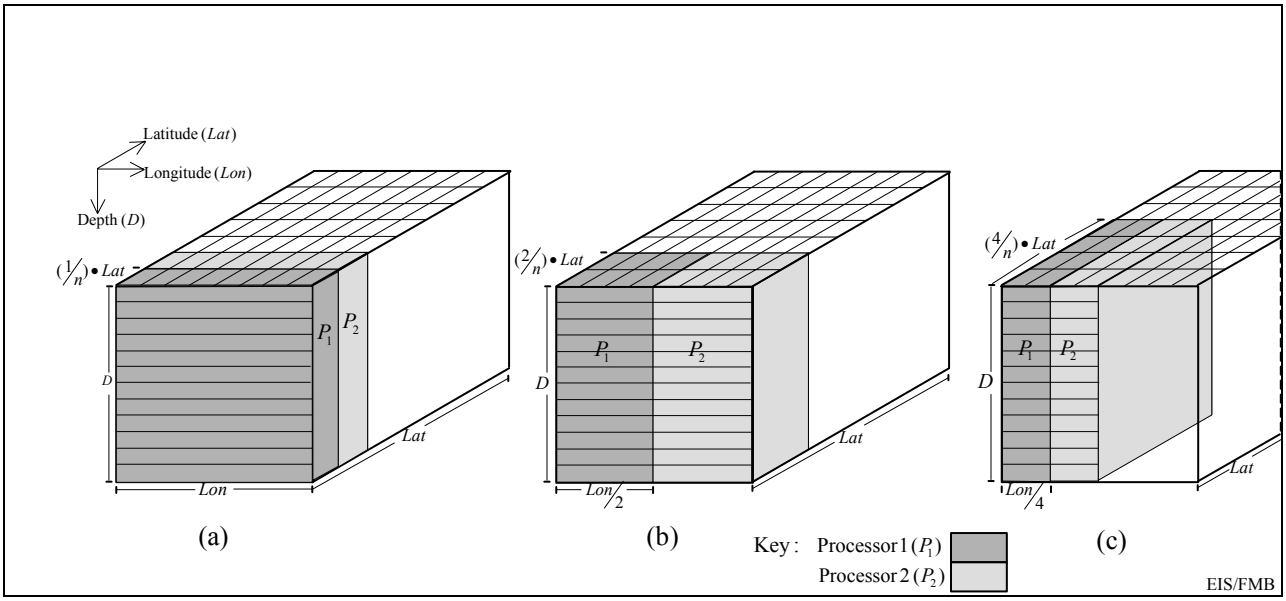


Figure 1. The region of the grid assigned to processor 1 under three different domain decompositions.

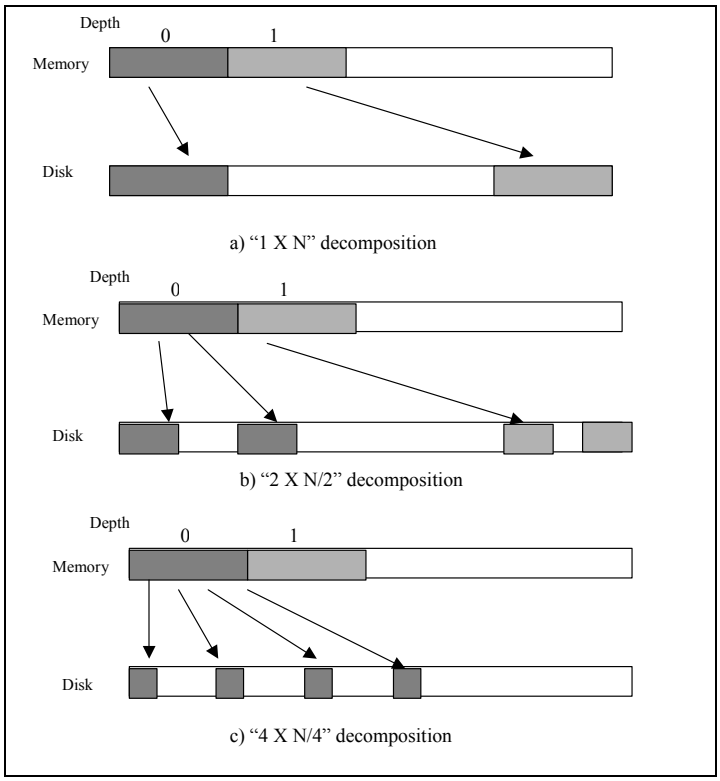


Figure 2. Illustration of the effect of different domain decompositions.

In the case of parallel IO, each processor writes its own part of the data, and MPI-IO coordinates the writes. In this case, the data is written to several parts of the file, depending on the decomposition. The amount of data per write also depends on the decomposition, as illustrated in Figure 2.

In both cases, a larger number of processors requires more network traffic and more overhead to coordinate the IO.

In this report, we called the amount of data written at each timestep one *unit* of output data. This data includes small variables and several problem sized arrays (i.e., latitude, longitude, depth). The amount of data written by each processor depends on the problem size, the number of processors, and the domain decomposition. Also, the number and size of writes depends on the decomposition.

