Parallel File I/O Profiling Using Darshan

Wadud Miah, Numerical Algorithms Group (NAG)

EU H2020 Centre of Excellence (CoE)

1 October 2015 – 31 March 2018

Grant Agreement No 676553

Agenda

- Introduction to parallel storage/file systems;
- Parallel I/O models available for MPI codes;
- Darshan profiling tool, including a live demo;
- Darshan case studies;
- Programming hints and tips on how to write efficient I/O for MPI codes;
- Questions and answers.

Introduction to Parallel Storage/File Systems

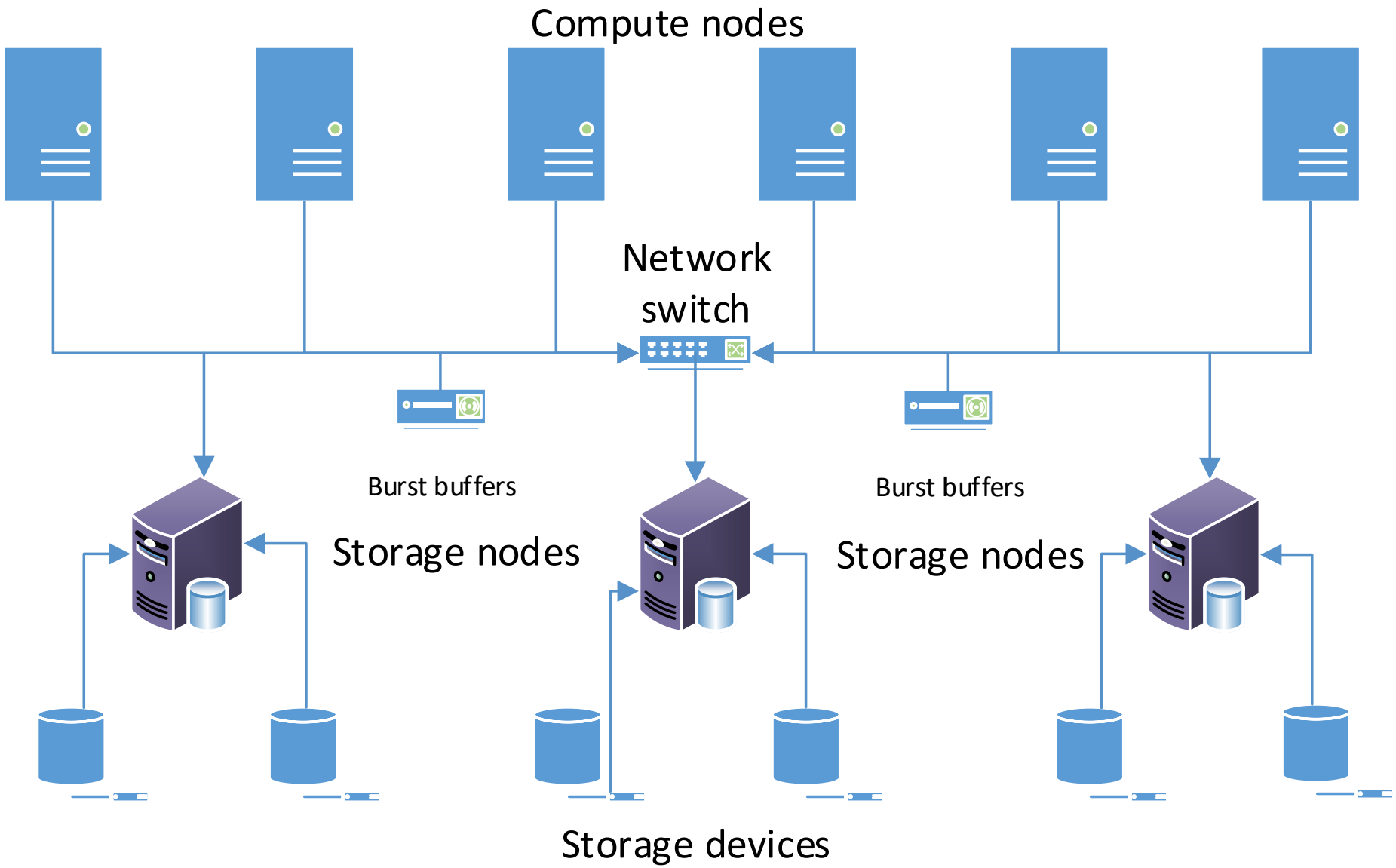
Parallel I/O Stack

- Parallel NetCDF and parallel HDF5;
- MPI-IO - Part of MPI 2.0 standard;
- CIO (IBM), DVS (Cray) - aggregates I/O;
- GPFS, Lustre, PanFS;
- Hard drives and RAID devices. Includes SSD, NVRAM and traditional spinning disks.

Data Intensive Science

- Science is also becoming increasingly data intensive, hence the importance of high performance I/O;
- File I/O is required for writing simulation data, check pointing, reading data for model validation. Example projects are SKA, particle physics (CERN), environmental sciences, genome sequencing;
- File I/O in computational science tends to be write-once and read-many.

Parallel Storage Architecture



Types of Storage Nodes

- There are two types of storage nodes: *meta-data* nodes and *data* nodes;
- Meta-data nodes store information such as file owner, access time - Linux inode data;
- Data nodes actually store the file data. There are more data nodes than meta-data nodes;
- Lustre and Panasas have dedicated meta-data nodes whereas GPFS strides the meta-data across storage nodes;
- *Parallel file systems are bandwidth bound*.

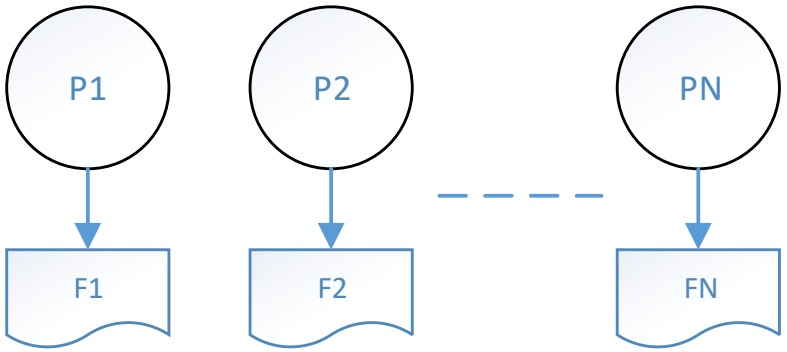
Parallel I/O Factors

- The number of MPI processes (N);
- The total amount of data to read or write;
- The size of the files involved, which should make file size / N sufficiently large;
- Number of files involved;
- Stripe count - number of storage nodes available;
- Stripe size of the parallel file system - block of data that is written to a storage node.

