Methodology and Application of HPC I/O Characterization with MPIProf and IOT

Yan-Tyng Sherry Chang, Henry Jin
NASA Advanced Supercomputing Division
NASA Ames Research Center
Moffett Field, CA 94035-1000, USA
{sherry.chang,haoqiang.jin}@nasa.gov

John Bauer
I/O Doctors, LLC
15054 County Road 20, P. O. Box 223
Hanska, MN 56041-0323
bauerj@iodoctors.com

Abstract*—Combining the strengths of MPIProf and IOT, an efficient and systematic method is devised for I/O characterization at the per-job, per-rank, per-file and per-call levels of HPC programs running on the NASA Advanced Supercomputing Center. This method is applied to answer four I/O questions in this paper. A total of 13 MPI programs and 15 cases, ranging from 24 to 5968 ranks, are analyzed to establish the I/O landscape from answers to the four questions. Four of the 13 programs use MPI I/O and the behavior of their collective writes depends on the specific implementation of the MPI library used. The SGI MPT library, the prevailing MPI library for our systems, was found to gather small writes from a large number of ranks to perform larger writes by a small subset of collective buffering ranks. The number of collective buffering ranks invoked by MPT depends on the Lustre stripe count and the number of nodes used for the run. A demonstration of varying the stripe count to achieve double-digit speedup of one program's I/O was presented. Another program, which concurrently opens private files by all ranks and could potentially create a heavy load on the Lustre servers, was identified. The ability to systematically characterize I/O for a large number of programs running on a supercomputer, seek I/O optimization opportunity and identify programs that could cause a high load and instability on the filesystems is important for pursuing exascale in a real production environment.

Keywords—MPI I/O; POSIX I/O; Lustre; Stripe

I. Introduction

The advances in high-performance computing (HPC) processor, memory, network and storage technologies are pushing to materialize exascale computing in the near future. The June 2016 Top500 list [1] shows that 95 supercomputers in the world have achieved more than 1 PetaFlops. Aside from the temporary glory of such achievement, a few important tasks of HPC centers are: (i) ensuring the availability and stability of resources for their users, (ii) educating users on how to effectively utilize these shared resources, and (iii) planning for future infrastructure where scientists and engineers can continue pursuing larger scale simulations and generating increasing amounts of data to solve problems with higher fidelity and complexity.

When pursuing exascale for real applications, I/O is becoming more important. An application cannot scale well if it is hampered by its I/O. A large-scale simulation can fail due to single points of failure in the processor, memory or network, and HPC users are advised to increase the frequency of checkpointing to avoid losing valuable results due to system failures. The shared parallel filesystems commonly deployed at HPC centers also require users to know not only how to better exploit them but also how not to create issues that may impact other users. HPC centers often create their own monitoring tools to check the health of their filesystems and identify user jobs that cause harm to them. A more challenging task is how to correlate issues seen at the system level to the characteristics of I/O performed by user codes.

At the NASA Advanced Supercomputing (NAS) Center, the Lustre [2] shared filesystems have grown over the years to a total of approximately 1300 Object Storage Targets (OSTs) and 30 PetaBytes (PB). A software tool [3] based on SGI's Performance Co-Pilot was developed recently to provide, at the end of each user job, Lustre statistics such as (i) the amount of data read and written, (ii) the number of files opened and closed, and (iii) the I/O size distribution. Such job-level statistics, though useful to the system administrators, are harder for the users to comprehend. In this study, a robust methodolgy which combines the strengths of two performance analysis tools, MPIProf [4] and IOT [5], is applied to reveal in-depth I/O characteristics of real user programs. The end results of this study include:

- Establishing the I/O landscape of codes at NAS
- Exposing details in the MPI I/O implementation
- Improving I/O for better scaling for some code(s)
- Identifying code(s) which cause heavy load on Lustre

The ability to provide quantitative I/O measurement of real applications to identify potential optimization in user codes and/or the institution's I/O infrastructure will bring users and HPC centers one step closer to pursuing exascale in the long term.

II. Related Work

Throughout the years, many research and commercial tools have been developed to analyze application performance and study I/O characteristics. Only a few that have relevance to the current work for I/O characterization are cited here. Full-scale profiling tools, such as TAU [6] and OpenSpeedshop [7], provide certain level of support for I/O profiling and tracing. But it is often challenging for a user to fully master these tools, since they are designed as a workhorse for application performance optimization with an emphasis on deep-diving, comprehensive performance analysis. A closer tool to MPIProf in functionality is MPInside [8] from SGI, which has a simple interface for collecting MPI and I/O statistics of MPI applications. Unfortunately the tool is a commercial product only available on SGI machines and is not currently well-maintained.

Darshan [9] is an I/O characterization tool that was designed to report per-file I/O behavior and access patterns of applications. It has low overhead in collecting I/O statistics (both MPI I/O and POSIX I/O) and has been successfully demonstrated for applications at extreme scale. Ftracer [10] is a trace-based I/O analysis tool with an aim at more detailed tracing information. It uses a double-buffered mechanism to reduce tracing overhead. Attempts have been made in defining high-level metrics for I/O characterization and understanding different workloads at data centers. Uselton and Wright [11] introduced a metric called file system utilization (FSU) to connect different I/O characteritics. Work has also been carried out to characterize I/O bebaviors of scientific applications in data centers [12,13]. The main difference of our approach in this work is that we define a methodology combining the strengths of two tools for I/O characterization and apply it to a set of production NASA applications.

