Improving I/O Performance in POP (Parallel Ocean Program)

Wang Di [2] Galen M. Shipman [1]
Sarp Oral[1] Shane Canon [1]

[1] National Center for Computational Sciences, Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA,
{gshipman,oralhs,canonrs}@ornl.gov
[2] Santa Clara, CA 95054, USA,
wangdi@sun.com

Abstract. As clusters and supercomputers rapidly approach the petaflop scale, the performance demand on file I/O systems cannot be overlooked. The overall balance of these systems will shift dramatically towards FLOP performance and leave a number of applications spending a higher percentage of time in file I/O operations. Improving an application's file I/O performance may therefore have a dramatic effect on application run time and overall system throughput. The National Center for Computational Sciences (NCCS) Lustre Center of Excellence (LCE) has been established to address this challenge by working with science code teams in tuning application I/O performance. This paper describes our work in optimizing file I/O performance in the Parallel Ocean Program (POP). Our work shows that with proper tuning and modification substantial I/O performance improvements can be achieved. Our experiments show a 13x speedup in the I/O performance of POP.

1 Introduction

Large scale HPC systems have traditionally balanced file I/O performance with computational capacity by providing 2 byte/sec of I/O throughput for each 1000 FLOP of computational capacity [1]. To maintain this balance in the petascale regime requires delivering a minimum of 2 TB/s I/O performance for each PFLOP of computational capacity. Based on current and near term technologies this level of file I/O performance equates to installation and maintenance of tens of thousands of magnetic disks. An I/O system of this magnitude is cost prohibitive due to the sheer number of magnetic disks, RAID controllers and fibre channel links. Near term petascale systems will therefore be handicapped with what would traditionally be considered a suboptimal I/O / FLOP ratio.

Oak Ridge National Lab is scheduled to procure a petaflop system in the 2008 / 2009 time-frame. File I/O performance will not meet the traditional target ratio. Our target I/O performance is currently 200 GB/s for a petascale supercomputer which is a ful order of magnitude lower than what traditional balanced systems would provide. Given these constraints our challenge is to increase the end-to-end efficiency of the I/O subsystem. This involves work at multiple layers including hardware, OS, a number of software layers and the application's I/O architecture.

In order to better address this challenge a partnership with Cluster File Systems Inc. (CFS, now part of Sun Microsystems) has been established. A Lustre Center of Excellence (LCE) has been established as a result of this partnership at Oak Ridge National Lab. A primary activity of the LCE is assisting science teams in tuning their application I/O performances on NCCS supercomputers.

This paper describes our joint effort in optimizing I/O performance from an application perspective and involves enhancements at multiple layers. The end result of our work is a dramatic improvement in the I/O performance of the Parallel Ocean Program (POP) [2] on the NCCS Cray XT3/4 system, Jaguar.

The remainder of this paper is organized as follows: Section 2 provides an overview of POP including the current file I/O strategy. Improvements to the file I/O strategy are discussed in Section 3. Results of these improvements are demonstrated in Section 4. Finally, conclusions are discussed in in Section 5.

2 Background

2.1 The Parallel Ocean Program

POP was developed at Los Alamos National Lab (LANL) under the sponsorship of the DoE CHAMMP program [3] and is the ocean component of the larger Community Climate System Model (CCSM) [4]. POP is an ocean circulation model which solves the three-dimension primitive equations for fluid motions on the sphere [2]. It uses a 3-D mesh representation of the ocean model wrapping the sphere. In our representative test case POP grid dimensions are set as 3600x2800x42. For this particular setup, 42 is the depth of the ocean. POP is written in Fortran 90 and uses MPI [5] for communication. POP has been to shown to scale up to 22,000 MPI tasks on ORNL's Jaguar system [6].

2.2 The Original POP I/O pattern

POP, like many other scientific applications, exhibits a read-once-write-many I/O pattern. At application initialization POP reads a setup file and then throughout the run it performs write operations at each time step. These write operations are roughly 1 hour apart in our representative test case. Two different file I/O methods are used by POP; a Fortran record read/write method and a NetCDF method. Among these two implementations, only the Fortran record read/write method supports parallel I/O.