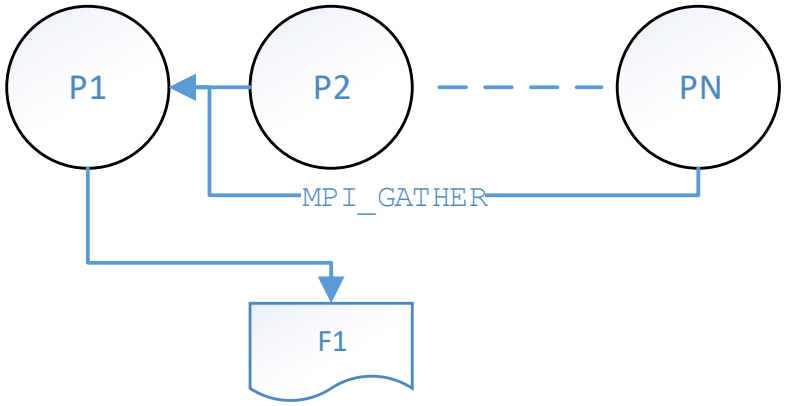
Parallel I/O Models Available for MPI Codes

Parallel I/O Models (1)

- One file per MPI process (N:N):

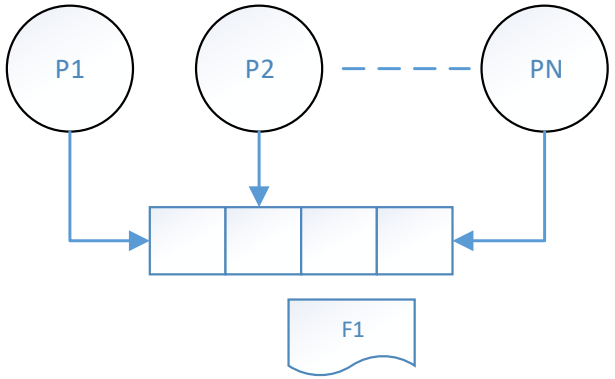


- Single file (1:1) model:

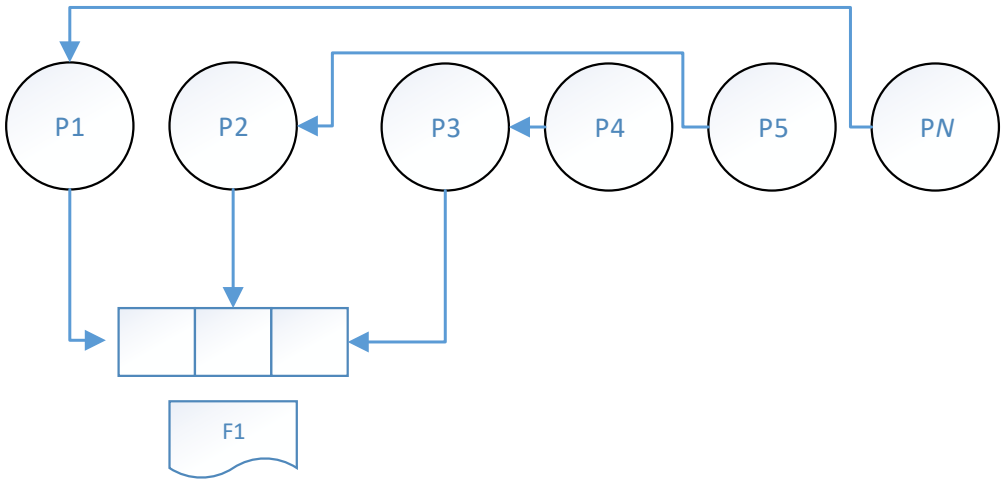


Parallel I/O Models (2)

- Shared file (N:1) model:

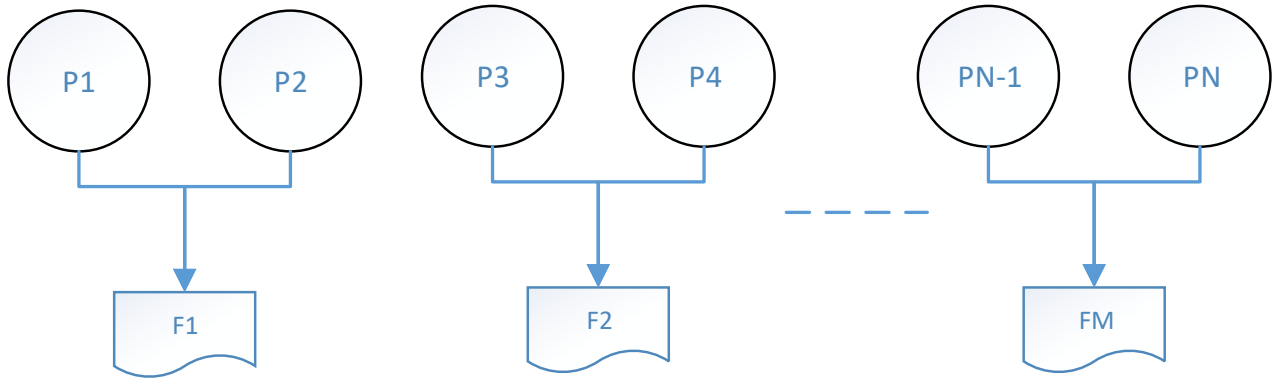


- Alternative shared file (M:1) model (M < N):



Parallel I/O Models (3)

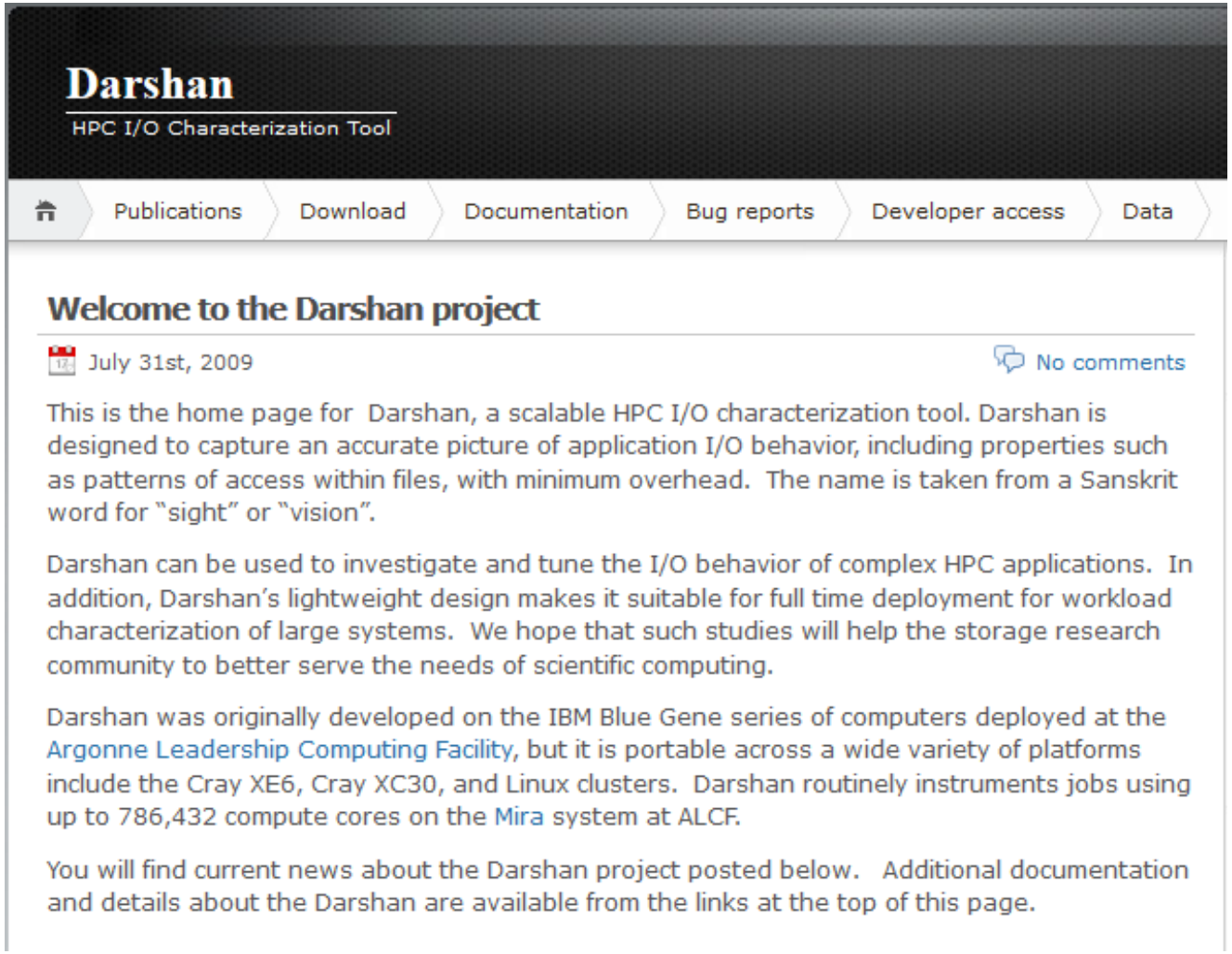
- Hybrid model (N:M) where M < N with M groups:



