A UT/ORNL PARTNERSHIP
NATIONAL INSTITUTE FOR COMPUTATIONAL SCIENCES
NICS

I/O Techniques and Performance Optimization

Lonnie Crosby
lcrosby1@utk.edu

NICS Scientific Computing Group

Petascale Programming Environments and Tools

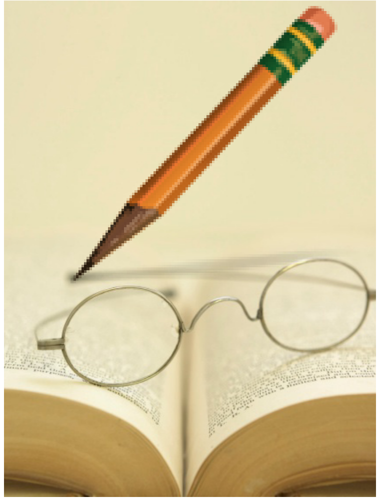
July 7, 2010

Outline

- **Introduction to I/O**
- **Path from Application to File System**
  - Data and Performance
  - I/O Patterns
  - Lustre File System
  - I/O Performance Results
- **MPI-IO**
  - General File I/O
  - Derived MPI DataTypes
  - Collective I/O
- **Common I/O Considerations**

Factors which affect I/O.

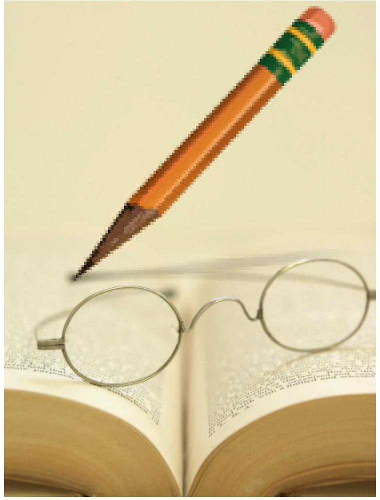
- **I/O is simply data migration.**
  - Memory ⟷ Disk
- **I/O is a very expensive operation.**
  - Interactions with data in memory and on disk.
- **How is I/O performed?**
  - I/O Pattern
    - Number of processes and files.
    - Characteristics of file access.
- **Where is I/O performed?**
  - Characteristics of the computational system.
  - Characteristics of the file system.



NICS

I/O Performance

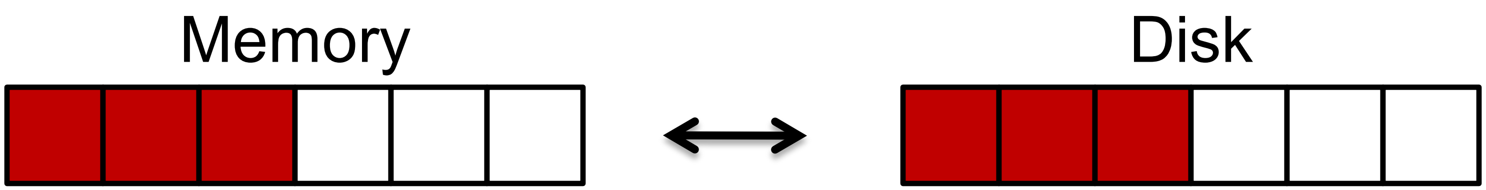
- There is no "One Size Fits All" solution to the I/O problem.
- Many I/O patterns work well for some range of parameters.
- Bottlenecks in performance can occur in many locations. (Application and/or File system)
- Going to extremes with an I/O pattern will typically lead to problems.



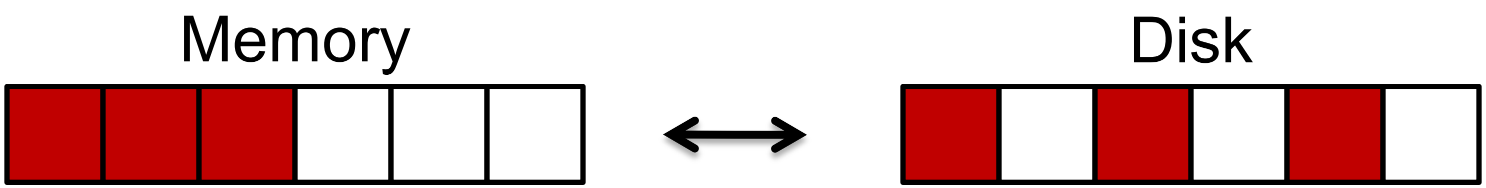
NICS

Data and Performance

- **The best performance comes from situations when the data is accessed contiguously in memory and on disk.**

