Parallel I/O

International HPC Summer School

July 11, 2018

Elsa Gonsiorowski
HPC I/O Specialist, LLNL

LLNL-PRES-751922
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Outline

Motivation

I/O in Parallel

- Step 1: Recognize a need
- Step 2: Existing I/O Libraries and Tools
- Step 3: I/O Patterns
- Step 4: Understand the File System
- Step 6: Profit

Technical Details: MPI I/O

Pro-Tips!

Motivation

Types of I/O

Input

- Launching an executable & it’s linked libraries
- Reading configuration file
- Loading data files

Types of I/O

Input

- Launching an executable & it's linked libraries
- Reading configuration file
- Loading data files

Output

- Checkpoints
- Results

Types of I/O

Input
- Launching an executable & it's linked libraries
- Reading configuration file
- Loading data files

Output
- Checkpoints
- Results

Science
- Moving files from one machine to another
- Cleaning up after experiments

Types of I/O

Input
- Launching an executable & it's linked libraries
- Reading configuration file
- Loading data files

Output
- Checkpoints
- Results

Science
- Moving files from one machine to another
- Cleaning up after experiments

Everyone interacts with a file system therefore everyone does I/O!

Why should I care?

Data movement is expensive and must be optimized

Why should I care?

Data movement is expensive and must be optimized

$$\text{Total execution time} = \text{Computation time}$$

Why should I care?

Data movement is expensive and must be optimized

$$\text{Total execution time} = \text{Computation time} + \text{Communication time}$$

Why should I care?

Data movement is expensive and must be optimized

$$\begin{aligned}\text{Total execution time} &= \text{Computation time}\\ &+ \text{Communication time}\\ &+ \text{I/O time}\end{aligned}$$

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s
- CPU Memory (DDR4): 120 GB/s

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s
- CPU Memory (DDR4): 120 GB/s
- Node-local storage or `/tmp` (SSD): 1.1 GB/s

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s
- CPU Memory (DDR4): 120 GB/s
- Node-local storage or `/tmp` (SSD): 1.1 GB/s
- PFS (HDD + SSD + Magic): 40 GB/s

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s
- CPU Memory (DDR4): 120 GB/s
- Node-local storage or `/tmp` (SSD): 1.1 GB/s
- PFS (HDD + SSD + Magic): 40 GB/s
  - burst buffer

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s
- CPU Memory (DDR4): 120 GB/s
- Node-local storage or `/tmp` (SSD): 1.1 GB/s
- PFS (HDD + SSD + Magic): 40 GB/s
  - burst buffer
  - "project" storage

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s
- CPU Memory (DDR4): 120 GB/s
- Node-local storage or `/tmp` (SSD): 1.1 GB/s
- PFS (HDD + SSD + Magic): 40 GB/s
  - burst buffer
  - "project" storage
  - "campaign store"

HPC Storage Stack

- GPU Memory (HBM2): 900 GB/s
- CPU Memory (DDR4): 120 GB/s
- Node-local storage or `/tmp` (SSD): 1.1 GB/s
- PFS (HDD + SSD + Magic): 40 GB/s
  - burst buffer
  - "project" storage
  - "campaign store"
- HPSS (Tape + Robots): 0.2 GB/s

HPC Storage Stack

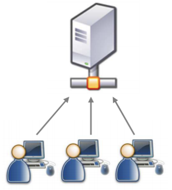
- GPU Memory (HBM2): 900 GB/s per GPU
- CPU Memory (DDR4): 120 GB/s per socket
- Node-local storage (SSD): 1.1 GB/s per node
- PFS (HDD + SSD + Magic): 40 GB/s shared by a system
  - burst buffer
  - "project" storage
  - "campaign store"
- HPSS (Tape + Robots): 0.2 GB/s shared by a center

File Systems

Laptop

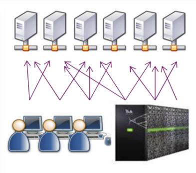
1 user
1.1 GB/s

Network File System (NFS)



