ANL/MCS-TM-331

Performance Analysis of Darshan 2.2.3 on the Cray XE6 Platform

Mathematics and Computer Science Division

About Argonne National Laboratory

Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

Availability of This Report

This report is available, at no cost, at http://www.osti.gov/bridge. It is also available on paper to the U.S. Department of Energy and its contractors, for a processing fee, from:

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
phone (865) 576-8401
fax (865) 576-5728
reports@adonis.osti.gov

Disclaimer

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

ANL/MCS-TM-331

Performance Analysis of Darshan 2.2.3 on the Cray XE6 Platform

by
P. Carns, K. Harms, R. Latham, and R. Ross
Mathematics and Computer Science Division, Argonne National Laboratory

October 31, 2012

Performance Analysis of Darshan 2.2.3 on the Cray XE6 Platform

Philip Carns, Kevin Harms, Robert Latham, and Robert Ross
Argonne National Laboratory
October 2012

***Abstract*—Darshan is a production-quality I/O characterization tool that captures and summarizes the I/O behavior of parallel applications. It records a variety of information with minimal overhead, including access patterns, number of files accessed, and the amount of time consumed by I/O routines. Darshan's lightweight design makes it suitable for full-time deployment for workload characterization of large HPC systems.**

Although Darshan was designed for portability, Darshan 2.2.3 is the first release to feature fully integrated support for the Cray XE6 platform, including support for PGI, Cray, Intel, and GNU compilers as well as both static and dynamic linking. This document presents a brief study of Darshan performance and runtime overhead on the Beagle Cray XE6 system operated by the Computation Institute and the Biological Sciences Division of the University of Chicago and Argonne National Laboratory. We find that Darshan introduces negligible end-to-end overhead for I/O-intensive applications and can store characterization data for over 12 million unique files in less than 5 seconds.

I. Background

Previously published studies outline the design of Darshan [1] and the types of analysis that it enables [2]. One of Darshan's key design goals is to be as efficient and unobtrusive as possible so that it does not interfere with application behavior. This allows Darshan to transparently instrument the behavior of all production activity on large-scale HPC systems without disturbing user productivity.

Previous studies have demonstrated the efficiency of Darshan on the IBM Blue Gene/P platform. This report seeks to validate that Darshan exhibits the same efficiency on the Cray XE6 platform.

II. Test environment

The experiments in this section were performed on Beagle, a Cray XE6 computer operated by the University of Chicago Computation Institute. Beagle contains 17,856 processor cores distributed across 744 compute nodes. All Darshan log files and application data were stored on Beagle's Lustre file system, which is built atop a DDN SFA 10000 storage array.

All experiments were performed with Darshan version 2.2.3-pre1. The Darshan library itself was built by using GNU GCC compilers. All test applications were built by using the default PGI compilers. The Darshan library is compatible with PGI, GCC, Intel, and Cray compilers; but we do not expect the choice of application compiler to play a significant role in I/O performance or the performance of Darshan itself.

III. Evaluation

In this section we describe experiments on a large-scale I/O-intensive benchmark and on low-level benchmarks.

A. Darshan impact on overall application run time

We begin by evaluating the impact of Darshan on the overall run time of a large-scale, I/O-intensive benchmark application. We used version 2.10.3 of the IOR benchmark for this purpose. IOR is a synthetic benchmark developed by Lawrence Livermore National Laboratory that can be configured to emulate a variety of workloads [3]. In this case, we configured IOR to use the MPI-IO API to write and read 128 MiB per process from 3,072 processes for an aggregate of 384 GiB of data. Each process wrote to a unique file using an access size of 128 KiB. The data was then read back by an adjacent process (rather than the process that wrote the data) in order to mitigate client-side read caching to some extent. The IOR configuration file is shown below.

```
IOR START
 api=MPIIO
 repetitions=1
 verbose=1
 blockSize=128M
 fsync=0
 writeFile=1
 readFile=1
 keepFile=0
 transferSize=128K
 filePerProc=1
 reordertasksconstant=1
 testFile = /lustre/beagle/carns/data/ior.dat
 RUN
IOR STOP
```

This IOR configuration requires Darshan to capture information from 3,072 processes, 3,072 unique files, and over 6 million I/O operations at two distinct API instrumentation levels: POSIX and MPI-IO. Because of the expected variability in I/O performance at this scale [2], we cannot reliably measure Darshan overhead based on just two job samples. We instead submitted 40 independent jobs: 20 compiled with Darshan support, and 20 compiled without Darshan support. We used scheduler dependencies to prevent concurrent execution and ensure that the samples alternated fairly between each of the two configurations. The execution time for each job was measured as the elapsed time taken for the `aprun` command to complete. We chose this measurement, rather than the

