Introduction to Parallel I/O

Ritu Arora and Si Liu

with contributions from

Robert McLay, John Cazes, and Doug James

Texas Advanced Computing Center

April 16, 2015

Email: {rauta, siliu}@tacc.utexas.edu

Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- Lab Session 1
- Break
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies
- Lab-Session2

I/O in HPC Applications

- High Performance Computing (HPC) applications often
  - Read initial conditions or datasets for processing
  - Write numerical data from simulations
    - Saving application-level checkpoints
- In case of large distributed HPC applications, the total execution time can be broken down into the computation time, communication time, and the I/O time
- Optimizing the time spent in computation, communication and I/O can lead to overall improvement in the application performance
- However, doing efficient I/O without stressing out the HPC system is challenging and often an afterthought

Addressing the I/O Bottlenecks

- Avail the software support for parallel I/O that is available in the form of
  - Parallel distributed file systems that provide parallel data paths to storage disks
  - MPI I/O
  - Libraries like PHDF5, pNetCDF
  - High-level libraries like T3PIO

- Understand the I/O strategies for maintaining good citizenship on a supercomputing resource

Some Examples of Parallel File Systems

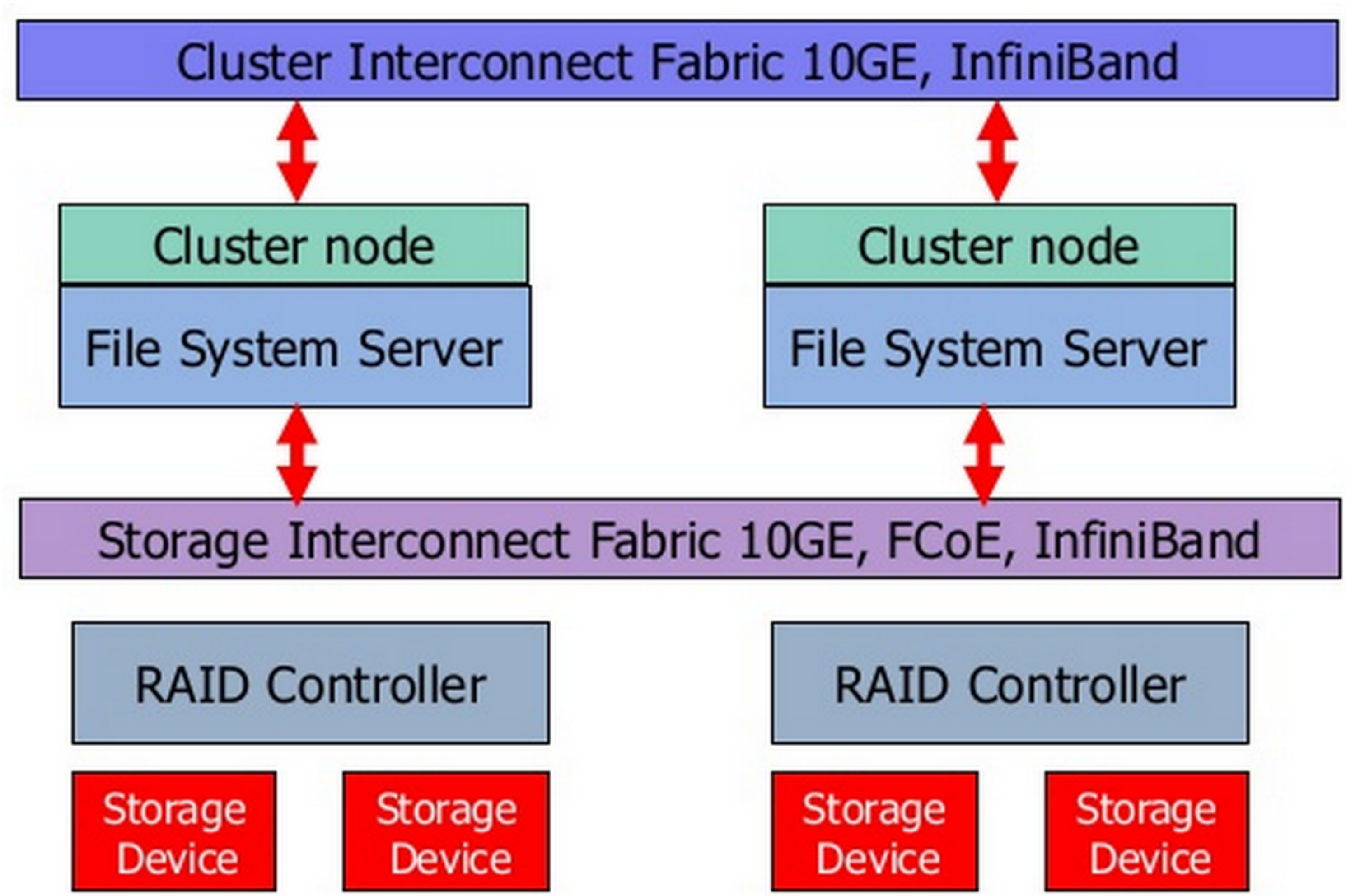
- General Parallel File System (GPFS)
  - Now rolled into IBM's Spectrum Scale product
  - Multiple topologies: direct-attached storage, network-attached storage, and hybrid
- Lustre File System

Other Parallel File Systems

- Panasas Parallel File system (PanFS)
- Parallel Virtual File System (PVFS)