Experiment 1: Output time for different maximum record sizes

Figure 3 - Figure 5 compares the output time between Parallel NetCDF and Sequential NetCDF for the three different sizes of output (Table 1). These runs had the same domain decomposition, (1 X 16). All runs were on the NCSA IBM P690 with 16 processors.

Table 1. Maximum Record Size for Experiment 1

Size	Number of elements in the largest record
Small size	82*80*16 = 104,960
Medium size	246*240*16 = 944,640
Large size	656*640*16 = 6,717,440

In the case of sequential NetCDF, each processor sends its data to a single IO processor, which then writes the whole array to disk. In this case, the IO time is primarily the time to execute a single write, for the whole array. This time includes the time for the network data transfer to the IO processor, followed by the write to the disk.

In contrast, for the arrays written with Parallel NetCDF, each processor writes data from its local memory to the file. This requires in 16 writes to the disk, one for each plane of depth. Each write is one plane of data, e.g., 80*82 elements for the small case. The MPI-IO library and parallel file system coordinate the writes from multiple processors.

With the small and medium record size, sequential NetCDF outperforms Parallel NetCDF. For the large record sizes, sequential and parallel are comparable (Figure 5). The smaller record sizes gives relatively poor performance with parallel IO because the amount of data written by each processor is not enough to overcome overhead of the message passing.

Figure 3. Comparison of Parallel and Sequential NetCDF.

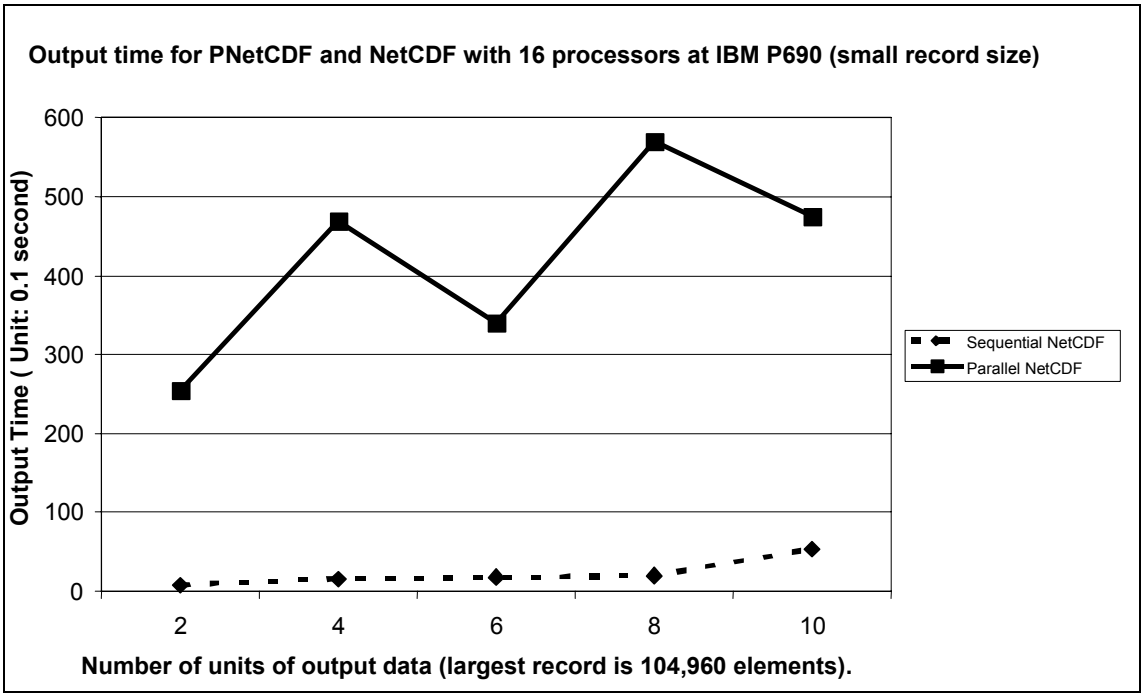


Figure 4. Comparison of Parallel and Sequential NetCDF (medium).

