



جامعة الملك عبد الله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Parallel I/O

KSL HPC seminar

Bilel Hadri
Computational Scientist
KAUST Supercomputing Lab
bilel.hadri@kaust.edu.sa

Hardware



- **Shaheen Cray XC40**
 - 6174 nodes of 32 cores Haswell with a total of 792 TB of memory
 - Cray Sonexion® 2000 Storage System with 17.2 PB of usable capacity with performance exceeding 500 GB/sec.
 - Cray DataWarp with a capacity of 1.5 PB and a performance exceeding 1.5 TB/sec (Will be presented in April Seminar)
 - Cray Tiered Adaptive Storage (TAS)r with 20 PB of capacity (up to 100 PB)

Data Storage

Area	Path	Type	Quota	Backups	Purged
User home	/home/username	NFS	200 G	Yes	No
User project	/project/kxxx	Lustre	20T	No	No
User scratch	/scratch/username	Lustre	-	No	Yes 60 days
Scratch project	/scratch/kxxx	Lustre	-	No	Yes 60 days
Scratch tmp	/scratch/tmp	Lustre	-	No	Yes 3 days

- Home directory, designed for development. Previous versions of files can be recovered from /home/<username>/.snapshot directory.
- /scratch/<username>: Temporary individual storage for data needed for execution. Files not accessed in the last 60 days will be deleted.
- /project/k###: 20 TB per project. All files are copied to tape. Once a project has used 20 TB of disk storage, files will be automatically deleted from disk with a weighting based on date of last access. Stub files will remain on disk that link to the tape copy.

HPC systems and I/O

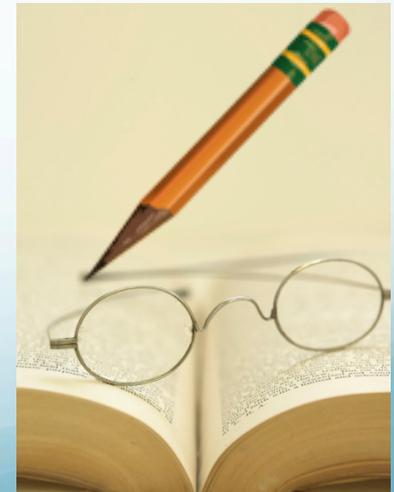
- "A supercomputer is a device for converting a CPU-bound problem into an I/O bound problem." [Ken Batcher]
- Machines consist of three main components:
 - Compute nodes
 - High-speed interconnect
 - I/O infrastructure
- Most optimization work on HPC applications is carried out on
 - Single node performance
 - Network performance (communication)
 - I/O only when it becomes a real problem

Why do we need parallel I/O?

- I/O subsystems are typically very slow compared to other parts of a supercomputer
 - You can easily saturate the bandwidth
- Once the bandwidth is saturated scaling in I/O stops
 - Adding more compute nodes increases aggregate memory bandwidth and flops/s, but not I/O
- Imagine a 24 hour simulation on 16 cores.
 - 1% of run time is serial I/O.
 - You get the compute part of your code to scale to 1024 cores.
 - 64x speedup in compute: I/O is 39% of run time (22'16" in computation and 14'24" in I/O).
- Parallel I/O is needed to
 - Spend more time doing science
 - Not waste resources
 - Prevent affecting other users

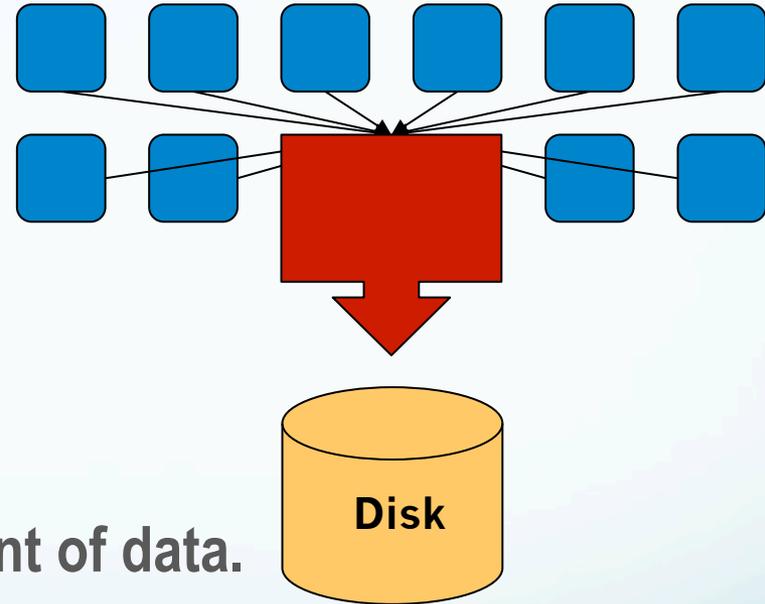
I/O Performance

- There is no “One Size Fits All” solution to the I/O problem.
- Many I/O patterns work well for some range of parameters.
- Bottlenecks in performance can occur in many locations. (Application and/or File system)
- Going to extremes with an I/O pattern will typically lead to problems.
- Increase performance by decreasing number of I/O operations (latency) and increasing size (bandwidth).



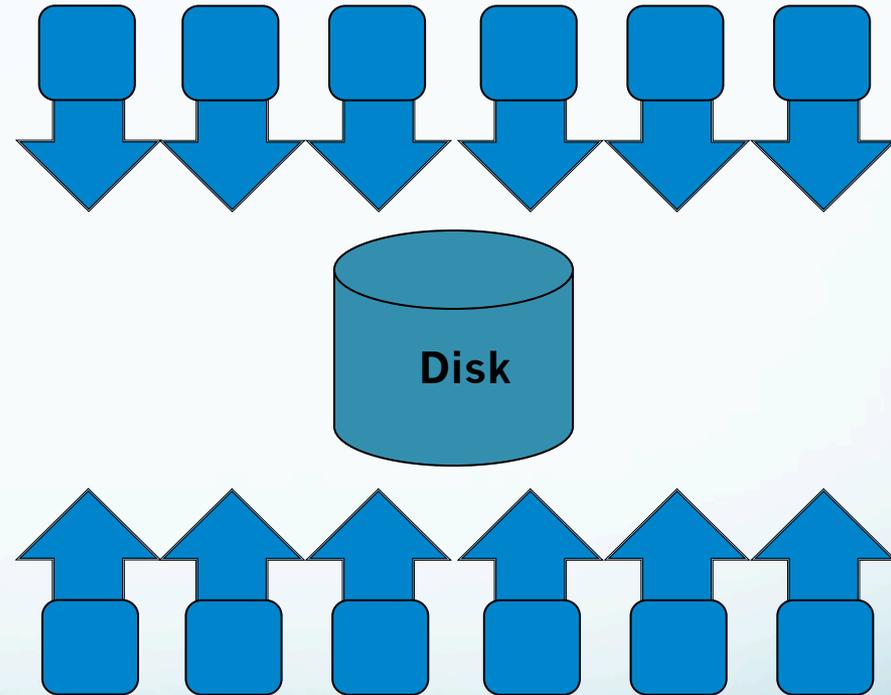
Serial I/O: Spokesperson

- One process performs I/O.
 - Data Aggregation or Duplication
 - Limited by single I/O process.
- Simple solution, easy to manage, but
 - Pattern does not scale.
 - Time increases linearly with amount of data.
 - Time increases with number of processes.