GPFS Topology 1

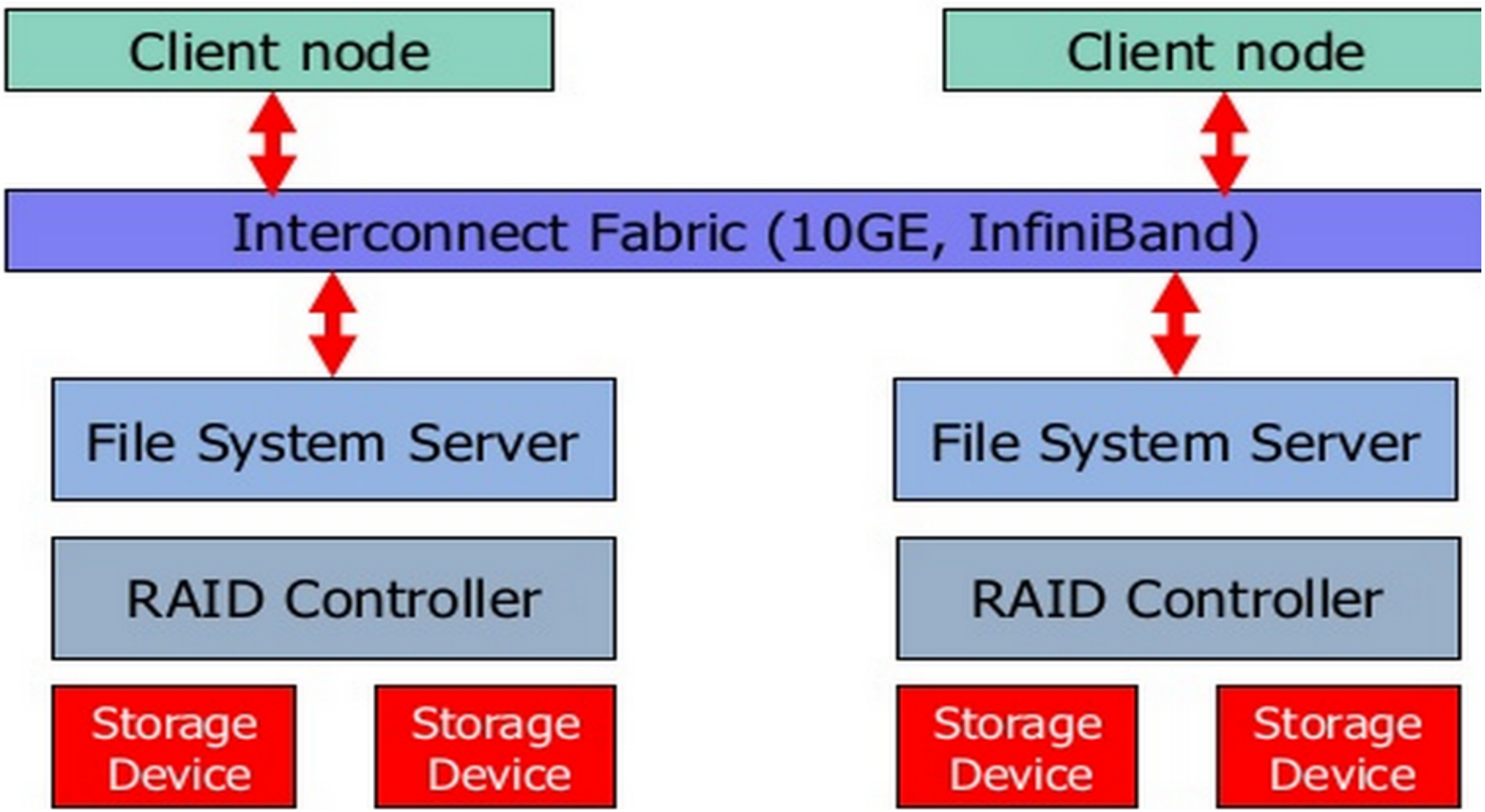
Direct Attached Storage



Source: http://www.slideshare.net/GabrielMateescu/sonas-44390281

GPFS Topology 2

Network Attached Storage

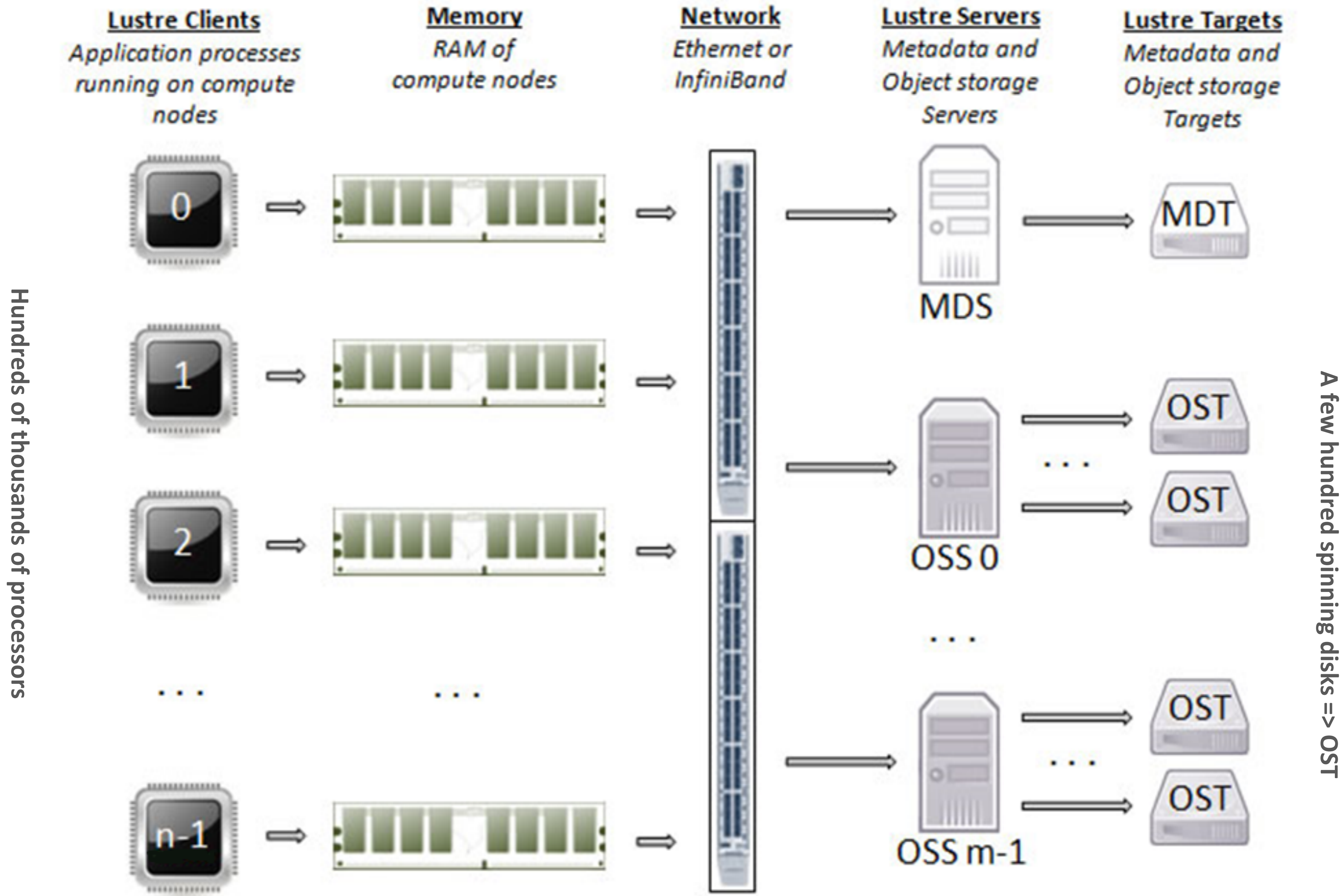


Source: http://www.slideshare.net/GabrielMateescu/sonas-44390281

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Lustre File System



Source: Reference 2, 4

GPFS versus Lustre

	GPFS	Lustre
MDS	In direct-attached storage topology, all nodes acts like MDS, whereas in network-attached topology, some nodes (server nodes) act like MDS	Often 1 primary + 1 failover; since version 2.4, supported for clustered MDS is available
Storage Type	RAID, SAN, …	RAID, SAN, …
User Control for Tuning	None; optimized by administrators at the time of installation	User can change some parameters like stripe size and stripe count
Daemon Communication	TCP/IP	Portal
License	Proprietary (IBM product)	Open-Source

Source: Reference 6

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

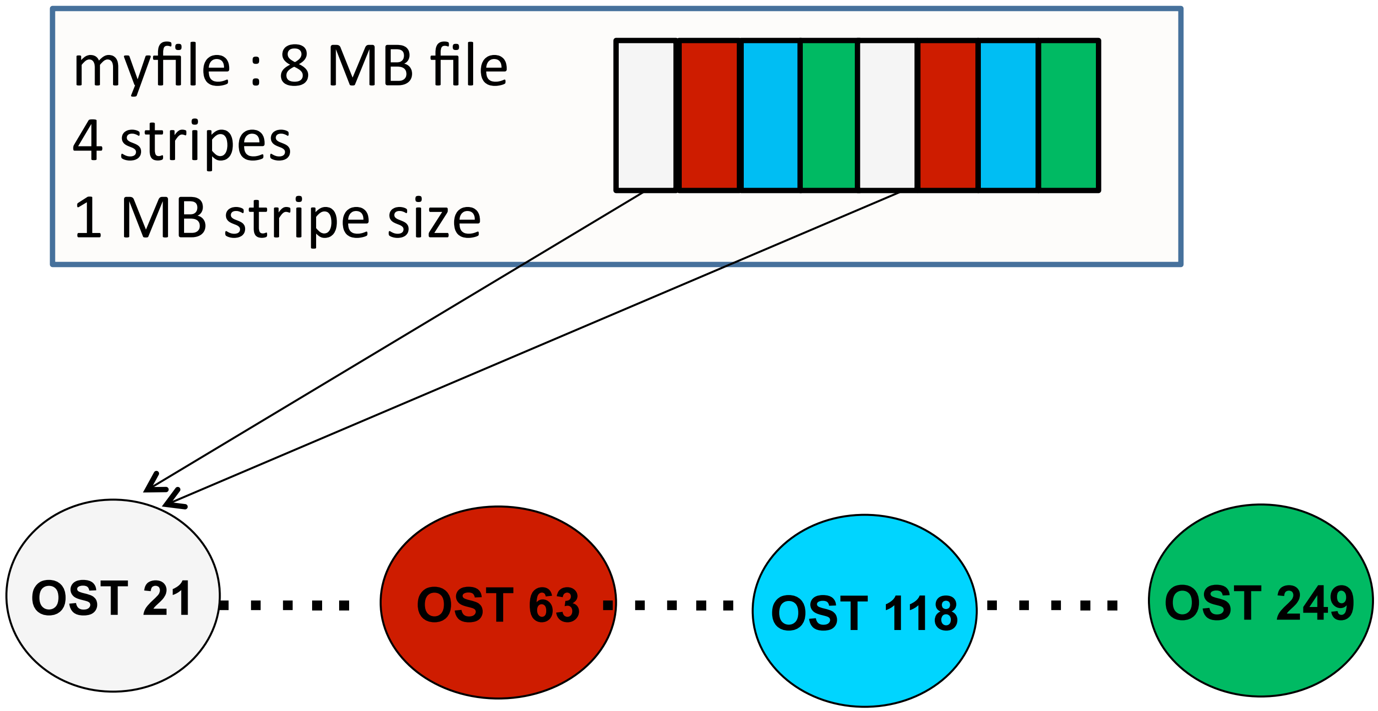
Lustre File System at TACC

- Each Lustre file system has a different number of OSTs
- The greater the number of OSTs the better the I/O capability