There are four files regularly written in POP: *history*, *movie*, *restart* and *tavg* files. The *history* file is written for each day in the model, while the other 3 files are written at each time step. The history file provides a static snapshot of the ocean model state and is written N times per averaging time-step. The *movie* file provides a visualization and animation of the ocean model and is relatively small compared with the other files; about 300 MB in size. A write operation for the movie file only costs several seconds (less than 5 seconds for most cases). The *tavg* file stores time average history of the ocean model in contrast to the static snapshot history. The *tavg* and *restart* files have similar I/O patterns where the *tavg* file is about 13 GB in size and *restart* file is about 28 GB. The I/O size of each file can be calculated as $(3600*2400*elementSize)$, where the $elementSize$ is the byte-length of an element. For example an integer element will be 32 MB $(3600*2400*4)$, whereas a floating point element will be 65 MB $(3600*2400*8)$.

Of the four files regularly written by POP the history file was shown to dominate most of the I/O and was therefore where we first focused our optimization effort.

The history file is segmented based on the model's horizontal layering of the ocean which resulted in 42 segments in our configuration.

Among two aforementioned file I/O implementations, only Fortran read/write method supports parallel I/O. Parallelization of the write I/O operations using Fortran formatted writes is done as follows: M processes within an N process job (where $M < N$) acts as aggregators and writers. The number of aggregators/writers is configurable from 1 up to the number of horizontal layers in the ocean. Data to be written is aggregated at each one of the M processes and each of these process writes to its designated area of a single shared file. In order to prevent conflicts each process writes to a distinguished record number which equates to a unique offset within the file. The NetCDF method uses a single aggregator/writer process and does not support parallel writers.

3 Improving POP I/O Performance

Our initial approach was to use the parallel Fortran write method to reduce the total time spent in history file I/O operations. With this method, each layer could easily be mapped to an aggregator/writer process. Unfortunately, this method resulted in file corruption and file size inconsistencies when writing to a single file from multiple writers. Although, at first it was assumed this was a bug in Jaguar's Lustre file system, more tests on Jaguar with a different file system also resulted in the same file corruption, ruling out a possible Lustre bug. The main reason for this problem is not yet known.

3.1 HDF5

Since the Fortran write method resulted in inconsistent file sizes with more than one processor performing I/O and netCDF does not support parallel I/O operations, the original I/O pattern was reduced to serial I/O. This resulted in I/O overhead of 10%-20% in our representative test case. In an effort to reduce this overhead, parallelization of I/O operations via HDF5 [7] was explored. HDF5 was chosen over other alternatives such as Parallel netCDF [8] due to better performance at scale [9].

HDF5 provides a file specification and library for parallel I/O applications. Given the shortcomings in the I/O structure of the original design, porting POP I/O to HDF5 was explored. Our implementation of POP with HDF5 layers the file I/O over MPI-IO. This design provides both collective and independent file I/O methods using $MPI\_FILE\_WRITE\_AT$ and $MPI\_FILE\_WRITE\_AT\_ALL$ internally and also ensures portability of the HDF5 library across a wide variety of parallel computing environments.

3.2 Data Sieving

Initial tests with the HDF5 for parallel I/O showed that the MPI-IO implementation on Jaguar uses data sieving [10]. Data sieving is a mechanism which allows non-contiguous file access from the process level via contiguous access at the file system level. For write operations multiple processes may be writing to different portions of a non-contiguous file via a read-modify-write operation. The first step of a data sieve write operation is to choose the sieve buffer size (default of 512KB). Data is then read from the correct offset in the file into the sieve buffer as illustrated in Figure 1. In this figure the different colors represent each processes view of the file. The first process' view of the file is indicated in white while the second process' is indicated in blue. The numbers are used to indicate the separate segments of data in the file.

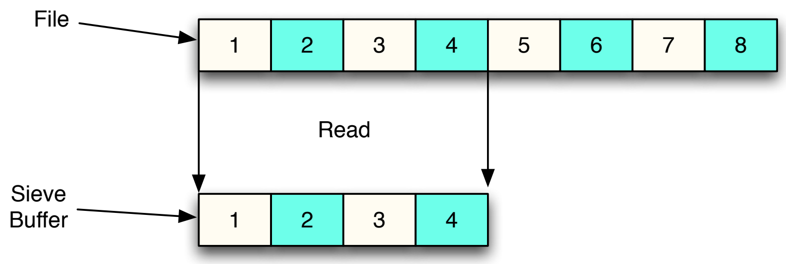

Fig. 1. Read File Data to Sieve Buffer

The data to be written from the process is then copied into the correct area of the data sieve buffer as in Figure 2. This corresponds to the modify stage of the read-modify-write operation. The color red in this figure is used to indicate data ready to be written. The numbering and arrows indicate that segments 1 and 2 which are contiguous in the write buffer are to be written to non-contiguous areas of the file. The color blue is used to indicate the unmodified segments of the file which correspond to another process' view. Rather than performing two separate writes of segments 1 and 2 of the write buffer to segments 1 and 3 to the file the intermediate sieve buffer is modified via a copy operation.