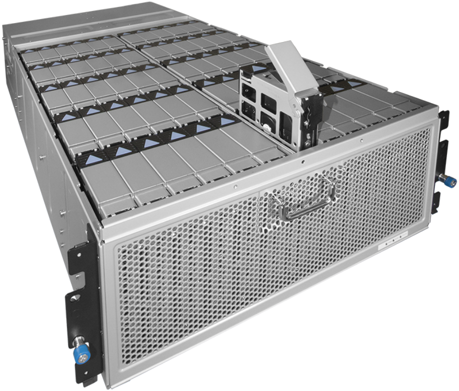
m servers, *n* clients
home directory
2 GB/s throughput
280K IOPS

Parallel File System (PFS)

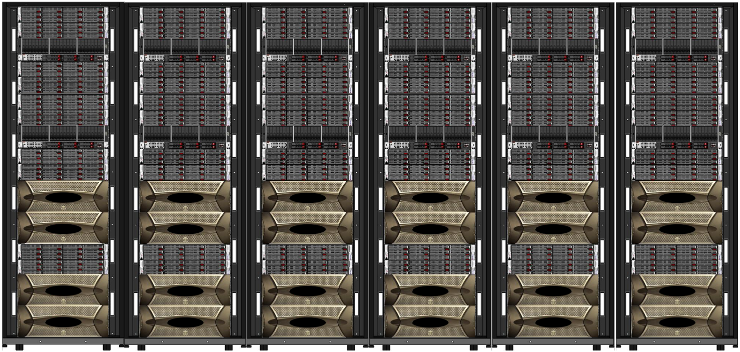


Used by HPC jobs
System specific
scratch or project storage
40 GB/s throughput
Millions of IOPS

Parallel File System



Parallel File System



Parallel File System

I/O in Parallel

Steps for Dealing with I/O

1. Recognize the need

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*
   - Deal with files efficiently

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*
   - Deal with files efficiently
2. Investigate I/O libraries and tools, one may be common in your field.

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*
   - Deal with files efficiently
2. Investigate I/O libraries and tools, one may be common in your field.
3. Implement an I/O pattern

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*
   - Deal with files efficiently
2. Investigate I/O libraries and tools, one may be common in your field.
3. Implement an I/O pattern
4. Understand the file system you are working on

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*
   - Deal with files efficiently
2. Investigate I/O libraries and tools, one may be common in your field.
3. Implement an I/O pattern
4. Understand the file system you are working on
5. ???

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*
   - Deal with files efficiently
2. Investigate I/O libraries and tools, one may be common in your field.
3. Implement an I/O pattern
4. Understand the file system you are working on
5. ???
6. Profit!

Step 1: Recognize a need

Profiling

- Darshan
- Tau

Profiling

- Darshan
- Tau

Attend tomorrow’s performance analysis session!

Step 2: Existing Libraries + Tools

Parallel I/O Libraries and Tools

Reading & Writing Files:

- HDF5
- PnetCDF
- Others: ADIOS, TyphonIO,SILO
- MPI-IO

Managing Files:

- Spindle
- mpiFileUtils
- SCR

Library: HDF5

- Hierarchical Data Format
- File-system in a file
- Datasets: multidimensional arrays of a homogeneous type
- Groups: container structures which can hold datasets and other groups
- Official support for C, C++, Fortran 77, Fortran 90, Java
- Implementations in R, Perl, Python, Ruby, Haskell, Mathematica, MATLAB, etc.

Library: PNetCDF

- Built on netCDF and MPI-IO

netCDF:

- self-describing, machine-independent format
- designed for arrays of scientific data
- netCDF is implemented in C, C++, Fortran 77, Fortran 90, Java, R, Perl, Python, Ruby, Haskell, Mathematica, MATLAB, etc.