Darshan Profiling Tool

Obtaining Darshan

- Darshan is open-source and can be downloaded from [1];
- Must be built with the same MPI distribution as your application;
- You can use the GNU compilers for building - doesn't have to be the same as the compiler used to build your application;
- It is developed by Argonne National Laboratory.



File I/O Profiling (1)

- Just as computation and communication can be profiled, so can file I/O be profiled;
- Subsequently, file I/O can also be optimised;
- Following I/O methods are used in HPC applications: POSIX, MPI-IO, parallel NetCDF and parallel HDF5;
- Darshan [1] is able to profile all four methods and *can only profile MPI codes.* Must call `MPI_FINALIZE`, so Darshan will not work if `MPI_ABORT` is called;
- Serial codes can be profiled be enclosing the code with `MPI_INIT` and `MPI_FINALIZE` and running it with `mpirun -n 1 ./app.exe`;

File I/O Profiling (2)

- Hybrid MPI + OpenMP is also supported;
- Darshan only profiles codes written in C, C++ and Fortran. Has been used for MPI4PY (Python) but not fully supported and tested;
- Darshan instruments I/O - not statistical sampling. Thus, profiles are accurate;
- Each I/O call is intercepted by the library;
- Each MPI process collates I/O metrics and collected when an `MPI_FINALIZE` call is made;
- The memory footprint of each MPI process is around 2 MiB.

Invoking Darshan

- No code changes are required to use Darshan. Dynamic executables do not require recompilation;
- Static executables require re-compilation using Darshan MPI wrappers;
- Darshan provides a summary of I/O statistics as well as a timeline;
- Darshan can be loaded as a dynamic library if profiling a Linux dynamic executable:

```
LD_PRELOAD=/lib/libdarshan.so mpirun -n 128 ./wrf.exe
```

- For profiling MPI4PY, you can *only* use the `LD_PRELOAD` variable method.

Processing the Darshan Trace File

- After application execution, the trace file can be found in the Darshan log directory. The filename has the following naming format:

```
<user>_<experimentID>_<executable>_id<JOB_ID>_<timestamp>.darshan.gz
```

- Darshan can create a PDF report from the trace file:

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

- Or individual statistics can be viewed in text format:

```
darshan-parser <trace file>
```


Processing the Darshan Trace File

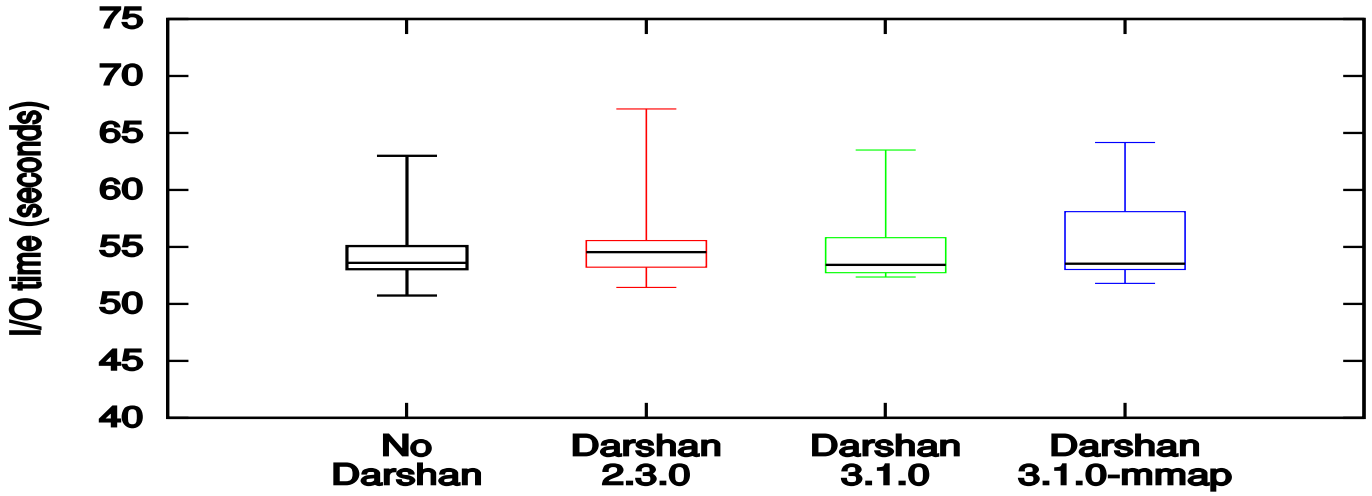
- PDF reports on individual files can also be created using:

```
darshan-summary-per-file.sh <trace file> <output-
directory>
```