III. Methodology

A. MPIProf

MPIProf is a NASA in-house tool developed initially as an MPI profiling tool and later expanded to add support for memory usage and I/O profiling. Similar to the SGI MPInside [8] and Intel MPI Performance Snapshot (MPS) [14], the tool was designed to gather statistics of MPI functions, including MPI I/O, and POSIX I/O calls made by an application. MPIProf provides information such as time spent, number of calls, and number of bytes transferred at both the job level and the per-MPI rank level.

Detailed description of MPIProf can be found in [4]. Briefly, MPIProf gathers statistics in a counting mode and instruments MPI functions by the PMPI interface and POSIX I/O via wrappers for shared libraries. The tool takes a lightweight approach in collecting performance data and has a very simple command-line interface without the need for modifying or recompiling applications. The most straightforward way to perform profiling with MPIProf is as follows:

```
mpiexec –np <n> mpiprof [-options] a.out [args]
```

where the tool loads the proper profiling environment, including libraries, and writes results at the end of the run. In its text output, MPIProf provides an overall run summary, per-function summary, data size histograms, aggregated and per-rank time spent, number of calls and transfer size. To focus on I/O profiling, the *–ios* option in MPIProf reports I/O statistics only, including MPI I/O functions, POSIX I/O called by MPI I/O, and other POSIX (non-MPI) I/O.

B. IOT

IOT [5] is a licensed toolkit developed by I/O Doctors, LLC, for I/O instrumentation and optimization of HPC programs. It allows detailed analysis at various levels: per-job, per-rank, per-file, and per I/O operation. In addition to information such as time spent, number of calls and number of bytes transferred, IOT also provides information on when and where in the file the I/O occurs.

iot is a utility program of the IOT toolkit that configures the runtime environment for instrumentation. All child processes of *iot* can be subject to *iot* instrumentation. The user-supplied *iot* configuration allows fine-grained control of *iot* layers and layer options that will be applied to selected programs and files. The instrumentation result of each child process is recorded in an *ilz* stream for post-mortem analysis with IOT's *Pulse* GUI. There is minimal change to the user script as the *iot* command is a pre-command, similar to *time*. On NASA Supercomputing system Pleiades [15], the following command is used for instrumenting an executable, represented as a.out below:

```
iot -m mpiflavor -f cfg.icf -c pfe:`pwd`/prg.${JOB_ID}.ilz \
        mpiexec -np <n> a.out [args]
```

where

- *-m* specifies the MPI implementation used, typically it is *mpt* for applications running on Pleiades
- *-f* specifies the *iot* configuration file
- *-c* merges all *ilz* streams from all instrumented processes into a single stream
- *pfe* is one of the Pleiades front-end nodes where iot collects all the *ilz* streams from the child processes running on various compute nodes
- *${JOB_ID}* is the JOBID from Pleiades batch job scheduling system PBS

Among the layers available in IOT, the *trc* layer provides low overhead instrumentation of intercepted I/O calls. The *trc* layer has multiple levels of user selectable instrumentation allowing the user to control the level of detail and volume of instrumentation data. The detail can range from a per process aggregation of the counts, sizes, and wall clock deltas of each I/O operation type to the collection of the metrics of every I/O operation for every file of every process.

In addition to the process-specific *trc* layer instrumentation, IOT can also monitor system performance information such as memory, network, disk, and cpu utilization (via the */proc* filesystem), at a user selectable time interval. This allows correlation of a process's activity with system activity. IOT also has Lustre-specific monitoring for lnetstat (Lustre network statistics) and OSC (Object Storage Client) behavior.

The resulting ilz stream can then be analyzed via *Pulse*:

java –jar Pulse.jar prg.jobid.ilz

Pulse will present the data in multiple hierarchical fashions, including a hierarchical tree format, table format, and graphical format, to facilitate the interpretation of the data.

C. *Procedure*

In this section, a procedure is described for analyzing a given test case of a program. This procedure combines the strengths of MPIProf and IOT where MPIProf is used to provide a quick overview of the program's I/O at every single MPI rank, and IOT is then deployed on selected ranks for deeper analysis at the per-file or even per-call levels. As a result, I/O analysis can be done quickly without the burden of creating a huge amount of analysis data from all MPI ranks.

There are multiple questions one can address with this procedure. A small subset is included in this paper:

- Does the program do MPI I/O and/or POSIX I/O?
- Is the I/O dominated by a single rank, multiple ranks or all ranks?
- What is the amount of time spent doing I/O and what I/O operation (open, close, read, write, etc.) dominates (in terms of time)?
- When does I/O occur during the run?

Answers obtained from the analysis of these questions at a per-job and/or per-rank level will be useful for an HPC center to establish the I/O patterns of programs running on a system and to help determine future I/O infrastructure better suited for its user community. Additional information obtained at a per-file and/or per I/O call level from the analysis may be useful to the program developer and users for potential I/O optimization.

