Darshan Hands-on Tutorial

Building Darshan

Darshan is an open-source tool and can be downloaded from:

ftp://ftp.mcs.anl.gov/pub/darshan/releases/darshan-3.1.5.tar.gz

To build Darshan from source, you will need to use the same MPI implementation used to build your application code. In addition, Darshan requires LaTeX for creating a report.

1. `tar -zxvf darshan-3.1.5.tar.gz`
2. `cd darshan-3.1.5/darshan-runtime`
3. `./configure --with-mem-align=8 --with-log-path-by-env=DARSHAN_LOGPATH --with-jobid-env=NONE CC=mpicc --prefix=/usr/local/darshan-3.1.5-ompi`
   The `--prefix` flag is where you would like to install the Darshan tool;
4. `make`
5. `make install`
   Then build the Darshan command line utilities:
6. `cd darshan-3.1.5/darshan-util`
7. `./configure --prefix=/usr/local/darshan-3.1.5-ompi`
8. The above prefix directory should be the same used in command 3 above;
9. `make`
10. `make install`

Profiling an application with Darshan

The easiest way to profile an application with Darshan is to build a dynamic executable, namely an executable that is dynamically linked with the MPI library. To determine if your executable is dynamic or not, type:

```
ldd bin/bt.A.4
[ ... ]
libmpi.so.1 => /usr/lib64/openmpi/lib/libmpi.so.1
[ ... ]
```

If the output shows the MPI library, then it is dynamically linked. To profile your code, set the Darshan log path variable and simply prefix the MPI execution command with the Darshan library as shown:

```
export DARSHAN_LOGPATH=.
LD_PRELOAD=/usr/local/darshan-3.1.5-ompi/lib/libdarshan.so mpirun -n 9 \
../bt.A.9.mpi_io_full
```

The above command will create a trace file in the current working directory. The trace file name should start with the user name and end with the text “darshan”.

Darshan Profiling Report

To create a PDF report of the profile, type:

```
darshan-job-summary.pl <trace file>.darshan
```

The PDF report will have the same name as the trace file but with the “pdf” extension. Use any PDF viewer to view the report.

Obtaining and Installing NPB

The NAS parallel benchmark can be used to learn more about the Darshan tool. The benchmark can be downloaded from:

https://www.nas.nasa.gov/assets/npb/NPB3.3.1.tar.gz

Then follow these instructions to build the I/O benchmark:

1. `tar -zxvf NPB3.3.1.tar.gz`
2. `cd NPB3.3.1/NPB3.3-MPI`
3. `cp config/make.def.template config/make.def`
4. Set the following make variables in `config/make.def`:
    a. `MPIF77 = mpif90 # or the MPI Fortran wrapper`
    b. `FFLAGS = -O2 -g`
5. `make bt NPROCS=9 CLASS=A SUBTYPE=full`
6. This will create the executable in `bin/bt.A.9.mpi_io_full`

Profiling NPB with Darshan

To profile the NPB benchmark, type the following command:

```
export DARSHAN_LOGPATH=.
LD_PRELOAD=/usr/local/darshan-3.1.5-ompi/lib/libdarshan.so mpirun -n 9 \
bin/bt.A.9.mpi_io_full
```

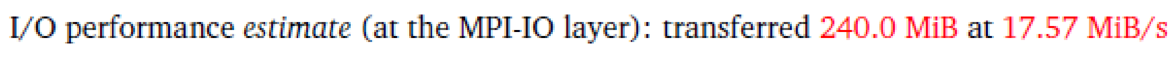
The above command will create a trace file in the current working directory. The trace file name should start with the user name and end with the text “darshan”.