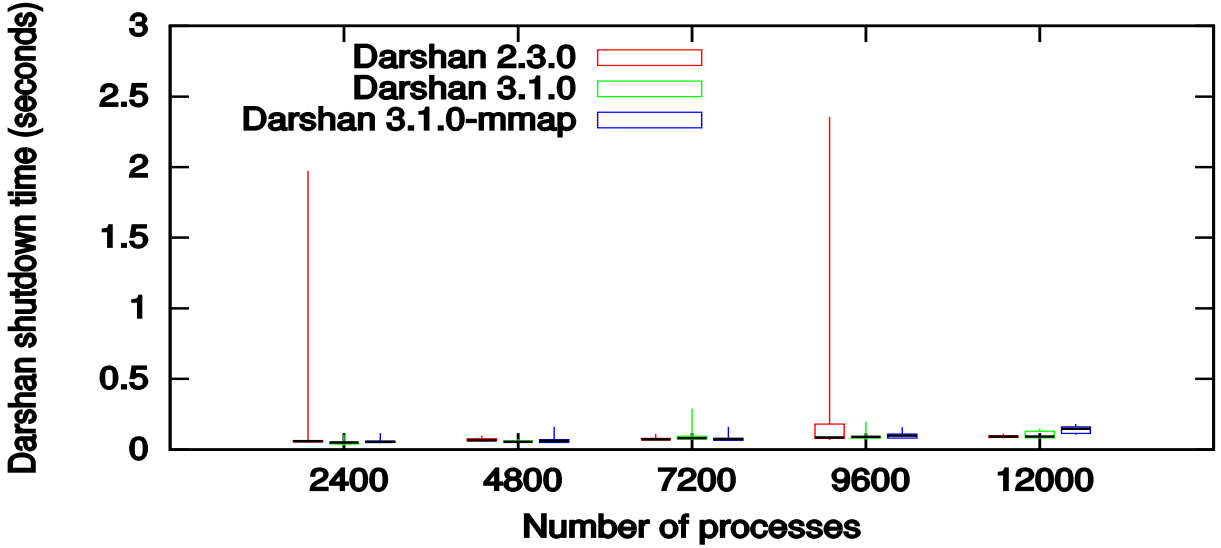
- This is useful to focus on performance metrics on specific input files or output files;
- The trace files are in binary format and are compressed with the zlib compression library.

Overhead of Darshan (1)

- Darshan overhead of a 6,000 MPI process job with one file per process [2, 5]:

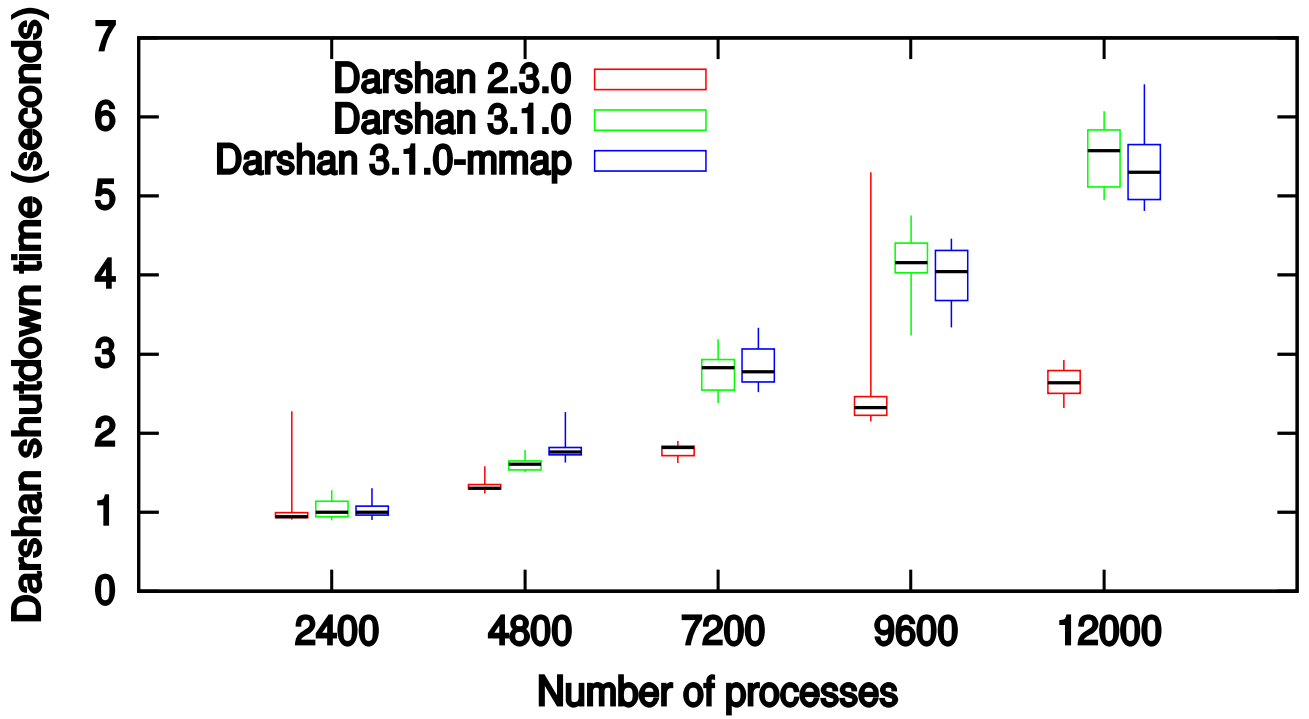


- The shutdown time for a shared file of Darshan is nearly constant with increasing MPI process counts.

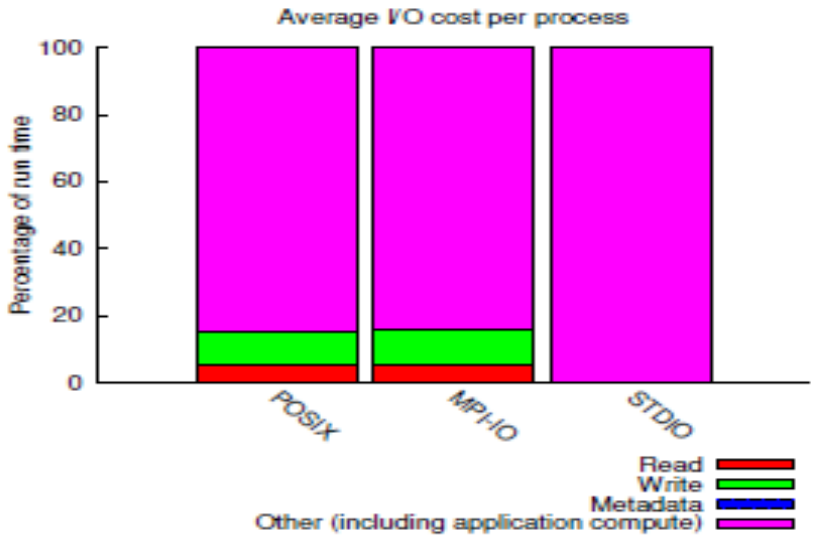


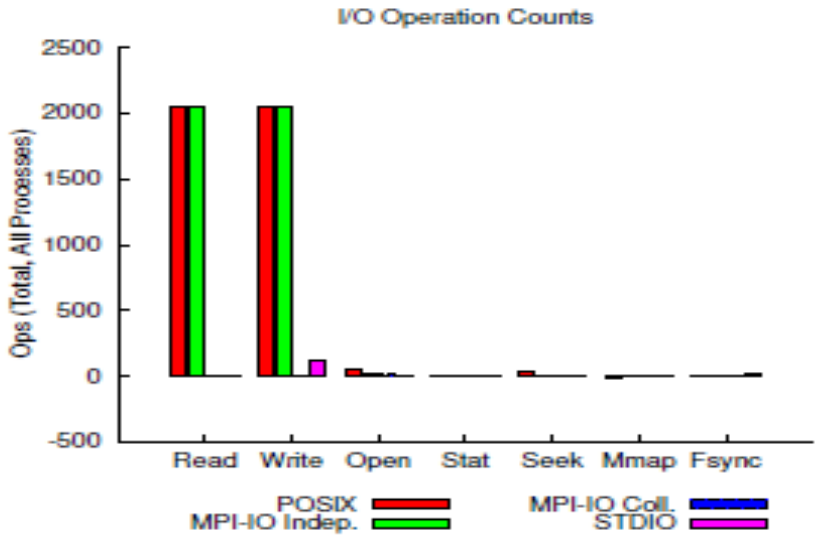
Overhead of Darshan (2)

