Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

I/O-profiling with Darshan

Bjørn Lindi*

Norwegian University of Science and Technology (NTNU), Trondheim, NO-7491, Norway

Abstract

Darshan is a set of libraries that can characterize MPI-IO and POSIX file access within typical HPC applications in a non-intrusive way. It can be used to investigate I/O behavior of a MPI-program. An application's I/O behavior can easily be an obstacle to achieving petascale performance. Hence, to be able to characterize the I/O of a HPC-application is an important step on the path to develop scaling properties. Darshan have been used on selected applications from task 7.1 and 7.2. This whitepaper describes the work carried out and the results achieved.

Introduction

It is critical for large scale simulations to leverage parallelism in all phases. With increasingly challenging computations, the requirement for parallelism in the I/O-phase has often lead to one file per process or MPI-task. On a petascale system this simple scheme could lead to the unpleasant experience of managing hundred thousands of files and possibly deplete the file system resources used for metadata handling. On the other hand, having I/O-operations spread across to few processes will lead to poor I/O-performance and degrade the application scalability. Hence, enabling sustained I/O performance at the application level, where I/O-parallelism is implemented above the global file system, is a necessity for application scalability.

Application owners often have selected a very simple I/O-strategy, or even worse, the application owner might not have any notion on how I/O is performed at all. A group of computer researcher at Argonne National Laboratory, US have addressed the issue of I/O-characterization by developing a tool call Darshan ("sight" or "vision" in Sanskrit).

The design of Darshan

Darshan is a light weight profiling tool that can be used to characterize I/O-load at petascale. It gives an accurate picture of the I/O-access pattern, how read-, write- and metadata operations are performed by the application. Further, it characterizes the access pattern within file and files, describing whether the access is MPI-based or POSIX. All this is done with a minimum of overhead.

Darshan is designed to reflect application I/O behavior while being transparent to users. Since it has the ambition to be a petascale characterization tool, it also must have strong scaling properties. It is implemented as set of user space liberaries. These are linked into the application during the linking phase. The applications I/O-calls are substituted with calls to the darshan libraries. No application source code modification is necessary.

During execution of the application, Darshan collects statistics which is stored in a file record per process. As the application ends its computation by a call to MPI_Finalize(), a shutdown routine is executed to gather all the darshan file records. The overhead introduce is negligible as profile data produced is small on smaller

* Corresponding author. *E-mail address*: bjorn.lindi@ntnu.no

computations. On larger computations (petascale), the time used for handling darshan profile data is small compared to the overall shutdown time of the computation. Accordingly, Darshan does not influence the computation. The profile achieved will be representative for the application’s I/O behavior.

Installing Darshan

Darshan can be downloaded from Argonne National Laboratory, US as a gzipped tar archive. To unpack the package do:

```
$ tar zxf darshan-2.1.2.tar.gz
```

To build and install the libraries, do configure, make and make install:

```
$ cd darshan-2.1.2
$ ./configure --with-mem-align=16 --with-log-path=$WORK/darshan/log --
prefix=$HOME/local/darshan --with-jobid-env=PBS_JOBID
$ make
$ make install
$ $HOME/local/darshan/bin/darshan-mk-log-dirs.pl
```

The –log-path argument points to the directory where Darshan writes the profile data files. The –with-jobid-env argument states which environment variable which express the job identifier assigned to each job by the batch system. After ‘make install’ the darshan libraries and scripts are installed in the directory pointed to by the –prefix argument. The command ‘darshan-mk-log-dirs.pl’ creates the log file directories.

