

The Parallel-netCDF I/O Library

Robert Latham
Argonne National Laboratory

January 12, 2010

1 Synonyms

pnetcdf

2 Definition

Parallel-NetCDF provides an API for parallel access to traditionally-formatted netCDF files. Parallel-NetCDF both produces and consumes files compatible with serial netCDF, while providing a familiar (though not identical) programming interface better suited to express parallel I/O.

3 Discussion

Before going into more detail about Parallel-NetCDF, it will be helpful to see how it fits in the bigger picture of scientific computing and application I/O.

3.1 Background

Today's leadership computing platforms contain hundreds of thousands of processors. In order to efficiently utilize these computers, applications make use of simplifying abstractions provided by tools and libraries. MPI libraries provide a standard programming interface for parallel computation on a wide array of hardware. Math libraries like BLAS hide the details of CPU-specific optimizations. These lower-level math and communication libraries contribute to an overall software stack built to allow developers to focus on the science behind their applications.

I/O performance on high-end computing platforms follows a similar story. Performance comes from aggregating many storage devices. In fact, CPU performance has consistently improved at a rate much faster than that of storage devices. This discrepancy makes the need for parallel I/O across multiple devices even more acute. Yet as the number of devices involved in I/O grows, coordinating I/O across those devices becomes more of a challenge. In the same

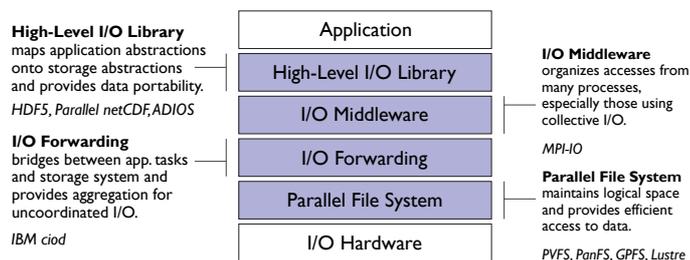


Figure 1: The I/O software stack.

way that communication and math libraries assist the parallel computation side of scientific simulations, an “I/O software stack” assists data management.

The large number of storage devices mentioned above form the foundation of the software stack. Several additional layers provide abstractions to help make the task of extracting maximum performance accessible to applications programmers. The programming API and data model Parallel-NetCDF presents to the scientific programmer are conceptually closer to the tasks common to scientific applications and hide many lower level details. Though hidden by Parallel-Netcdf, discussing the lower levels of the software stack will provide a better appreciation of Parallel-NetCDF’s role at the upper level.

- Storage systems contain thousands of individual devices to increase the potential bandwidth. The *parallel file system* manages those separate devices and collects them into a single logical unit. The file system does lack some important features: typically file systems have no mechanism for coordinated I/O, and the data model is still fairly low level for computational scientists to use directly.
- The *I/O forwarding* layer typically exists only on the largest computer systems. Applications do not interact with this layer directly. Rather, this layer simplifies the scalability challenge: a large number of compute nodes communicate with a smaller number of I/O forwarding nodes, and these nodes in turn talk to the file system.
- the *Middleware*, or *MPI-IO*, layer introduces more sophisticated algorithms tailored to parallel computing. This layer coordinates I/O among a group of processes, introducing collective I/O to the stack. MPI datatypes allow applications to describe arbitrary I/O access patterns. Applications can use the MPI-IO layer directly, but the programming model is still rather low-level and not a perfect fit to the needs of common applications.
- Parallel-NetCDF belongs to the *High-Level I/O Library* layer. Libraries in this layer introduce concepts such as multidimensional arrays of typed data which are better suited to scientific applications. For example, a

climate code can represent temperature in the atmosphere with a four-dimensional array: latitude, longitude, altitude, and number of degrees Celsius. Further, these libraries define the layout of data on disk. These self-describing file formats make exchanging data with colleagues now and in the future much easier.

3.2 History

The history of the Parallel-NetCDF project starts with Unidata’s netCDF project. The netCDF library has long provided a straightforward programming API and file format for serial applications. In the summer of 2002, Argonne National Laboratory and Northwestern University started a joint effort to build a parallel I/O library while keeping the best parts of netCDF. Without changing the existing netCDF file format, Parallel-NetCDF introduced a new programming API. This new API was not identical to serial netCDF, but would be comfortable to anyone familiar with the existing serial netCDF API. Section 3.3 will contain more discussion of the Parallel-NetCDF API.

Parallel-NetCDF has since become an important tool for high-performance I/O in the climate and weather domains. These domains have long-established netcdf-based workflows, using netcdf datasets for archiving, analysis, and data exchange. Since Parallel-NetCDF retains the same file format as netCDF, climate and weather codes could make changes to their simulation codes while leaving other components of their workflow the same. Parallel-NetCDF sees use in other domains as well: the programming API and file format make it possible to deliver good parallel I/O performance without having to master all of MPI-IO.

