Investigation of Parallel NetCDF with ROMS

Muqun Yang
Mike Folk
Robert E. McGrath
NCSA HDF group
May 4, 2004

I. Introduction

This paper reports results and lessons learned from an investigation of parallel NetCDF is implemented by Argonne National Lab and Northwestern University [4]. Parallel netCDF is beta software. The current version is 0.9.3. Source code and other information is available from:
http://www-unix.mcs.anl.gov/parallel-netcdf/

This project had two goals:

1. To help scientists understand MPI-IO and parallel NetCDF as part of the MEAD project [8].
2. To improve Parallel HDF5 work.

This report is based on three resources:

1) Related paper reviews

- Two papers from parallel NetCDF website, at Argonne National Lab and Northwestern University:
  "Parallel netCDF: A High-Performance Scientific I/O interface" [1]
  and
  "A parallel API for Creating and Reading NetCDF files" [2]

- Chapter 3 of William Gropp, Ewing Lusk and Rajeev Thakur, *Using MPI-2* [3].

- Studies of Parallel NetCDF by John Tannahill at Lawrence Livermore National Laboratory, reported in [4].

2) Reading the Parallel NetCDF source code.

3) Experience from installing Parallel NetCDF and integrating it with the Regional Ocean Modeling System (ROMS) [5].

This limited investigation may not have discovered many details of about the implementation. The authors apologize for any incorrect statements in this report. Corrections are welcome.

III. MPI-IO

In MPI-IO, there are two IO options: Independent IO and Collective IO. Independent IO means that each process does IO independently. Collective IO requires that all processes should participate when doing IO. With collective IO, MPI-IO can do optimization to improve IO performance. The reason is as follows:

> General OS system IO calls such as on UNIX and Windows can only handle contiguous data in a file. For non-contiguous data, it has to read/write in many small IO accesses; therefore the IO performance becomes worse.

Using the independent IO options means that each process does its own IO and MPI-IO library won't do any optimization for this option; so using independent IO is just like doing general IO with many processes. If an application is only handling contiguous data, the performance is mostly not going to be worse. However, for many applications, including Weather Research Forecast (WRF) [9] and Regional Ocean Modeling System (ROMS) [10], each process needs to access noncontiguous data, which will greatly degrade the performance. On the other hand, the MPI-IO library can help improve performance by using the MPI-IO function call MPI_FILE_SET_VIEW and collective IO. The basic idea is to assemble a big contiguous IO collectively by combining the noncontiguous data layout of each process.

For example, suppose we have four processes with each process view of the data as illustrated on the following chart:

When doing independent IO, since the view of each process is noncontiguous, essentially writing four blocks on the above chart will require 8 individual IO access to the disk. However, when using collective IO for this case, the IO access to the disk can be illustrated as follows:

This layout is contiguous. With an appropriate parallel file system, the previous 8 IO can become one IO. This example is oversimplified. In real applications, MPI-IO can handle more

customized cases by using collective IO. *MPI_Type_create_subarray* and *MPI_Type_create_darray* function calls are usually used for building MPI derived data types. ROM-IO, an MPI-IO implementation at Argonne National Lab, can use the data-sieving technique to improve IO performance [6]. MPI_INFO hints, which pass to MPI-IO library, can sometimes help improve the performance on specific platforms. For a detailed explanation, please see chapter 3 of reference [4] for more details.

III. Why Parallel NetCDF?

NetCDF is a simple, straightforward and widely used file format. NetCDF is widely used, especially in computational environment science.

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 both ROMS and WRF.

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. Some Details of the parallel netCDF Implementation

1. Design of the library

Parallel netCDF reuses small sections of the netCDF3 library from Unidata, but mostly is a new implementation of netCDF.

2. Use of netCDF File

The structure of a netCDF file is well-suited for parallel I/O. Figure 1 shows the NetCDF file structure: a file header contains metadata of the stored arrays, then the fixed-size arrays are laid out in the following contiguous file space in a linear order, with variable-sized arrays appended at the end of the file in an interleaved pattern. (adapted from Figure 1 of reference [1]).

Because each fixed-size variable are stored in a fixed contiguous region of the file, and it is straightforward to figure out the interval between the current record position and the next record position of each record variable, NetCDF file is ideal for using Set_file_view and collective IO to improve performance through MPI-IO.