Linking darshan to an application

In the directory ‘bin’ in the Darshan installation directory there is three scripts that generates wrappers for the C/C++ and Fortran compilers. Here is how these scripts are called for creating wrappers for the C and Fortran compilers:

```
$ export PREFIX=$HOME/local/darshan
$ cd $PREFIX/bin
$./darshan-gen-cc.pl  `which mpicc` --output $PREFIX/bin/mpicc
$./darshan-gen-fortran.pl  `which mpif90` --output $PREFIX/bin/mpif90
```

The scripts install a wrapper for mpicc and mpif90 in the PREFIX/bin directory. These scripts wraps all call I/O-calls (POSIX, MPI-IO and HDF5) to special calls in the Darshan libraries. An inspection of the wrapper-scripts will show that during the linking phase calls like open64 get substituted by __wrap_open64() in the Darshan POSIX library. All other I/O-calls are wrapped in a similar way.

The wrappers $PREFIX/bin/mpicc or $PREFIX/bin/mpif90 must be used to build the actual application by being called from the application’s makefile(s), substituting the ordinary mpicc or mpif90 commands.

Creating the Darshan summary report

For each execution of an application which is linked with the Darshan library, the shutdown routine of Darshan will create a logfile in the logfile directory. The logfile may be decompressed and processed manually. The Darshan package also provides a tool for creating a summary of the profiling.

The example below show different log-files after several executions of an application enabled with Darshan. The perl-script ‘darshan-job-summary.pl’ creates a PDF-report named ‘nonc48_io.pdf’ of one log file . Gnuplot and pdflatex must be available in the execution path for the script to produce a PDF-report.

```
$ pwd
/darshan/log/2011/10/22
$ls  -l
total 192
-r-------- 1 bjorn ts 166 Oct 22 11:32 id128914_10-22-41559-3836569230660266244_1.darshan.gz
-r-------- 1 bjorn ts 253 Oct 22 11:36 id128915_10-22-41751-3836569230660266244_1.darshan.gz
-r-------- 1 bjorn ts 258 Oct 22 11:41 id128916_10-22-42088-3836569230660266244_1.darshan.gz
-r-------- 1 bjorn ts 256 Oct 22 11:48 id128917_10-22-42511-3836569230660266244_1.darshan.gz
-r-------- 1 bjorn ts 276 Oct 22 17:33 id128924_10-22-63223-1546178374410473002 7_1.darshan.gz
```

```
-r-------- 1 bjorn ts 273 Oct 22 17:38 id128925_10-22-63490-15461783744104730027_1.darshan.gz
$ export PREFIX=$HOME/local/darshan
$ $PREFIX/bin/darshan-job-summary.pl id128916_10-22-42088-
3836569230660266244_1.darshan.gz --output ~/tmp/nonc48_io.pdf
```

Profiling OpenFOAM

We selected Open Source Field Operation and Manipulation (OpenFOAM) as the community code for the CFD and engineering community. OpenFOAM is a popular Computational Fluid Dynamic (CFD) library used to build CFD applications and data tools. It is used to study different problems in continuum mechanics like combustion as well as flow in different types of media.

As one user case with OpenFOAM from ICHECH showed poor scalability in terms of longer execution time with an increasing number of processes, Darshan was used to better understand the application’s I/O-behaviour. Five tests with 64 to 1024 processes based on the same problem case where executed on Curie with Darshan enabled. Strong scaling or indications thereof should be expected as the number of computing processes is doubled for each test.

As can be seen from the table 1 the compute time increases with increasing number of processes with almost a doubling for the last test case with 1024 processes. Metadata handling not only takes a large share of overall compute time, but the time used on metadata handling grows disproportionately compared to the time used on compute.

Table 1 Compute time and time for metadata handling for different number of processes for OpenFOAM with the ICHEC case

Number of Processes	**64**	**128**	**256**	**512**	**1024**
Compute time [s]	686	801	890	1161	2248
Cumulative metadata [s]	64	202	274	389	892
The share of Meta data handling of the overall compute time	9.3%	25%	31%	34%	39%

The increase in time spent on metadata handling is partly due to an increasing volume of small files. The number of files created and read doubles with each doubling of number of processes. The average file size is at same time approximately halved.