	$HOME	**$WORK**	**$SCRATCH**
Stampede	24	672	348
Lonestar	N/A (NFS)	30	90

Lustre File System - Striping

- Lustre supports the striping of files across several I/O servers (similar to RAID 0)
- Each stripe is a fixed size block



Lustre File System – Striping on TACC Resources

- Administrators set a default stripe count and stripe size that applies to all newly created files
  - Stampede: `$SCRATCH`: 2 stripes/1MB
    `$WORK`: 1 stripe /1MB
  - Lonestar: `$SCRATCH`: 2 stripes/1MB
    `$WORK`: 1 stripe /1MB
- However, users can reset the default stripe count or stripe size using the Lustre commands

Lustre Commands

- Get stripe count

```
 % lfs getstripe ./testfile
./testfile
lmm_stripe_count:   2
lmm_stripe_size:    1048576
lmm_stripe_offset:  50
        obdidx           objid          objid           group
            50         8916056       0x880c58               0
            38         8952827       0x889bfb               0
```

- Set stripe count

```
% 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
            21         8891547       0x87ac9b               0
            13         8946053       0x888185               0
            57         8906813       0x87e83d               0
            44         8945736       0x888048               0
```

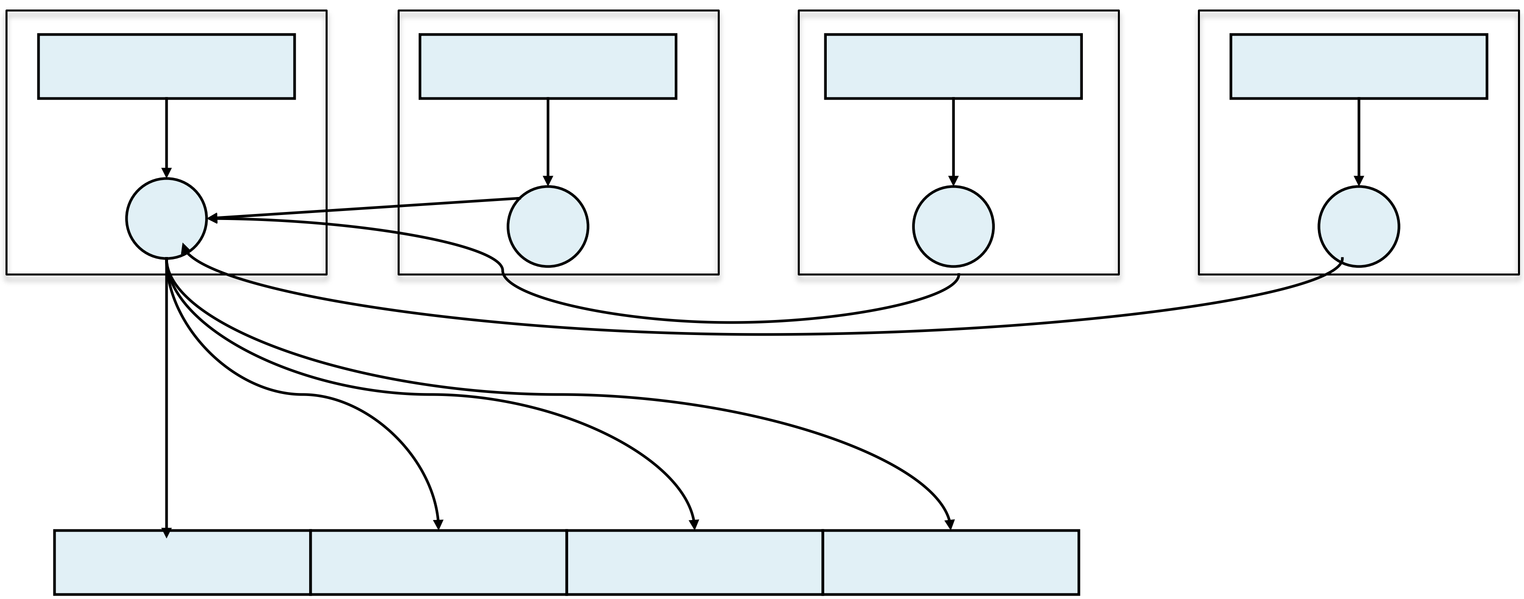
Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- MPI I/O Example – Distributing Arrays
- Lab Session 1
- Break
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies
- Lab-Session2

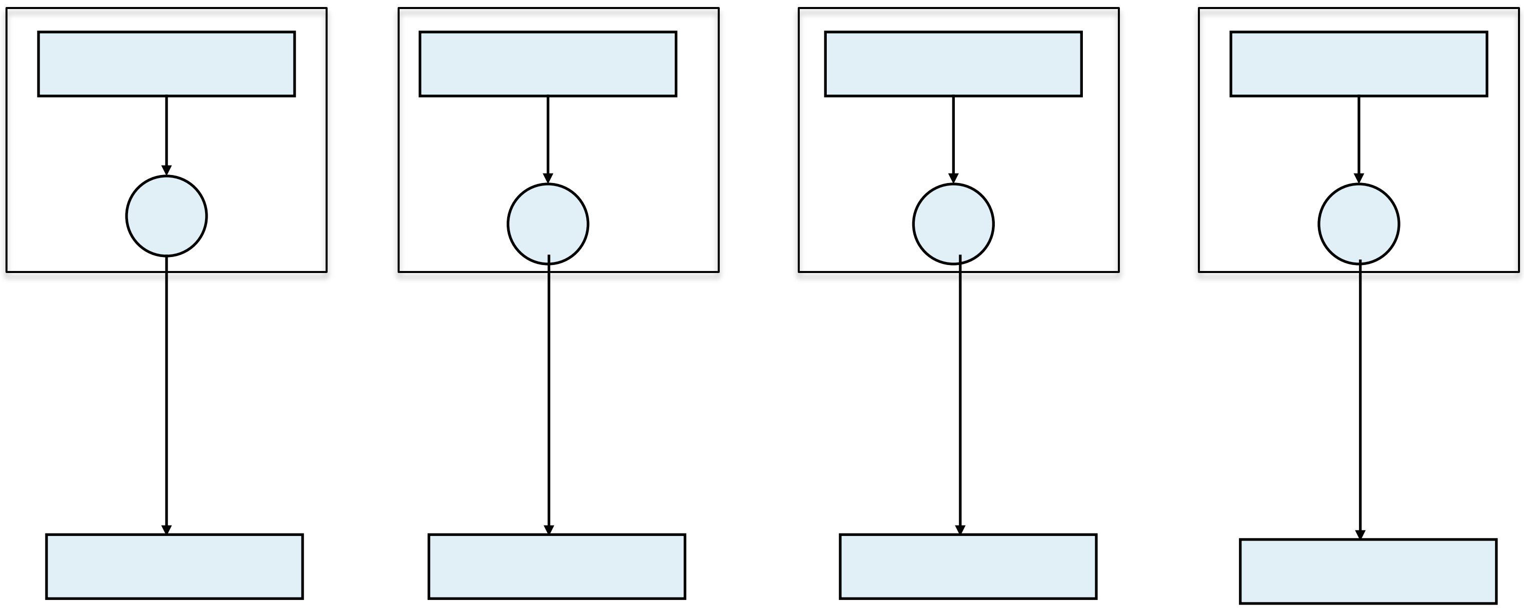
Typical Pattern: Parallel Programs Doing Sequential I/O

- All processes send data to master process, and then the process designated as master writes the collected data to the file

- This sequential nature of I/O can limit performance and scalability of many applications