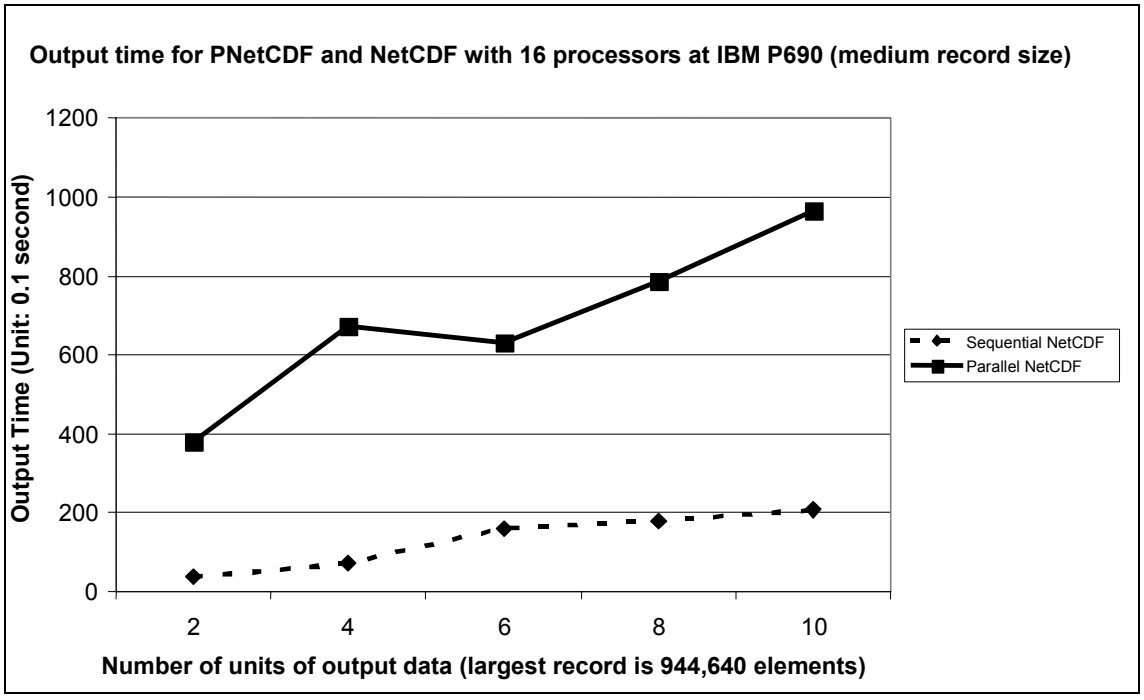
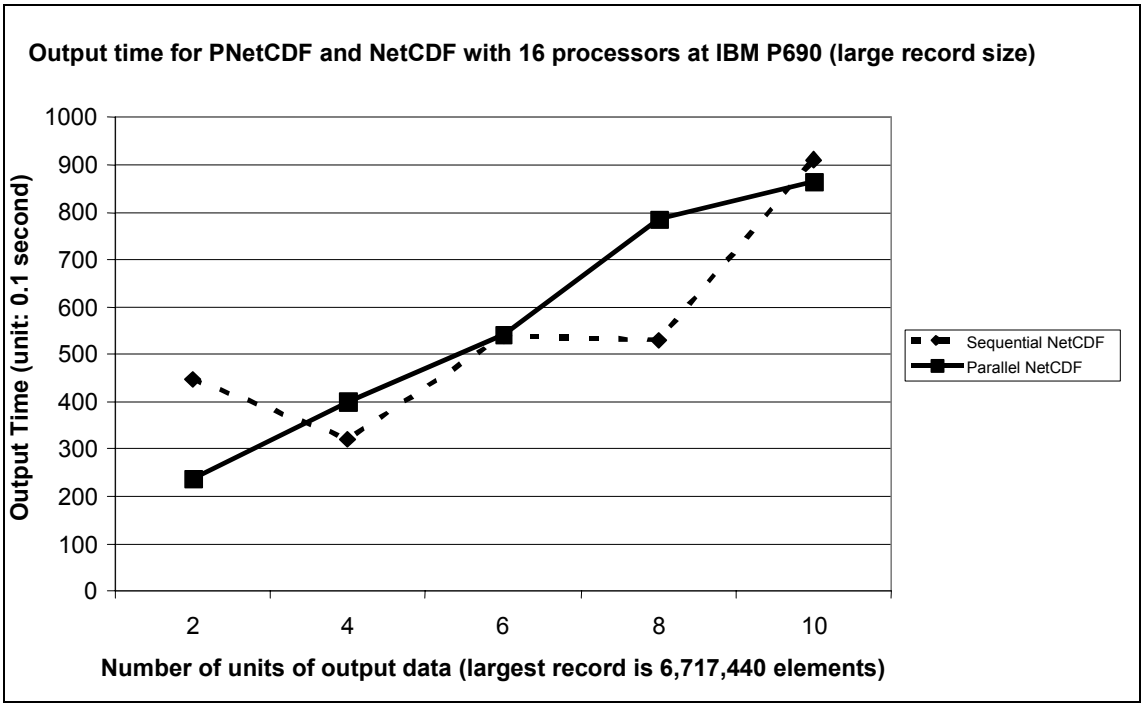


Figure 5. Comparison of Parallel and Sequential NetCDF (large).



At the time of this report, no FORTRAN 64-bit Parallel NetCDF library was available on IBM SP systems. Therefore, the experiment was required to use the 32-bit Parallel NetCDF library. The largest record size in Figure 5 is the maximum size that can be created with the 32-bit library. As long as this restriction exists, using ROMS with Parallel NetCDF won't get better performance than with the sequential NetCDF on the NCSA IBM P690. Using ROMS with

parallel NetCDF may outperform using ROMS with sequential NetCDF on other systems, especially those that support a 64-bit Parallel netCDF library.

Experiment 2: Output time with different domain decompositions

This experiment used three different domain decompositions of the model data. Table 2 shows the different domain decompositions used. As explained in Section IV above, in each case, the total amount of data per write is the same, but the decomposition means the data is more or less contiguous on disk.

Figure 6 and Figure 7 compare the output time of Parallel NetCDF and Sequential NetCDF with the different domain decompositions on the NCSA IBM P690.