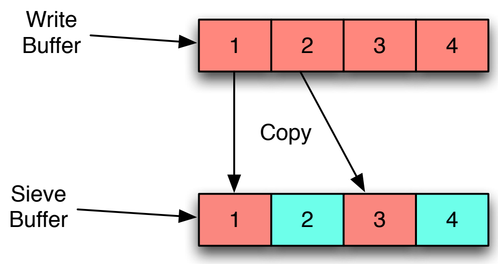

Fig. 2. Copy Write Buffer Data to Sieve Buffer

Figure 3 illustrates the write stage of the read-modify-write operation. At this stage the sieve buffer is written to the file via a single write operation. Note that the sieve buffer is contiguously written to a contiguous area of the file. In this example the read-modify-write operation reduces the number of writes issued by a process from two to one due to aggregation via the sieve buffer. Write operations are expensive on jaguar due to a number of factors including the overhead of flock (file locks to maintain consistency and coherency) as well as the overhead of RPCs (remote procedure calls) in Lustre. Lustre uses RPCs not only to transfer data to be written to remote physical storage but also to handle metadata operations such as granting locks to a file.

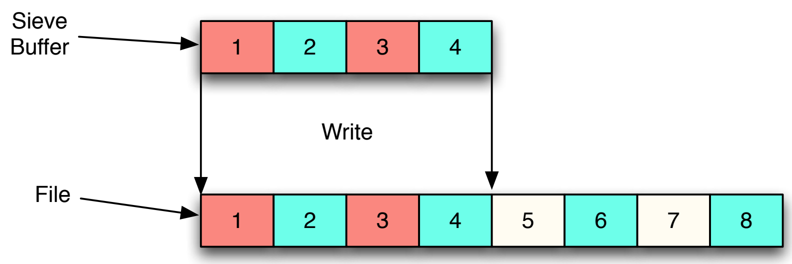

Fig. 3. Write Sieve Buffer to Disk

This default method of data sieving is inefficient when using HDF5 parallel writes in POP. The POP application writes contiguous blocks of memory to contiguous sections of a shared file as illustrated in Figure 4. The end result of data sieving was a read of the file blocks into the sieve buffer followed by a copy of the contiguous write buffer into the sieve buffer. As the write buffer was already contiguous the read (including an flock) and copy operations were unnecessary. The write buffer could have been directly written to the file. To eliminate these unnecessary operations the ADIO [11] *ADIO_GEN_WriteStrided* was modified to bypass the data sieving operation when the write buffer was contiguous.

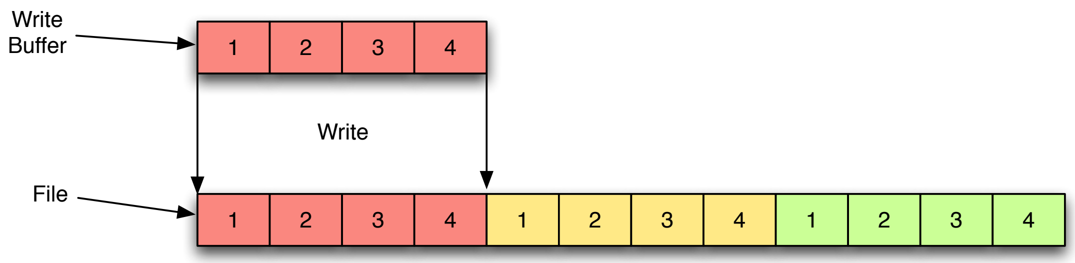

Fig. 4. POP I/O Writes Contiguous Regions

4 Results

4.1 Experimental Setup

The Jaguar supercomputer at the Oak Ridge National Lab was used to test our I/O improvements in POP. Jaguar is a Cray XT3/4 supercomputing platform and provides computational services for a broad spectrum of select applications. A performance evaluation of Jaguar has been recently published [6]. This, as well as several other studies have evaluated the I/O performance of Cray XT platforms [12],[9]. These studies have provided relevant information for scientific applications, such as peak system throughput and the impact of file striping patterns, etc.