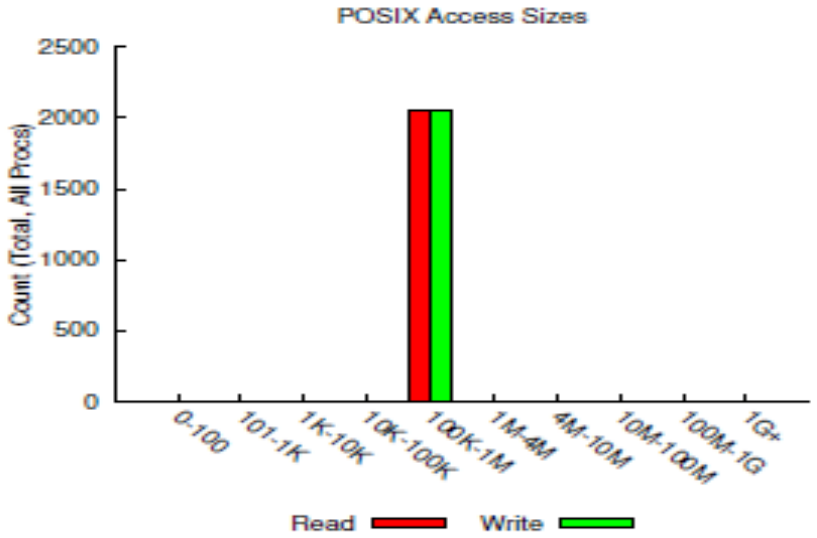
- The shutdown time for one file per process of Darshan scales linearly with increasing MPI process counts;
- Some HPC systems switch on Darshan profiling for all their jobs, so it is very lightweight.

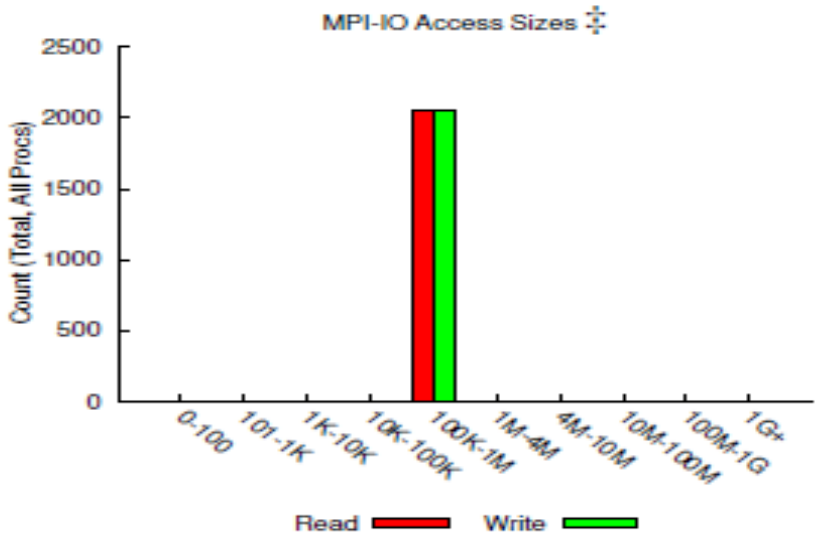


Darshan Report Graphs









Darshan Report Tables (1)

- Table below shows file statistics:

Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	1048576	4096
MPI-IO ‡	1048576	4096

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

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

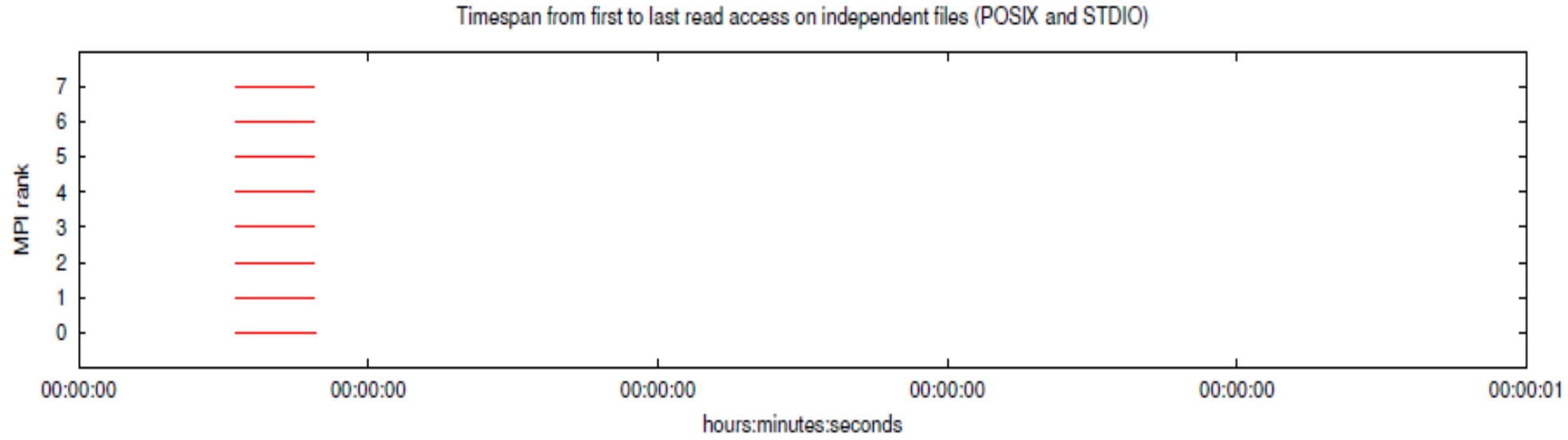
type	number of files	avg. size	max size
total opened	9	228M	256M
read-only files	0	0	0
write-only files	1	1.6K	1.6K
read/write files	8	256M	256M
created files	9	228M	256M