Figure 6 shows that the different domain decomposition does not affect the performance for sequential NetCDF IO. Since the sequential IO is collected in memory and written in a single write, the decomposition only affects the size and number of transfers to the array in the memory of the IO process. The effect of the decomposition is small relative to the time to write the array to disk.

Table 2. Domain Decompositions Used for Experiment 2 (The depth is 16 in the experiment.)

Domain decomposition (I*J)	Contiguous Regions on Disk
1*16	16
2*8	160
4*4	320

Figure 6. Performance of sequential NetCDF with different decompositions.

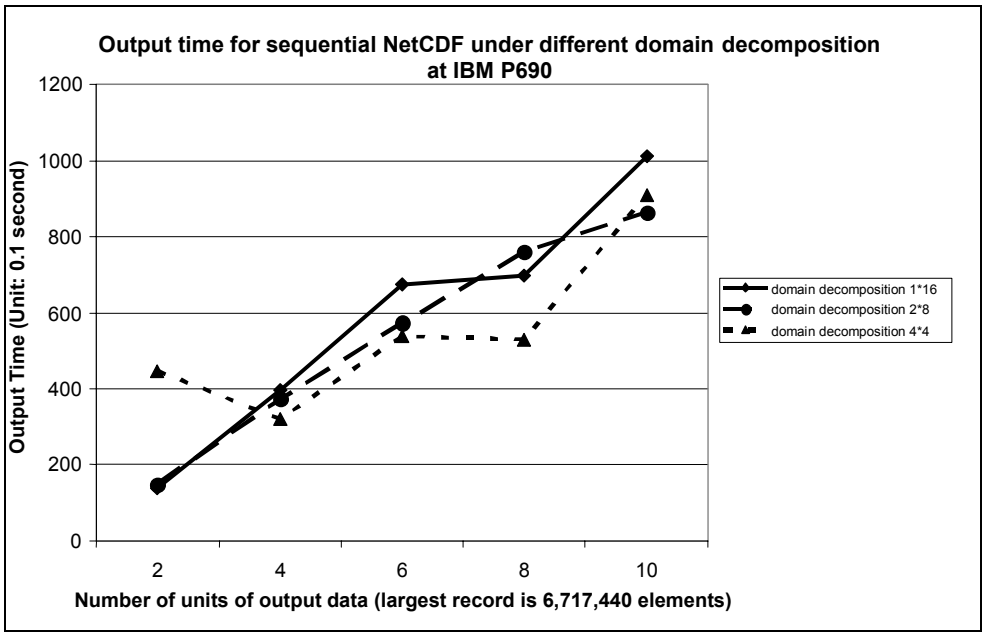


Figure 7. Performance of parallel NetCDF with different decompositions.

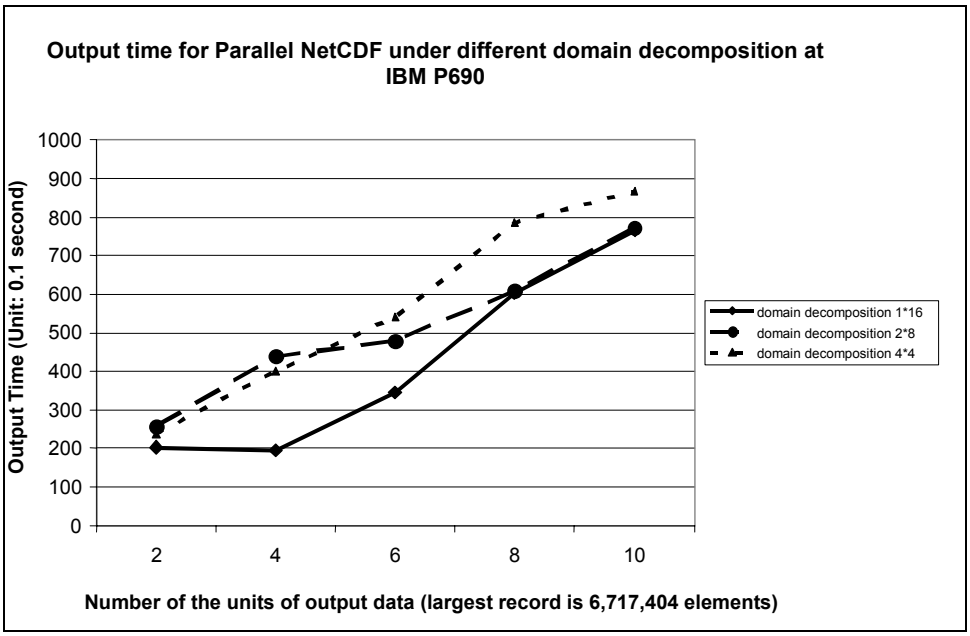


Figure 7 shows the performance for Parallel NetCDF. The more contiguous (1*16) domain layout gives better performance than less contiguous (4*4) domain layout. However, the difference is surprisingly small, considering that the latter would require 20 times as many disk writes, as the former if the writes were independent MPI-IO function calls. It appears that the MPI-IO collective write is very effective in these experiments.

These results suggest that Parallel NetCDF handles different domain decompositions very efficiently, so users can choose their domain decomposition according to their own computational requirements.