Library: MPI-IO

- API for interacting with files with MPI concepts
  - blocking vs. non-blocking
  - collective vs. non-collective
- Lower level than other libraries
- Fine-grain control of files and offsets
- C and Fortran interfaces
- Separate effort from regular MPI

Tool: Spindle

- Scalable dynamic library and Python loading
- Caches linked libraries
- Life saver for NFS issues

`https://github.com/hpc/spindle`

Tool: mpiFileUtils

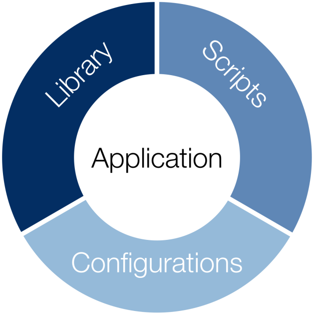
Use parallel processes to perform file operations

- Executed within a job allocation
- `dbcast`: broadcast a file from PFS to node-local storage
- `dcp`: copy multiple file in parallel
- `drm`: delete files in parallel
- *many more*

`https://github.com/hpc/mpifileutils`

Library: SCR

- Scalable Checkpoint Restart
- Enable checkpointing applications to take advantage of system storage hierarchies
- Efficient file movement between storage layers
- Data redundancy operations



Step 3: I/O Patterns

Parallel I/O Patterns

- Single file, accessed by 1 task

Parallel I/O Patterns

- Single file, accessed by 1 task
- Single shared file, accessed by all tasks

Parallel I/O Patterns

- Single file, accessed by 1 task
- Single shared file, accessed by all tasks
- Many shared files, accessed by groups of tasks

Parallel I/O Patterns

- Single file, accessed by 1 task
- Single shared file, accessed by all tasks
- Many shared files, accessed by groups of tasks
  - Baton-passing

Parallel I/O Patterns

- Single file, accessed by 1 task
- Single shared file, accessed by all tasks
- Many shared files, accessed by groups of tasks
  - Baton-passing
  - Coordinated "View"

Parallel I/O Patterns

- Single file, accessed by 1 task
- Single shared file, accessed by all tasks
- Many shared files, accessed by groups of tasks
  - Baton-passing
  - Coordinated "View"
- Many independent files, accessed by a subset of tasks

Parallel I/O Patterns

- Single file, accessed by 1 task
- Single shared file, accessed by all tasks
- Many shared files, accessed by groups of tasks
  - Baton-passing
  - Coordinated "View"
- Many independent files, accessed by a subset of tasks
- One file per process

Step 4: Understand the PFS

Parallel File System Policies

- Allocation: how much space you have

Parallel File System Policies

- Allocation: how much space you have
- Backups: if backups or snapshots are created

Parallel File System Policies

- Allocation: how much space you have
- Backups: if backups or snapshots are created
- Purges: when data is deleted

Parallel File System Policies

- Allocation: how much space you have
- Backups: if backups or snapshots are created
- Purges: when data is deleted
- Configuration: I/O pattern system is configured for

Parallel File Systems

- Black Magic: IBM's GPFS (general parallel file system)
  - Closed source
  - aka *Elastic Scale Storage*™ or *Spectrum Scale*™
  - HPC users do not have knobs to tune

Parallel File Systems

- Black Magic: IBM's GPFS (general parallel file system)
  - Closed source
  - aka *Elastic Scale Storage*™ or *Spectrum Scale*™
  - HPC users do not have knobs to tune
- White Magic: Lustre
  - Open source
  - Users can deviate from default behavior

Lustre Striping

- HDDs are logically grouped into OSTs (Object Storage Targets)
- Users can *stripe* a file across multiple OSTs
  - Explicitly take advantage of multiple OSTs
  - Depends on the total amount of I/O you are doing
  - There is a system default
- Use the correct striping for your use case

Lustre Striping Commands

```
$ lfs setstripe -c 4 -s 4M testfile2
$ lfs getstripe ./testfile2
./testfile2
lmm_stripe_count: 4
lmm_stripe_size: 4194304
lmm_stripe_offset: 21
    obdidx       objid       objid     group
        50     8916056    0x880c58         0
        38     8952827    0x889bfb         0
```

Lustre Striping Commands

```
$ lfs getstripe ./testfile
./testfile
lmm_stripe_count: 2
lmm_stripe_size: 1048576
lmm_stripe_offset: 50
    obdidx       objid        objid     group
        21     8891547     0x87ac9b         0
        13     8946053     0x888185         0
        57     8906813     0x87e83d         0
        44     8945736     0x888048         0
```

Step 6: Profit

Steps for Dealing with I/O

1. Recognize the need
   - Get some data out of the application
   - Get some data out of the application *faster*
   - Deal with files efficiently
2. Investigate I/O libraries and tools, one may be common in your field.
3. Implement an I/O pattern
4. Understand the file system you are working on
5. ???
6. Profit!

Technical Details: MPI I/O

Locking and Atomicity

```
$ export BGLOCKLESSMPIO_F_TYPE=1

int MPI_File_set_atomicity ( MPI_File mpi_fh, int flag );
```

Opening Files

```
int MPI_File_open(MPI_Comm comm, const char *filename,
                  int amode, MPI_Info info, MPI_File *fh);
```

AMode	Description
MPI_MODE_RDONLY	read only
MPI_MODE_RDWR	reading and writing
MPI_MODE_WRONLY	write only
MPI_MODE_CREATE	create the file
MPI_MODE_EXCL	error if file already exists
MPI_MODE_DELETE_ON_CLOSE	delete file on close
MPI_MODE_UNIQUE_OPEN	file will not be concurrently opened
MPI_MODE_SEQUENTIAL	file will only be accessed sequentially
MPI_MODE_APPEND	position of all file pointers to end

Organizing Data

- Use `MPI_Datatype` to define the structure of your data
- Corresponds to C `struct`
- Read and write instances of this data
- Use `MPI_File_set_view` for working with non-contiguous data in a shared file

Useful MPI Function

```
offset = (long long) 0;
MPI_Exscan(&contribute, &offset, 1, MPI_LONG_LONG,
           MPI_SUM, file_comm);
```

Useful MPI Function

```
offset = (long long) 0;
MPI_Exscan(&contribute, &offset, 1, MPI_LONG_LONG,
           MPI_SUM, file_comm);
```

Rank	0	1	2	3	4
contribute	3	4	2	7	3

Useful MPI Function

```
offset = (long long) 0;
MPI_Exscan(&contribute, &offset, 1, MPI_LONG_LONG,
           MPI_SUM, file_comm);
```

Rank	0	1	2	3	4
contribute	3	4	2	7	3
offset	0	3	7	9	16

Accessing Files with MPI

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_IREAD_AT_ALL MPI_FILE_IWRITE_AT_ALL
	split collective	N/A	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_IREAD_ALL MPI_FILE_IWRITE_ALL
	split collective	N/A	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	N/A
	split collective	N/A	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

Accessing Files with MPI

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_IREAD_AT_ALL MPI_FILE_IWRITE_AT_ALL
	split collective	N/A	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_IREAD_ALL MPI_FILE_IWRITE_ALL
	split collective	N/A	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	N/A
	split collective	N/A	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

Accessing Files with MPI

Level 0

independent file ops, explicit offset, sequential data

Level 1

collective file ops, explicit offset, sequential data

Level 2

independent file ops, derived or non-contiguous data

Level 3

collective file ops, derived or non-contiguous data

MPI I/O & Lustre

- Can be built by HPC resource providers with Lustre integration

```
mpi_info_set(myinfo, "striping_factor", stripe_count);
mpi_info_set(myinfo, "striping_unit", stripe_size);
mpi_info_set(myinfo, "cb_nodes", num_writers);
```

Pro-Tips!

Pro-Tip!

Step One

Profile your code. Fix up the I/O until it doesn't suck.

Pro-Tip!

Be Smart

Don’t re-invent I/O, use an existing library or tool.

Pro-Tip!

Working with File Systems

Use the PFS for Parallel I/O, do NOT use NFS.

Pro-Tip!

I/O Pattern
Create 1 file per node and make this a tune-able parameter.

Pro-Tip!

Ask an Expert
Find the "I/O person" at your HPC center and ask for guidance.