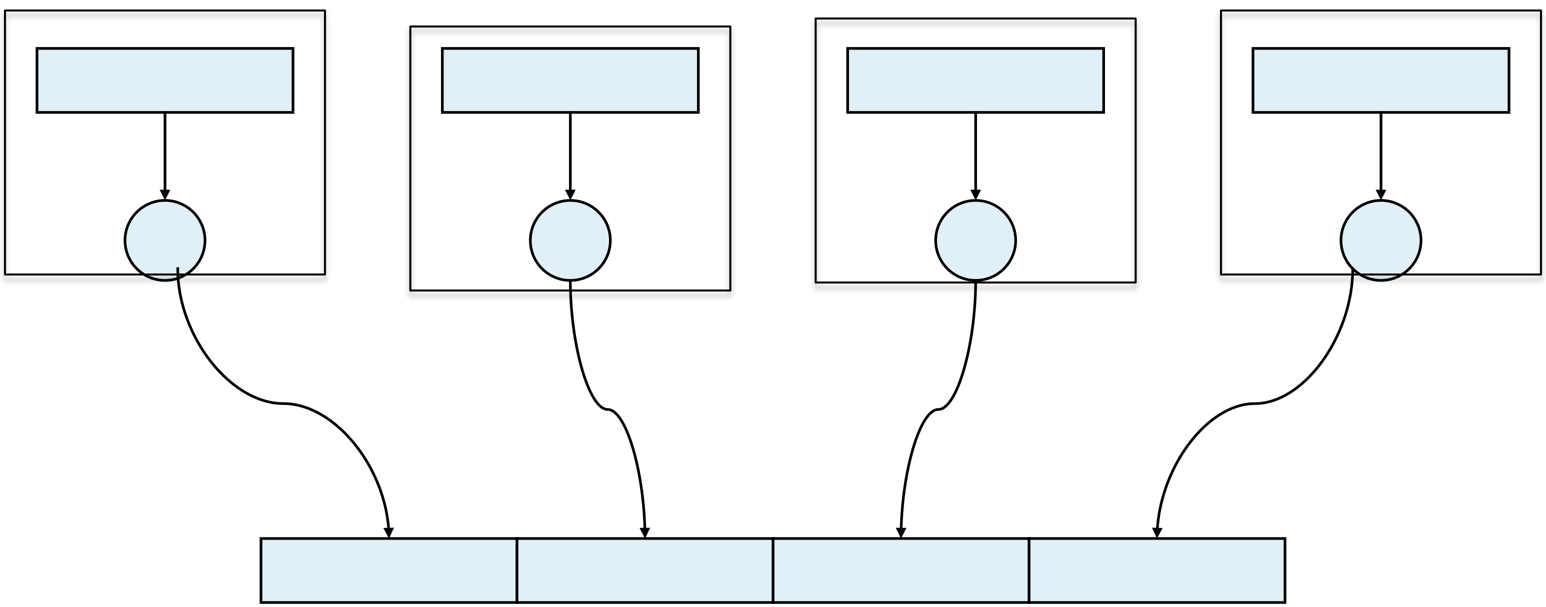


Another Pattern: Each Process Writing to a Separate File



Desired Pattern: Parallel Programs Doing Parallel I/O

- Multiple processes participating in reading data from or writing data to a common file in parallel
- This strategy improves performance and provides a single file for storage and transfer purposes



Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- Lab Session 1
- Break
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies

Need for High-Level Support for Parallel I/O

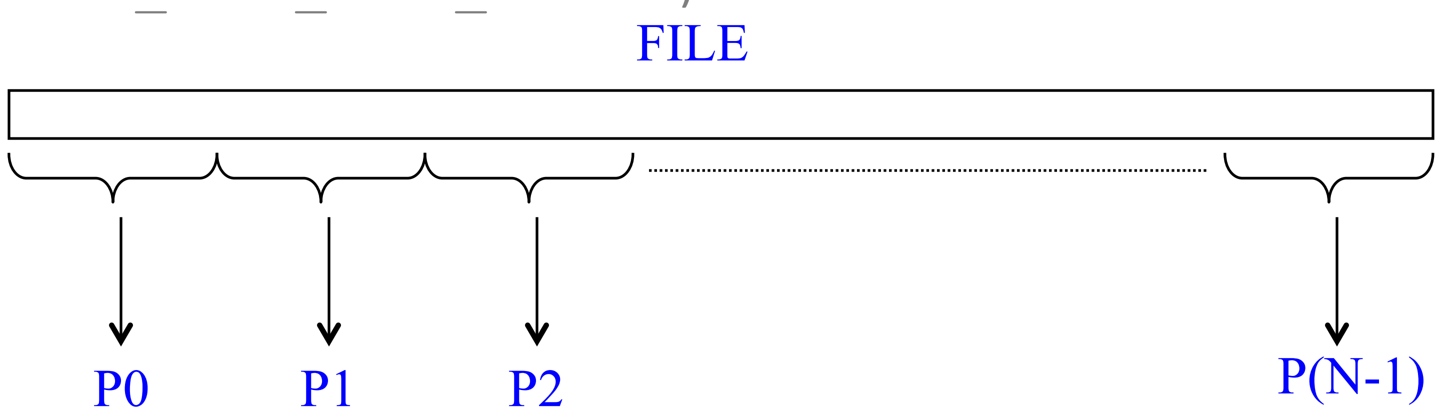
- Parallel I/O can be hard to coordinate and optimize if working directly at the level of Lustre API or POSIX I/O Interface (not discussed in this tutorial)
- Therefore, specialists implement a number of intermediate layers for coordination of data access and mapping from application layer to I/O layer
- Hence, application developers only have to deal with a high-level interface built on top of a software stack, that in turn sits on top of the underlying hardware
  - MPI-I/O, parallel HDF5, parallel netCDF, T3PIO,…

MPI for Parallel I/O

- A parallel I/O system for distributed memory architectures will need a mechanism to specify collective operations and specify noncontiguous data layout in memory and file
- Reading and writing in parallel is like receiving and sending messages
- Hence, an MPI-like machinery is a good setting for Parallel I/O (think MPI communicators and MPI datatypes)
- MPI-I/O featured in MPI-2 which was released in 1997, and it interoperates with the file system to enhance I/O performance for distributed-memory applications

Using MPI-I/O

- Given N number of processes, each process participates in reading or writing a portion of a common file
- There are three ways of positioning where the read or write takes place for each process:
  - Use individual file pointers (*e.g.,* `MPI_File_seek`/`MPI_File_read`)
  - Calculate byte offsets (*e.g.,* `MPI_File_read_at`)
    - Explicit offset operations perform data access at the file position given directly as an argument — no file pointer is used nor updated
  - Access a shared file pointer (*e.g.,* `MPI_File_seek_shared`, `MPI_File_read_shared`)



Source: Reference 3

MPI-I/O API Opening and Closing a File

- Calls to the MPI functions for reading or writing must be preceded by a call to `MPI_File_open`
  - `int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)`
- The parameters below are used to indicate how the file is to be opened

MPI_File_open mode	**Description**
MPI_MODE_RDONLY	read only
MPI_MODE_WRONLY	write only
MPI_MODE_RDWR	read and write
MPI_MODE_CREATE	create file if it doesn't exist

- To combine multiple flags, use bitwise-or "|" in C, or addition "+" in Fortran
- Close the file using: `MPI_File_close(MPI_File fh)`

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

MPI-I/O API for Reading Files

After opening the file, read data from files by either using `MPI_File_seek` & `MPI_File_read` Or `MPI_File_read_at`

int MPI_File_seek(MPI_File *fh*, **MPI_Offset** *offset*,
int *whence* **)**

int **MPI_File_read(**MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status**)**

whence in **MPI_File_seek** updates the individual file pointer according to

`MPI_SEEK_SET`: the pointer is set to offset

`MPI_SEEK_CUR`: the pointer is set to the current pointer position plus offset

`MPI_SEEK_END`: the pointer is set to the end of file plus offset

int **MPI_File_read_at**(MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

Reading a File: readFile2.c

```
#include<stdio.h>
#include "mpi.h"
#define FILESIZE 80
int main(int argc, char **argv){
  int rank, size, bufsize, nints;
  MPI_File fh;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  bufsize = FILESIZE/size;
  nints = bufsize/sizeof(int);
  int buf[nints];
  MPI_File_open(MPI_COMM_WORLD,"dfile",MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
  MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
  MPI_File_read(fh, buf, nints, MPI_INT, &status);
  printf("\nrank: %d, buf[%d]: %d", rank, rank*bufsize, buf[0]);
  MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

Reading a File: readFile2.c

```
#include<stdio.h>
#include "mpi.h"
#define FILESIZE 80
int main(int argc, char **argv){
  int rank, size, bufsize, nints;
  MPI_File fh;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  bufsize = FILESIZE/size;
  nints = bufsize/sizeof(int);
  int buf[nints];
  MPI_File_open(MPI_COMM_WORLD,"dfile",MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
  MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
  MPI_File_read(fh, buf, nints, MPI_INT, &status);
  printf("\nrank: %d, buf[%d]: %d", rank, rank*bufsize, buf[0]);
  MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