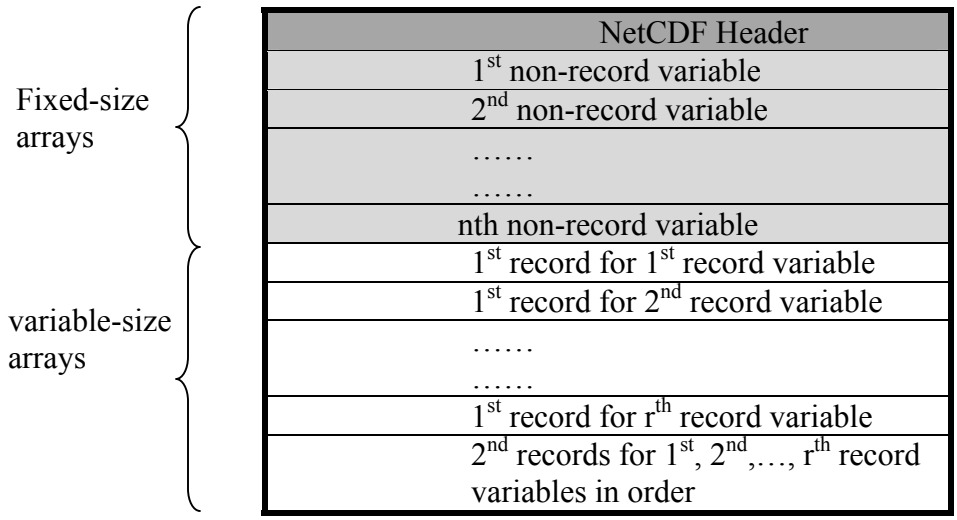

Figure 1. The organization of a netCDF file.

3. The netCDF programming model: Define mode and Data mode

3.1. Define mode

Functions such as inquiry, attribute, dimension, are all done with collective I/O calls.

Collective calls allow for better error detection than independent calls. Error detection is performed at the end of the define mode. This is not as costly as one might expect since it doesn't imply any communication at the time the call is made [2]. The error detection calls indeed helped us find a bug when we investigated ROMS-Parallel NetCDF IO module.

3.2. Data mode

Reading and writing data arrays (netCDF variables) supports both independent IO and collective IO. Collective IO combined with MPI_SET_FILE_VIEW yields a substantial improvement in IO performance.

4. High-level APIs and flexible APIs

Parallel NetCDF includes high-level APIs and flexible APIs.

From the limited experience in this project, it seemed that each Unidata NetCDF API has a corresponding Parallel NetCDF API. Each API prefix is simply changed from *nc_* to *ncmpi_* for C interface and from *nf_* to *nfmpi_* for Fortran interface. There are no C++ and Java interfaces for the parallel netCDF prototype.

The main change to the standard netCDF API is the addition of two arguments to *ncmpi_create* and *ncmpi_open*. These functions require MPI_Comm and MPI_INFO structures required by

MPI-I/O. Through MPI_Comm, the MPI communicator can be passed. Hints of IO optimizations can be passed through MPI_INFO.

The MPI Datatype MPI_offset replaces C datatype size_t in some functions.

Two new functions were added for doing independent IO inside parallel NetCDF. These functions are *ncmpi_begin_indep_data(int ncid)* and *ncmpi_end_indep_data(int ncid)*.

For other APIs, the suffix *all* is added for collective IO calls such as, *ncmpiget_vars_float_all*.

Finally, there are flexible Data Mode APIs that allow users to use MPI_Datatype themselves for better performance than standard netCDF data types. For example, Flexible function

*ncmpi_put_vara(int ncid, int varid, const MPI_Offset start[], const MPI_Offset count[], const void *buf, int bufcont, MPI_Datatype datatype)*

allows the programmer to use MPI datatypes to describe the in-memory organization of the values. The datatype passed to the flexible API should be a basic datatype such as MPI_FLOAT and MPI_INT.

V. Dimension scales

There are no special arrangements of dimensions in parallel NetCDF. All dimension information except the dimensional scale data is stored inside the file header. This is handled with collective IO as mentioned above. Just as Unidata NetCDF does, the dimensional variable is treated as an ordinary variable with the same dimensional name.

VI. Comparison of Parallel NetCDF with Parallel HDF5

Li et al [1] compared the performance of parallel NetCDF with parallel HDF5. This paper identified two important differences between parallel netCDF and HDF5.

1. The linear data layout (regular and highly predictable) minimizes the overhead of parallel NetCDF. In contrast, parallel HDF5 uses a tree-like file structure; in which results the data is irregularly laid out using super block, header block, extended header block, extended data blocks. The result is that it is difficult to pass user access patterns directly to MPI-IO.
2. The I/O for the parallel NetCDF's header is low. There is only one header that contains all necessary information. By comparison, in parallel HDF5, the header metadata is dispersed in separate header blocks for each object. It is necessary to iterate through the entire namespace to get all the header information, and in general, HDF5 requires many small reads and writes to manage the metadata in the files. These are inefficient for parallel access.