Table 2. The number of files create, read for OpenFOAM with the ICHEC case

Number of processes	**64**	**128**	**256**	**512**	**1024**
Number of files created	512	1024	2048	4096	8192
Number of files read	1089	2177	4353	9729	17409
Average file size	597K	317 K	163K	84K	47K
Number of stat() calls	500 000	1000 000	2000 000	4400 000	8500 000

Leading to even further increased metadata traffic is the large volume of stat()-calls. Inspection of the code shows that OpenFOAM uses files as a way of communication between the processes. MPI is used for process creation, but data is scattered and gathered through updates of files created by each process. OpenFOAM uses an instance of an object named FileMonitor to issue stat()-calls which check the timestamp of the different files during the path of simulation. A newer timestamp indicates that the parameter/result has been updated. Increasing the number of processes effectively kills scaling since this increases the number of files to stat(). Accordingly, metadata handling materializes as the bottle neck.

OpenFOAM’s I/O characteristics are contradictory to features needed to achieve good I/O-performance during scaling. Good I/O-performance is achieved by utilizing the service a global parallel file system provides which are reading and writing parallel streams of data in large chunks. The volume of metadata handling must be low compared to the volume in data provided to the computing process. Typical chunk or block sizes are in the range of 1 MB – 4 MB. OpenFOAM on the contrary creates files which average size are 10-20% of the block size of the file system (the 1024 processes case). Furthermore, each process creates eight files. Accordingly the number of files is proportional to the number of processes, leading to substantial metadata traffic.

A way to reduce the number of file operations was investigated. OpenFOAM has different parameters controlling the computation. The parameter 'RunTimeModifable' toggles the reading of the different dictionaries. With 'RunTimeModifable: True' the dictionaries are read for each time step.

Tests with 64 to 512 processes where carried out with 'RunTimeModifieable' set to 'No'. For the tests with the higher core counts, from 128 processes and above, this cuts the compute time by more than 50%. As can be seen from table 3 a better scaling was also achieved for the first doubling of the number of processes.

Table 3 Compute time and time for metadata handling for different number of processes with RunTimeModifiable set to 'No'

Number of Processes	**64**	**128**	**256**	**512**
Compute time [s]	542	381	343	411
Cumulative metadata [s]	0.99	2.62	6.03	14.4
The share of Meta data handling of the overall compute time	0.2%	0.7%	1.8%	3.5%

As the number of file increases with the number of processes, metadata handling starts to influence the computation. Increasing the number of processes from 256 to 512 gives a negative scaling as the compute time of the latter is larger.

Table 4. The number of files created and read with RunTimeModifiable set to 'No'.

Number of processes	**64**	**128**	**256**	**512**
Number of files created	512	1024	2048	4096
Number of files read	1089	2177	4353	9729
Average file size	597K	317 K	163K	84K
Number of stat() calls	5000	10 000	20 000	44 000

Toggling the parameter off also reduced the number of stat()-calls by a factor 100, which also corresponds to the number of timesteps, as the computations are carried out from T_o=0 to T_1=1x10^{-6} with a timestep of 1x10^{-8}. The number of files created remained the same.

Conclusion

The profiling of OpenFOAM with Darshan clearly indicates that the scaling properties are constrained by a poor I/O-design. Currently, OpenFOAM uses a IOStream object to read and write updated parameters. IOstream is used throughout the application code of more than a million lines of C++-code. A new scheme of communication and I/O are under such conditions not easily introduced. Though, it is worth investigating whether a I/O-library like parallel-NetCDF can be used by OpenFOAM. By encapsulating parallel-NetCDF and overload the IOStream operator, new way of both communication and I/O could be implemented.

The Darshan library has proven valuable for the work with OpenFOAM. It has given insight to the application behavior, indicating a bottleneck often not taken into account as one strive for better scaling properties. Using Darshan on applications early in the PRACE selection and evaluation process will prove valuable both for application owners as well as for the work groups providing the PRACE support service.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557. The work is achieved using the PRACE Research Infrastructure resources CURIE.

References

1. Philip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, "24/7 Characterization of Petascale I/O Workloads"
2. Darshan homepage http://www.mcs.anl.gov/research/projects/darshan.