Experiment 3: Output time with different numbers of processors

The third experiment measured the performance of sequential and parallel output with different numbers of processors on NCAR's IBM cluster. Figure 8 -

Figure 10 compares the output time between parallel NetCDF and sequential NetCDF with 16, 32, and 128 processors, in 8, 16, or 64 nodes with 2 processors per node.

In each case, the same output was generated, i.e., the same number of bytes was written to disk. Each processor writes part of the data, so increasing the number of processors decreased the amount of data written by each processor. The record size used in this test is the maximum that can be created with the 32-bit library.

In this architecture, Parallel NetCDF performs better than sequential NetCDF for any number of processors. Figure 11 compares the output time for sequential NetCDF to generate the data with different number of processors. Figure 12 shows the parallel NetCDF runs.

Figure 8. Comparison of Parallel NetCDF and NetCDF with 16 processors.

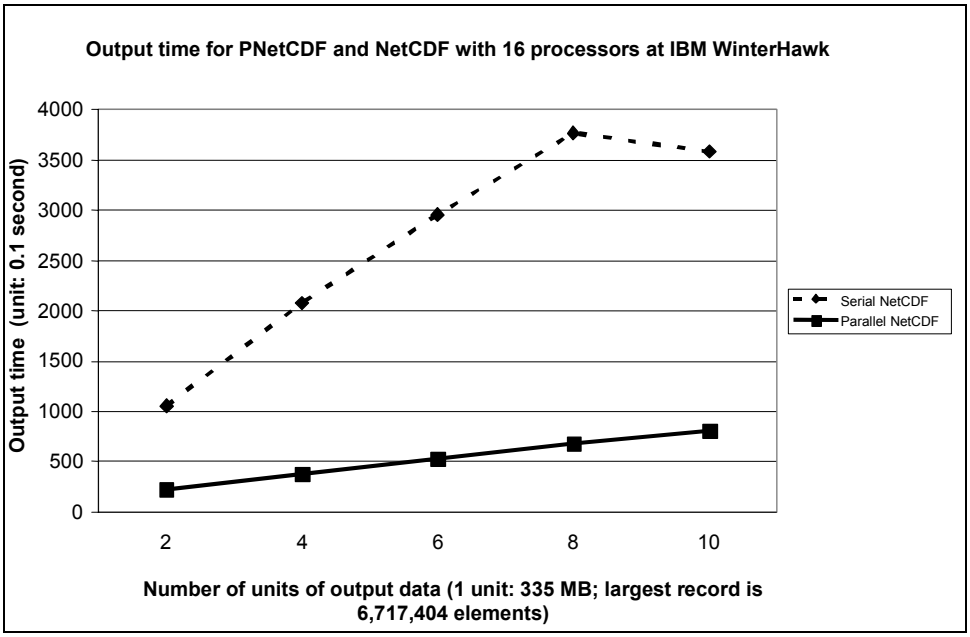


Figure 9. Comparision of Parallel NetCDF and NetCDF with 32 Processors.

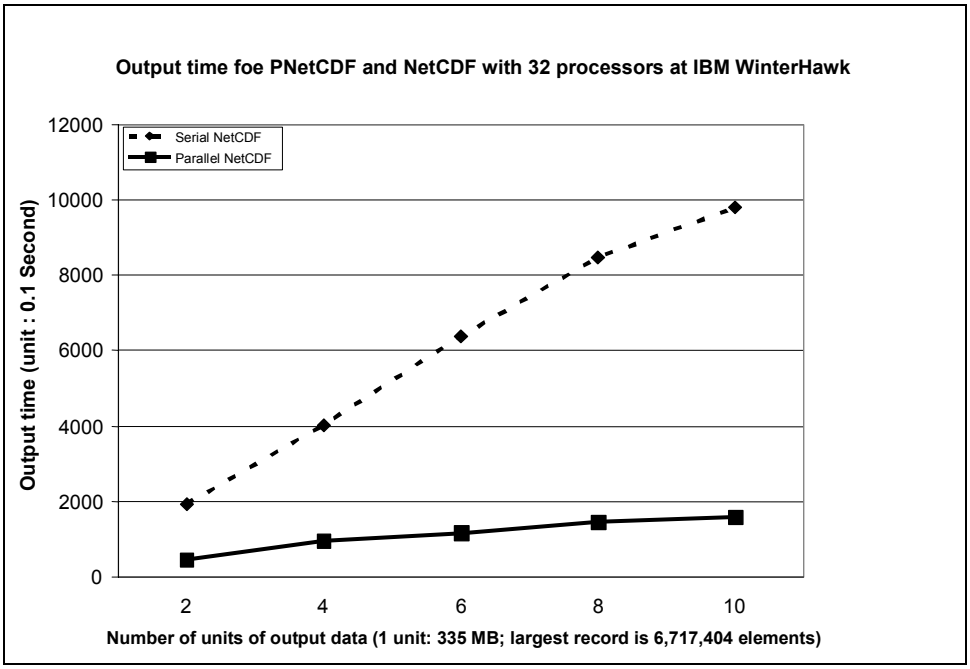


Figure 10. Comparison of Parallel NetCDF and NetCDF with 128 processors.