The procedure for analyzing each un-recompiled program and/or test case is as follows:

- Modify user's job script and run the executable under MPIProf with the *–ios* option to focus on I/O.

  The first three questions above at the per-job and per-rank levels are addressed from this step. In addition, results obtained for the first two questions will be useful to help choose the MPI ranks to track by IOT in the next two steps.

- Modify job script and run the executable under IOT with the *trc.totaliops* and *trc.iops* options enabled on selected MPI ranks.

  This step will address the last two questions at a per-rank and per-file level. Results obtained from this step also verify the results obtained from MPIProf.

- Modify the IOT configuration file to enable the *trc.totaliops, trc.iops* and *trc.events* options and run the executable under iot on selected MPI ranks.

  This step will provide more detailed information on a per-I/O-call level.

- If needed, change the runtime environment (such as the filesystem, stripe count, or stripe size used, etc.) and repeat steps.

IV. Application

The methodology is applied to user programs running on the NAS Pleiades supercomputer. Pleiades [15] is comprised of 11,472 compute nodes of four Intel processor generations (Sandy Bridge, Ivy Bridge, Haswell and Broadwell) connected via two InfiniBand (IB) fabrics in a partial 11-dimensional hypercube topology. One IB fabric is used for MPI communication while the other is mainly used for I/O. Multiple Lustre filesystems (/nobackupp[1-9]), with a total of 1300 OSTs and 30 PB, are shared among ~1600 users. Compute resources are allocated to each user project in Standard Billing Units (SBU) [16], where 1 SBU is 1 node-hour on the recently retired Westmere processors. For Sandy Bridge, Ivy Bridge, Haswell and Broadwell nodes, using 1 node-hour will cost more SBUs due to the better capability of these processors. The total allocatable SBUs per month are more than 10 million. The top-10 codes, in terms of SBU usage, frequently consume more than 50% of the total SBUs.

Table I lists a total of 13 programs and 15 cases analyzed. Among these 13 programs, ATHENA, ENZO, FLASH, MCONV and SOLARBOX are Astrophysics programs. ECCO, FVCORE and WRF are used for weather modeling. FUN3D, LOCI-STREAM, OVERFLOW and USM3D are Computational Fluid Dynamics (CFD) codes for various aero-related fields. All executables and test cases were obtained directly from users except FLASH, which was downloaded from the Parallel I/O Benchmarking Consortium web site [17]. The ATHENA program, provided by a user, comes with two versions, C and C++, and they are labeled in Table I accordingly. The programs marked with * in the table, are among the top-10 codes on Pleiades.

TABLE I. List of Programs and Cases Analyzed

Program	**Resources**	**Number of MPI Ranks**
*ATHENA-C	84 Has x 24	2048
*ATHENA-C++	136 Has x 24	3264
*ATHENA-C++	214 Bro x 28	5968
*ECCO	5 Has x 24	120
*ENZO	12 Ivy x 20	240
FLASH	1 Has x 24	24
FLASH	5 Has x 24	120
*FUN3D	40 Has x 24	960
FVCORE	59 Ivy x 20	1176
*LOCI-STREAM	120 Ivy x 20	2400
*MCONV	202 Ivy x 20	4032
*OVERFLOW	5 Has x 24	120
*SOLARBOX.	80 Ivy x 20	1600
USM3D	10 Has x 24	240
WRF	16 Has x 24	384

*One of Pleiades' top-10 programs

All of these programs and cases use only MPI for parallel processing (i.e., no OpenMP), which is typical for jobs running on Pleiades. The computing resources used for each of these MPI programs and cases are shown in the 2nd column, where Ivy, Has and Bro represent the Intel Ivy Bridge (20 cores/node), Haswell (24 cores/node) and Broadwell (28 cores/node) processor nodes [15]. The number of ranks used, shown in the last column, ranges from 24 to 5968, which is also representative of the workload on Pleiades. For ATHENA-C++, two test cases were provided by the user, one with 3264 MPI ranks and the other with 5968 MPI ranks. For FLASH, two cases were tested, one with 24 MPI ranks, and the other with 120 MPI ranks. All programs, with one exception, were built by users with SGI MPT library and run with MPT version 2.12r26. The MCONV program was built by the user with Intel MPI library and run with Intel MPI version 5.0.3.048.

Unless stated explicitly, all runs were performed on /nobackupp8, which includes a total of 312 OSTs and 6.6 PB of disk space, with a stripe count of 1 and a stripe size of 4 MB on the run directories and files.

V. Results

A. *MPI I/O and/or POSIX I/O*

The MPIProf text output file contains a section of "Instrumented function list" which displays the functions intercepted from a program. Fig. 1 shows an example from the ATHENA-C++ program where multiple MPI I/O functions are called. For clarity purpose, MPIProf uses short names (e.g. mread, mwrite, ewrite, ewritec, etc.,) to represent the MPI functions in its tabulated output sections. Since the MPI I/O functions eventually call POSIX I/O functions, the table marks those POSIX I/O called by MPI I/O functions with "<". In addition to POSIX I/O called by MPI I/O, there are other POSIX I/O which are not called by MPI I/O and they are shown without "<".

```
==> Instrumented function list
1          mopen - (3) MPI_File_open
2          mread - (4) MPI_File_read
3         mwrite - (4) MPI_File_write
4         ewrite - (4) MPI_File_write_at
5        ewritec - (3) MPI_File_write_at_all
6         mclose - (3) MPI_File_close
7           open - (5) IO_open
8          write - (5) IO_write
9          close - (5) IO_close
10         <open - (6) <IO_open
11         <read - (6) <IO_read
12        <write - (6) <IO_write
13        <close - (6) <IO_close
(3)       c-mpio = collective MPI I/O
(4)      nc-mpio = non-collective MPI I/O
(5)        px-io = posix I/O
(6)       <px-io = posix I/O from MPI I/O
+         Ovhead - MPIProf overhead time
```