To create a PDF report of the profile, type:

```
darshan-job-summary.pl <trace file>.darshan
```

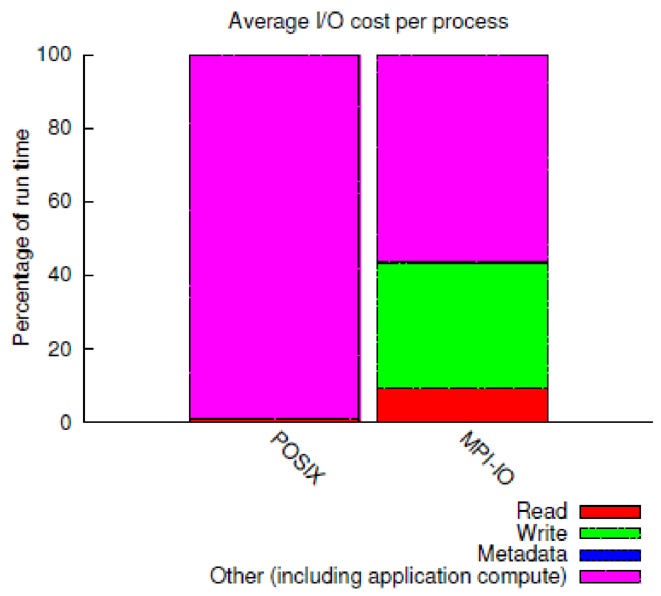
Interpreting the Darshan Report

The Darshan report contains a number of sections each outlining the I/O characteristics of an application run. At the very top of the report, it shows how much data was transferred and what the bandwidth was. The bandwidth data could be used to determine the I/O scalability of the application at different MPI process counts to determine if I/O is scaling linearly. An example output of this section is shown below:



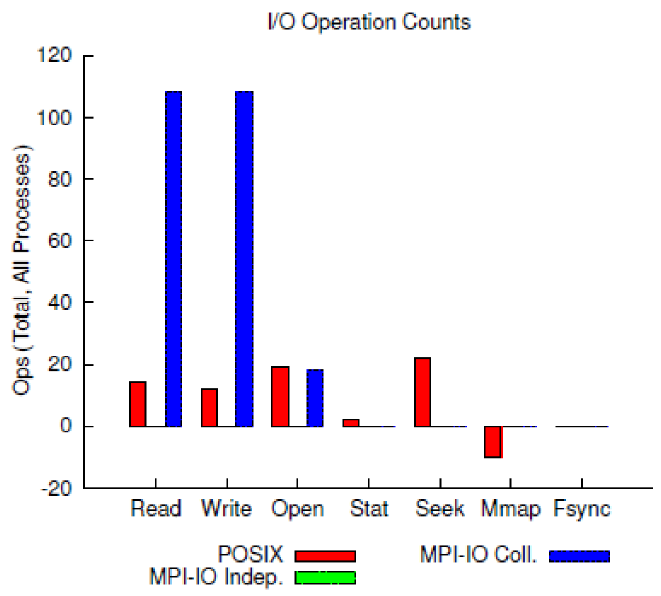
I/O performance *estimate* (at the MPI-IO layer): transferred 240.0 MiB at 17.57 MiB/s

The bar chart below shows, as a percentage of runtime, how much I/O contributed to the runtime.



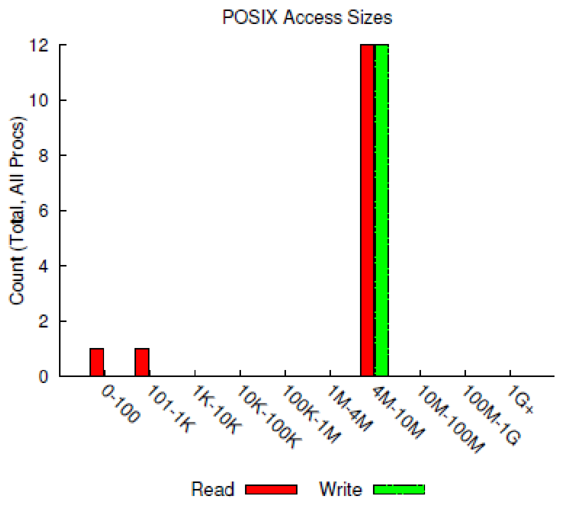
From this chart, one can determine if I/O is a performance issue or not. In this example, over 40% of the application runtime is being spent in I/O, so the I/O of this code can be addressed. It also shows the contribution to the runtime in read, write, metadata operations as well how much time in everything else, namely “other”. The “other” category includes time in computation and communication. Note that parallel NetCDF and parallel HDF5 are labelled as MPI-IO.