However, Parallel NetCDF has some essential drawbacks compared with Parallel HDF5. These include:

1. Parallel NetCDF doesn't support chunking storage.

2. HDF5 can create new objects and metadata at any time. Unidata NetCDF and parallel netCDF creates all metadata during the define phase. If metadata is added during the data phase, Parallel NetCDF has to copy the entire file header (essentially rewrites the whole file), which may be expensive.

VII. Installation and Integration with an Application

This study tested Parallel NetCDF on two IBM SP machines (IBM P690 at NCSA and IBM WinterHawkII clusters at NCAR). IBM GPFS was stable and easy to use on these machines, so it was possible to experiment with MPI-I/O.

It was not difficult to install Parallel NetCDF on copper, NCSA's IBM P690. The Parallel NetCDF test suite has some misleading information. For example, the Fortran test depends on a C test. It was not difficult to figure this out.

The Fortran APIs did not work when turning on large file support with 64-bit object mode. We've sent a bug report to the Argonne group.

It was relatively easy implement a Parallel NetCDF ROMS writer because many previous NetCDF calls can be converted easily to Parallel NetCDF calls. The ROMS writer is discussed in the next section.

VIII. Performance Studies

1. A Parallel NetCDF-ROMS History File Writer

In this report, the performance results come from a real application: the Regional Ocean Model System (ROMS). ROMS is an oceanographic prediction model [10]. The model can write output data into a history file at every time step.

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 ROMS history file with parallel NetCDF.

These changes include

1. Adding parallel NetCDF function calls,
2. Assuring that each process writes correctly,
3. Linking with parallel netCDF.

2. Description of the Experiments

The performance of the ROMS parallel writer was measured in two experiments. The first experiment was run on the NCAR IBM SP WinterHawk. The second experiment was run on the NCSA IBM P690.

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 same large history file and several small files.

In parallel component of both experiments, we used parallel NetCDF to generate the history file but serial NetCDF to generate other small files.

We called the amount of data at each timestep one unit of output data. The size of the biggest record is a key factor to affect the parallel IO performance, as is explained in part 4 of this section.

Table 1 summarizes the experimental conditions.

Table 1. Summary of Experiments

	Experiment 1	**Experiment 2**
Platform	IBM SP WinterHawk	NCAR IBM P690
Number of processor	128	16
Unit of data (MB)	335	48
The size of the biggest record (elements)	656*640*16 = 6,717,440	246*240*16 = 944,640

For each run, the wall-clock time for writing output data of the model was the performance measurement for both sequential NetCDF and Parallel NetCDF.

3. Analysis and Discussion

Figure 2 and Figure 3 show how IO performance behavior in experiment 1 and 2, respectively.

With 128 processors, at NCAR IBM, the wall-clock time to write output data with Parallel NetCDF is much shorter than sequential NetCDF in all model runs. However, for the second experiment, with 16 processors, at NCSA IBM p690, wall-clock time to write output data using Parallel NetCDF was much longer than sequential NetCDF in all model runs. However, when the model was adjusted so the unit of data was identical, the results for the 16 processors at NCSA P690, were similar to the 128 processors at NCAR IBM SP. We conclude that the difference between the results in these two experiments was caused by the size of the biggest record in the output history file.

There are more than 20 1-element variables inside every ROMS file. There are also around 10 fixed-size small NetCDF variables inside the file. These small data accesses will pay tremendous penalty when accessed using Parallel IO.

As the number of timesteps increases, the number of small writes doesn't change. However, as the number of timestep increases, the number of big records written to the history file increases. So it is reasonable to say that MPI-IO overhead becomes relatively less and the overall Parallel NetCDF IO performance should become better as the number of timesteps increases.

In Figure 3 the wall clock time to write output data was less for a file with ten total timesteps than for a file with eight timesteps. We do not have an explanation for this finding. It is possible that these results are due to caching mechanisms in the file system or storage hardware.

In these experiments there are three factors that seem to affect the performance of the ROMS parallel IO:

1. the size of the largest record
2. the number of processors used in a run
3. the type of platform

The NCAR IBM and the NCSA IBM are essentially the same processors and platform. Our results suggest that the key factor is the size of the largest record.