Fig. 1. I/O functions called by ATHENA-C++

Among all the programs analyzed, ATHENA-C++ (A), FLASH (F), MCONV (M) and SOLARBOX (S) are the only four that use MPI I/O in addition to non-MPI POSIX I/O. The MPI I/O functions called by each of these four programs are listed in Table II. Interestingly, all four of them fall in the Astrophysics discipline.

TABLE II. MPI I/O Functions Called By Programs

MPI I/O Function	**A**	**F**	**M**	**S**
MPI_File_open	x	x	x	x
MPI_File_read_all			x	
MPI_File_write_all			x	
MPI_File_write_at_all	x	x		
MPI_File_read_ordered				x
MPI_File_write_ordered				x
MPI_File_read	x			
MPI_File_write	x			
MPI_File_write_at	x	x		
MPI_File_read_shared				x
MPI_File_write_shared				x
MPI_File_close	x	x	x	x

B. *I/O Dominant Rank(s)*

MPIProf text output contains multiple sections about the time used, number of function calls and amount of data read/written by each rank. From these sections, one can determine whether the program's I/O is dominated by a single rank, multiple ranks or all ranks. Although there are exceptions, in many programs analyzed, these sections depict the same dominance pattern. For the programs that do not use MPI I/O, all except ATHENA-C show I/O dominated by rank 0. Except for FUN3D, these programs also have small amounts of I/O done by all other ranks. For ATHENA-C, I/O is evenly distributed among all ranks.

For programs that do MPI I/O, the behavior is more complicated. In particular, the ranks that dominate the collective writes, if called by the program, could be different depending on the specific implementation of the MPI library used. With the SGI MPT library, when multiple processes are writing to the same file in a coordinated manner, the different processes send their writes to a subset of collective buffering ranks to do a smaller number of bigger writes, which could potentially improve the I/O performance. The two main factors in the MPT algorithm for choosing how many ranks to do the writes are: the stripe count and number of nodes. When the number of nodes is greater than the stripe count, the number of collective buffering ranks is the stripe count. Otherwise, the number of collective buffering ranks is the largest integer less than the number of nodes that evenly divides the stripe count. In addition, MPT chooses the first rank from the first n nodes to come up with n collective buffering ranks. This is demonstrated in Fig. 2 and Fig. 3 of the FLASH program where the <px-io call counts from the ranks that actually perform the underlying POSIX I/O are shown in blue while those of collective MPI I/O (c-mpio) calls from all ranks are in red.

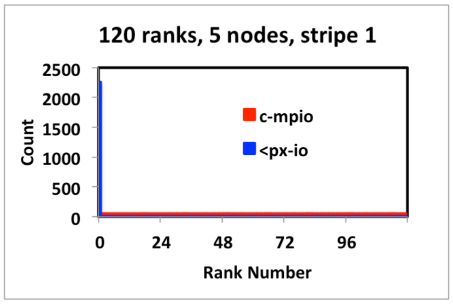

Fig. 2. Rank 0 dominating case of FLASH

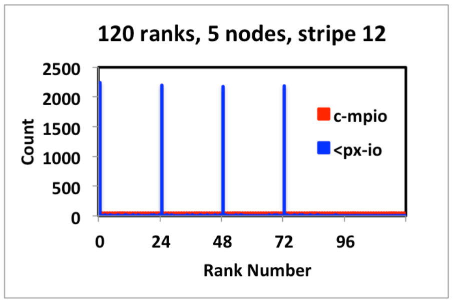

Fig. 3. 4 ranks dominating case of FLASH

Similar to FLASH, all MPI ranks for the 3264-rank and 5968-rank runs of ATHENA-C++ and the 1600-rank runs of SOLARBOX call collective MPI write functions. These programs also show the change from rank 0 dominating <px-io with a stripe count of 1 to multiple ranks dominating <px-io with a stripe count > 1. For example, running SOLARBOX with 80 Ivy Bridge nodes, each with 20 MPI ranks, and using a stripe count of 16, the 16 ranks dominating <px-io are ranks 0, 20, 40, …, 300.