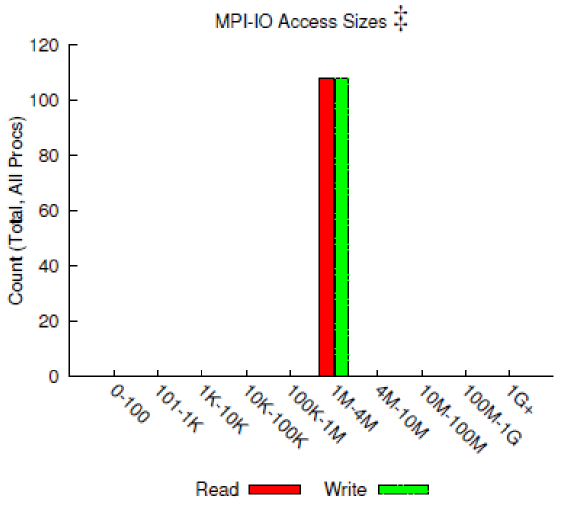
The next graph shows the I/O operation counts. Ideally, the operation counts should be small as possible as a large number could indicate a performance issue.



The operation counts are summed across all MPI processes. In this example, there are a large number of read/write operations. However, they are MPI-IO collective operations which are more optimal than independent MPI-IO operations. A large number of seek operations could also indicate performance issues as this operation is expensive.

The bar charts below show the read/write access sizes for both POSIX and MPI-IO. Small access sizes could indicate performance issues. Ideally, the number of I/O operations should be low with large access sizes. In this example, the access sizes are sufficiently large for both POSIX and MPI-IO.





The table on the below left gives a breakdown of the access sizes, and the table on the right shows the file count summary. Ideally, the file operation counts should be low, particularly the write-only files. This is because meta-data creation time is expensive.

Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	10485760	24
	253	1
MPI-IO ‡	1165080	144
	1164240	48
	1166800	24

‡ NOTE: MPI-IO accesses are given in terms of aggregate datatype size.

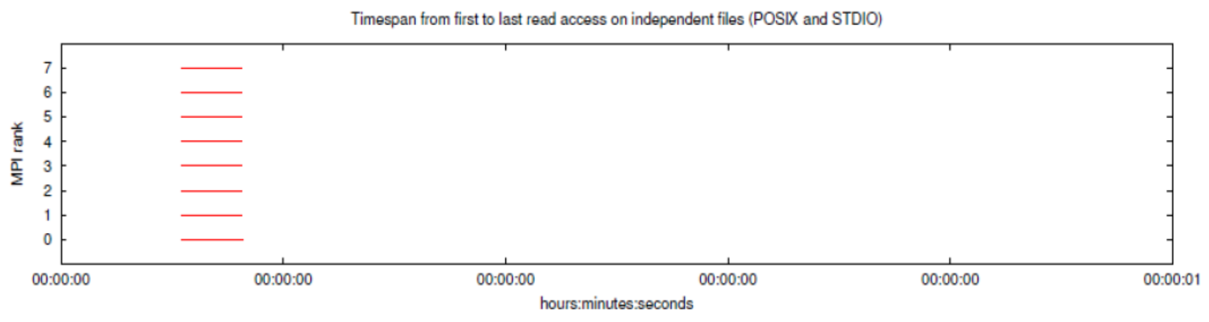
File Count Summary
(estimated by POSIX I/O access offsets)

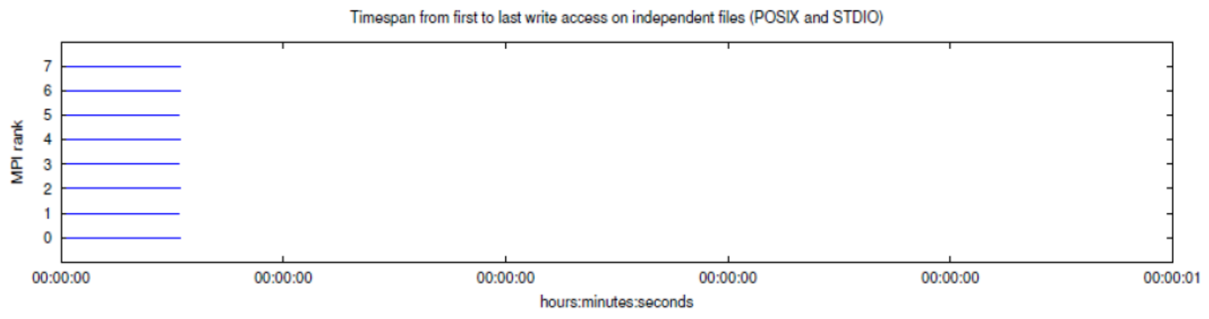
type	number of files	avg. size	max size
total opened	2	61M	120M
read-only files	1	253	253
write-only files	0	0	0
read/write files	1	120M	120M
created files	1	120M	120M