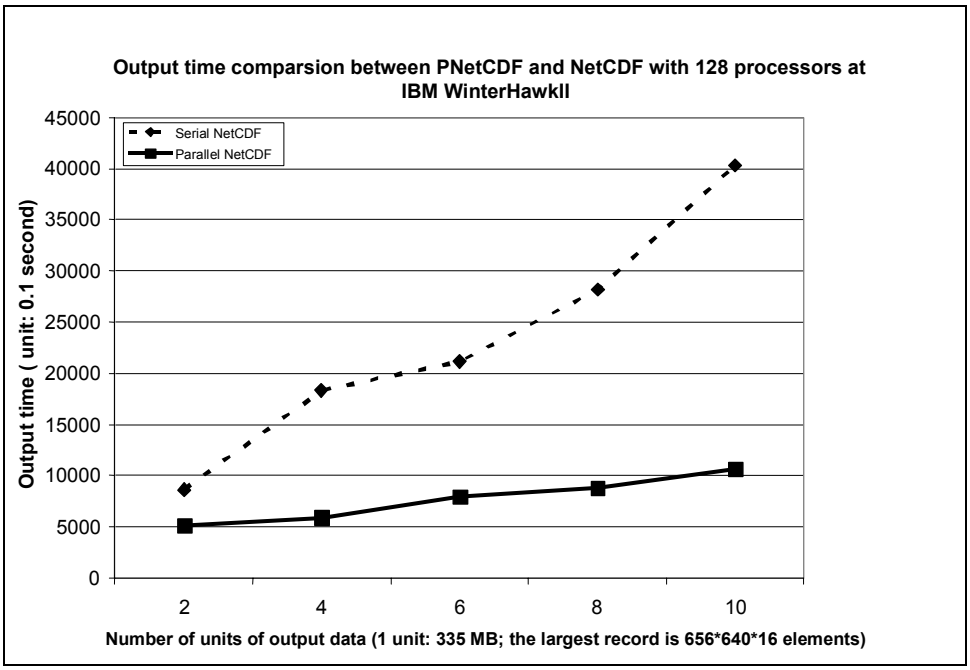


Figure 11. Performance of sequential NetCDF with different numbers of processors.

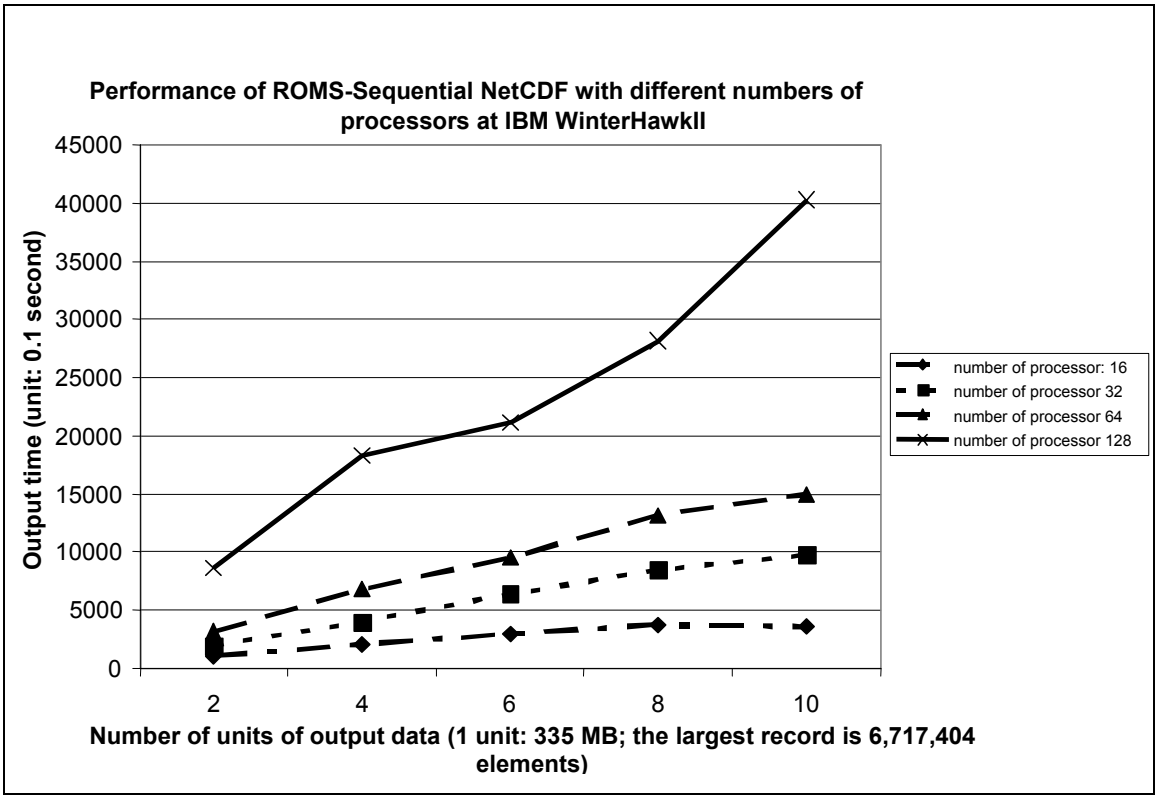


Figure 12. Performance of parallel NetCDF with different numbers of processors.