A detailed description of Jaguar and its I/O subsystem is provided in [13]. Jaguar is a massively parallel processor architecture and is an aggregation of Cray XT3 and XT4 technologies. Both Cray XT3 and XT4 have similar architectures, whereas the XT4 has faster DDR2-667MHz memory and faster SeaStar2 interconnect compared to its predecessor, the XT3. The basic building block of Cray XT4 architecture is the Processing Element (Compute PE or Service PE), as illustrated in Figure 5. Each PE is currently equipped with a dual-core AMD processor with 2 GB per core of memory conencted by Hyper-Transport links, and a dedicated SeaStar2 communication chip. In our experiments Jaguar ran the Catamount [14] operating system on its compute PEs. A new OS, called Compute Node Linux (CNL), has recently been developed by Cray [15] as an alternative to Catamount and installed on the ORNL's Jaguar compute PEs. For both the Catamount and CNL versions, service PEs run a modified stock Linux kernel based on the SuSE distro. Portals [16] is used for inter-node communication.

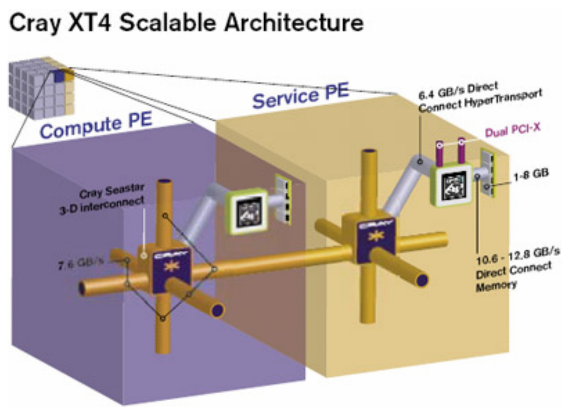
Cray XT4 Scalable Architecture
Compute PE
Service PE
6.4 GB/s Direct Connect HyperTransport
Dual PCI-X
1-8 GB
Cray Seastar 3-D interconnect
7.6 GB/s
10.6 - 12.8 GB/s Direct Connect Memory

Fig. 5. Cray XT System Architecture of Jaguar (Courtesy of Cray)

Jaguar uses the Lustre parallel file system [17]. Lustre is a POSIX compliant, object-based file system composed of three components: Metadata Servers (MDS), Object Storage Servers (OSS), and clients. Jaguar is configured with three Lustre file systems in various sizes, providing the required scratch storage space for data produced on Jaguar. 72 service nodes are configured as OSSes for these 3 file systems. In addition, each file system has its own dedicated MDS node. The backend storage hardware is S2A 9550 storage devices.

4.2 Results and Analysis

As detailed in Section 3 POP was modified to use HDF5 for I/O operations. Table 1 presents the comparative POP I/O overheads for the Fortran read/write, NetCDF, and the two aforementioned HDF5 methods. Here overhead is defined as the percent of time spent for I/O compared to the total execution time. As can be seen from Table 1 the HDF5 independent implementation significantly reduced the total I/O cost as well as decreasing the I/O overhead. However, the HDF5 collective I/O method did not perform as expected and even lowered the file I/O performance compared to a single aggregator/writer using Fortran record writes.

I/O method/library	I/O Processes	Time Step (min)	I/O Duration (min)	Overhead %
NetCDF	1	60	26	43
Fortran record	1	60	9	15
HDF5 Collective	42	60	12	20
HDF5 Independent	42	60	2	3

Table 1. POP I/O breakdown

To validate our findings and gain a deeper understanding for the reason of poor HDF5 collective I/O performance, we have performed further experiments using the IOR benchmark [18].

The first test we have performed uses HDF5 collective I/O over 42 processes. The same test was then repeated using independent I/O. Table 2 shows that HDF5 independent I/O performs better by two orders of magnitude when compared to parallel I/O.

HDF5 Mode	Access Mode	BW (MB/s)	Open (s)	wr/rd (s)	Close (s)
Collective I/O	Write	47.01	0.038791	55.50	0.237275
Independent I/O	Write	5048	0.056582	0.423846	0.235394

Table 2. IOR I/O comparison for collective and independent HDF5 methods

Through additional investigation the poor performance of HDF5 collective I/O was found to be a result of inefficient handling of contiguous writes to a non-contiguous file via ADIO data sieving as detailed earlier in Section 3.

Corrections to the ADIO layer as described in Section 3 made in order to improve the performance of HDF5 Collective I/O. Due to the OS change on the Jaguar supercomputer we were only able to verify our modifications to the ADIO layer for Catamount on a small-scale test system. As detailed in Table 3 HDF5 independent I/O performs better than HDF5 collective I/O. When data sieving is bypassed the performance of HDF5 collective I/O surpasses that of independent I/O.