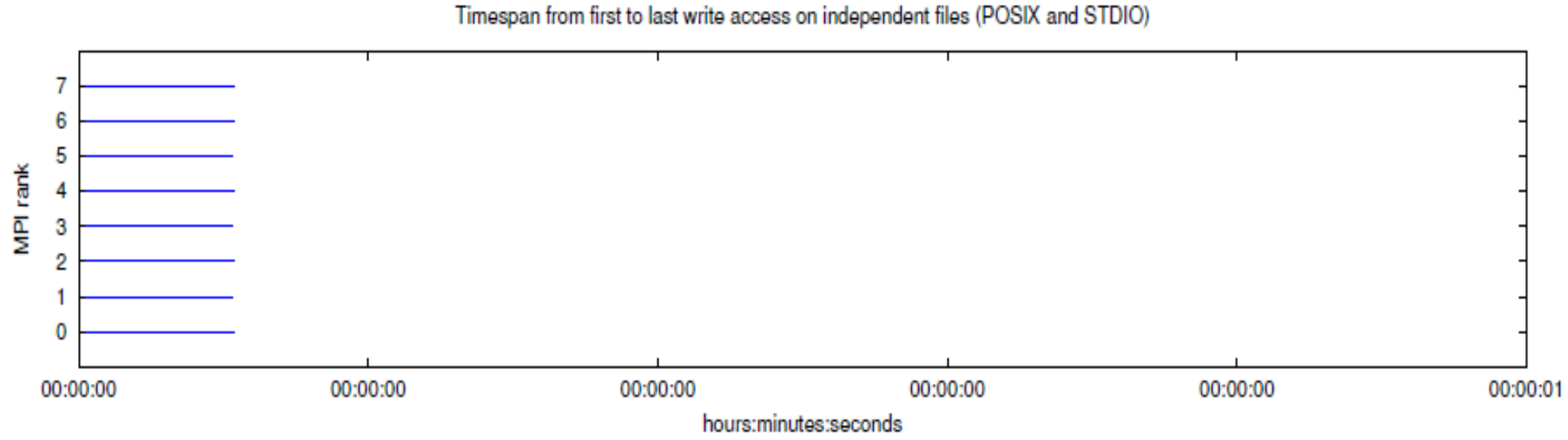
- Opening a large number of small files and/or a large number of I/O operation counts could be a cause of performance problems;

Darshan Report Tables (2)

- Ideally, the code should have large access sizes as parallel file systems are bandwidth bound;
- Parallel NetCDF and parallel HDF5 appear as MPI-IO.

Darshan Timeline



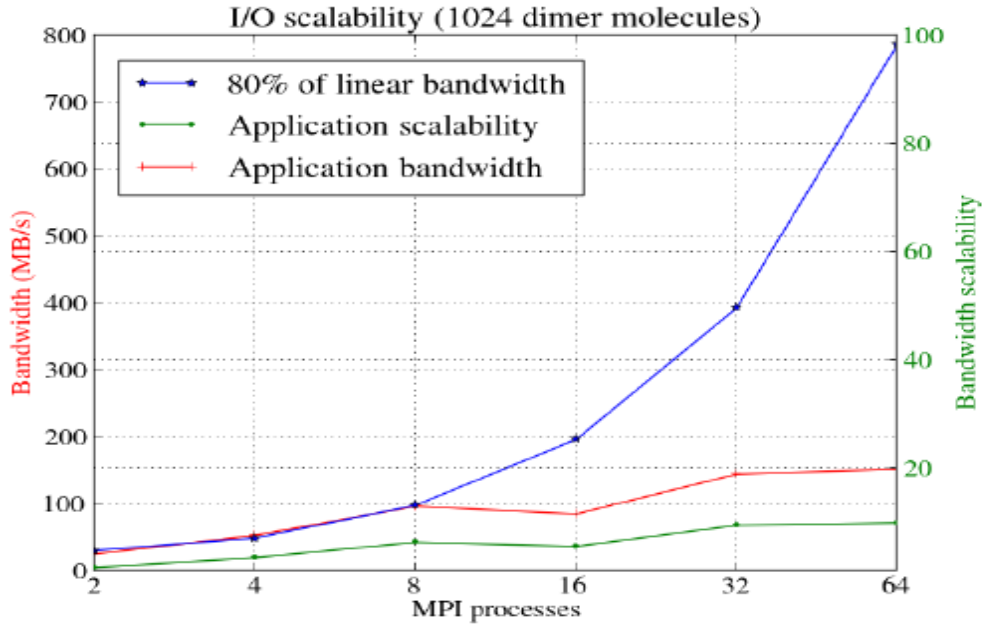


Darshan Demonstration

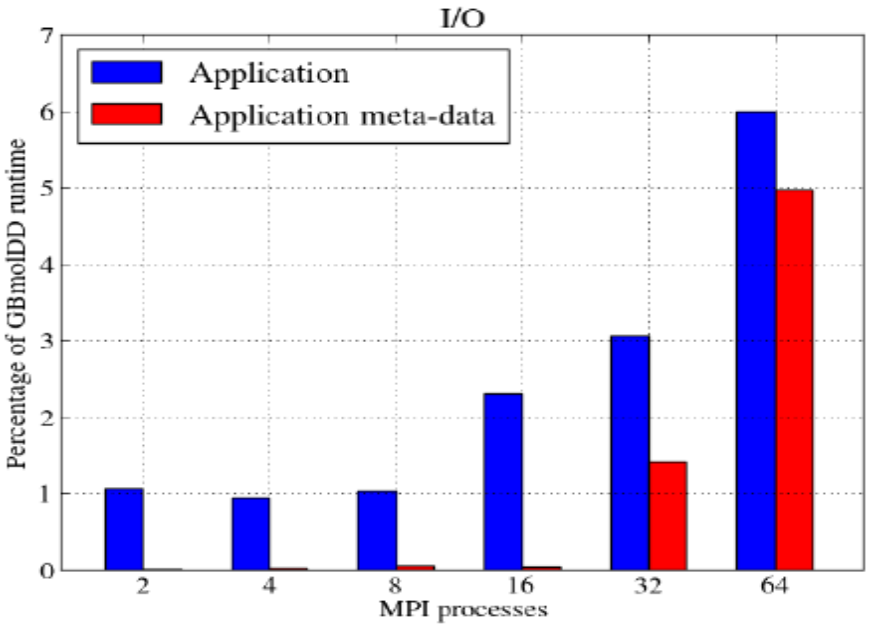
Darshan Case Studies

GBmolDD -Computational Chemistry Code (1)

- GBmolDD simulates coarse-grained systems of isotropic and/or anisotropic particles. Uses the Lennard-Jones potential function to approximate interaction;
- Molecules' position, energy and temperature is written using one file per MPI process using POSIX I/O (N:N);



(a) I/O scalability



(b) I/O percentage of runtime

GBmolDD -Computational Chemistry Code (2)

- For 64 MPI processes onwards, the I/O is spending more time in file meta-data mode than in data mode;
- This is because a Linux inode has to be created for each file (64 MPI process run created 192 files);
- Parallel file systems have fewer meta-data servers, so this quickly becomes a bottleneck;
- The separate files have to be post-processed, so this takes more time;
- Recommendation was to use a parallel file format such as MPI-IO, parallel NetCDF or parallel HDF5 using the N:1 or N:M model.