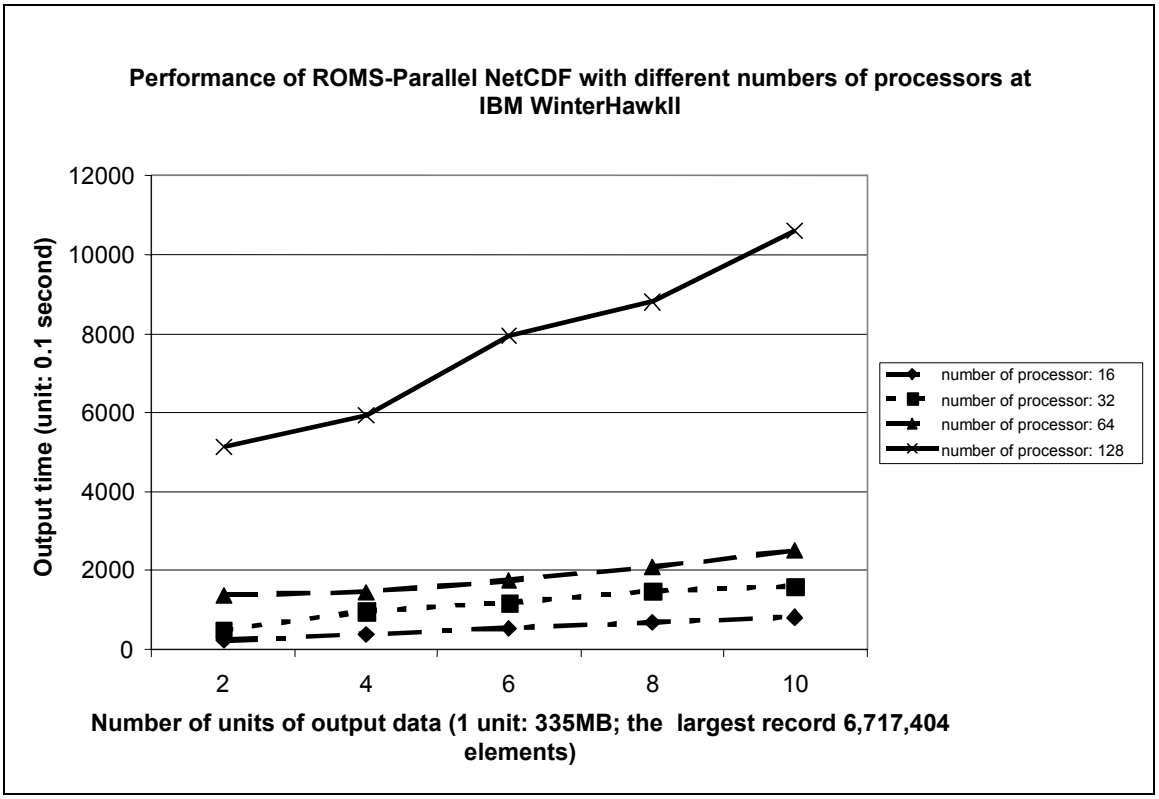


Figure 11 and Figure 12 show that as the number of processors increases, the output time *increases* dramatically for both Parallel NetCDF and sequential NetCDF. At the first glace, this result is unexpected.

Part of this performance degradation could be due to the smaller amount of data per processor as the number of processors increases. For example, for 16 processors, the maximum contiguous write would be (656 / 16 * 640) = 26,240 elements, compared to (656 / 128 * 640) = 3,280 elements for 128 processors.

However, we hypothesize that the performance bottleneck is primarily caused by network communication time among the nodes of the cluster. Note that the NCSA IBM P690 used in Experiments 1 and 2 is a single 16 processor node. The network communication time is not a significant bottleneck for that architecture, compared to a cluster.

Whatever the cause of the degradation for additional processors, these results clearly indicate that the Parallel NetCDF IO module performed significantly better than sequential NetCDF IO module. This result shows the potential advantage of using parallel NetCDF library to reduce network communication time on a cluster. In contrast, on the P690, even with all 16 processors assigned, the Parallel NetCDF IO module did no better than equal to sequential NetCDF IO module in all cases.

VI. Summary

These experiments suggest the following conclusions.

1. It was straightforward to add Parallel NetCDF module from Argonne in ROMS.
2. The hardware architecture, the number of IO nodes, the maximum record size of the file will greatly affect the performance of parallel NetCDF IO.
3. On a cluster, increasing the number of processors decreased the overall performance of IO for both sequential and parallel.
4. Different domain decompositions may slightly affect the performance of parallel NetCDF IO.
5. Parallel IO does not improve performance in all cases, especially for small amounts of data.

The most important conclusion is that optimal IO performance depends on many factors, including the architecture of the platform. On most systems, the best IO performance for ROMS will require a mix of sequential and parallel IO.