3.3 The Parallel-NetCDF API and File Format

Discussions about Parallel-NetCDF need to cover two main points: the programming model and the on-disk data format. The API simplifies many MPI-IO concepts while retaining several key optimizations. The on-disk format affects both I/O performance as well as collaborations now and in the future.

If applications were to use MPI-IO directly, the “basic type” they would build upon would be a linear stream of bytes. In contrast, Parallel-NetCDF offers a more application-friendly model based on multidimensional arrays of typed data. Operations on those arrays could be as simple as reading the entire array, or a more complicated operation such as having each process write a sub-cube representing a chunk of the Earth’s atmosphere. In addition to operations for manipulating data, the API contains routines to annotate the data in the file or the file itself. Such annotations may include timestamps, machine information, experiment or workflow information, or other provenance information to better understand how an application produced the data in this file. Section 3.4.1 goes into more detail about the programming interface as well as how Parallel-NetCDF interacts with the underlying MPI-IO library.

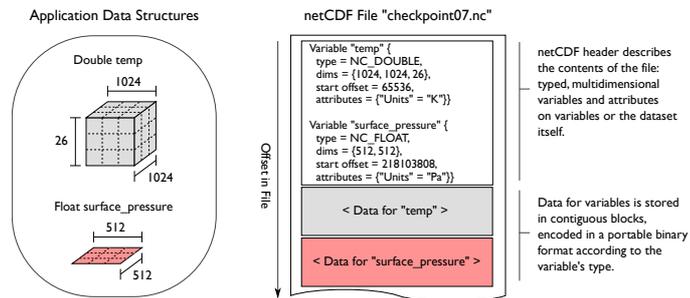


Figure 2: On the left, how an application views netCDF objects. On the right, how netCDF and Parallel-NetCDF store those objects on disk

In addition to the programming interface, the file format plays a major role in how useful Parallel-NetCDF can be to application groups. Parallel-NetCDF applications create a self-describing, portable file with support for embedded metadata. API routines allow programs to query the contents of data files without any prior knowledge of the contents. The structure of the datasets extends to the representation of bytes on disk: even if a dataset is moved to a different machine, the library will still be able to understand that file. The IBM BlueGene system, for example, has a 32-bit, big-endian architecture, while the Cray XT5 has a 64-bit little-endian architecture. Despite these differences, files created on one can be read on the other and vice versa.

Figure [reffig:netcdf-overview](#) depicts a simple Parallel-NetCDF application. A climate code wishes to create a checkpoint file to store some key pieces of information. Atmospheric temperature is stored as a three-dimensional array of double-precision floating point values. Barometric pressure at the Earth's surface requires only two dimensions, and only single-precision floating points. Further, the application wants to ensure future consumers of this dataset know which units these variables use. It stores this information as an attribute on each variable. On disk, the dataset contains a small header describing the size and type of both variables and any attributes, followed by the actual data for the variables. This fairly simple file format comes with some restrictions, but the trade-off results in efficient file accesses for parallel I/O.

Parallel-NetCDF uses the exact same file format as netCDF, making it easier for applications groups to adopt Parallel-NetCDF. A programmer can change the computationally intensive simulation code to use Parallel-NetCDF while the other components of the workflow (e.g. visualization, analysis, archiving) can continue to use serial netCDF.

3.4 Annotated Examples

Parallel-NetCDF shares many netCDF concepts, but, as mentioned previously, uses a slightly different API. Some simple example programs should make the

```

1 #include <mpi.h>
2 #include <pnetcdf.h>
3
4 int main(int argc, char **argv) {
5     int ncfile, nprocs, rank, dimid, varid1, varid2, ndims=1;
6     MPI_Offset start, count=1;
7     char buf[13] = "Hello World\n";
8
9     MPI_Init(&argc, &argv);
10    ncmpi_create(MPI_COMM_WORLD, "demo.nc",
11                NC_WRITE|NC_64BIT_OFFSET, MPI_INFO_NULL, &ncfile);
12
13    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
14    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
15
16    ncmpi_def_dim(ncfile, "d1", nprocs, &dimid);
17    ncmpi_def_var(ncfile, "v1", NC_INT, ndims, &dimid, &varid1);
18    ncmpi_def_var(ncfile, "v2", NC_INT, ndims, &dimid, &varid2);
19    ncmpi_put_att_text(ncfile, NC_GLOBAL, "string", 13, buf);
20
21    ncmpi_enddef(ncfile);
22
23    start = rank;
24    ncmpi_put_vara_int_all(ncfile, varid1, &start, &count, &rank);
25    ncmpi_put_vara_int_all(ncfile, varid2, &start, &count, &rank);
26
27    ncmpi_close(ncfile);
28    MPI_Finalize();
29    return 0;
30 }

```

Figure 3: A basic Parallel-NetCDF program using the standard interface. For brevity, error checking/handling has been omitted.

differences clear.

3.4.1 The Standard Interface

Figure 3 contains a full, correct Parallel-NetCDF program to create a simple dataset. This program, though brief, demonstrates many key features of the library.

Because Parallel-NetCDF relies on MPI-IO, the program must include the `mpi.h` header file (line 1) and initialize (line 10) and finalize (line 30) the MPI library.