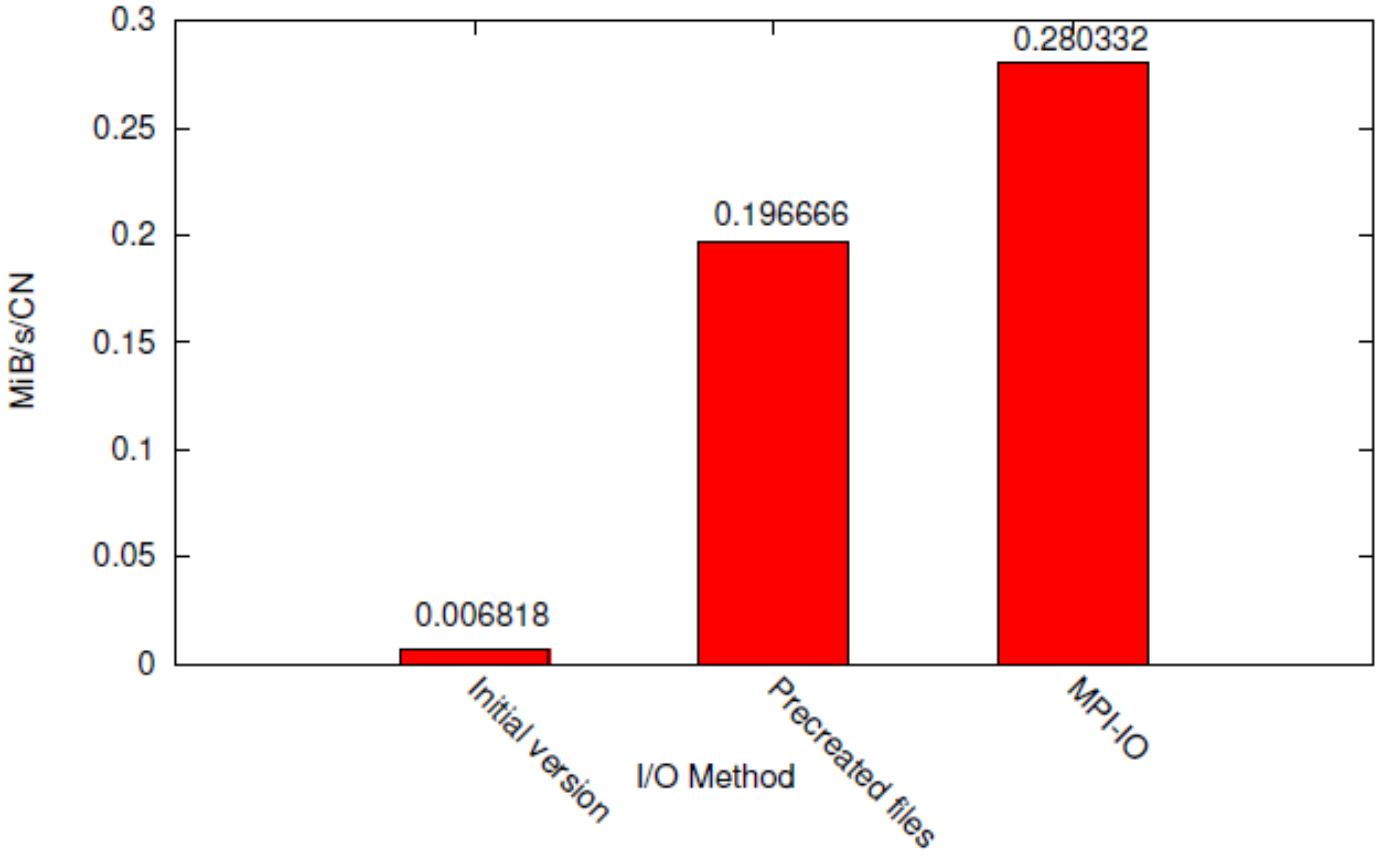
Combustion Physics Code (1)

- Combustion code [3] was check pointing at intervals using one file per MPI rank (N:N);
- Writing 2^9 mesh points and creating two 20 GiB checkpoint files;
- The file creation time (Linux inode) was considerable and reduced the overall I/O bandwidth;
- Each checkpoint took 728 seconds to complete. The checkpoint files were pre-created prior to the simulation which reduced the I/O to 25 seconds;

Combustion Physics Code (2)

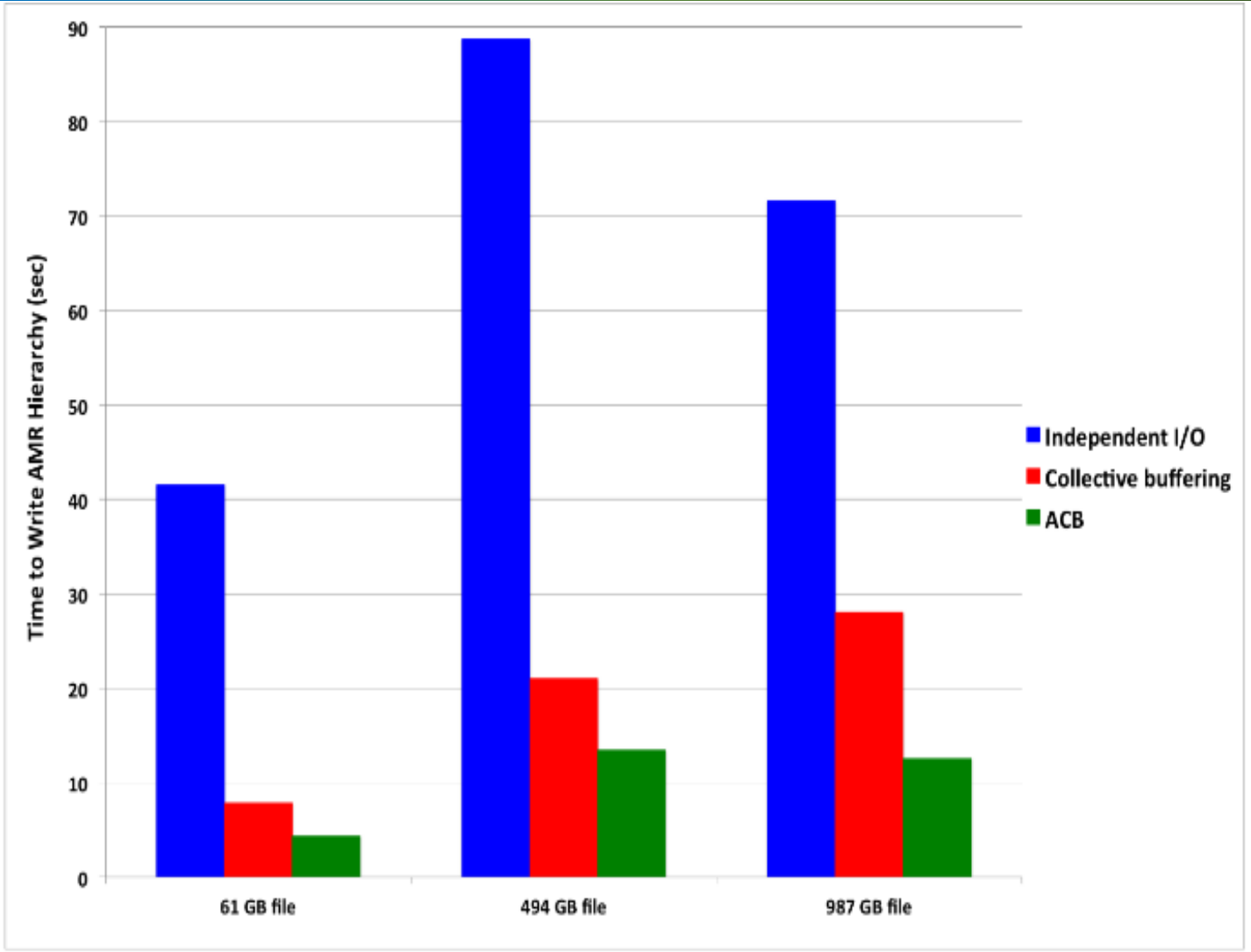
- The code was profiled with Darshan to measure bandwidth per compute node and a shared file MPI-IO version was implemented;

- The code also used MPI collectives to aggregate write operations with block alignment to increase bandwidth;
- Writes of 1 to 4 MiB were aggregated to 16 MiB writes;
- Number of write operations was reduced from 16k to 4k.