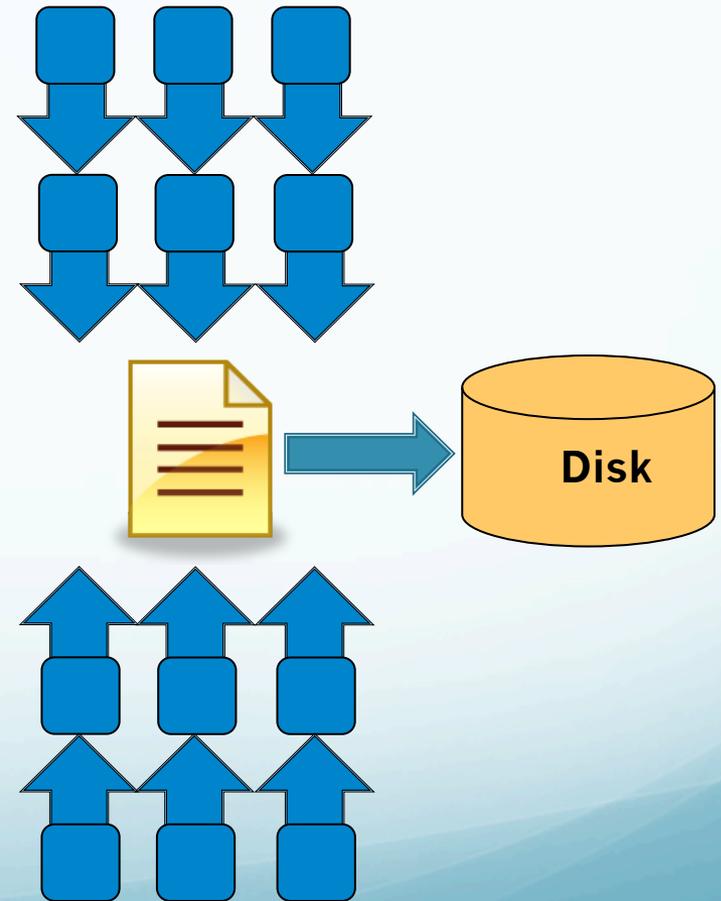
Parallel I/O: File-per-Process

- All processes perform I/O to individual files.
 - Limited by file system.
- Pattern does not scale at large process counts.
 - Number of files creates bottleneck with metadata operations.
 - Number of simultaneous disk accesses creates contention for file system resources.



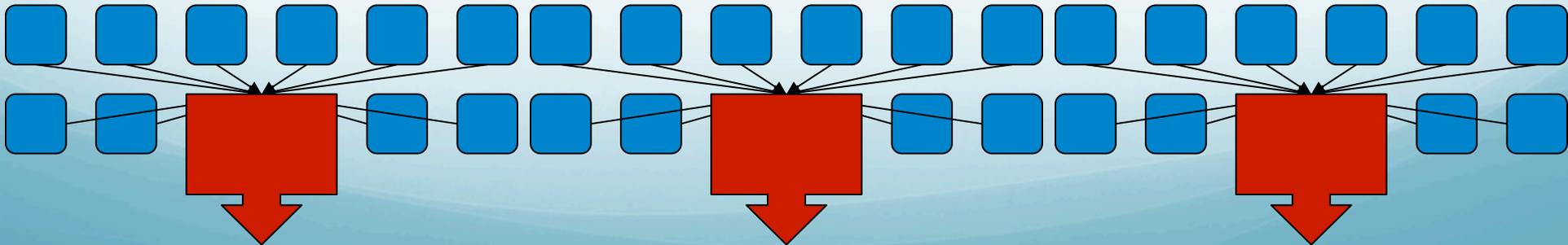
Parallel I/O: Shared File

- **Shared File**
 - Each process performs I/O to a single file which is shared.
 - Performance
 - Data layout within the shared file is very important.
 - At large process counts contention can build for file system resources.



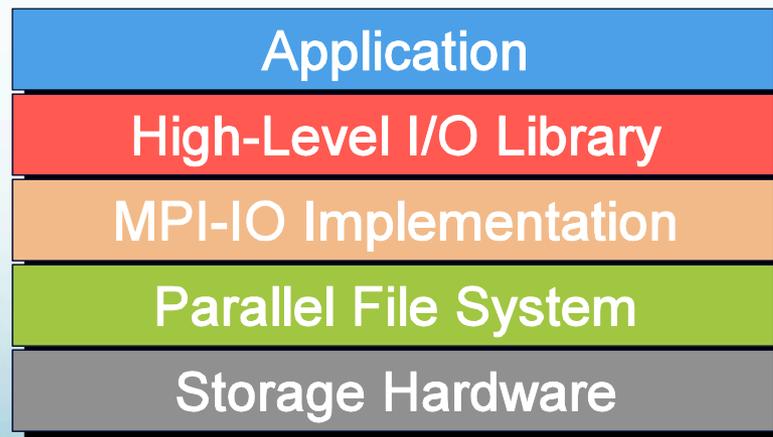
Pattern Combinations

- **Subset of processes which perform I/O.**
 - **Aggregation of a group of processes data.**
 - Serializes I/O in group.
 - **I/O process may access independent files.**
 - Limits the number of files accessed.
 - **Group of processes perform parallel I/O to a shared file.**
 - Increases the number of shared files
 - increase file system usage.
 - **Decreases number of processes which access a shared file**
 - decrease file system contention.



Lustre

- **Lustre file system is made up of an underlying:**
 - **set of I/O servers called Object Storage Servers (OSSs)**
 - **disks called Object Storage Targets (OSTs).**
- **The file metadata is controlled by a Metadata Server (MDS) and stored on a Metadata Target (MDT).**



Shaheen II Sonexion

- **Cray Sonexion 2000 Storage System consisting of 12 cabinets containing a total of 5988 4TB SAS disk drives.**
- **The cabinets are interconnected by FDR Infiniband Fabric .**
- **Each cabinet can contain up to 6 Scalable Storage Units (SSU); Shaheen II has a total of 72 SSUs.**
- **As there are 2 OSS/OSTs for each SSU, this means that there are 144 OSTs in total**

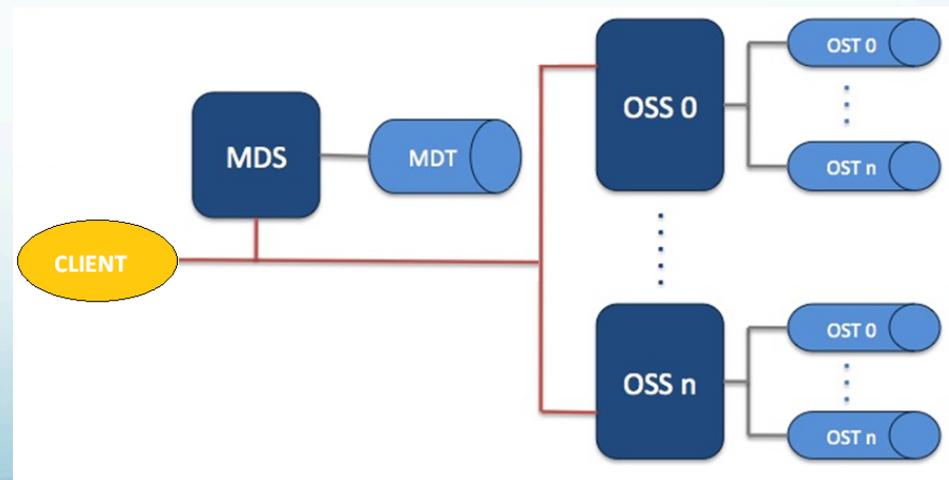


File I/O: Lustre File System



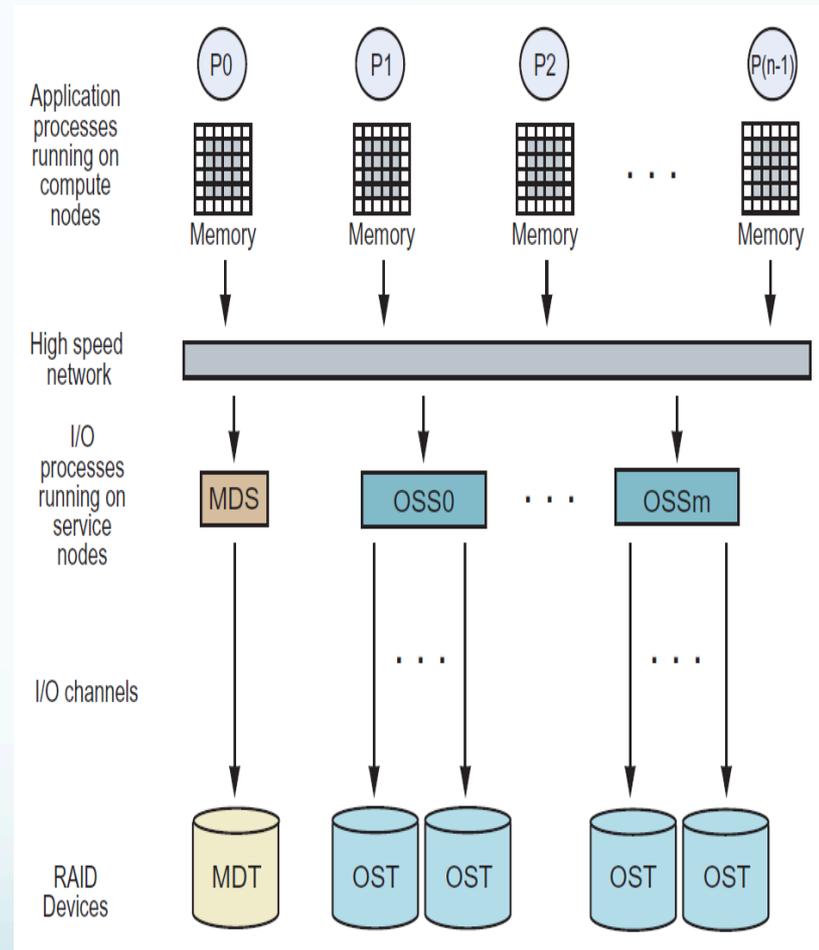
جامعة الملك عبد الله
للعلوم والتقنية
King Abdullah University of
Science and Technology

- **Metadata Server (MDS)** makes metadata stored in the MDT(Metadata Target) available to Lustre clients.
 - The MDS opens and closes files and stores directory and file Metadata such as file ownership, timestamps, and access permissions on the MDT.
 - Each MDS manages the names and directories in the Lustre file system and provides network request handling for the MDT.
- **Object Storage Server(OSS)** provides file service, and network request handling for one or more local OSTs.
- **Object Storage Target (OST)** stores file data (chunks of files).



Lustre

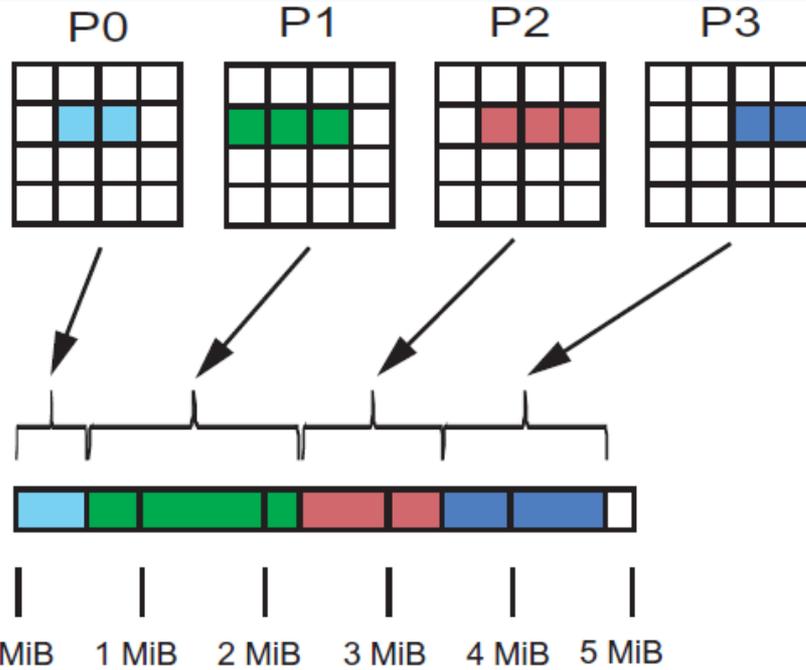
- Once a file is created, write operations take place directly between compute node processes (P0, P1, ...) and Lustre object storage targets (OSTs), going through the OSSs and bypassing the MDS.
- For read operations, file data flows from the OSTs to memory. Each OST and MDT maps to a distinct subset of the RAID devices.



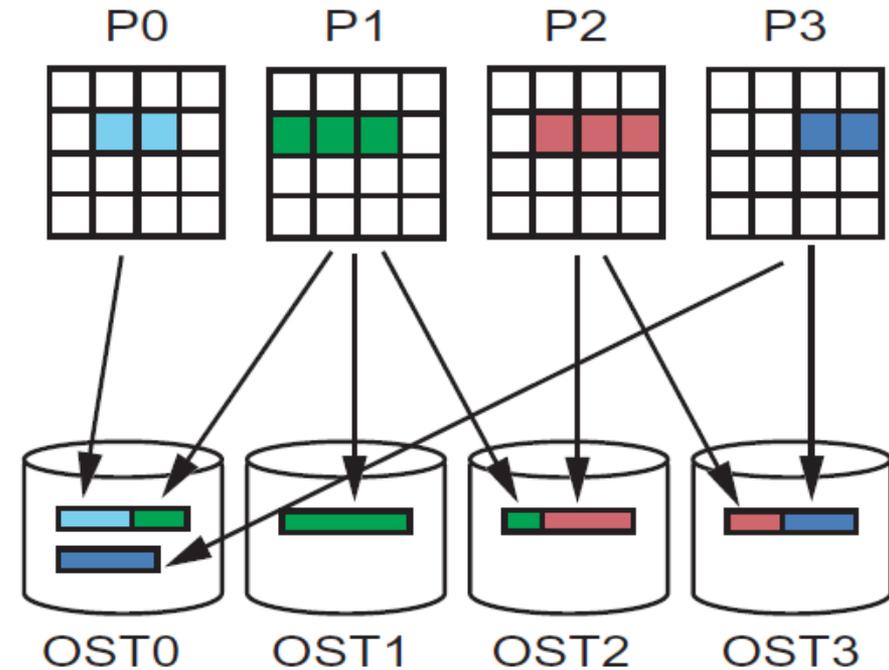
Lustre filestripping

- **Files on the Lustre filesystems can be striped**
 - transparently divided into chunks that are written or read simultaneously across a set of OSTs within the filesystem.
 - The chunks are distributed among the OSTs using a method that ensures load balancing.
- **Benefits include:**
 - Striping allows one or more clients to read/write different parts of the same file at the same time, providing higher I/O bandwidth to the file because the bandwidth is aggregated over the multiple OSTs.
 - Striping allows file sizes larger than the size of a single OST. In fact, files larger than 100 GB *must* be striped in order to avoid taking up too much space on any single OST, which might adversely affect the filesystem.

File Striping: Physical and Logical Views



Four application processes write a variable amount of data sequentially within a shared file. This shared file is striped over 4 OSTs with 1 MB stripe sizes.

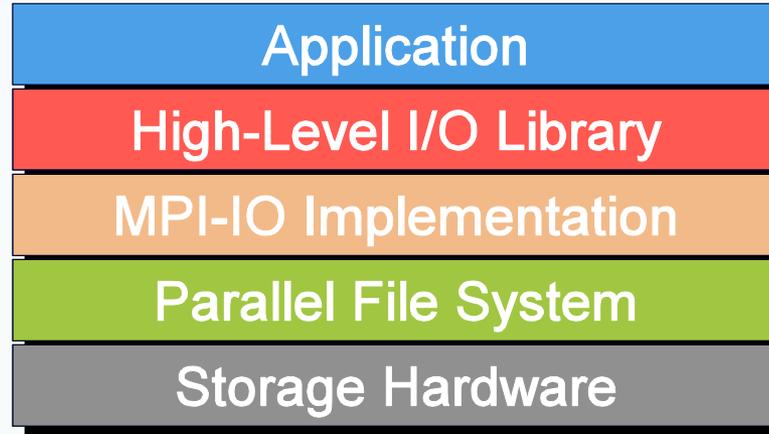


This write operation is not stripe aligned therefore some processes write their data to stripes used by other processes. Some stripes are accessed by more than one process

→ May cause contention !

OSTs are accessed by variable numbers of processes (3 OST0, 1 OST1, 2 OST2 and 2 OST3).

Parallel I/O tools for Computational Science



- High level I/O library maps app. abstractions to a structured, portable file format (e.g. HDF5, Parallel netCDF)
- Middleware layer deals with organizing access by many processes (e.g. MPI-IO)
- Parallel file system maintains logical space, provides efficient access to data (e.g. Lustre)

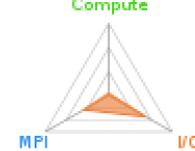
➔ Is my code I/O bound ?

Allinea DDT

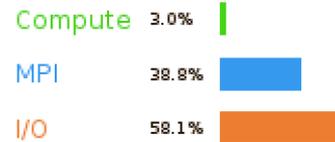
- **Compile the code**
 - `username@cdl4:~> module load allinea-report/5.1-43967`
 - `username@cdl4:~> make-profiler-libraries`
 - `username@cdl4:~> cc -O3 myapp.c -o myapp.exe -dynamic -L$PWD -lmap-sampler-pmpi -lmap-sampler -Wl,--eh-frame-hdr`
- **Edit the submission script to add:**
 - `export LD_LIBRARY_PATH=/path/to/map-sampler-libs:$LD_LIBRARY_PATH`
 - `export ALLINEA_MPI_WRAPPER=/path/to/map-sampler-libs/libmap-sampler-pmpi.so`
 - `perf-report srun ./myapp.exe myargs`
- **When the job is submitted and executed, it will produce a .html and .txt file that can be open on the login node afterwards:**
- `username@cdl4:~> firefox myapp.exe_Xp_DD-MM-YYYY-HH-MM.html` or check the txt file
- **Note for python code:**
 - `perf-report srun /sw/xc40/python/3.4.2/cnl5.2_gnu4.9.2/bin/python3 test.py`



Command: `strun -n 16 ./benchio_allinea`
 Resources: 1 node (32 physical, 64 logical cores per node)
 Memory: 126 GB per node
 Tasks: 16 processes
 Machine: nid00797
 Start time: Mon Feb 1 18:48:06 2016
 Total time: 8 seconds (0 minutes)
 Full path: `/lustre/scratch/hadrib/io-exercises/benchio`
 Input file:
 Notes:



Summary: benchio_allinea is **I/O-bound** in this configuration



Time spent running application code. High values are usually good. This is **very low**; focus on improving MPI or I/O performance first.

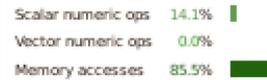
Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it.

Time spent in filesystem I/O. High values are usually bad. This is **very high**; check the I/O breakdown section for optimization advice.

This application run was **I/O-bound**. A breakdown of this time and advice for investigating further is in the **I/O** section below.

CPU

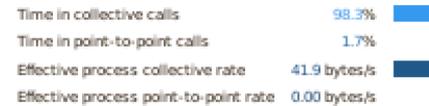
A breakdown of the **3.0%** CPU time:



The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance. No time is spent in **vectorized instructions**. Check the compiler's vectorization advice to see why key loops could not be vectorized.

MPI

A breakdown of the **38.8%** MPI time:



Most of the time is spent in **collective calls** with a very low transfer rate. This suggests load imbalance is causing synchronization overhead; use an MPI profiler to investigate.

I/O

A breakdown of the **58.1%** I/O time:



Transfer rate unavailable (no `/proc/<pid>/io`)
 Most of the time is spent in **write operations** with a very low effective transfer rate. This may be caused by contention for the filesystem or inefficient access patterns. Use an I/O profiler to investigate which write calls are affected.

Threads

A breakdown of how multiple threads were used:



No measurable time is spent in multithreaded code. **Physical core utilization** is low. Try increasing the number of processes to improve performance.

Memory

Per-process memory usage may also affect scaling:



The peak node memory usage is very low. Running with fewer MPI processes and more data on each process may be more efficient.

Allinea (2)

Craypat

- **Load Craypat performance tools**
 - **module unload darshan**
 - **module load perftools**
- **Build application keeping .o files**
 - **make clean ; make**
- **Instrument application for automatic profiling analysis**
 - **You should get an instrumented program a.out+pat**
 - **pat_build -u -g hdf5,mpi,io,sysio a.out+pat**
- **Run application to get top time consuming routine**
 - **srun... a.out+pat**
- **You should get a performance file (“<sdatafile>.xf”) or multiple files in a directory <sdadir>.**
 - **To generate report pat_report -o report.txt <sdatafile>.xf**
 - **For exact line pat_report -b function,source,line <sdatafile>.xf**

Craypat (2)

Table 1:

Time%	Time	Imb. Time	Imb. Time%	Calls	Function Source Line
100.0%	13,461.594081	--	--	666,344.0	Total
32.1%	4,326.121649	--	--	3,072.0	mpi_barrier_(sync)
24.4%	3,284.591116	--	--	48,630.0	MPI_FILE_WRITE_ALL
14.0%	1,884.152065	--	--	71,930.0	h5dwrite_c cray-hdf5-1.8.14-ccel- parallel/fortran/src/H5Df.c 3
12.7%	1,704.005636	--	--	88,516.0	line.334 nc4_put_vara_tc cray-netcdf-4.3.3.1-ccel/ netcdf-4.3.3.1/libsrc4/nc4var.c 3
9.9%	1,338.717666	--	--	49,539.0	line.1431 write_var cray-netcdf-4.3.3.1-ccel/ netcdf-4.3.3.1/libsrc4/nc4hdf.c 3
3.0%	397.666538	--	--	128.0	line.2262 mpi_init_(sync)

Additional details

I/O Utility: Darshan

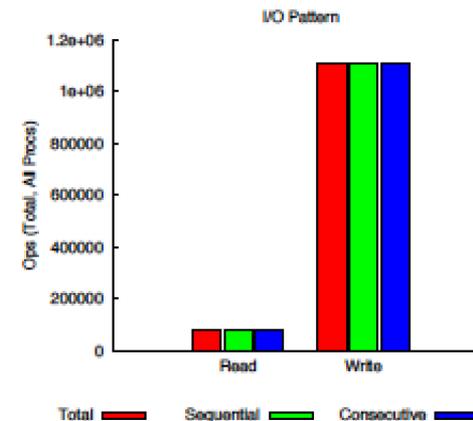
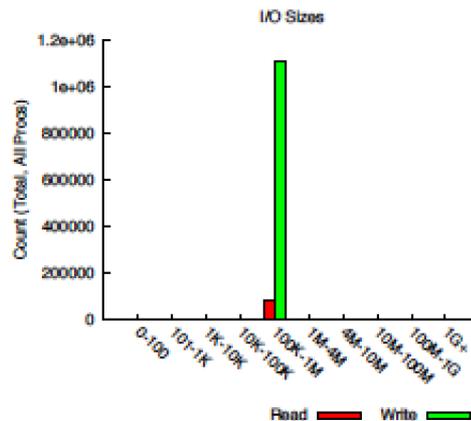
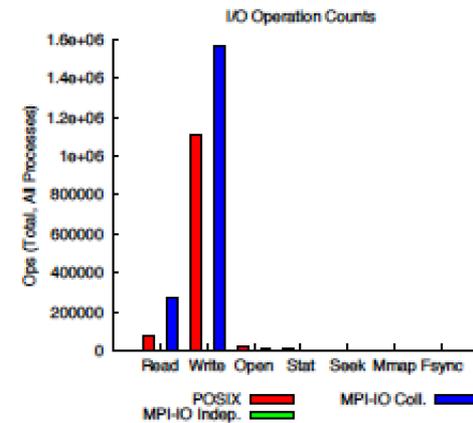
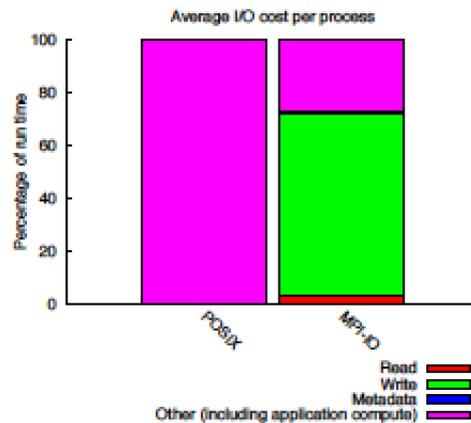
- Darshan is “a scalable HPC I/O characterization tool... designed to capture an accurate picture of application I/O behavior... with minimum overhead”
- I/O Characterization
 - Sheds light on the intricacies of an application’s I/O
 - Useful for application I/O debugging
 - Pinpointing causes of extremes
 - Analyzing/tuning hardware for optimizations
- Installed by default on Shaheen
 - Requires no code modification (only re-linking)
 - Small memory footprint, no-overhead
 - Includes a job summary tool
 - Location of the gz:\$DARSHAN_LOGPATH/YYYY/MM/DD/username_exe_jobid_xxx.gz

Darshan Specifics

- **Darshan collects per-process statistics (organized by file)**
 - Counts I/O operations, e.g. unaligned and sequential accesses
 - Times for file operations, e.g. opens and writes
 - Accumulates read/write bandwidth info
 - Creates data for simple visual representation
- **Get your report :**
 - In pdf : `darshan-job-summary.pl $DARSHAN_LOGPATH/YYYY/MM/DD/username_exe_name_idjobid.xxxxx_darshan.gz`
 - Summary of performance : `darshan-parser -perf $DARSHAN_LOGPATH/YYYY/MM/DD/username_exe_name_idjobid.xxxxx_darshan.gz`



jobid: 667358	uid:]	nprocs: 1216	runtime: 2884 seconds
---------------	--------	--------------	-----------------------

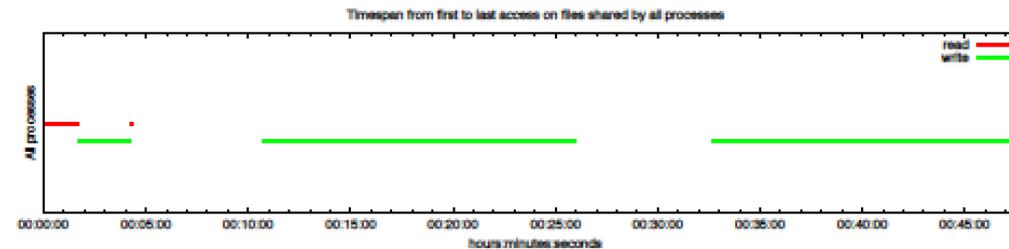
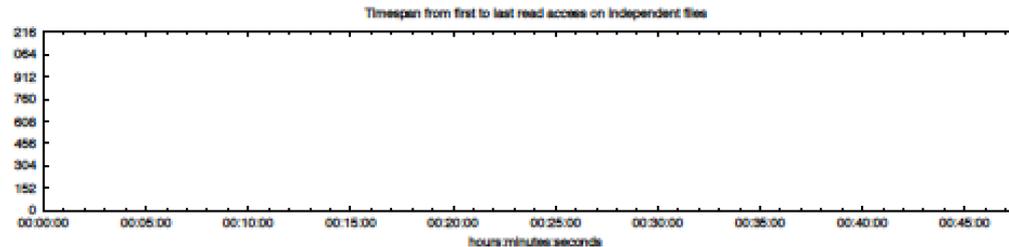


Most Common Access Sizes

access size	count
1048576	1182182
4	81
108	60
16	18

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

type	number of files	avg. size	max size
total opened	2446	485M	411G
read-only files	3	27G	77G
write-only files	6	180G	411G
read/write files	0	0	0
created files	6	180G	411G



Average I/O per process

	Cumulative time spent in I/O functions (seconds)	Amount of I/O (MB)
Independent reads	0.000002	0.000588
Independent writes	0.000000	0.000055
Independent metadata	0.016473	N/A
Shared reads	0.061813	64.928630
Shared writes	0.979258	908.205085
Shared metadata	0.038747	N/A

Data Transfer Per Filesystem

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
/lustre	1104377.45073	1.00000	78953.92893	1.00000

Variance in Shared Files

File Suffix	Processes	Fastest			Slowest			σ	
		Rank	Time	Bytes	Rank	Time	Bytes	Time	Bytes
...-12-18-00_30.00	1216	993	757.912663	0	0	758.125728	411G	13.1	1.26e+10
...-12-18-01_00.00	1216	466	757.339776	0	0	757.524430	411G	12.9	1.26e+10
...-12-18-01_00.00	1216	1159	164.842321	0	0	164.939220	86G	2.8	2.64e+09
...-12-18-00_30.00	1216	423	156.757853	0	0	156.869395	86G	2.75	2.64e+09
...-12-18-00_00.00	1216	1206	152.131270	0	0	152.283136	86G	2.64	2.64e+09
...59/wrfinput_d01	1216	1	87.747921	0	0	100.346905	77G	2.14	2.36e+09
...1759/wrfbdy_d01	1216	8	3.009217	0	0	3.568701	280M	0.0117	8.39e+06
.../namelist.input	1216	956	0.000348	0	0	0.015663	0	0.00224	0

Optimizations

- ***file striping*** to increase IO performance
- **IOBUF**
 - Serial I/O operations
- **MPI-IO hints**
- **Use I/O libraries: HDF5, NetCDF, ADIOS**

Lustre Utility

- Lustre provides a utility to query and set access to the file system that are faster and more efficient than GNU command. They are all sub commands to the program lfs .

```
hadrib@cd14:lfs help
```

```
Available commands are:
```

```
setstripe
```

```
getstripe
```

```
setdirstripe
```

```
getdirstripe
```

```
mkdir
```

```
rm_entry
```

```
pool_list
```

```
find
```

```
check
```

```
join
```

```
osts
```

```
mdts
```

```
df
```

```
getname
```

```
For more help type: help command-name
```

```
quotacheck
```

```
quotaon
```

```
quotaoff
```

```
setquota
```

```
quota
```

```
flushctx
```

```
lsetfacl
```

```
lgetfacl
```

```
rsetfacl
```

```
rgetfacl
```

```
cp
```

```
ls
```

```
changelog
```

```
changelog_clear
```

```
fid2path
```

```
path2fid
```

```
data_version
```

```
hsm_state
```

```
hsm_set
```

```
hsm_clear
```

```
hsm_action
```

```
hsm_archive
```

```
hsm_restore
```

```
hsm_release
```

```
hsm_remove
```

```
hsm_cancel
```

```
swap_layouts
```

```
migrate
```

```
help
```

```
exit
```

```
quit
```

Useful Lustre commands

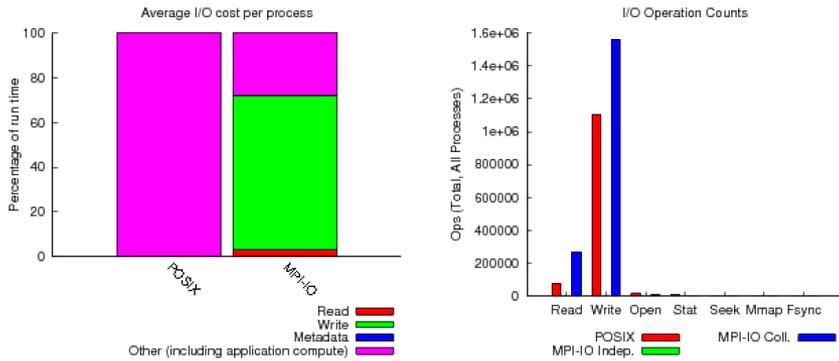
- **Listing Striping Information**
 - `lfs getstripe filename`
 - `lfs getstripe -d directory_name`
- **File stripping:**
 - `lfs setstripe -s stripe_size -c stripe_count dir/filename`
- **Note:** The stripe settings of an existing file cannot be changed. If you want to change the settings of a file, create a new file with the desired settings and copy the existing file to the newly created file.

Combustion code 2x speedup

Stripe count	1	2	4	5	10
time I/O	79	48	37	42	39
time code	122	91	83	87	85
%I/O	65%	53%	45%	48%	46%
Speedup IO	1.00	1.65	2.14	1.88	2.03
Speedup code	1.00	1.34	1.47	1.40	1.44

WRF 12x speedup with file striping

jobid: 667358 nprocs: 1216 runtime: 2884 seconds

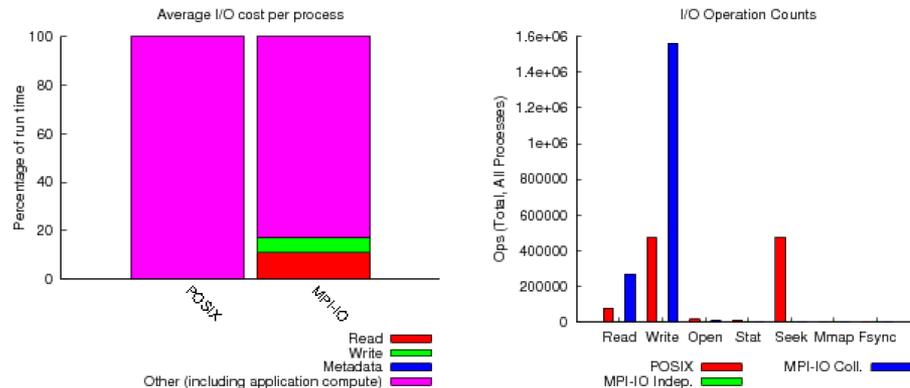


**Default striping 1,
I/O time: 2094 sec
Total time: 2884 sec**

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

type	number of files	avg. size	max size
total opened	2446	485M	411G
read-only files	3	27G	77G
write-only files	6	180G	411G
read/write files	0	0	0
created files	6	180G	411G

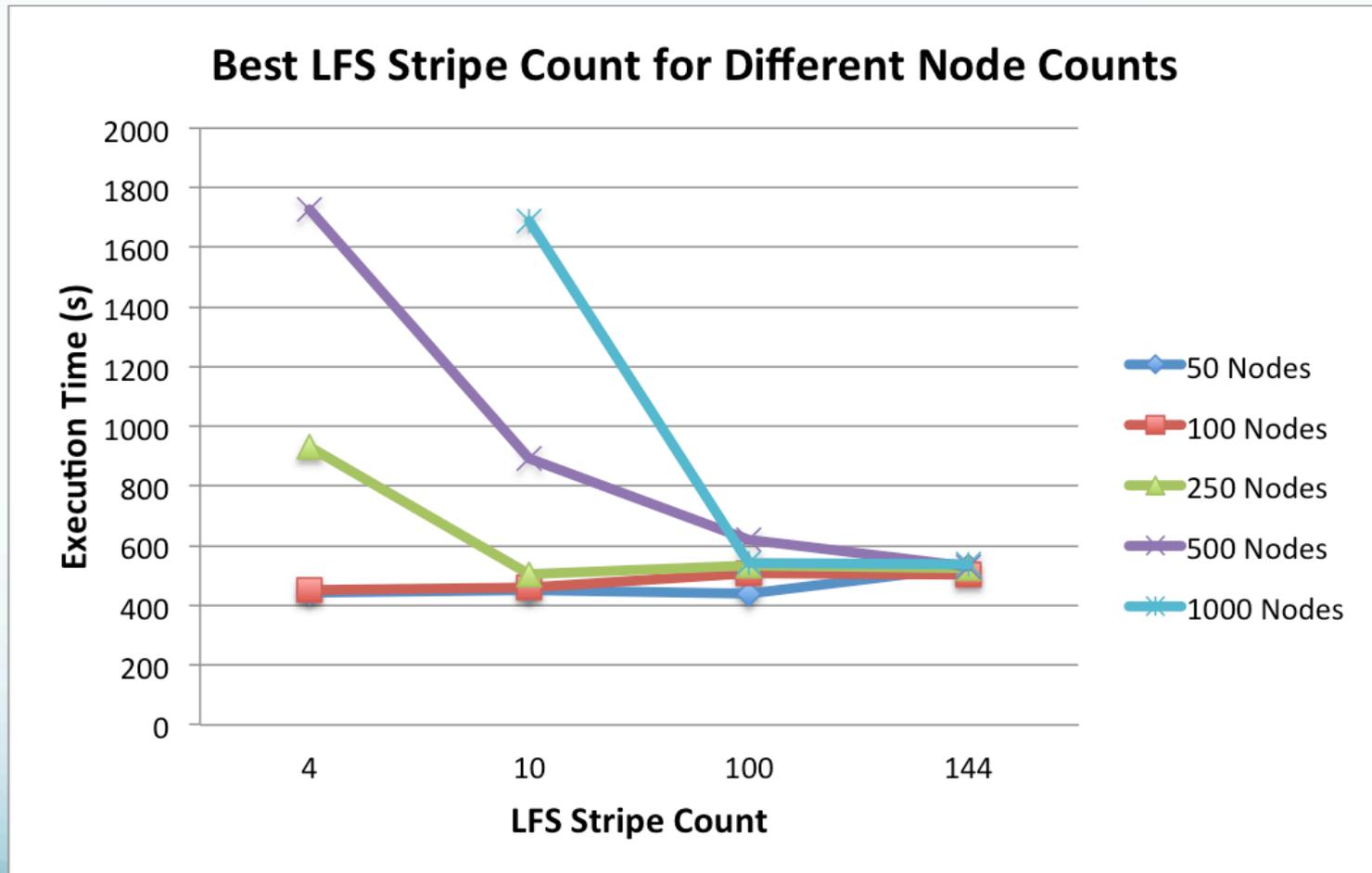
jobid: 669210 nprocs: 1216 runtime: 959 seconds



**Striping over 144 I/O
time: 174 sec
Total time: 959 sec**

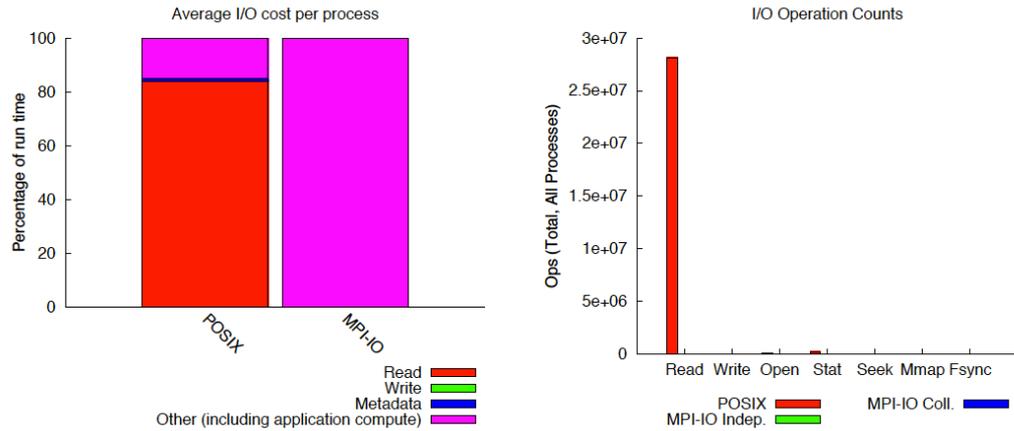
**I/O speedup: 12x
Total time speedup: 3x**

Stripe Tuning for Natural Migration Code



Stripe Tuning for Natural Migration Code

Darshan output before tuning

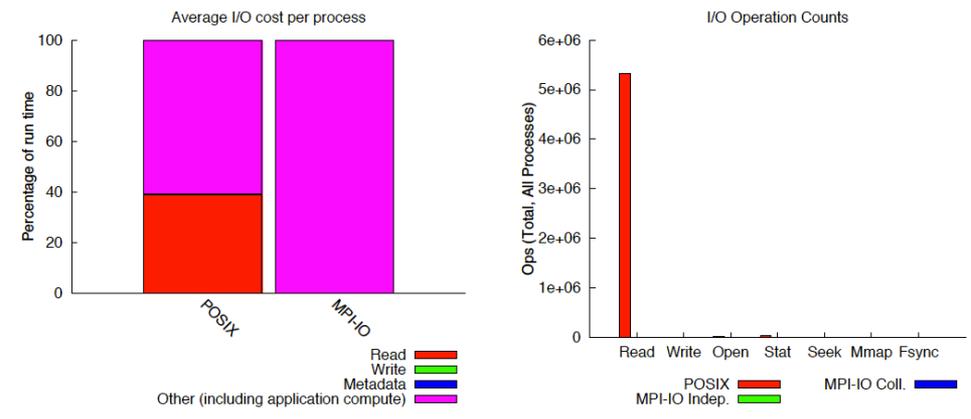


Most Common Access Sizes

access size	count
17310596	28058209
4935	58267
21188	26485
3268	5297

File Count Summary

type	number of files	avg. size	max size
total opened	5305	17M	86G
read-only files	6	15G	86G
write-only files	1	108M	108M
read/write files	0	0	0
created files	1	108M	108M



Darshan output after tuning

IOBuf for serial

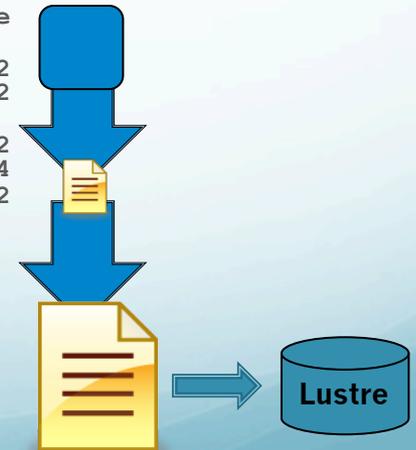
- **Advantages**
 - Aggregates smaller read/write operations into larger operations.
 - Examples: OS Kernel Buffer, MPI-IO Collective Buffering
- **Disadvantages**
 - Requires additional memory for the buffer.
 - Can tend to serialize I/O.
- **Use I/O buffering for all sequential I/O**
 - IOBUF is a library that intercepts standard I/O (stdio) and enables asynchronous caching and prefetching of sequential file access
 - No need to modify the source code but just
 - Load the module iobuf
 - Rebuild your application

Case Study: Buffered I/O

- A post processing application writes a 1GB file.
- This occurs from one writer, but occurs in many small write operations.
 - Takes 1080 s (~ 18 minutes) to complete.
- IO buffers were utilized to intercept these writes with 4 64 MB buffers.
 - Takes 4.5 s to complete. A 99.6% reduction in time.

```

File "ssef_cn_2008052600f000"
      Calls      Seconds      Megabytes      Megabytes/sec      Avg Size
Open              1      0.001119
Read             217      0.247026      0.105957      0.428931      512
Write          2083634      1.453222     1017.398927     700.098632      512
Close              1      0.220755
Total          2083853      1.922122     1017.504884     529.365466      512
Sys Read           6      0.655251      384.000000     586.035160     67108864
Sys Write         17      3.848807     1081.145508     280.904052     66686072
Buffers used           4 (256 MB)
Prefetches         6
Preflushes        15
  
```



MPI I/O hints

- The `MPICH_MPIIO_HINTS` variable specifies *hints* to the MPI-IO library that can, for instance, override the built-in heuristic and force collective buffering on:
- `setenv MPICH_MPIIO_HINTS="*:romio_cb_write=enable:romio_ds_write=disable"`
 - Placing this command in your batch file before calling your executable will cause your program to use these hints.
 - The `*` indicates that the hint applies to any file opened by MPI-IO,
- `MPICH_MPIIO_HINTS_DISPLAY=1` will dump a summary of the current MPI-IO hints to `stderr` each time a file is opened.
 - Useful for debugging and as a sanity check against spelling errors in your hints.
- Full list and description of MPI-IO hint is available from the `intro_mpi` man page.

I/O Best Practices

- **Read small, shared files from a single task**
 - Instead of reading a small file from every task, it is advisable to read the entire file from one task and broadcast the contents to all other tasks.
- **Limit the number of files within a single directory**
 - Incorporate additional directory structure
 - Set the Lustre stripe count of such directories which contain many small files to 1. (default on Shaheen)
- **Place small files on single OSTs**
 - If only one process will read/write the file and the amount of data in the file is small (< 1 MB to 1 GB) , performance will be improved by limiting the file to a single OST on creation.
 - ➔ This can be done as shown below by: `# lfs setstripe PathName -s 1m -i -1 -c 1` (default on Shaheen)

I/O Best Practices (2)

- **Place directories containing many small files on single OSTs**
 - If you are going to create many small files in a single directory, greater efficiency will be achieved if you have the directory default to 1 OST on creation

→# Ifs setstripe DirPathName -s 1m -i -1 -c 1 (default on Shaheen)
- **Avoid opening and closing files frequently**
 - Excessive overhead is created.
- **Use ls -l only where absolutely necessary**
 - Consider that “ls -l” must communicate with every OST that is assigned to a file being listed and this is done for every file listed; and so, is a very expensive operation. It also causes excessive overhead for other users. "ls" or "Ifs find" are more efficient solutions.
- **Consider available I/O middleware libraries**
 - For large scale applications that are going to share large amounts of data, one way to improve performance is to use a middleware library; such as ADIOS, HDF5, or MPI-IO.

I/O Best Practices (3)

- **Reduce I/O as much as possible:**
 - only relevant data must be stored on disks
 - Save data in binary/unformatted form
 - asks for less space comparing with ASCII/formatted ones } It is faster (less OS interaction)
 - Save only what is necessary to save for restart or checkpointing, everything else, unless for debugging reason or quality check, should be computed on the fly .
- Think parallel for I/O like for your computations
- A bad behaving application hurts not only itself but ALL running applications!

More docs and refs.

- <http://www.mcs.anl.gov/research/projects/darshan>
- <https://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips>
- http://researchcomputing.github.io/meetup_fall_2014/pdfs/fall2014_meetup10_lustre.pdf
- http://www.nas.nasa.gov/hecc/support/kb/lustre-best-practices_226.html
- http://docs.cray.com/cgi-bin/craydoc.cgi?mode=View;id=S-2490-40;idx=books_search;this_sort=release_date%20desc;q=getting%20started;type=books;title=Getting%20Started%20on%20MPI%20I/O

Thank You !