VII. Some suggestions for future work

This work is a partial implementation. There is much more that must be done to provide full parallel IO for ROMS. This work includes:

1. Add parallel NetCDF IO writers for other types of ROMS output files
2. Implement parallel NetCDF IO readers for all types of ROMS input files
3. Test on other platforms, including larger numbers of processors and other architectures
4. Optimize one-element array reading and writing, e.g., by consolidating many small operations

In the future, netCDF version 4 from Unidata will provide parallel IO and other advanced features [8]. It will be useful to measure the performance of NetCDF4 Parallel IO in the ROMS model.

Acknowledgements

This work is part of NSF-funded Modeling Environment for Atmospheric Discovery Expedition (MEAD) (http://www.ncsa.uiuc.edu/AboutUs/FocusAreas/MEADExpedition.html)

References:

[1] Jianwei Li, Wei-keng Liao, Alok Choudhary, Robert Ross, Rajeev Thakur, William Gropp, Rob Latham, Andrew Siegel, Brad Gallagher, Michael Zingale. “Parallel netCDF: A High-

Performance Scientific I/O Interface", SC'2003 November 15-21,2003, Phoenix, Arizona, USA
[2] A Parallel API for Creating and Reading NetCDF Files, http://www-unix.mcs.anl.gov/parallel-netcdf/
[3] Parallel netCDF Study, John Tannahill, Lawrence Livermore National Laboratory, PowerPoint slides
[4] Regional Ocean Model System, Web site: http://marine.rutgers.edu/po/index.php?page=&model=roms&print=true
[5] Muqun Yang, Mike Folk, Robert E. McGrath, "Investigation of parallel NetCDF with ROMS", NCSA HDF group, April 15, 2004, http://hdf.ncsa.uiuc.edu/apps/WRF-ROMS/parallel-netcdf.pdf
[6] William Gropp, Ewing Lusk, Rajeev Thakur, 1999: Using MPI-2. The MIT Press, 51-118.
[7] Unidata, netCDF, http://my.unidata.ucar.edu/content/software/netcdf/index.html
[8] Unidata, netCDF4: Merging the NetCDF and HDF5 Libraries, http://my.unidata.ucar.edu/content/software/netcdf/netcdf-4/index.html

Appendix: Changes and Additions to ROMS

This appendix summarizes the changes to the ROMS model to use the parallel NetCDF library. For more information see:
http://hdf.ncsa.uiuc.edu/apps/WRF-ROMS/

These changes include

1. Adding parallel NetCDF function calls,
   Source files that need to be modified:
   1) cppdefs.h
      We added another C-preprocessor option HISTORYP. If this macro is defined, then parallel NetCDF is invoked to generate history file.
   2) globaldefs.h
      Several parallel NetCDF function macros need to be added in this file.
   3) output.F
      We needed to add a #ifdef HISTORYP block to generate history file with parallel NetCDF.
      Adding another line: USE mod_pnetcdf
   4) inp_par.F
      Around line 140,

```
!-------------------------------------------------------
!  Check C-preprocessing options and definitions.
!-------------------------------------------------------
```

      A string to store CPP definitions has been input only through Master Node. For parallel IO, that has to be the same for all processors.
      It can be fixed in the future by ROMS developers

```
!-------------------------------------------------------
! For parallel output, these two lines can be commented out
!      IF (Master) CALL checkdefs
!      IF (Master) CALL my_flush (out)
! For parallel output, all processors should get these
! values.
CALL checkdefs
CALL my_flush(out)
```

   5) Files added
      Parallel NetCDF IO source files need to be added:

```
def_hisp.F
def_infop.F
def_varp.F
mod_pnetcdf.F
nf_fwritep.F
opencdfp.F
wrt_hisp.F
wrt_infop.F
```

      The corresponding sequential NetCDF IO source files are:

```
def_his.F
def_info.F
def_var.F
```

```
mod_netcdf.F
nf_fwrite.F
opencdf.F
wrt_his.F
wrt_info.F
```

Generally parallel NetCDF function prefix `nfmpi` replaces `nf` in parallel NetCDF source files. Some exceptions will be discussed in the next section.
For example, mod_pnetcdf.F is very similar to mod_netcdf.F. We only used

```
#include "pnetcdf.inc"
```

to replace

```
#include "netcdf.inc"
```

6) Makefile and MakeDepend
Linking option to parallel NetCDF library needs to be added in Makefile.
New IO source files need to be added in MakeDepend.

2. Assuring that each process writes correctly
Two exceptions for parallel NetCDF implementation are:
1)
There are about 20 1-element NetCDF variables in ROMS that have to be written independently through parallel NetCDF library. So functions *nfmpi_begin_indep_data(int ncid)* and *nfmpi_end_indep_data(int ncid)* need to be included for every independent Parallel NetCDF write function calls.
These two functions are used extensively at wrt_infop.F.
2)
Inside nf_fwritep.F, the array index for parallel IO has to be adjusted since
a) the model currently used halo-points for its upper and lower bounds;
b) raw data at each processor should be written to the disk in parallel.
The detailed changes include:
Set Nghost = 0; use subroutine get_bounds to obtain the starting and ending points of physical subdomain. Remember to adjust stagger-C points offset by 1.

3. Linking with parallel NetCDF
Makefile and MakeDepend should be changed. Currently the history file writer is only tested on IBM SP platforms. Mpxlf90_r should be used to compile with MPI-IO library.