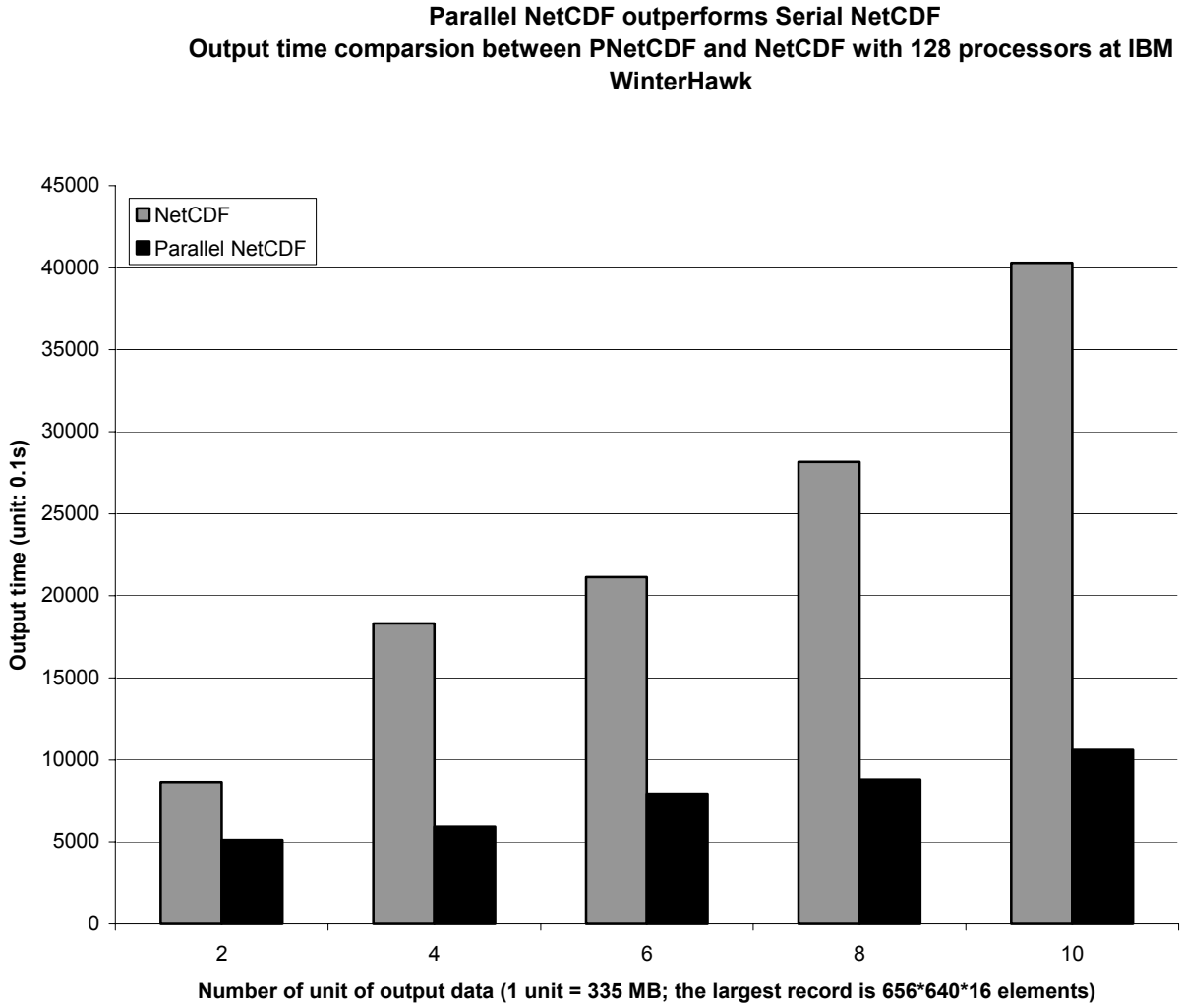

Figure 2. Comparison between PNetCDF and NetCDF with 128 processors at IBM WinterHawk.