5 Conclusions

Scaling applications to petaflop systems will require efforts on numerous fronts. I/O performance which has often been overlooked in application scalability will become increasingly important as near term petascale systems will be handicapped with

HDF5 Mode	Access Mode	BW (MB/s)	Block (KB/s)	Xfer (KB)	Open (s)	wr/rd (s)	Close (s)	iter
Independent I/O	Write	9.03	30720	30720	0.103011	9.85	6.23	0
Collective I/O (with sieving)	Write	3.47	30720	30720	0.094072	25.15	3.75	0
Collective I/0 (without sieving)	Write	14.41	30720	30720	0.078956	6.15	1.77	0

Table 3. IOR I/O comparison for collective and independent HDF5 methods after modifications on a small-scale test system.

a suboptimal I/O / FLOP ratio. Given this challenge, improving end-to-end I/O performance is of increasing importance.

Through careful analysis and modification of multiple layers we have shown that application I/O performance can be improved by a significant factor. Due to correctness and performance issues POP was modified to use HDF5. This modification allowed the use of multiple aggregators/writers which showed performance improvement when HDF5 independent I/O was used. Further modifications to the ADIO layer showed that HDF5 collective I/O may additionally improve performance. These modifications improved POP I/O performance by a factor of 13 and reduced the overall I/O overhead from 15% to 3%.

6 Acknowledgements

The authors would like to thank the staff and colleagues who have contributed material to this paper. Thanks to Trey White and Patrick Worley for assistance, suggestions and corrections. Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

References

1. Nowak, D.A., Seagar, M.: ASCI terascale simulation: Requirements and deployments. http://www.ornl.gov/sci/optical/docs/Tutorial19991108Nowak.pdf
2. LANL: Parallel ocean program (POP). http://climate.lanl.gov/Models/POP
3. Program, D.C. http://www.csm.ornl.gov/chammp/
4. UCAR: Community Climate System Model. http://www.ccsm.ucar.edu/
5. Geist, A., Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Saphir, W., Skjellum, T., Snir, M.: MPI-2: Extending the Message-Passing Interface. In: Euro-Par '96 Parallel Processing, Springer Verlag (1996) 128–135
6. Alam, S.R., Barrett, R.F., Fahey, M.R., Kuehn, J.A., Larkin, J.M., Sankaran, R., Worley, P.H.: Cray XT4: An early evaluation for petascale scientific simulation. In: Proceedings of the ACM/IEEE conference on High Performance Networking and Computing (SC07), Reno, NV (2007)
7. UIUC: Hierarchical data format (HDF5). http://hdf.ncsa.uiuc.edu/products/hdf5/index.html
8. Li, J., Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R.: Parallel netcdf: A scientific highperformance i/o interface (2003)
9. Shan, H., Shalf, J.: Using IOR to analyze the I/O performance of XT3. In: Proceedings of the 49th Cray User Group (CUG) Conference 2007, Seattle, WA (2007)
10. Thakur, R., Gropp, W., Lusk, E.: Data sieving and collective I/O in ROMIO. In: Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation, IEEE Computer Society Press (1999) 182–189
11. Thakur, R., Gropp, W., Lusk, E.: An abstract-device interface for implementing portable parallel-I/O interfaces. In: Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation. (1996) 180–187

12. Laros, J., Ward, L., Klundt, R., Kelly, S., Tomkins, J., Kellogg, B.: Red Storm IO performance analysis. In: Proceedings of the IEEE Cluster 2007, Austin, TX (2007)
13. Yu, W., Vetter, J., Oral, S.: Performance characterization and optimization of parallel I/O on the Cray XT. In: Submitted to the IPPDS 2007
14. Kelly, S.M., Brightwell, R.: Software architecture of the light weight kernel, Catamount. In: Proceedings of the 47th Cray User Group (CUG) Conference 2005. (2005)
15. Wallace, D.: Compute node linux: New frontiers in compute node operating systems. In: Proceedings of the 49th Cray User Group (CUG) Conference 2007, Seattle, WA (2007)
16. Brightwell, R., Riesen, R., Lawry, B., Maccabe, A.B.: Portals 3.0: Protocol building blocks for low overhead communication. In: Proceedings of the 2002 Workshop on Communication Architecture for Clusters (CAC). (2002)
17. Lustre. "http://wiki.lustre.org/"
18. LLNL: IOR benchmark. http://www.llnl.gov/asci/purple/benchmarks/limited/ior