Declaring a File Pointer

Calculating Buffer Size

Opening a File

File seek & Read

Closing a File

Reading a File: readFile1.c

```
#include<stdio.h>
#include "mpi.h"
#define FILESIZE 80
int main(int argc, char **argv){
  int rank, size, bufsize, nints;
  MPI_File fh;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  bufsize = FILESIZE/size;
  nints = bufsize/sizeof(int);
  int buf[nints];
  MPI_File_open(MPI_COMM_WORLD,"dfile",MPI_MODE_RDONLY,MPI_INFO_NULL,&fh);
  MPI_File_read_at(fh, rank*bufsize, buf, nints, MPI_INT, &status);
  printf("\nrank: %d, buf[%d]: %d", rank, rank*bufsize, buf[0]);
  MPI_File_close(&fh);
  MPI_Finalize();
  return 0;
}
```

Combining file seek & read in one step for thread safety in **MPI_File_read_at**

MPI-I/O API for Writing Files

- While opening the file in the write mode, use the appropriate flag/s in **`MPI_File_open`**: **`MPI_MODE_WRONLY`** Or **`MPI_MODE_RDWR`** and if needed, **`MPI_MODE_CREATE`**
- For writing, use **`MPI_File_set_view`** and **`MPI_File_write`** or **`MPI_File_write_at`**

```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp,
MPI_Datatype etype, MPI_Datatype filetype, char
*datarep, MPI_Info info)
```

```
int MPI_File_write(MPI_File fh, void *buf, int count,
MPI_Datatype datatype, MPI_Status *status)
```

```
int MPI_File_write_at(MPI_File fh, MPI_Offset offset,
void *buf, int count, MPI_Datatype datatype,
MPI_Status *status)
```

Writing a File: writeFile1.c (1)

```
#include<stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
  int i, rank, size, offset, nints, N=16 ;
  MPI_File fhw;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  int buf[N];
  for ( i=0;i<N;i++){
        buf[i] = i ;
  }
...
```

Writing a File: writeFile1.c (2)

```
offset = rank*(N/size)*sizeof(int);

MPI_File_open(MPI_COMM_WORLD, "datafile",
  MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_NULL, &fhw);

printf("\nRank: %d, Offset: %d\n", rank, offset);

MPI_File_write_at(fhw, offset, buf, (N/size),
  MPI_INT, &status);

MPI_File_close(&fhw);

MPI_Finalize();
return 0;
}
```

Compile & Run the Program on Compute Node

```
c401-204$ mpicc -o writeFile1 writeFile1.c
c401-204$ ibrun -np 4 ./writeFile1
```

TACC: Starting up job 1754636

TACC: Setting up parallel environment for MVAPICH2+mpispawn.

Rank: 0, Offset: 0

Rank: 1, Offset: 16

Rank: 3, Offset: 48

Rank: 2, Offset: 32

TACC: Shutdown complete. Exiting.

```
c401-204$ hexdump -v -e '7/4 "%10d "' -e '"\n"' datafile
```

0	1	2	3	0	1	2
3	0	1	2	3	0	1
2	3					

File Views for Writing to a Shared File

- When processes need to write to a shared file, assign regions of the file to separate processes using **`MPI_File_set_view`**

- File views are specified using a triplet - (*displacement*, *etype*, and *filetype*) – that is passed to **`MPI_File_set_view`**

  displacement = number of bytes to skip from the start of the file

  etype = unit of data access (can be any basic or derived datatype)

  filetype = specifies which portion of the file is visible to the process

-
```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp,
MPI_Datatype etype, MPI_Datatype filetype, char *datarep,
MPI_Info info)
```

- Data representation (datarep above) can be `native`, `internal`, or `external32`

Writing a File: writeFile2.c (1)

```c
#include<stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
  int i, rank, size, offset, nints, N=16;
  MPI_File fhw;
  MPI_Status status;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  int buf[N];
  for ( i=0;i<N;i++){
        buf[i] = i ;
  }
  offset = rank*(N/size)*sizeof(int);
...
```

Writing a File: writeFile2.c (2)

```
MPI_File_open(MPI_COMM_WORLD, "datafile3",
   MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_NULL,
   &fhw);
  printf("\nRank: %d, Offset: %d\n", rank,
   offset);
  MPI_File_set_view(fhw, offset, MPI_INT,
   MPI_INT, "native", MPI_INFO_NULL);
  MPI_File_write(fhw, buf, (N/size), MPI_INT,
   &status);
  MPI_File_close(&fhw);
  MPI_Finalize();
  return 0;
}
```

Compile & Run the Program on Compute Node

```
c402-302$ mpicc -o writeFile2 writeFile2.c
c402-302$ ibrun -np 4 ./writeFile2
```

TACC: Starting up job 1755476

TACC: Setting up parallel environment for MVAPICH2+mpispawn.

Rank: 1, Offset: 16

Rank: 2, Offset: 32

Rank: 3, Offset: 48

Rank: 0, Offset: 0

TACC: Shutdown complete. Exiting.

```
c402-302$ hexdump -v -e '7/4 "%10d "' -e '"\n"' datafile3
```

```
     0     1     2     3     0     1     2
     3     0     1     2     3     0     1
     2     3
```

Note about atomicity Read/Write

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

- Use this API to set the atomicity mode – 1 for true and 0 for false – so that only one process can access the file at a time
- When atomic mode is enabled, MPI-IO will guarantee sequential consistency and this can result in significant performance drop
- This is a collective function

Collective I/O (1)

- Collective I/O is a critical optimization strategy for reading from, and writing to, the parallel file system

- The collective read and write calls force all processes in the communicator to read/write data simultaneously and to wait for each other

- The MPI implementation optimizes the read/write request based on the combined requests of all processes and can merge the requests of different processes for efficiently servicing the requests

- This is particularly effective when the accesses of different processes are noncontiguous

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Collective I/O (2)

- The collective functions for reading and writing are:
  - `MPI_File_read_all`
  - `MPI_File_write_all`
  - `MPI_File_read_at_all`
  - `MPI_File_write_at_all`
- Their signature is the same as for the non-collective versions

MPI-I/O Hints

- MPI-IO hints are extra information supplied to the MPI implementation through the following function calls for improving the I/O performance
  - `MPI_File_open`
  - `MPI_File_set_info`
  - `MPI_File_set_view`

- Hints are optional and implementation-dependent
  - you may specify hints but the implementation can ignore them

- `MPI_File_get_info` used to get list of hints, examples of Hints: **`striping_unit, striping_factor`**

Lustre – setting stripe count in MPI Code

- MPI may be built with Lustre support
  - MVAPICH2 & OpenMPI support Lustre

- Set stripe count in MPI code
  Use MPI I/O hints to set Lustre stripe count, stripe size, and # of writers

  Fortran:
  ```
  call mpi_info_set(myinfo,"striping_factor",stripe_count,mpierr)
  call mpi_info_set(myinfo,"striping_unit",stripe_size,mpierr)
  call mpi_info_set(myinfo,"cb_nodes",num_writers,mpierr)
  ```

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

- Default:
  - # of writers = # Lustre stripes

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- Lab Session 1
- Break – for 15 minutes
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies
- Lab-Session2

Lab-Sessions: Goals & Activities

- You will learn
  - How to compile and execute MPI code on Stampede
  - How to do parallel I/O using MPI, HDF5, and T3PIO

- What will you do
  - Compile and execute the code for the programs discussed in the lecture and exercises
  - Modify the code for the exercises to embed the required MPI routines, or calls to high-level libraries

Accessing Lab Files

Please see the hand-out for the username (login name) and password

- Log on to Stampede using **your_login_name**

- Uncompress the
- file, **SEA2015.tgz** , that is located in the **~train00** directory into your HOME directory.

```
ssh <your_login_name>@stampede.tacc.utexas.edu

tar -xvzf ~train00/SEA2015.tgz

cd SEA2015/mpi
```

Please Note

- The project number for this tutorial is:

  SEA-Parallel-2015-04-16

- In the job submission script, provide the project number mentioned above (replace the "A-xxxxx" in the line "-A A-xxxxx" with the appropriate project number)

- The reservation name is SEA-Parallel-2015-04-16 and the queue to be used is **normal**

- Add the following line to your SLURM job-script

  #SBTACH --reservation SEA-Parallel-2015-04-16

Exercise 0 (if you are not familiar with Stampede)

- **Objective**: practice compiling and running MPI code on Stampede
- Compile the sample code `mpiExample4.c`

```
login3$ mpicc -o mpiExample4 mpiExample4.c
```

- Modify the job script, `myJob.sh`, to provide the name of the executable to the `ibrun` command
- Submit the job script to the SGE queue and check it's status

```
login3$ sbatch myJob.sh
```
(you will get a `job id`)

```
login3$ squeue
```
(check the status of your job)

- When your job has finished executing, check the output in the file `myMPI.o<job id>`