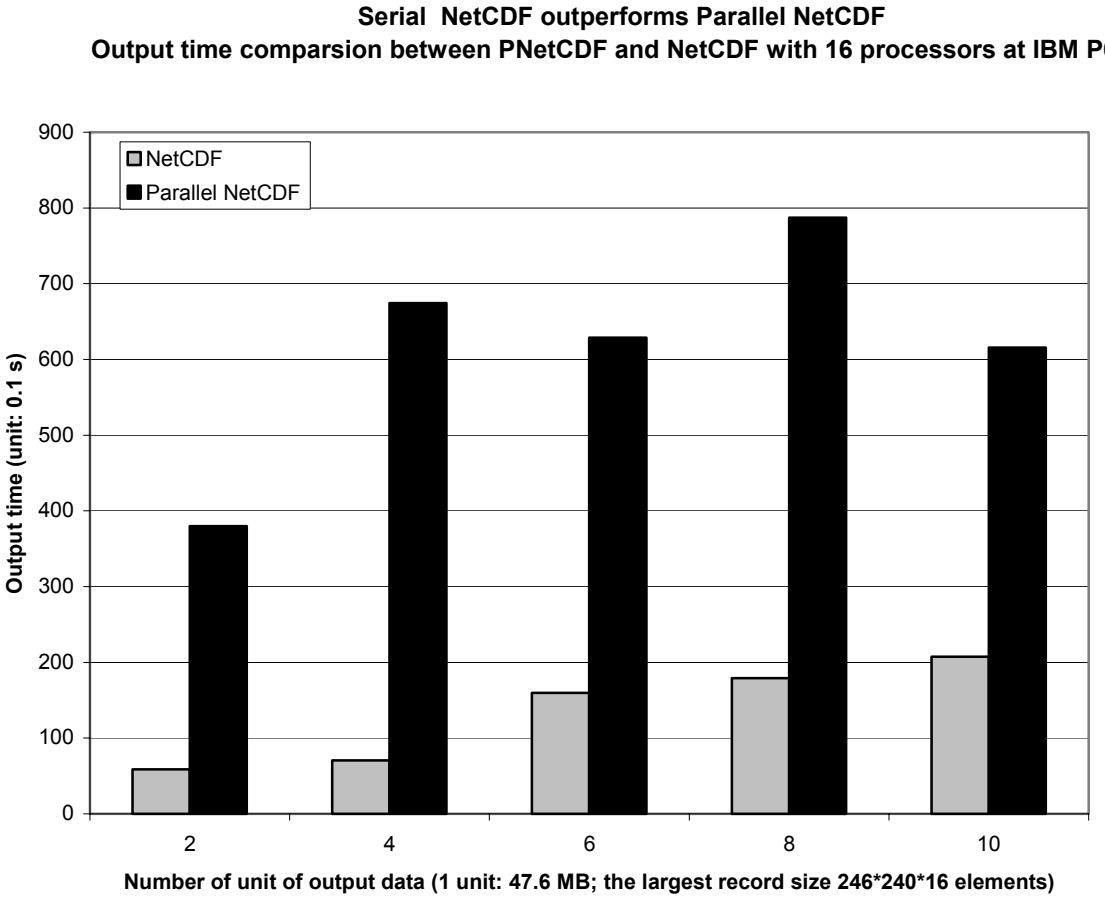

Figure 3. Comparison between PNetCDF and NetCDF with 16 processors at IBM P690

We conclude that the parallel NetCDF will outperform the serial NetCDF when the size of the records written to the output file is large enough to overcome the overhead from MPI-IO. The break even point will depend on platform specific factors. This finding is consistent with earlier reports that indicated that the number of processors in use and different platforms appeared to affect the performance [4]. We will further analyze the role of different factors in another report.

X. Summary

1. Using collective IO as well as MPI_SET_FILE_VIEW properly can greatly improve IO performance in parallel applications.

2. Parallel NetCDF from Argonne National Lab and Northwestern University is easy to install and to use though it still needs some tuning.

3. Using Parallel NetCDF can greatly improve performance for ROMS with the increasing of big record size. Parallel netCDF gives better performance for large writes, serial netCDF is better for small writes.

4. ROMS writes some small arrays and a few large arrays. The best I/O performance will require a hybrid, using both serial and parallel I/O.

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] William Gropp, Ewing Lusk, Rajeev Thakur, 1999: Using MPI-2. The MIT Press, 51-118.

[4] Parallel netCDF Study, John Tannahill, Lawrence Livermore National Laboratory, PowerPoint slides.

[5] Regional Ocean Model System, Web site: http://marine.rutgers.edu/po/index.php?page=&model=roms&print=true

[6] Rajeev Thakur, William Gropp, Weing Lusk, 1999: "Data Sieving and Collective I/O in ROMIO," in Proceeding of the 7th Symposium on the Frontiers of Massively Parallel Computation, pp. 182-189

[7] http://hdf.ncsa.uiuc.edu/apps/WRF-ROMS/

[8] http://www.ncsa.uiuc.edu/AboutUs/FocusAreas/MEADExpedition.html

[9] http://www.wrf-model.org/

[10] http://marine.rutgers.edu/po/index.php?model=roms&page=

Acknowledgments

This work is part of NSF-funded Modeling Environment for Atmospheric Discovery Expedition (MEAD) [8].