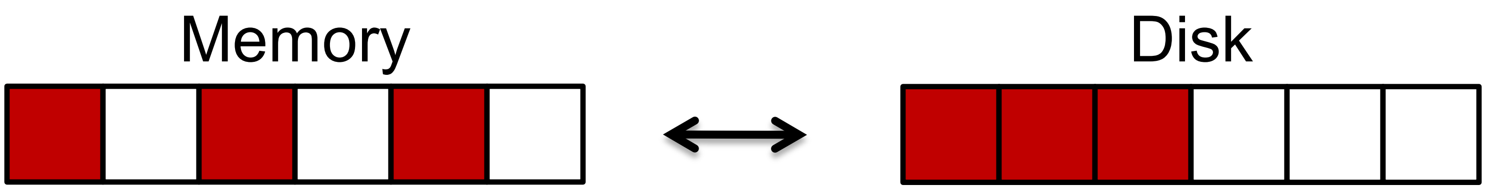


- **Commonly, data access is contiguous in memory but noncontiguous on disk. For example, to reconstruct a global data structure via parallel I/O.**

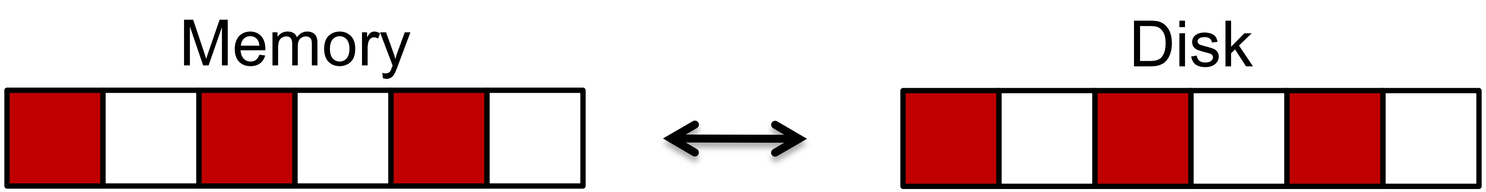


Data and Performance

- **Sometimes, data access may be contiguous on disk but noncontiguous in memory. For example, writing out the interior of a domain without ghost cells.**