The next graph shows the I/O timeline which can be aggregated or shown individually for each MPI process. To obtain the I/O timeline for individual MPI processes, set the following environment variable prior to the application execution:

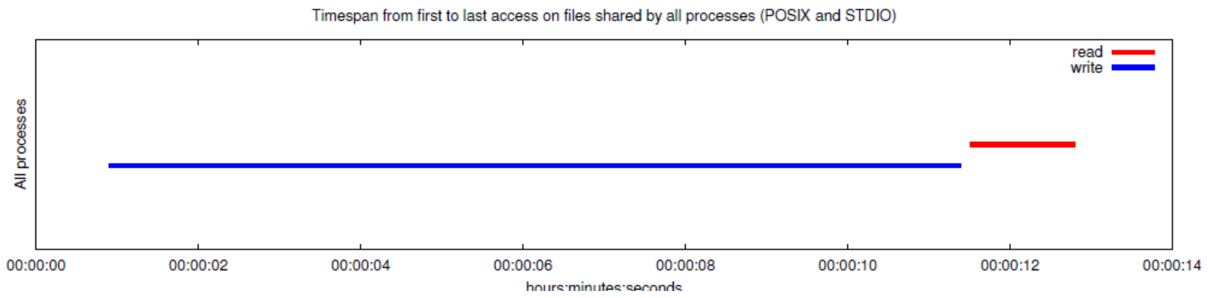
```
export DARSHAN_DISABLE_SHARED_REDUCTION=1
```

An example I/O timeline for each MPI process is shown below, and from this timeline, any load imbalances can be observed.





An example of an aggregate I/O timeline is shown below which is the default view for Darshan.



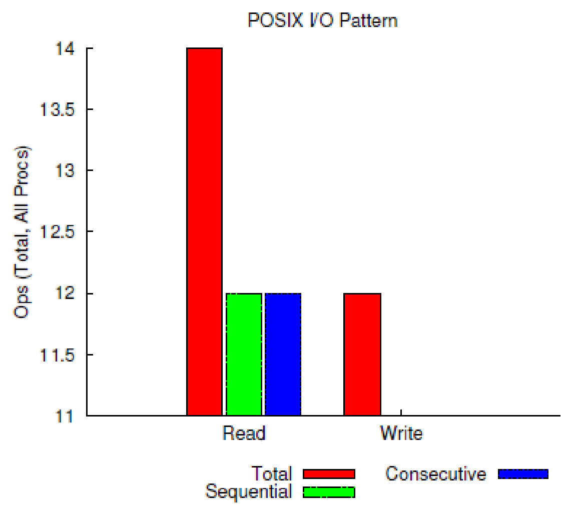
The next table shows which file systems were accessed during the application run.

Data Transfer Per Filesystem (POSIX and STDIO)

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/fserver	120.00000	1.00000	120.00024	1.00000

This is particularly useful to determine if the application run was using the correct file system, e.g. a high performance file system and not NFS which is not designed for parallel I/O.

The next graph shows the POSIX I/O pattern.



Consecutive I/O means contiguous I/O which is optimal. Sequential I/O means non-contiguous.

The last table shows the variance of POSIX I/O data across the MPI processes and can be used to determine if there are any load imbalances across the MPI processes.

Variance in Shared Files (POSIX and STDIO)

File Suffix	Processes	Fastest			Slowest			σ	
		Rank	Time	Bytes	Rank	Time	Bytes	Time	Bytes
...tio.full.out	9	8	0.001480	0	0	1.174813	240M	0.369	7.91e+07

In this example, this shows that process number 8 has transferred 0 bytes, whereas process number 0 has transferred 240 MB. There is clearly a load imbalance in this example and needs to be addressed.