Chombo - AMR PDE Solver (1)

- Chombo is PDE solver on adaptive meshes (AMR);
- Each MPI process writes its own box, resulting in a large number of independent write operations;
- Uses aggregated collective buffering (ACB). Performance was compared with MPI-IO collective buffering [4];

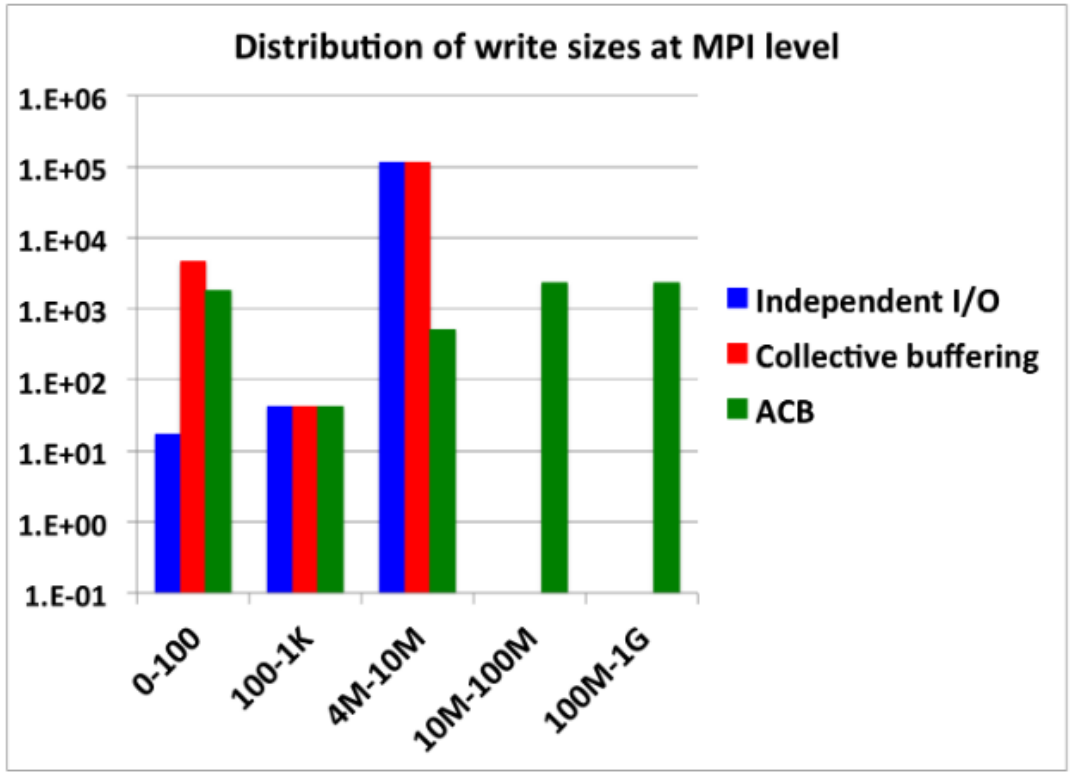


Chombo - AMR PDE Solver (2)

- Darshan profiling showed the following characteristics:

	Ind. I/O	Coll. I/O	ACB. I/O
MPI-IO #writes	115268	119808	6912
Max access size	4 MB	4 MB	8 MB

- ACB is aggregate collective buffering:



Programming Hints and Tips

Programming Tips (1)

- Parallel file systems are bandwidth bound, so try to write/read large amounts of data with fewer operations;
- Reduce the number of new files created. File creation is expensive;
- Avoid POSIX I/O - acceptable for configuration files but not for large data files;
- Write data contiguously to avoid expensive file seek operations;
- Avoid opening and closing files multiple times. Open it once, read/write the data and close it at the end;
- Use either MPI-IO, parallel NetCDF or parallel HDF5 for data;

Programming Tips (2)

- Use I/O aggregation for small writes;
- Configuration files should be read by a single process and then broadcasted to other MPI processes;
- Parallel I/O offers further optimisation opportunities using MPI-IO hints using:

```
MPI_INFO_SET( hints, key, value, ierr )
```

- If a shared file model is not suitable for your parallel file system, e.g. because of file lock contention, then try an N:M approach. N is the number of MPI processes and M is the number of files where M < N;
- Use collective I/O subroutines instead of independent calls;

Programming Tips (3)

- What is the best approach? N:M, M:1 or N:1? This is a trade-off between file lock contention and meta-data creation time;
- Suggestion: N:1 for small N, M:1 for medium N, and N:M for large N;
- For very large MPI process counts, create two communicators: (N - M) processes do computation and M processes do I/O asynchronously;
- For the M:1 or N:M approaches, select M = √N assuming the right balance between number of I/O nodes and compute nodes;
- Always profile your code! This could be included as part of acceptance testing;
- The recent CPU bugs are likely to affect I/O performance [6, 11], hence the need for profiling.

Acknowledgement

- Glenn Lockwood of Argonne National Laboratory - Darshan developer;
- Phil Carns of Argonne National Laboratory - Darshan developer.

Questions and Answers

References (1)

[1] http://www.mcs.anl.gov/research/projects/darshan/

[2] "HPC I/O for Computational Scientists: Understanding I/O", P. Carns, *et al*. ATPESC 2017

[3] “Understanding and Improving Computational Science Storage Access through Continuous Characterization”, P. Carns, et al. Proceedings of 27th IEEE Conference on Mass Storage Systems and Technologies (MSST 2011), 2011

[4] “Collective I/O Optimizations for Adaptive Mesh Refinement Data Writes on Lustre File System”, D. Devendran, *et al*. CUG 2016

[5] “Modular HPC I/O Characterization with Darshan”. S. Snyder, *et al*. Proceedings of 5th Workshop on Extreme-scale Programming Tools (ESPT 2016), 2016.

[6] https://www.hpcwire.com/2018/01/10/will-meltdown-spectre-patches-affect-hpc-workloads/

References (2)

[7] “High Performance I/O”, A. Jackson, et al. 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2011;

[8] “High Performance Parallel I/O”, Prabhat and Q. Koziol. CRC Press, 2014;

[9] “Scalable Input/Output”, D. Reed, MIT Press, 2004;

[10] “Scientific Data Management”, Arie Shoshani and Dorom Rotem. CRC Press, 2009;

[11] https://www.hpcwire.com/2018/01/17/researchers-measure-impact-meltdown-spectre-patches-hpc-workloads/

Performance Optimisation and Productivity

A Centre of Excellence in Computing Applications

Contact:

https://www.pop-coe.eu

mailto:pop@bsc.es

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 676553.