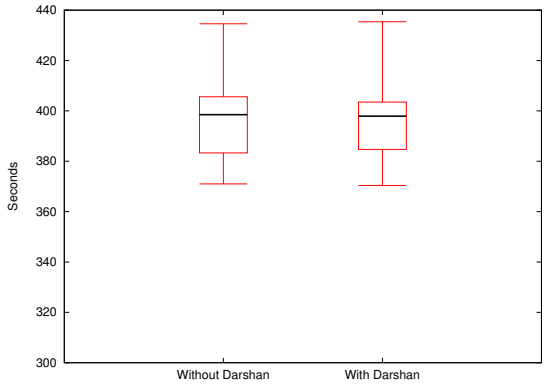


Fig. 1. Box plot of IOR execution time on Beagle (20 samples with Darshan, 20 samples without Darshan)

run time reported by the scheduler, in order to eliminate any potential variance introduced by the scheduler or job environment itself.

Figure 1 shows a box plot of the samples obtained with this methodology. The plot indicates the min, max, median, first quartile, and third quartile for each sample population. The two scenarios produced similar variability. The median execution time with Darshan was 397.9 seconds, while without Darshan it was 398.5 seconds. A typical job achieved approximately 1.6 GiB/s of write performance and 2.5 GiB/s of read performance.

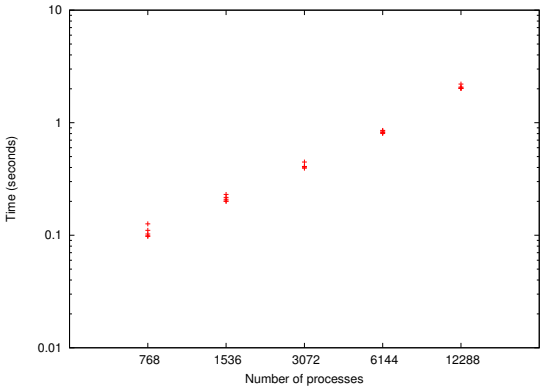
We used a t test, as computed using R [4], to check for the presence of a statistically significant difference in execution time between the two independent sample sets. The results of that analysis are presented in Table I. With a p value of 0.9463, and a 95% confidence interval for the difference in means of -10.09511 to 10.79374, there is no statistical evidence for a difference in the average run time between the IOR runs that used Darshan and the IOR runs that did not.

B. Darshan data aggregation and storage costs

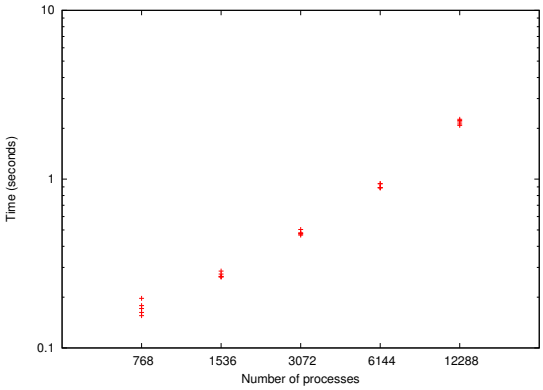
We can also use low-level benchmarks to isolate the runtime overhead of individual Darshan components. There are two potential sources of runtime overhead in Darshan. The first is the cost of the wrapper functions that intercept and instrument I/O function calls. In previous work we have found that this wrapper overhead is minimal [1] relative to the cost of an I/O operation, even on systems with relatively low CPU performance per core. The wrappers themselves perform no communication or I/O activity. Data is collected independently

TABLE I
TWO-SIDED INDEPENDENT SAMPLE T-TEST SUMMARY FOR THE DIFFERENCE IN MEANS BETWEEN THE SAMPLES WITH AND WITHOUT DARSHAN INSTRUMENTATION

t	0.0678
degrees of freedom	35.989
p value	0.9463
95% confidence interval	-10.09511 to 10.79374



(a) 1 shared file



(b) 1024 shared files

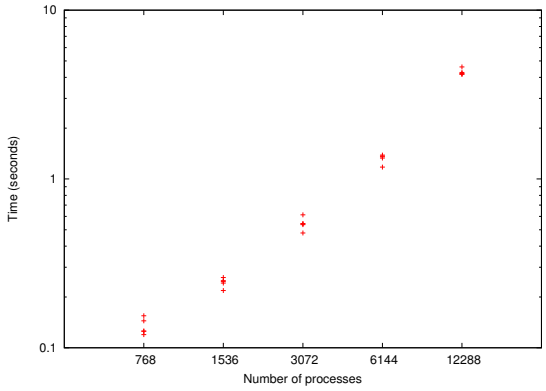
Fig. 2. Elapsed time consumed by the Darshan shutdown procedure when instrumenting shared file access

at each process by using efficient data structures and a bounded amount of memory.