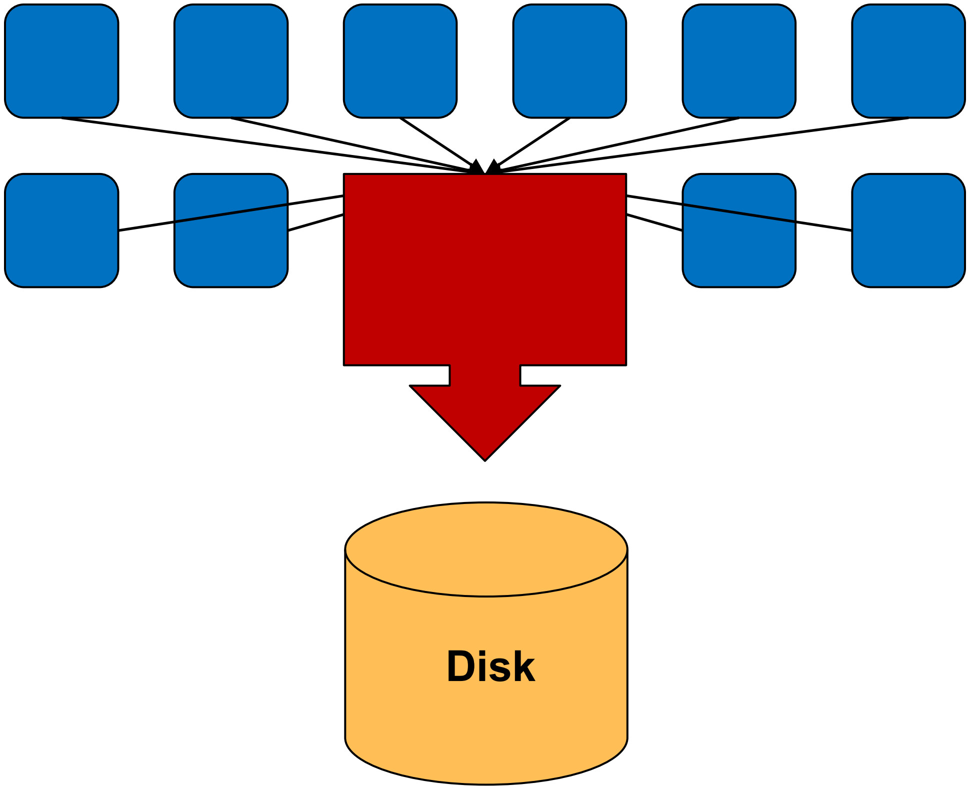


- **A large impact on I/O performance would be observed if data access was noncontiguous both in memory and on disk.**



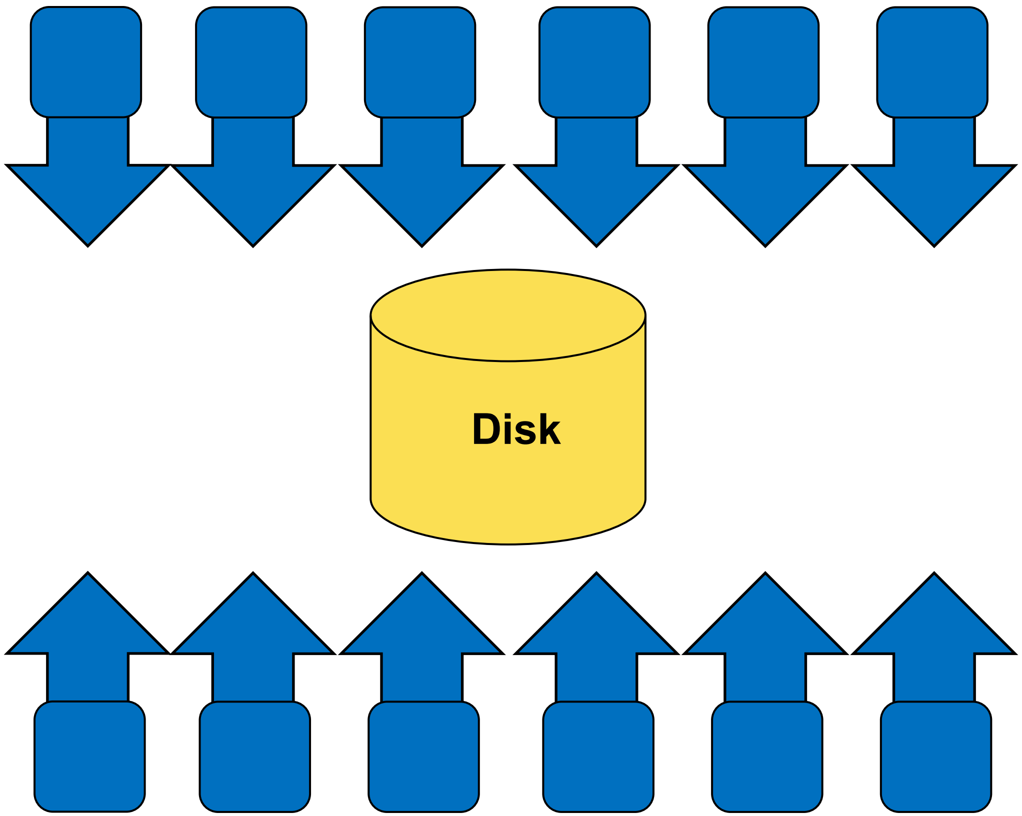
Serial I/O: Spokesperson

- Spokesperson
  - One process performs I/O.
    - Data Aggregation or Duplication
    - Limited by single I/O process.
  - Pattern does not scale.
    - Time increases linearly with amount of data.
    - Time increases with number of processes.