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Exercise 1

- **Objective**: **Learn to use MPI I/O calls**
- Modify the code in file `exercise1.c` in the subdirectory **`exercise`** within the directory **`SEA2015`**
  - Read the comments in the file for modifying the code
    - Extend the variable declaration section as instructed
    - You have to add MPI routines to open a file named "`datafile_written`", and to close the file
    - You have to fill the missing arguments of the routine **`MPI_File_write_at`**
  - See the lecture slides for details on the MPI routines
- Compile the code and execute it via the job script using 10 MPI processes (see Exercise 0 for the information related to compiling the code and the jobscript)

Exercise 2

- **Objective**: **Learn to use collective I/O calls**
- Modify the code in file `exercise2.c` in the subdirectory **`exercise`** within the directory **`SEA2015`**
  - Read the comments in the file for modifying the code
    - Use the `MPI_File_write_all` function in the specified place in the program
- Compile the code and execute it via the job script using 10 MPI processes (see Exercise 0 for the information related to compiling the code and the jobscript)

Viewing the output file

```
staff$ module swap intel/13.0.2.146 intel/14.0.1.106

staff$ srun -p development -A TG-ASC130034 -t 01:00:00 -n
16 --pty /bin/bash -l

c557-201$ mpicc -o exercise2 exercise2.c
c557-201$ ibrun -n 10 -o 0 exercise2
c557-201$ hexdump -v -e '7/4 "%10d "' -e '"\n"' datafile
         0          1          2          3          0
1          2
         3          0          1          2          3
0          1
         2          3
c557-201$
```

Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- Lab Session 1
- Break
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies
- Lab-Session2

HDF5: Hierarchical Data Format

HDF5 is a file format

- Managing any kind of data
- Software to manage data in the HDF5 format
- An HDF5 file can be viewed as a file system inside a file
- It uses a Unix style directory structure
- It is a mixture of entities: groups, datasets, and attributes
- Any entity can have descriptive attributes (metadata), e.g. physical units

HDF5 Nice Features

- Interface support for C, C++, Fortran, Java, and Python
- Supported by data analysis packages

  (Matlab, IDL, Mathematica, Octave, Visit, Paraview, Tekplot, etc.)
- Machine independent data storage format (self-describing)
- Supports user defined datatypes and metadata
- Read or write to a portion of a dataset (Hyperslab)
- Run on almost all systems

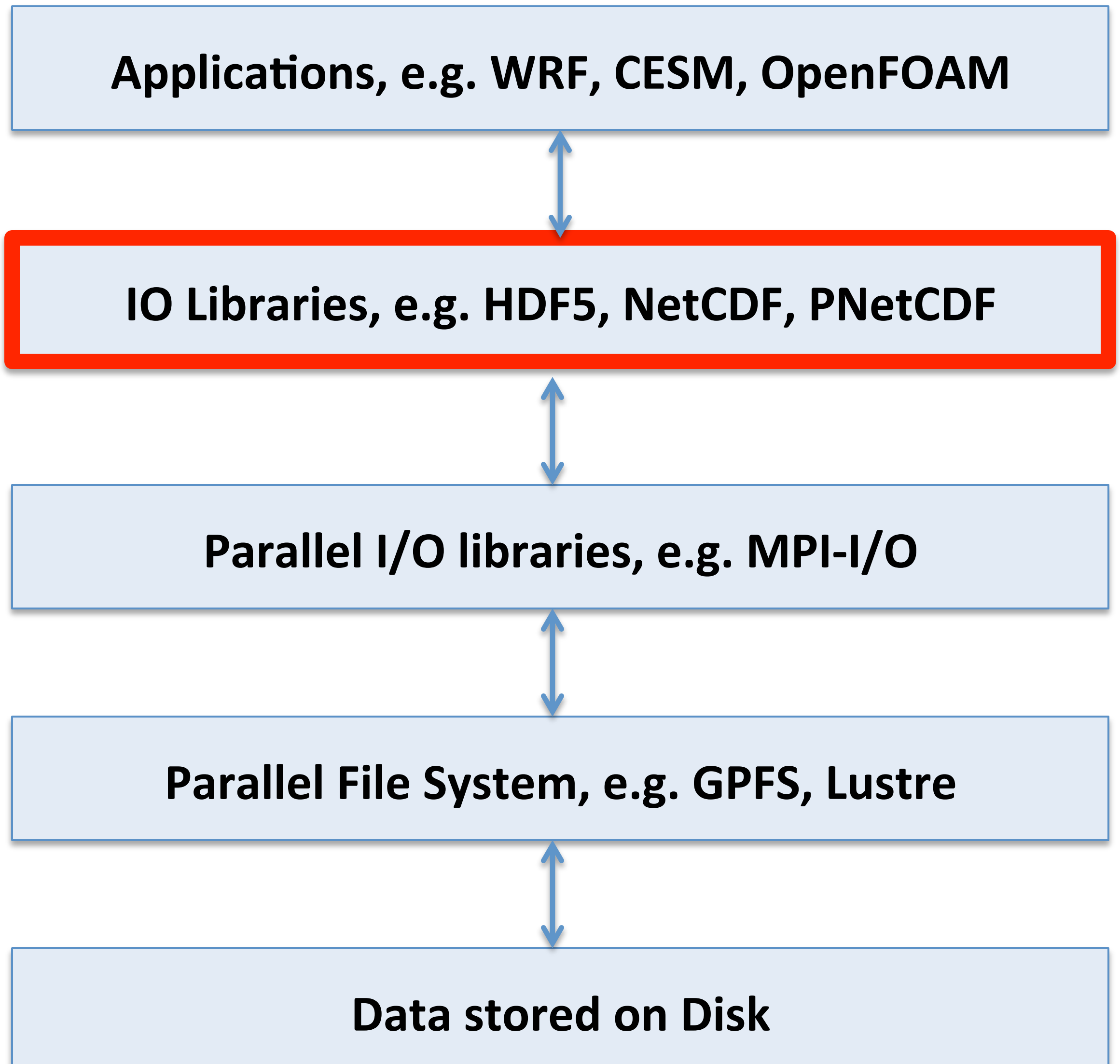
HDF5: The Benefits of Metadata

- It is easy to record many metadata items within a solution file
- Adding attributes later won't break any program that reads the data.
- With HDF5 it is easy to save with each solution file:
  - Computer Name, OS Version
  - Compiler and MPI name and version
  - Program Version
  - Physical unit
  - Etc.

PHDF5 Overview

- PHDF5 is the Parallel HDF5 library.
  - You can write one file in parallel efficiently
  - Parallel performance of HDF5 very close to MPI I/O
- Uses MPI I/O (Don't reinvent the wheel)
- MPI I/O techniques apply to HDF5
- Use MPI_Info object to control # writers, # stripes(Lustre), stripe size(Lustre), etc.

Overall Implementation Layers



Optimize HDF5 I/O Performance

- Only 1 file is opened → Efficient interaction with MDS.
- Every task calls HDF5 dataset write routines...
- ... but internally HDF5 and MPI move data to a small number of writer nodes (aggregators)
- We can control the number of writers, stripes and stripe size to tune I/O performance (MPI Info/T3PIO)

A Dump of a Simple HDF5 File

```
$ h5dump dset.h5
   HDF5 "dset.h5" {
   GROUP "/" {
      DATASET "T" {
      DATATYPE H5T_IEEE_F64LE
      DATASPACE SIMPLE { ( 10 ) / ( 10 ) }
      DATA {
            (0): 1.5, 1, 1.0625, 1.0625, 2.0625,
            (5): 1.4375, 1.4375, 0.625, 1.625, 1.625
      }
      ATTRIBUTE "Description" {
            DATASPACE SIMPLE { ( 1 ) / ( 1 ) }
            DATA {
                  (0): "thermal soln"
            }
      }
   . . .
```

Basic HDF5 Structure

- Open HDF5
- Open File
    - Open Group
        - Open Dataset
        - Write Dataset
        - Close Dataset
    - Close Group
- Close File
- Close HDF5

HDF5 Write: Simple Example

```
 … data prepared  …

// Open an existing file.
  file_id = H5Fopen(FILE, H5F_ACC_RDWR, H5P_DEFAULT);

  // Open an existing dataset.
  dataset_id = H5Dopen2(file_id, "/dset", H5P_DEFAULT);

  // Write the dataset.
  status = H5Dwrite(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
                                        H5P_DEFAULT, data);

//status = H5Dread(dataset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL,
                               H5P_DEFAULT, data);       //Read is similar

  // Close the dataset.
  status = H5Dclose(dataset_id);

  // Close the file.
  status = H5Fclose(file_id);
```

HDF5 Write: Another Example

```
// Set up file access property list with parallel I/O access
  plist_id = H5Pcreate(H5P_FILE_ACCESS);
  H5Pset_fapl_mpio(plist_id, comm, info);

//Create a new file collectively and release property list identifier.
   file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
   H5Pclose(plist_id);

//Create the dataspace for the dataset.
filespace = H5Screate_simple(RANK, dimsf, NULL);

//Create the dataset with default properties and close filespace.
dset_id = H5Dcreate(file_id, DATASETNAME, H5T_NATIVE_INT, filespace,
                                            H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

//Create property list for collective dataset write.
    plist_id = H5Pcreate(H5P_DATASET_XFER);
    H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

//Write the data
    status = H5Dwrite(dset_id, H5T_NATIVE_INT, H5S_ALL, H5S_ALL, plist_id, data);
```

Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- MPI I/O Example – Distributing Arrays
- Lab Session 1
- Break
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies
- Lab-Session2

T3PIO Library

- TACC's Terrific Tool for Parallel I/O
- Lustre parallel I/O performance depends on
  - Number of Writers (aggregators)
  - Number of Stripes (stripe count)
  - Stripe Size
  - Other parameters
- By default MPI I/O sets
  - Number of Writers = Number of nodes
  - Number of Stripes = directory default (typically 4, could be 1 or 2)
  - Stripe Size = 1 MB
- This T3PIO library will reset these parameters for you.

T3PIO Basic Heuristics

The T3PIO library resets the MPI_Info object

1. Decide the upper limit of reasonable stripe count s_{max}
   - s_{max} is bounded by the maximum possible stripe count a "friendly" user can/should use
   - s_{max} is also bounded by the Luster-imposed limit
2. Set the stripe count s to be
   - a small multiple of N (nodes), if $s_{max} > N$
   - $s = s_{max}$ (if $s_{max} \leq N$)

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

T3PIO Library: Fortran

Fortran interface

```
subroutine t3pio_set_info(comm,info,dir,err,   &
           GLOBAL_SIZE=size,                   &
           MAX_STRIPES=nstripes,               &
           FACTOR=factor,                      &
           RESULTS=results,                    &
           FILE="file"                         &
           ...)
```

T3PIO Library: C/C++

`C/C++` interface

```
include <t3pio.h>

int ierr = t3pio_set_info(comm, info, dir,
                  T3PIO_GLOBAL_SIZE, size,
                  T3PIO_MAX_STRIPES, maxStripes,
                  T3PIO_FACTOR,      factor,
                  T3PIO_FILE,        "file",
                  T3PIO_RESULTS,     &results
                  )
```

How to Use T3PIO Library (F90)

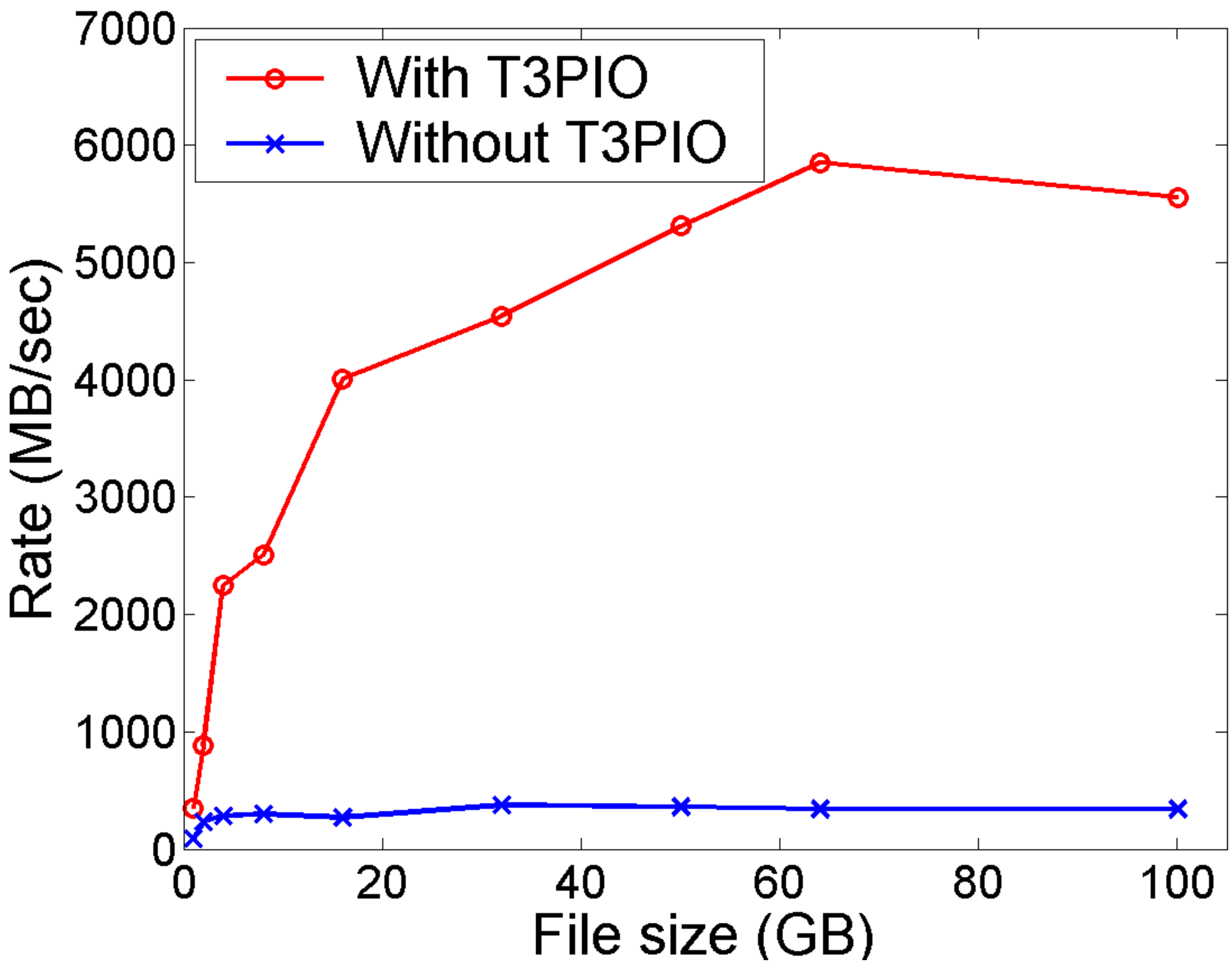
```
subroutine hdf5 writer(....)
use hdf5
use t3pio
integer info                  ! MPI Info object
integer comm                  ! MPI Communicator
integer(hid_t) :: plist_id ! Property list identifier
...
comm = MPI_COMM_WORLD
! Initialize info object.
call MPI_Info_create(info,ierr)
! use library to fill info with nwriters, stripe
call t3pio_set_info(comm, info, "./", ierr)
call H5open_f(ierr)
call H5Pcreate_f(H5P_FILE_ACCESS_F,plist_id,ierr)
call H5Pset_fapl_mpio_f(plist_id, comm, info, ierr)
call H5Fcreate_f(fileName, H5F_ACC_TRUNC_F, file_id, ierr, &
            access_prp = plist_id)
```

How to Use T3PIO Library (C)

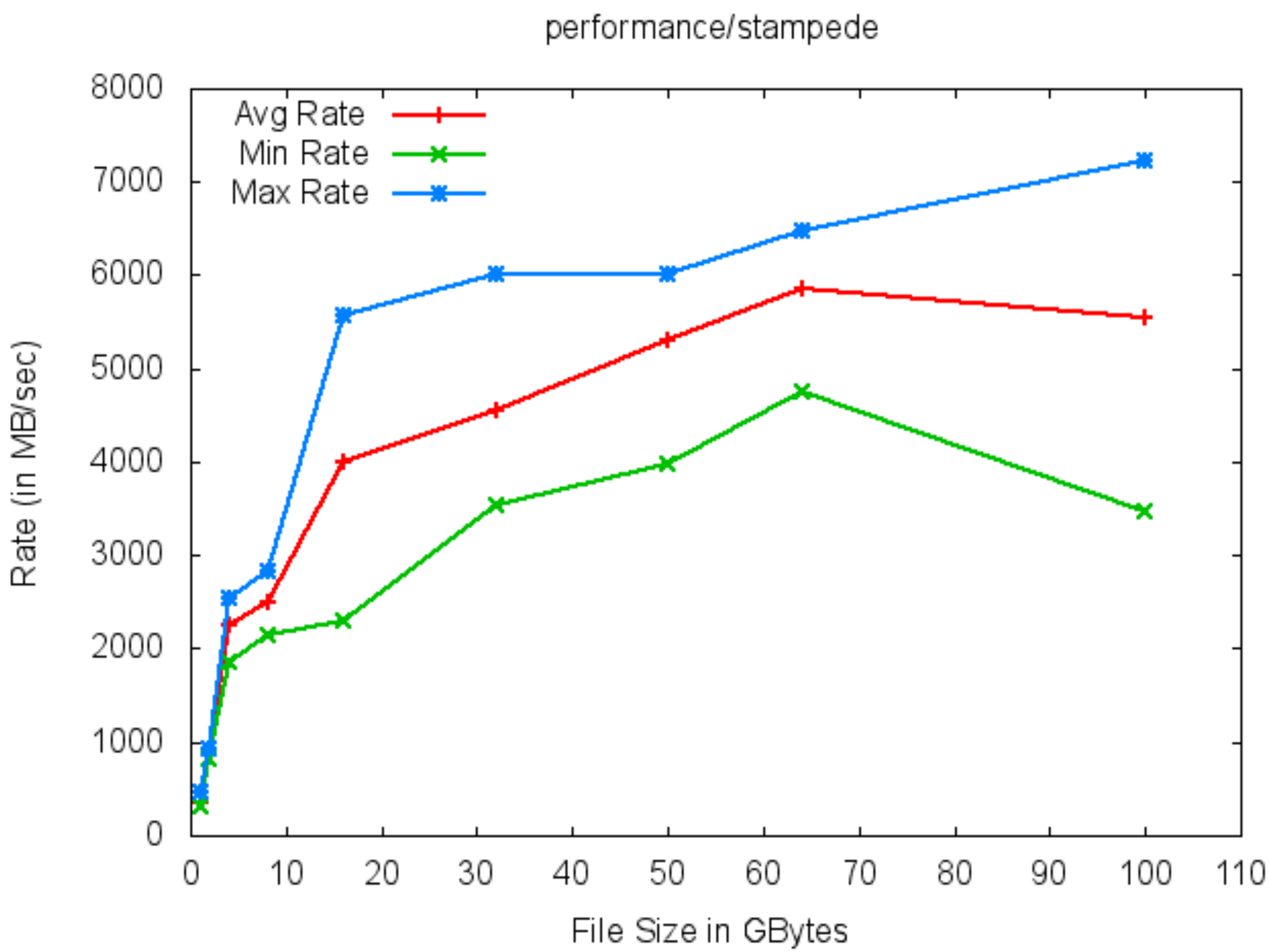
```
#include "t3pio.h"
#include "hdf5.h"
void hdf5 writer(....)
{
   MPI_Info info = MPI_INFO_NULL;
   hid_t plist_id;
   ...
   MPI Info create(&info);
   ierr = t3pio_set_info(comm, info, "./”);

   plist_id id = H5Pcreate(H5P_FILE_ACCESS);
   ierr = H5Pset_fapl_mpio(plist_id, comm, info);
   File_id = H5Fcreate(fileName, H5F_ACC_TRUNC, H5P_DEFAULT,
   plist_id);
   ...
}
```

Performance Benefit (on Stampede)



Parallel Performance



Shows variation in parallel performance on Stampede.

Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- Lab Session 1
- Break
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies
- Lab-Session2

I/O Issues Needing Attention

- Pay attention to the data storage pattern in your application
- Pay attention to the number of MDS (Meta Data Server) requests
- Pay attention to the number (or frequency) of processes accessing file simultaneously
- Pay attention to the stripe count choice of your program

General Strategies for I/O

- Access data contiguously in memory and on disk if possible
- Avoid "Too often, too many" access pattern
- Write large files to the file system if possible
- Write one global file instead of multiple files
- Use parallel I/O
  - MPI I/O
  - Parallel HDF5, parallel NetCDF
- Set file attributes (stripe count, stripe size, number of writers) properly
  - T3PIO

Summary

- I/O can impact performance a lot at large scale
- Take advantage of the parallel file system
- Consider using MPI-IO, Parallel HDF5, or Parallel NetCDF libraries（Non continuous, collective, hint)
- Analyze your code to determine if you may benefit from parallel I/O
- Set stripe count and stripe size for optimal use if on a Lustre file system

References

1. HDF5 Tutorial: www.hdfgroup.org/HDF5/Tutor/introductory.html

2. NICS I/O guide: http://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips#lustre-fundamentals

3. T3PIO: github.com/TACC/t3pio

4. Introduction to Parallel I/O: http://www.olcf.ornl.gov/wp-content/uploads/2011/10/Fall_IO.pdf

5. Introduction to Parallel I/O and MPI-IO by Rajeev Thakur

Outline

- Introduction to parallel I/O and parallel file system
- Parallel I/O Pattern
- Introduction to MPI I/O
- Lab Session 1
- Break
- Introduction to HDF5
- Introduction to T3PIO
- I/O Strategies
- Lab-Session2

HDF5 Hyperslab

- Allows hdf5 program to read or write to a portion of a dataset

- Hyperslab selection
  - logically contiguous collection of points in a dataspace
  - a regular pattern of points or blocks in a dataspace.

- A Hyperslab is a combo of the global offset and a local size

- Writing in parallel requires understanding Hyperslabs

Hyperslab example 1

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3

Hyperslab example 1 (cont.)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3

offset[0]=2
offset[1]=0

count[0]=2
count[1]=8

Hyperslab example 2

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3

Hyperslab example 2 (cont.)

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3
2	2	2	2	3	3	3	3

offset[0]=0
offset[1]=4

count[0]=4
count[1]=4

Hyperslab example 3

0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3

Hyperslab example 3 (cont.)

0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3

offset[0]=0
offset[1]=2

count[0]=2
count[1]=2

block[0]=2
block[1]=2

stride[0]=4
stride[1]=4

HDF5 Lab

- Login to Stampede with the training account or your personal account

- Change your directory to $SCRATCH

```
cds
```

- Untar the lab files (if you have not done so)

```
tar -xvzf ~train00/SEA2015.tgz
```

- Change your directory to the hdf5 lab

```
cd SEA2015/hdf5
```

HDF5 Lab

- The programs hyperslab_col_?.f90 and hyperslab_row_?.c are simple examples using HDF5 to write a distributed global array to a HDF5 file.

- The makefile will produce corresponding executables:

  hyperslab_col_?.exe -- from hyperslab_col_?.f90
  hyperslab_row_?.exe -- from hyperslab_row_?.c

  Add extra executable in the Makefile if necessary

- Running the executables will generate hdf5 data files:

  data_col.h5 or data_col.h5

  Use h5dump to check the data files

HDF5 Lab

To run the executables, follow these steps.

Build the executable:

You must build from the login node. The required libz.a library is not available on regular compute nodes. So, if you're still on a compute node from the previous exercise, please logout or you may open another terminal.

- Load the parallel hdf5 module before you build:

```
module reset
module load phdf5
```

Then build:

```
make
```

- Get an interactive session:

```
idev
```

HDF5 Exercise 1

Objective: Run a simple case to generate the "pattern" in hyperslab example 1

- Use ibrun command within an idev session to run the job:

  ibrun -np 4 ./hyperslab_col.1.exe (Fortran)

  ibrun -np 4 ./hyperslab_row.1.exe (C)

- Examine the hdf5 output file:

  h5dump data_row.h5

  h5dump data_col.h5

- You will see the data are kept as in the hyperslab example 1

  Note: Fortran users can set the parameters properly (switch the dimension) and see the same results (since h5dump is written in C)

HDF5 Exercise 2

Objective: complete hyperslab_col_2.f90 or hyperslab_row_2.c to generate the pattern in hyperslab example2

- Complete the code with proper values of offset, count
- Use ibrun command within an idev session to run the job:

  ibrun -np 4 ./hyperslab_col.2.exe (Fortran)

  ibrun -np 4 ./hyperslab_row.2.exe (C)
- Examine the hdf5 output file:

  h5dump data_row.h5

  h5dump data_col.h5
- You will see the data are kept as in the hyperslab example 2.

HDF5 Exercise 3

Objective: Complete hyperslab_col_3.f90 or hyperslab_row_3.c to generate the pattern in hyperslab example3

- Define stride and block in your code
- Complete the code with proper values of offset, count, block, stride
- Use ibrun command from within an idev session to run the job:

  ibrun -np 4 ./hyperslab_col.3.exe (Fortran)

  ibrun -np 4 ./hyperslab_row.3.exe (C)
- Examine the hdf5 output file:

  h5dump data_row.h5

  h5dump data_col.h5
- You will see the data are kept as in the hyperslab example 3.