The `ncmpi_create` routine (line 11) adds two arguments to netCDF's `nc_create`: a communicator and an MPI Info object. These two arguments do expose a bit of the underlying MPI library, but give the programmer some flexibility and the ability to tune operations if needed.

Parallel-NetCDF, like netCDF, has a *bi-modal* interface: when creating a dataset, an application starts off in *define mode*, when it must first describe what variables it will write, the size and type of those variables, and any attributes. This example defines a single dimension (line 17) and assigns that dimension to two variables (line 18 and 19).

Datasets can contain a great deal of metadata. Both variables and dimen-

sions have human-readable labels. In addition to those labels, attributes may be defined and placed on dimensions, variables, or the dataset itself. Line 20 places an attribute on the entire dataset.

In the example, line 22 calls `ncmpi_enddef` to switch from define mode to *data mode*. By asking the programmer to pre-define variables and attributes, the library can allocate space with very little overhead. Re-entering define mode can potentially trigger a very expensive re-writing of the dataset. For a large class of problems, however, it is known ahead of time which information will be written out.

Lines 25-26 do the actual work of storing information into the dataset. Parallel-NetCDF provides one set of functions for uncoordinated *collective I/O*, and another parallel set for coordinated *collective I/O*. This example uses the collective version (as shown by the `_all` suffix). Every MPI process participates in these writes, each writing out its MPI rank. In a traditional serial netCDF program, each rank would send data to a master process and this process would in turn write out the information. This “send-to-master” model quickly becomes untenable as the number of MPI processes increases and as the amount of memory available to each MPI process gets smaller.

The collective Parallel-NetCDF routines in turn call collective MPI-IO routines. MPI-IO collective routines can use powerful optimizations such as two-phase I/O ([?]) or data shipping ([?]). It’s possible to use independent I/O with Parallel-NetCDF, but the opportunities for optimization are greatly limited in that case.

Once executed, this example will produce a file “demo.nc”. This dataset will contain two variables, as can be verified with either netCDF or Parallel-NetCDF utilities.

3.4.2 The Flexible Interface

The Parallel-NetCDF standard interface shares much with the serial netCDF interface. The function name explicitly states if the operation is a read or write (`_put` or `_get`), the type of access (e.g. `_var`, `_vara`), and the type (e.g. `_int`, `_float`, `_double`). For example, the function `ncmpi_put_vara_int` clearly writes integers into a sub-array.

Parallel-NetCDF allows for even more flexibility in describing an application’s datatypes. In Figure ??, a write operation from Figure ?? is converted to the Flexible interface. The datatype in this example is trivial (`MPI_INT`), but MPI datatypes allow an application to describe how data is laid out in memory and then send along all that information in a single call. A more complicated MPI datatype might, for example, write out all the data in a multi-dimensional array while skipping over the “ghost cells” used to optimize communication but which are just copies of data belonging to other MPI processes. (XXX: figure of array with ghost cells?)

```

/* function prototype */
int ncmpi_put_vara_all(int ncid, int varid,
    /* start and count describe access in file */
    const MPI_Offset start[], const MPI_Offset count[],
    /* 'buf' 'bufcount' and 'datatype' describe data in memory */
    const void *buf, MPI_Offset bufcount, MPI_Datatype datatype);

...
start = rank;
ncmpi_put_vara_all(ncfile, varid1, &start, &count, &rank, count, MPI_INT);
...

```

Figure 4: Prototype for and usage of one of the flexible mode routines.

```

1
2 async interface mode code goes here

```

Figure 5: async mode code

3.5 Parallel-netCDF limitations

3.6 Tuning Parallel-NetCDF

3.7 Related Works

4 Related Entries

- Parallel-NetCDF fits squarely under the topic of *Parallel I/O*.
- It is closely related to *HDF5*, the first high-level I/O library and a project from which Parallel-NetCDF learned much.
- Parallel-NetCDF sees most use in *Distributed Memory* environments, particularly those using message passing (though nothing precludes shared memory platforms from using Parallel-NetCDF effectively).
- *Benchmarking* ties in to Parallel-NetCDF in that correctly benchmarking Parallel-NetCDF requires understanding the entire I/O software stack.
- *Weather and Climate Simulation* makes use of Parallel-NetCDF, particularly as per-node memory sizes become smaller and smaller. With less memory per-node, parallel approaches to I/O become important not just for performance, but to actually be able to run on these systems at all.

5 Bibliographic Notes and Further Reading

The Parallel-NetCDF web site is www.mcs.anl.gov/parallel-netcdf. The site hosts pointers to additional Parallel-NetCDF papers and projects as well as links to production releases and the latest code.

The Parallel-NetCDF community of users and developers communicates on the parallel-netcdf@mcs.anl.gov mailing list.

Parallel-NetCDF developers regularly give tutorials and workshops covering the library.

References

Acknowledgments

This work was supported by the U.S. Dept. of Energy under Contract DE-AC02-06CH11357.