Results from runs with IOT also verify the I/O dominant ranks seen with MPIProf. In addition, more details on how the SGI MPT library dispatches data to the collective buffering ranks for writes are revealed by runs using the IOT *trc.events* level enabled. With a stripe count of 1, data are dispatched in 4 MB chunks to rank 0. When the stripe count is n, data are dispatched in 1 MB chunks to the n collective buffering ranks in a round-robin manner. This is demonstrated in Fig. 4 using SOLARBOX's writing of a large restart file via the MPI_File_write_ordered as an example. Specifically, with a stripe count of 16 where 16 ranks (i.e., ranks 0, 20, 40, …, 300) collect data from all 1600 ranks for <write, rank 0 writes the 1st, 17th, 33th, 49th… 1 MB chunks, rank 20 writes the 2nd, 18th, 34th, 50th… chunks, …, and rank 300 writes the 16th, 32th, 48th, 64th, … chunks. For clarity, only ranks 0, 100, 200 and 300 are shown in Fig. 4. Furthermore, analysis with IOT at a per-call level also reveals that these <writes are done mostly sequentially, a common characteristic for all programs analyzed in this study.

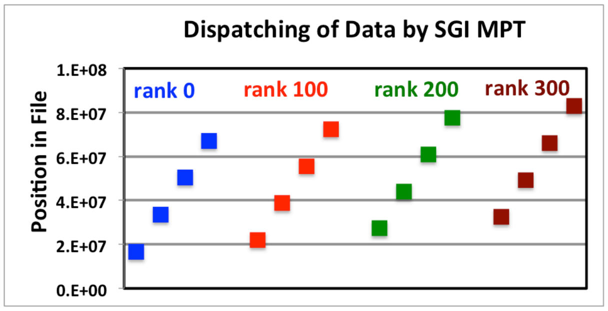

Fig. 4. Sequential, round-robin dispatching of data in SOLARBOX

Contrary to what is observed with SGI MPT, using the only program (MCONV) built with Intel MPI library, no collective buffering for <write is observed. That is, all the ranks that invoke the collective write MPI functions call the underlying POSIX write functions.

For MPI collective reads, neither SGI MPT nor Intel MPI implements the use of collective buffering ranks as in the MPI collective writes.

C. *Amount of Time Spent in I/O Functions*

Both MPIProf and IOT report the amount of time spent in I/O. Each one has its advantage over the other. MPIProf reports separate I/O time between MPI I/O and non-MPI I/O. For the MPI I/O portion, one can also deduce the contribution between communication and the POSIX I/O called underneath. IOT has the advantage of tracking some I/O functions, such as stat(), unlink(), etc., that are not tracked by MPIProf. It also shows the I/O on a per-file and per-call level.

To get a landscape of I/O time spent among the programs, results from IOT for each program/case with the *trc.iops* enabled but with *trc.events* disabled and with a stripe count of 1 are shown in Table III. For ease of comparison, timing in this table is from rank 0 only. Column 2 shows the total runtime in sec. Columns 3 and 4 show the total I/O time in sec (including POSIX I/O time from both MPI I/O and non-MPI I/O, but not the communication time from MPI I/O) by this rank and the percentage time spent in I/O relative to the total runtime.

TABLE III. RANK 0 TIMING WITH STRIPE COUNT OF 1 FROM IOT

Program	Runtime	I/O	I/O %
ATHENA-C	584	20.7	3.5
ATHENA-C++ (3264)	520	354.3	68.1
ATHENA-C++ (5968)	1659	1163.2	70.1
ECCO	35	12.4	35.4
ENZO	740	51.8	7.0
FLASH (24)	11	5.1	46.4
FLASH (120)	25	16.8	67.2
FUN3D	1299	14.8	1.1
FVCORE	1161	0.9	0.0
LOCI-STREAM	3692	103.0	2.8
MCONV	2969	47.1	1.6
OVERFLOW	195	17.1	8.8
SOLARBOX.	4207	86.4	2.1
USM3D	377	45.4	12.0
WRF	1358	16.4	1.2

Among these programs and cases, ATHENA-C++ and FLASH show more than 40% of runtime spent in I/O. The two ATHENA-C++ cases provided by a user contain a higher frequency of I/O operations than his normal production runs. The FLASH case was downloaded from web and is designed to study I/O. It is thus expected that the percentage time spent in I/O would be smaller for production cases of these two programs. Nonetheless, a long runtime and high I/O% as in the ATHENA-C++ cases provide incentive to seek I/O optimization opportunity. Timing analysis of four I/O function types in Fig. 5 shows that most I/O time of ATHENA-C++ cases is spent in write operations followed by open operations and no time is spent in read and close operations. The analysis described below illustrates the use of MPIProf and IOT to improve the I/O performance of the ATHENA-C++ program.

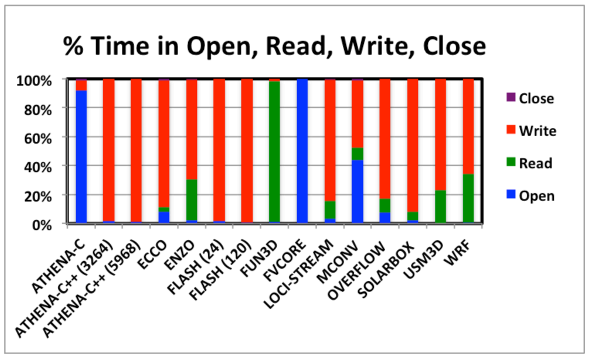

Fig. 5. Percentage I/O time spent in open, read, write, and close

As described earlier, with SGI MPT, only rank 0 is used as the collective buffering rank in the collective MPI write operations when the stripe count is 1, while multiple ranks are used when the stripe count is > 1. In addition, the write size is changed from 4 MB for stripe 1 to 1 MB for stripe > 1. Using either MPIProf or IOT, it was observed that changing the stripe count from 1 to a larger number also affects the I/O time. Figs. 6 and 7 compare the timing among different stripe counts for the ATHENA-C++ 3264-rank and the 5968-rank cases, respectively, using results obtained from IOT. In Fig. 6 for the 3264-rank case, it is observed that (i) the total write time dominates the total I/O time for all three stripe counts, (ii) the total write time decreases from 348 sec for stripe 1 to 33.2 sec for stripe 16 and 28.5 sec for stripe 64, (iii) the total open time stays about the same at ~6 sec for all three stripe counts. The improvement in the total write time with a stripe count of 16 vs a stripe count of 1 results in a speedup factor of 10x for the total write time, 9x for the total I/O time and 4x for the total runtime. Comparing timing between stripe count of 64 and stripe count of 1, the speedup factor is 12x for the total write time, 10x for the total I/O time and 4x for the total runtime. A similar trend is seen in Fig. 7 for the 5968-rank case, except that the improvement for the total write time is even more dramatic - 1152, 47.5, 26.0 and 21.0 sec for stripe counts of 1, 16, 64 and 128, respectively. This improvement results in (a) a speedup factor of 24x for total write, 20x for total I/O and 5x for total runtime for a stripe count of 16 vs 1, (b) a speedup factor of 44x for total write, 30x for total I/O and 6x for total runtime for a stripe count of 64 vs 1, and (c) a speedup factor of 55x for total write, 36x for total I/O and 8x for total runtime for a stripe count of 128 vs 1. Although timing results from MPIProf are not shown here, they portray very similar speedup factors as those obtained from IOT for both the 3264-rank and the 5968-rank cases.

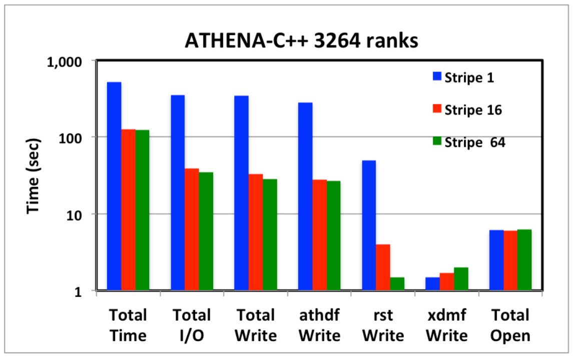

Fig. 6. Rank 0 timing of ATHENA-C++ 3264 ranks from IOT

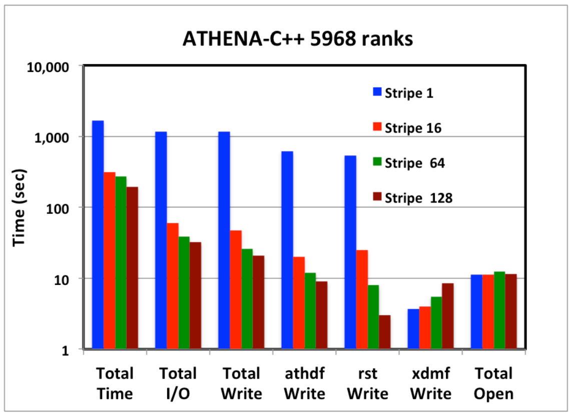

Fig. 7. Rank 0 timing of ATHENA-C++ 5968 ranks from IOT

Further characterization of the I/O timing for the ATHENA-C++ program at the per-file level was obtained by combining information from both MPIProf and IOT. As shown in Table II, in addition to the non-MPI POSIX write function, ATHENA-C++ calls three different MPI write functions – the collective MPI_File_write_at_all and the non-collective MPI_File_write and MPI_File_write_at. Since MPIProf does not profile I/O for each file, in the absence of the program source code, one can only speculate on the type of write calls for different files. Table IV combines results from different output sections of MPIProf for the ATHENA-C++ 5968-rank case with a stripe count of 1. It was deduced from this table that the POSIX writes from (i) the collective MPI_File_write_at_all was done by rank 0 with a total size of about 323 GB. This 323 GB is close to the total size of 6 output files with suffix .rst; (ii) the non-collective MPI_File_write_at was done by all ranks with a total size of about 14 GB. This 14 GB is close to the total size of 6 output files with suffix .athdf; (iii) the non-MPI write was done only by rank 0 for a total size of about 60 KB. This 60 KB is close to the total size of 6 output files with suffix .xdmf. Since the time spent in MPI_File_write is zero, no attempt was made to figure out what files were written by these calls. Changing the stripe count from 1 to > 1 only increases the number of collective buffering ranks used for the collective MPI_File_write_at_all. It does not affect the number of ranks used for POSIX I/O from the non-collective MPI write and the non-MPI write functions.