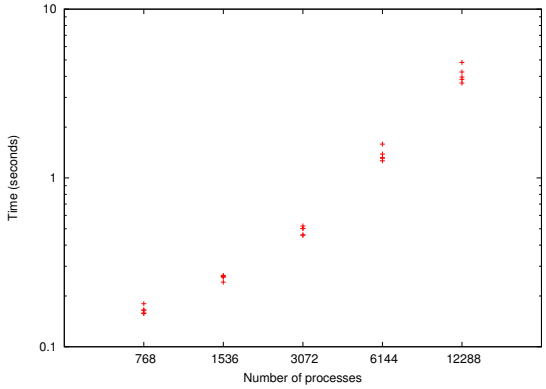
The second potential source of overhead is the cost of aggregating data across processes and writing the data to persistent storage when the application shuts down. This procedure is invoked by Darshan at `MPI_Finalize()` time. Darshan performs a sequence of steps at this point to prune the amount of data that it has collected, compress remaining data, and write the compressed data to a single log file. This procedure is generally the most significant source of measurable Darshan overhead. It is also the mechanism in Darshan that is most likely to perform differently across platforms because of differences in I/O capabilities and collective network algorithms.

In this section we measure the time consumed by the Darshan shutdown procedure using a synthetic benchmark. The synthetic benchmark does not read or write any application data. Instead, it inserts artificial counters into the Darshan library to mimic various I/O workloads. The Darshan shutdown routine is invoked after each I/O workload scenario and instrumented to report the elapsed time.

Figure 2 shows the amount of time consumed by the Darshan shutdown routine after instrumenting a shared-file access pattern in which each MPI process opened the same file. In this case, Darshan uses collective MPI routines to



(a) 1 unique file per process



(b) 1024 unique files per process

Fig. 3. Elapsed time consumed by the Darshan shutdown procedure when instrumenting unique files on each process.

combine the characterization records from each process into a single, unified record before storing the results. Figures 2(a) and 2(b) illustrate the Darshan performance in this scenario when recording access to a single shared file or 1,024 distinct shared files, respectively. Five data points were collected at each scale, ranging from 768 to 12,288 MPI processes. All log files were written to the /lustre/beagle file system. The worst case example added at most 2.2 seconds to the MPI_Finalize() execution time. This is a fixed cost regardless of the amount of data accessed by the application.

Figure 3 repeats the same experiments but with each process opening unique files rather than shared files. In Figure 3(a) there is one unique file per process, while in Figure 3(b) there are 1,024 unique files per process. In these scenarios, Darshan is unable to combine characterization records, but the shutdown procedure still consumes no more than 4.6 seconds in the worst case. Note that the worst-case example emulates an application that opens over 12 million unique files in a single job run. According to internal Darshan instrumentation, the biggest factor in the Darshan shutdown time is the amount of time required to write the compressed log files to disk.

IV. Conclusions

Our experiments conducted on the Beagle Cray XE6 show that Darshan introduces negligible overhead in the overall run time of an I/O-intensive benchmark that reads and writes 384 GiB of data using 3,072 MPI processes. In-depth instrumentation of the Darshan library itself further finds that as many as 12 million files can be accessed by 12,288 MPI processes while adding a fixed overhead of less than 4.6 seconds to the time needed to shut down an application. These results indicate that Darshan is unlikely to interfere with production activity on the Cray XE6 platform.

Acknowledgments

This research was supported in part by NIH through resources provided by the Computation Institute and the Biological Sciences Division of the University of Chicago and Argonne National Laboratory, under grant S10 RR029030-01. We specifically acknowledge the assistance of Lorenzo Pesce.

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[1] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale I/O workloads," in *Proceedings of 2009 Workshop on Interfaces and Architectures for Scientific Data Storage*, September 2009.

[2] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, p. 8, 2011.

[3] H. Shan, K. Antypas, and J. Shalf, "Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark," in *Proceedings of Supercomputing*, November 2008.

[4] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2012, ISBN 3-900051-07-0. [Online]. Available: http://www.R-project.org

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439-4847

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC