PARALLEL I/O ADVANCEMENTS IN AIR QUALITY MODELING SYSTEMS

Todd H Kordenbrock*, Ron A Oldfield
Computer Science Research Institute, Sandia National Laboratories, Albuquerque, NM, USA

1. INTRODUCTION

Over the past year, researchers at Sandia National Laboratories worked closely with researchers at the National Oceanic and Atmospheric Administration to analyze performance and make recommendations about how to improve performance and scalability of the Community Multiscale Air Quality (CMAQ) code. After some preliminary analysis, we identified a number of potential optimizations – particularly in the input/output libraries used by CMAQ. This paper describes one such optimization that allows CMAQ to write output files in parallel using MPI-IO and parallel-netCDF. These changes led to a 48% improvement in write performance of CMAQ using a representative dataset.

2. CMAQ I/O IN A CLUSTER

The CMAQ code uses the Models-3 I/O API (IOAPI3) to manage the I/O of all input and output files. IOAPI3 is widely used in the Community Models & Analysis System (CMAS) community because it performs much of the work involved with file creation and definition. IOAPI3 also performs necessary data transformations between memory and disk. These IOAPI3 features ensure that data files can be shared within the CMAS community.

IOAPI3 is built on netCDF, the de-facto standard in the atmospheric modeling field. NetCDF is an array-oriented I/O library with an easy to use API and a portable, self-describing, binary file format. NetCDF was originally designed for sequential computing and thus has a few limitations that restrict its usefulness for parallel applications. Most notably, it does not support concurrent writes from multiple processes. The netCDF metadata is cached in memory in each process and gets out of sync if multiple processes write to the same file.

*Corresponding author: Todd H Kordenbrock, Sandia National Laboratories, PO Box 5800, MS-1319, Albuquerque, NM 87185-1319; e-mail: thkorde@sandia.gov; phone: 505-844-7181; fax: 505-845-7442

The CMAQ code addresses this problem by selecting a single “I/O node” to perform the I/O on behalf of the other processors. This modification exists in a thin software layer (called PARIO) that mimics the IOAPI3 API. Instead of each process independently writing to a shared netCDF file, each processor sends its portion to the I/O node. The I/O node then gathers the data and writes to a single netCDF file using the original IOAPI3 library (Fig. 1a). This approach allows a parallel application to use netCDF, but it is not an efficient solution with respect to I/O. Dumping all data to a single node creates a significant I/O bottleneck at the I/O node, especially as the number of compute nodes increases to hundreds to thousands of processors. It also requires the I/O node to have enough memory to store the entire dataset before writing, which can be a problem for large applications. Finally, performing the write from a single node prevents the application from making efficient use of parallel file systems or parallel I/O libraries that may allow concurrent writes from different processors.

3. PARALLEL I/O FOR CMAQ

After reviewing the CMAQ I/O framework, we investigated the modifications required to replace netCDF with parallel-netCDF (pnetCDF) (Li 2003). PnetCDF is an implementation of the netCDF standard that extends the API to include support for parallel I/O. PnetCDF provides parallel I/O support by layering the pnetCDF library on top of the MPI-IO parallel I/O interface (Gropp 1999).

Parallel I/O improves I/O access through advanced techniques like two-phase I/O and data sieving (Thakur 1999). These optimizations enable the compute nodes to cooperate in a way that allows efficient parallel access to the storage system. For example, in two-phase I/O compute nodes first aggregate data on a subset of the nodes, and then write the aggregated data to the storage system in parallel (Fig. 1b). The aggregation of data reduces the number of I/O operations and allows for large contiguous writes—reducing the seek-time overhead incurred by a large number of non-contiguous write requests.

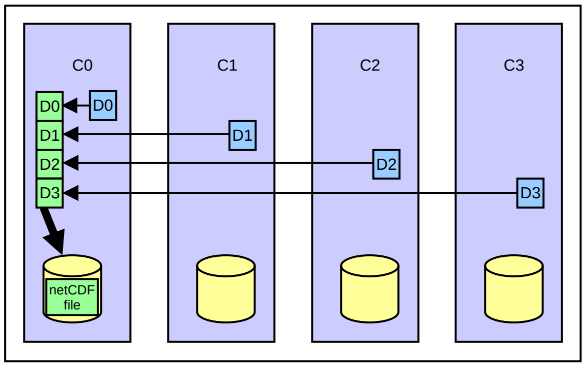

Fig. 1a. Collecting data at a single I/O node and writing to a locally attached disk

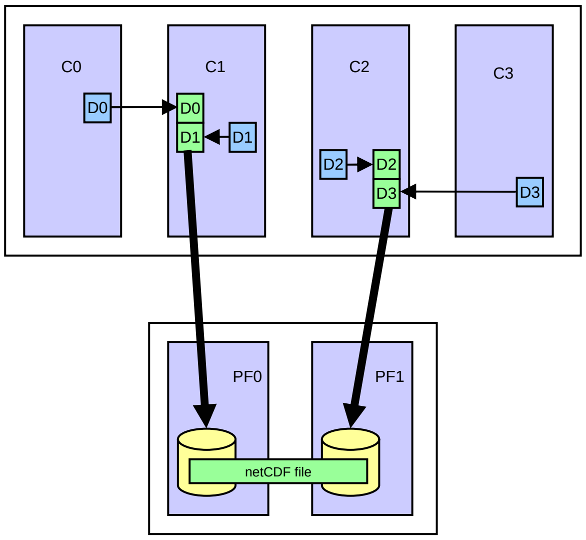

Fig. 1b. Using two-phase I/O to aggregate data and write to a parallel file system

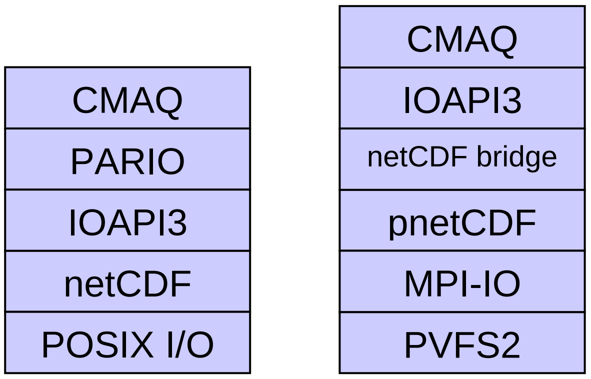

Fig. 2. Original (left) and parallel I/O (right) software stacks

To fully exploit the available I/O parallelism in a cluster, applications often write to a parallel file system that distributes the file to multiple storage devices to increase aggregate throughput.

Figure 2 illustrates the different software stacks for the original and parallel I/O version of CMAQ. To incorporate pnetCDF into the CMAQ code, we removed the PARIO library and developed a thin software layer to bridge the API gap between IOAPI3 and pnetCDF. The netCDF bridge has two functions: emulate the netCDF API on the frontend, and manage the pnetCDF API on the backend. We then stored pnetCDF files to the PVFS2 (Latham 2004) parallel file system to distribute data across storage devices. We chose PVFS2 because the MPI-IO library that sits below pnetCDF is optimized to use the native PVFS2 parallel I/O interface.

By managing the pnetCDF API on the backend, the netCDF bridge is able to supply the required MPI-IO parameters to pnetCDF. The netCDF bridge also allows users to provide hints[1] to MPI-IO and the underlying file system to further optimize performance.

The added support for parallel I/O required minimal changes to the main CMAQ code. All parallel I/O optimizations are configured at compile time. Users that do not have the libraries or storage system to support parallel I/O can use the original version without any code modifications.

4. TESTING AND PERFORMANCE EVALUATION

We evaluated the modifications to the CMAQ code by running a series of experiments on a small development cluster at Sandia National Laboratories. The cluster consisted of 32 IA32 dual-processor compute nodes with Myrinet interconnect. Each compute node had an 18 GB SCSI disk attached, and we used 6 compute nodes to support the PVFS2 file system.

The compute nodes and PVFS2 server nodes ran a Linux 2.4.x kernel with the GCC v4.0.1 C compiler and g95 FORTRAN compiler. The PVFS2 clients and servers were using PVFS2 v1.5.1. We used the MPICH2 v1.0.3 implementation of the MPI-2 specification, and we used the 4.5 release of CMAQ.

The results shown in Figure 3 are from the experiments on the original and parallel I/O implementations of CMAQ that used the 2km_ppm_aero3 data set with a total simulation

[1] The use of hints for MPI-IO is described in Gropp (1999).

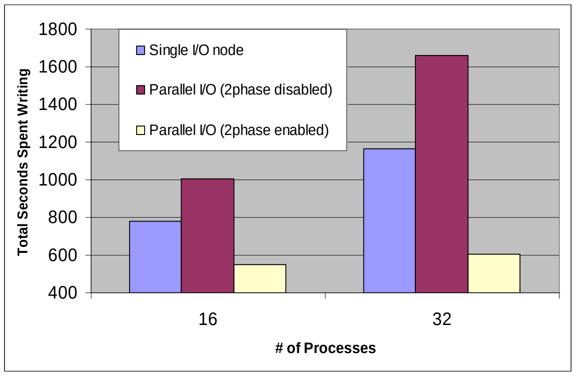

Fig. 3. Without the benefits of two-phase, total write time increases significantly as the number of processes increases.

time of 24 hours and a timestep of 1 hour. For the original implementation, we used the NFS server as a shared source of input files and wrote output files to the local disk on the I/O node. We tested the parallel I/O optimizations by writing to the PVFS2 parallel file system. We ran both implementations using 16 and 32 processors.

For the original implementation, 16 or 32 processors easily saturate the Fast Ethernet NIC on a single I/O node, even with the small data sizes that CMAQ uses. Just adding a parallel file system does not solve the problem. A parallel file system distributes the network load among the file servers, but without two-phase I/O enabled by MPI-IO, the performance is actually worse than using a single I/O node.

An interesting point to note is what happens to write time when the processor count doubles from 16 to 32 nodes. A single I/O node sees an increase in write time of 49%, while pnetCDF without the two-phase I/O optimization has an even more dramatic increase of 64%. In contrast to the other experiments, PnetCDF with two-phase I/O remains flat with an increase of only 10%.

5. RELATED WORK

Our approach to parallelizing the I/O by using the parallel netCDF library is not unique. A group at Northwestern University and ANL made similar modifications to an atmospheric modeling code called FLASH (Li 2003). In their study, they compared the performance of three I/O libraries: serial netCDF, parallel-netCDF and HDF5 (Cheng 2000). Our work is the first known effort to develop a truly parallel I/O interface for the CMAQ code.

6. FUTURE WORK

Research into CMAQ I/O and parallel file systems continues. There are optimizations possible at all levels of the parallel I/O software stack (Fig. 2). In particular, there are many potential optimizations for the MPI-IO library that warrant exploration.

CMAQ is an interesting tool for parallel file system developers, because it has small writes and interesting I/O patterns. We plan to create a CMAQ I/O kernel to study the behavior of the parallel I/O software stack. The I/O kernel will simulate the I/O patterns of the CMAQ code, but either skip or emulate the computation phase.

7. ACKNOWLEDGEMENTS

This work was supported by the High Performance Computing Consulting Project with the Environmental Protection Agency.

We thank Jeffrey Young from the National Oceanic and Atmospheric Administration for providing feedback, guidance, history and insight to the CMAQ code.

8. REFERENCES

Cheng, A. and Michael Folk, 2000: HDF5: High performance science data solution for the new millennium. *Proceedings of SC2000: High Performance Networking and Computing*, Dallas, TX, ACM Press and IEEE Computer Society Press.

Gropp, W., E. Lusk, and R. Thakur, 1999: *Using MPI-2: Advanced Features of the Message-Passing Interface*, MIT Press.

Latham, R., N. Miller, R. Ross, and P. Carns, 2004: A Next-Generation Parallel File System for Linux Clusters. *LinuxWorld Magazine*, Volume 2, Issue 1.

Li, J., W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, 2003: Parallel netCDF: A high-performance scientific I/O interface. *Proceedings of SC2003: High Performance Networking and Computing*, Phoenix, AZ, IEEE Computer Society Press.

Thakur, R., W. Gropp, and E. Lusk, 1999: Data sieving and collective I/O in ROMIO. *Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, IEEE Computer Society Press, 182-189.