TABLE IV. POSIX WRITES OF ATHENA-C++ 5968-RANK CASE

POSIX write from	Rank(s)	Time (sec)	Size (Byte)
MPI_File_write_at_all	0	~590*	~323G
MPI_File_write_at	all	~640 (rank 0)*	~14G
MPI_File_write	0	~0	~20K
Non-MPI	0	~2	~60K

* Time includes both communication and <write

Results from IOT confirm the inference obtained from multiple MPIProf output sections. That is (i) the .rst files by the collective MPI write function are written by 1 rank with a stripe count of 1, 16 ranks with a stripe count of 16, and so on. When multiple ranks are used, data in each file is equally divided and dispatched in 1 MB chunks in a round-robin fashion to the ranks; (ii) the .athdf files by the non-collective MPI write function are written by all ranks irrespective of the stripe count, where data in each file is equally divided and dispatched in 8 – 16 KB chunks among all ranks; and (iii) the .xdmf files by non-MPI POSIX write are written by only rank 0 regardless of the stripe count in either 4 or 8 KB chunks. As seen in Figs. 6 and 7, increasing the stripe count has the most dramatic speedup for the writing of the 6 .rst files due to a combination of (a) the large sizes of these files (~44 GB in total for the 3264-rank case and ~323 GB for the 5968-rank case), (b) the use of multiple ranks for parallel writes, and (c) the use of multiple OSTs of the Lustre filesystem which provides additional parallelism. The speedup for writing the 6 smaller .athdf files is significant but levels off faster than the writing of the 6 .rst files. Since the 6 .athdf files are written by all ranks regardless of the stripe count, the speedup only comes from further parallel handling of writing data through multiple OSTs of Lustre. For the 6 .xdmf files, only rank 0 is used for writes, and increasing the stripe count actually slows them down, though the effect is quite small due to the very small sizes of these files.

The analysis of the ATHENA-C++ 3264-rank and 5968-rank cases with both MPIProf and IOT allows for better understanding of the I/O characteristics of this program and why using a stripe count of 1, as was originally used in the user's production runs, prevents scaling to larger number of ranks. With a larger problem size and a larger number of MPI ranks, it is important to increase the stripe count accordingly to better take advantage of multiple parallelism from both the MPI I/O library and the Lustre filesystem in order to minimize the I/O time for better scaling. For the production cases of the ATHENA-C++ program where the I/O time is a much smaller fraction of the total runtime, the user has reported performance improvement of the total runtime by 10 – 20% simply by increasing the stripe count from 1 to 16. The ability to associate specific MPI I/O calls with files and their I/O time, as demonstrated in this study, would help the ATHENA-C++ developer to experiment different I/O approaches for individual files when trying to scale this program to more than 10,000 MPI ranks.

D. *When Does I/O Occur?*

In addition to knowing what ranks, what I/O sizes and the amount of time spent in the I/O functions, knowing when the I/O occurs can sometimes be useful. This is demonstrated with the ATHENA-C program, which does not use MPI I/O.

As shown in Fig. 5, for most programs and cases analyzed, more time is spent in writing than opening, reading or closing files. One exception is the ATHENA-C program where the time spent in open is much larger than the time spent in write. From both MPIProf and IOT, it was found that all 2048 ranks in this program do I/O. Furthermore, IOT reveals that each rank opens, writes and closes 48 private .rst files and 48 private .vtk files. In addition to the .rst and .vtk files, rank 0 also opens, writes and closes 48 .hst files. To understand why the open operations are so costly, it helps to examine when the open operations occur during the run. Fig. 8 shows the occurrence of the opens during the run from 3 representative ranks - ranks 0, 128 and 256. It is clear from this figure that the 48 sets of opening 2 private files (or 3 files from rank 0) happen at about the same time among all the ranks. Having 2048 ranks each sending 2 to 3 open calls to the Lustre server at the same time creates contention for Lustre, thus the costly time spent in opens.

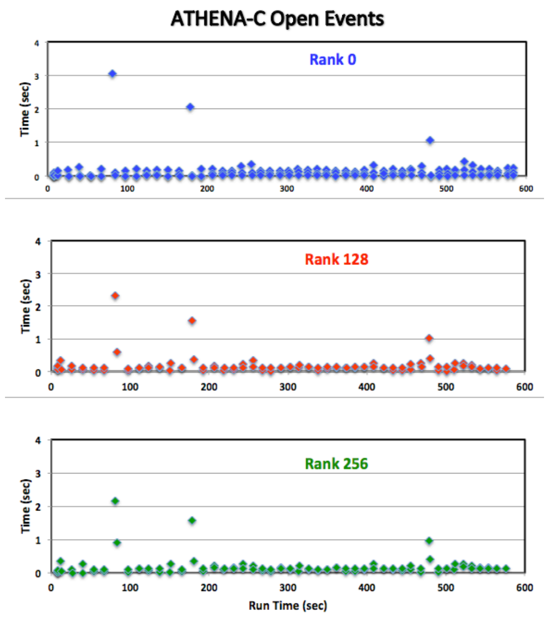

Fig. 8. ATHENA-C open events by 3 representative ranks

Programs that create a heavy load on Lustre, as exhibited in the ATHENA-C case, have the potential of not only hurting their own I/O performances, but also those of other users' jobs. Even more detrimental is the possibility of slowing down Lustre to the extent of making it inaccessible to all users. The NAS system administrators regularly monitor the Lustre filesystems and identify programs that cause heavy load on the systems. When necessary, the offending jobs are killed to bring Lustre back to life. Killing jobs sends a strong signal to users but does not help them figure out how to pinpoint the cause in their codes. The methodology with MPIProf and IOT provides a mechanism that users can employ to understand the I/O of their codes in order to find ways to eliminate problematic I/O behaviors.

VI. Conclusion

The combined use of MPIProf and IOT enabled an efficient and systematic analysis of many real programs with number of MPI ranks up to 5968. Of the 13 programs analyzed, the I/O landscape shows:

- 4 programs use MPI I/O and 9 do not.
- Except for ATHENA-C, I/O is dominated by rank 0 for all programs that do not use MPI I/O.
- I/O patterns for programs with MPI I/O vary depending on the specific MPI I/O calls used and the choice of MPI library.
- I/O is mostly dominated by write operations for most programs.
- All programs do mostly sequential I/O.

The prevalent use of single rank I/O in many existing programs implies that most users have not explored parallel I/O. Converting these programs from serial to parallel I/O will take significant education and effort. Supporting serial I/O is inevitable for the near term and options are to be investigated.

Since the SGI MPT library is the recommended MPI library for jobs running on Pleiades, the lessons learned from this study about the collective writes handled by MPT emphasize the need for users to understand their program's MPI I/O and find the optimum stripe counts for specific cases. This is demonstrated in the I/O characterization of ATHENA-C++ and the speedup obtained for this program. In some situations, it may require setting different stripe count for different files for better overall I/O performance. One of the recently introduced capabilities of IOT is a mechanism where different stripe counts and stripe sizes can be easily set for different files.

Most programs spend more time writing than opening, reading or closing files. Investing in new hardware or tuning Lustre system parameters for faster writes is a worthwhile effort.

The ATHENA-C program with concurrent opens from all ranks is one example where contention is created in Lustre servers. Using the methodology described in this study will help users identify such behavior in order to find a proper remedy which in turn will improve the stability of the filesystems.

Only 4 I/O questions are addressed in this paper for the 13 programs studied. Other questions such as the distribution of I/O sizes, correlation of I/O performance with system activities, why a program performs better on one filesystem vs another, etc., can be addressed through other features of MPIProf and IOT.

Acknowledgment

Communication with SGI MPT lead engineer, Michael Raymond, regarding MPI I/O implementation in MPT is greatly appreciated. Yanfei Jiang, Jahed Djomehri, Samson Cheung, Doug Westra, Robert Stein, Dennis Jespersen, Irina Kitiashvili and Alan Wray are acknowledged for making their programs and test cases available. We thank Johnny Chang and Dale Talcott for reviews, corrections, and comments on the manuscript.

References

[1] Top500 List - June 2016, https://www.top500.org/list/2016/06/

[2] P. J. Braam and P. Schwan, "Lustre: the intergalactic file system," in Proceedings of Ottawa Linux Symposium, 2002.

[3] S. Saini, J. Rappleye, J. Chang, D. Barker, P. Mehrotra, and R. Biswas, "I/O performance characterization of Lustre and NASA applications on Pleiades," 19th International Conference on High Performance Computing (HiPC), 2012.

[4] H. Jin, "Using MPIProf for performance analysis," http://www.nas.nasa.gov/hecc/support/kb/using-mpiprof-for-performance-analysis_525.html

[5] IOT FAQ, http://iodoctors.com/faq.html

[6] S. Shende and A.D. Malony, "The TAU parallel performance system," International Journal of High Performance Computing Applications, Vol. 20-2, pp. 287-331, 2006.

[7] The Krell Institute, Open|SpeedShop, https://openspeedshop.org/

[8] D. Thomas, J.-P. Panziera, and J. Baron, "MPInside: a performance analysis and diagnostic tool for MPI applications," WOSP/SIPEQ '10, Proceedings of the First Joint WISP/SIPEW International Conference on Performance Engineering, pp. 79-86, 2010.

[9] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale I/O workloads," in 2009 IEEE International Conference on Cluster Computing and Workshops, pp. 1-10. IEEE, 2009.

[10] W. Dong, G. Liu, J. Yu, and Y. Zuo, "Characterizing I/O workloads of HPC applications through online analysis," in 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC), pp. 1-2. IEEE, 2015.

[11] A. Uselton and N Wright, "A file system utilization metric for I/O characterization," in Proc. of the Cray User Group conference, 2013.

[12] P. C. Roth, "Characterizing the I/O behavior of scientific applications on the Cray XT," in Proc. of the 2nd International Petascale Data Storage Workshop (PDSW'07), November 11, 2007, Reno, NV.

[13] F. Pan, Y. Yue, J. Xiong, and D. Hao, "I/O characterization of big data workloads in data centers," in Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware, pp. 85-97. Springer International Publishing, 2014.

[14] Intel Developer Zone document, "Getting started with the MPI performance snapshot," https://software.intel.com/en-us/articles/getting-started-with-the-mpi-performance-snapshot.

[15] NASA High-End Computing Capability Project, Pleiades Supercomputer, http://www.nas.nasa.gov/hecc/resources/pleiades.html

[16] Standard Billing Units, http://www.hec.nasa.gov/user/policies/sbus.html

[17] Parallel I/O Benchmarking Consortium, http://www.mcs.anl.gov/research/projects/pio-benchmark/