Parallel I/O: File-per-Process

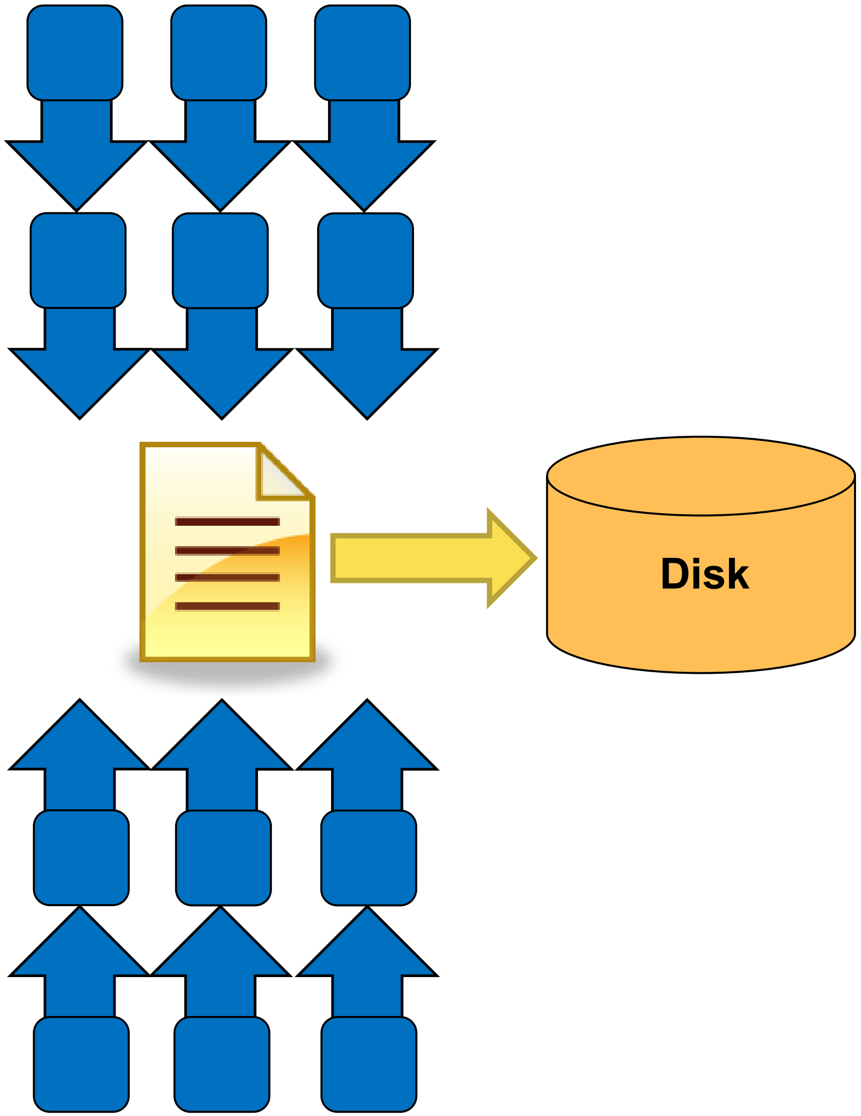
- File per process
  - All processes perform I/O to individual files.
    - Limited by file system.
  - Pattern does not scale at large process counts.
    - Number of files creates bottleneck with metadata operations.
    - Number of simultaneous disk accesses creates contention for file system resources.



NICS

Parallel I/O: Shared File

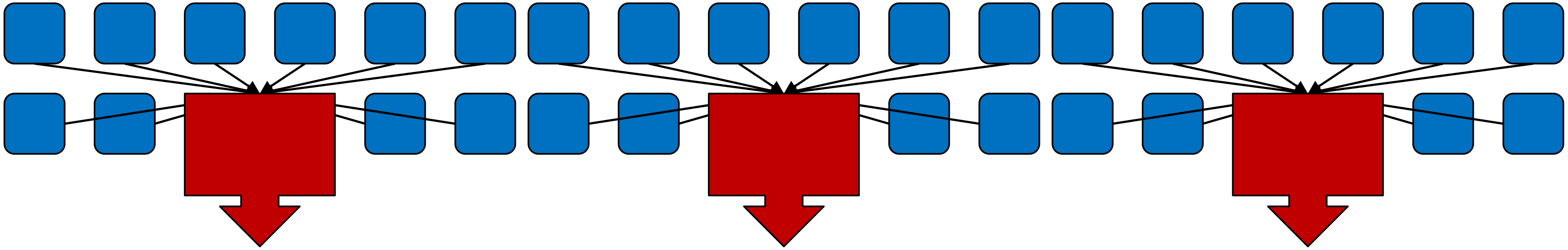
- Shared File
  - Each process performs I/O to a single file which is shared.
  - Performance
    - Data layout within the shared file is very important.
    - At large process counts contention can build for file system resources.



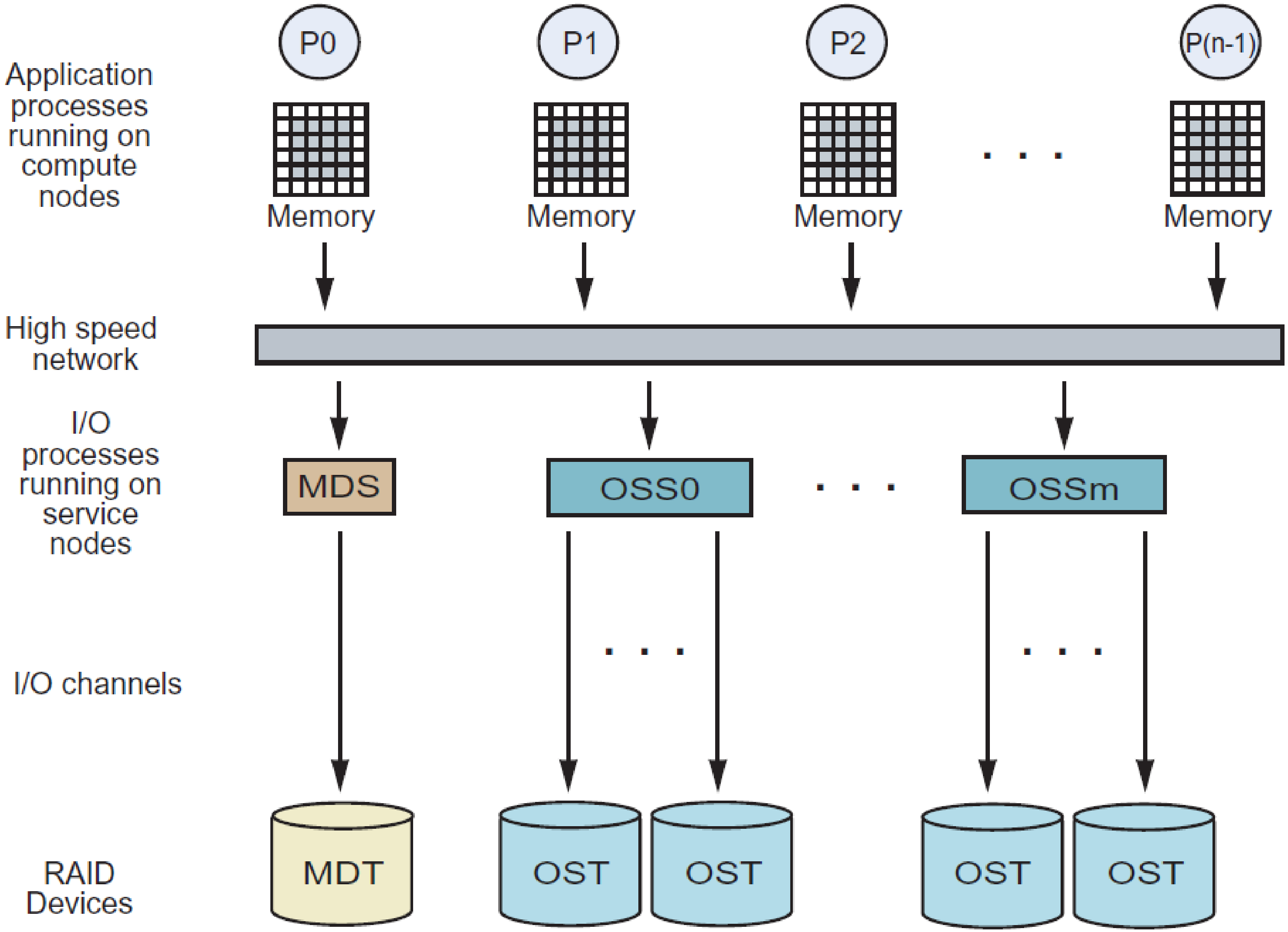
NICS

Pattern Combinations

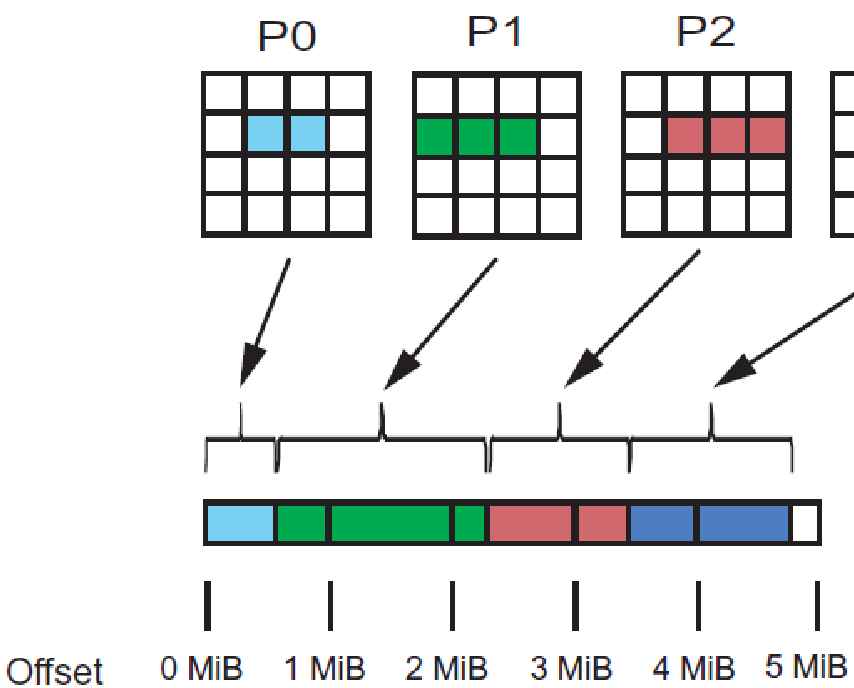
- **Subset of processes which perform I/O.**
  - **Aggregation of a group of processes data.**
    - **Serializes I/O in group.**
  - **I/O process may access independent files.**
    - **Limits the number of files accessed.**
  - **Group of processes perform parallel I/O to a shared file.**
    - **Increases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

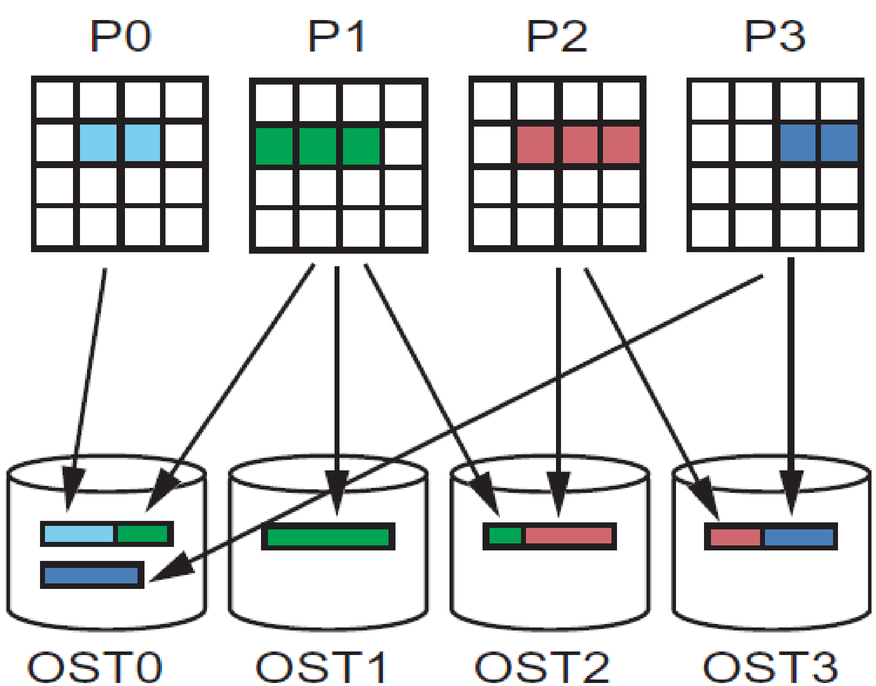


A Bigger Picture: Lustre File System



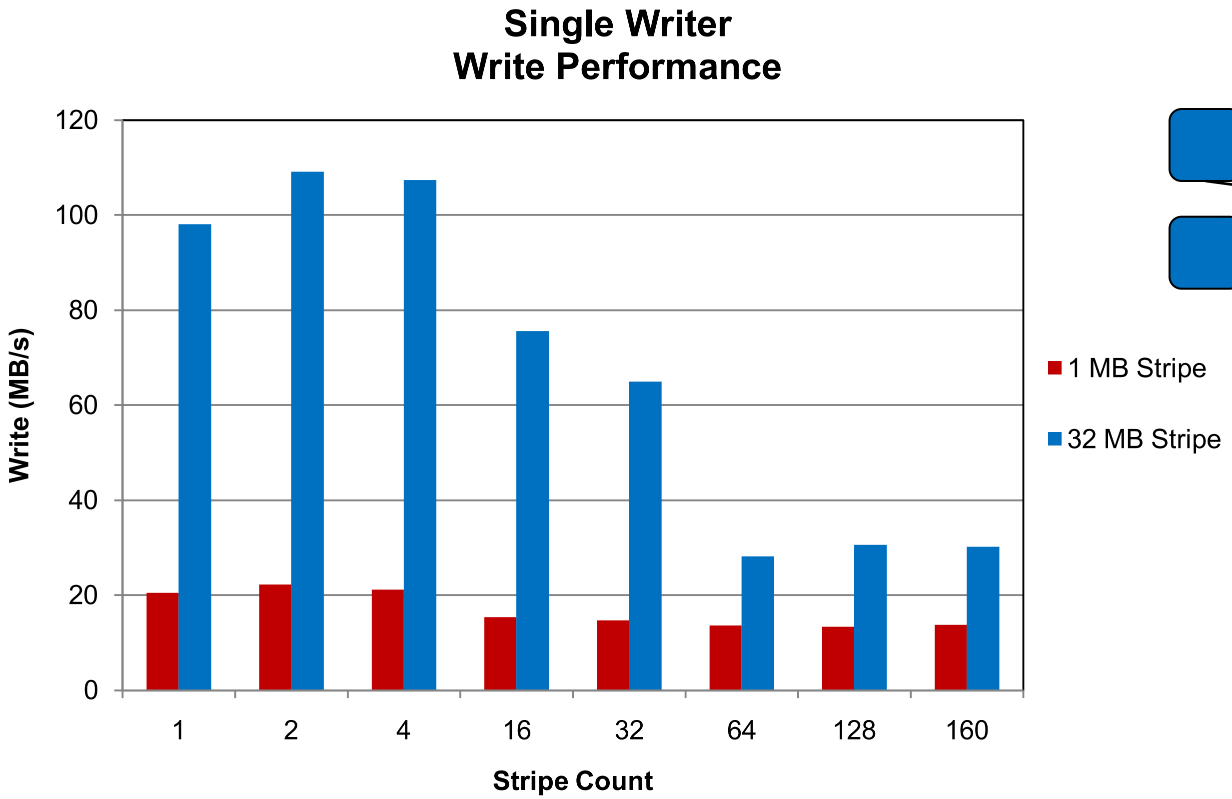
File Striping: Physical and Logical Views

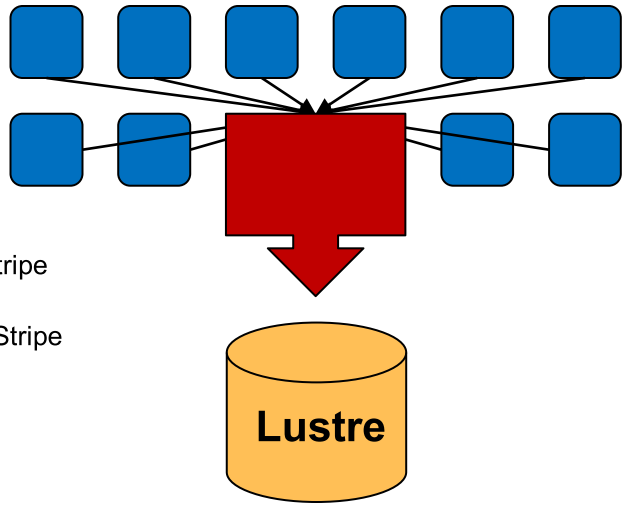




Single writer performance and Lustre

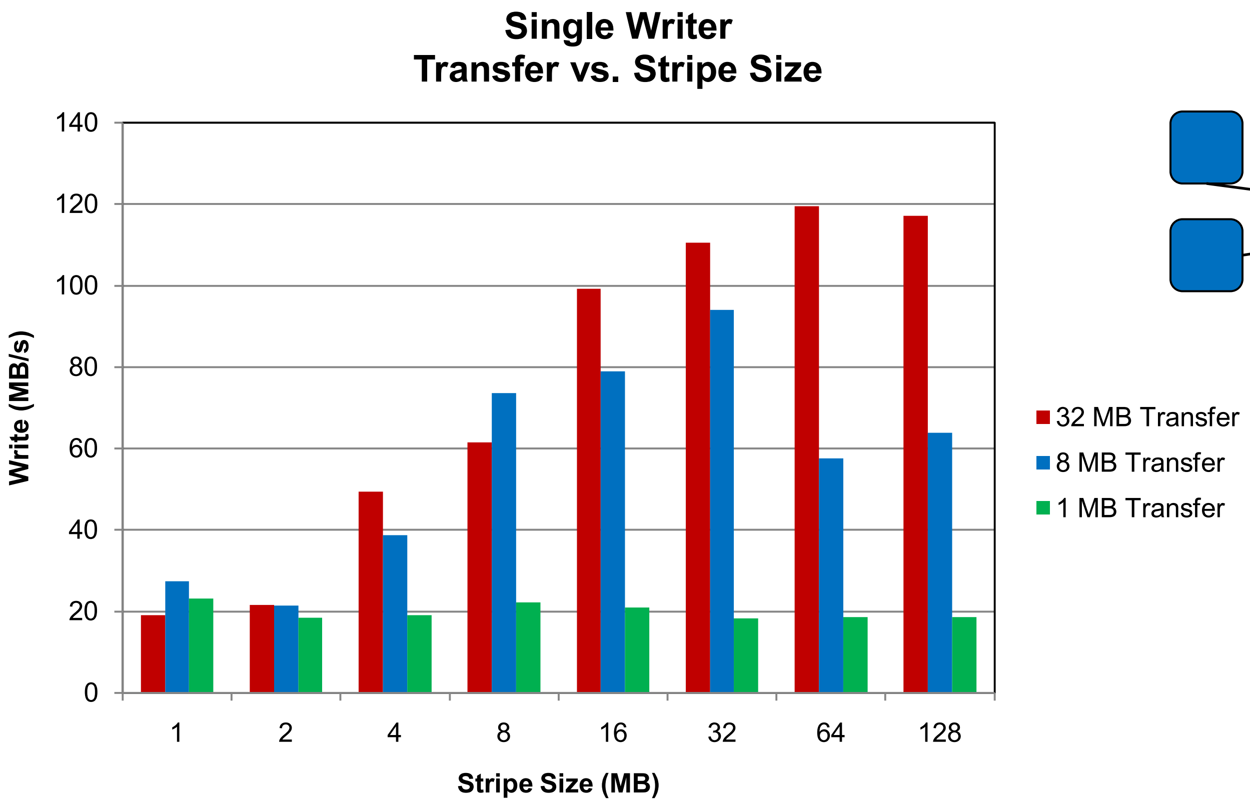
- **32 MB per OST (32 MB – 5 GB) and 32 MB Transfer Size**
  - **Unable to take advantage of file system parallelism**
  - **Access to multiple disks adds overhead which hurts performance**

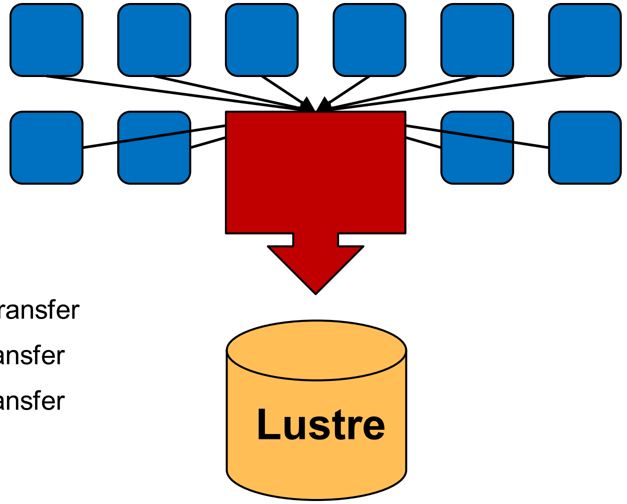




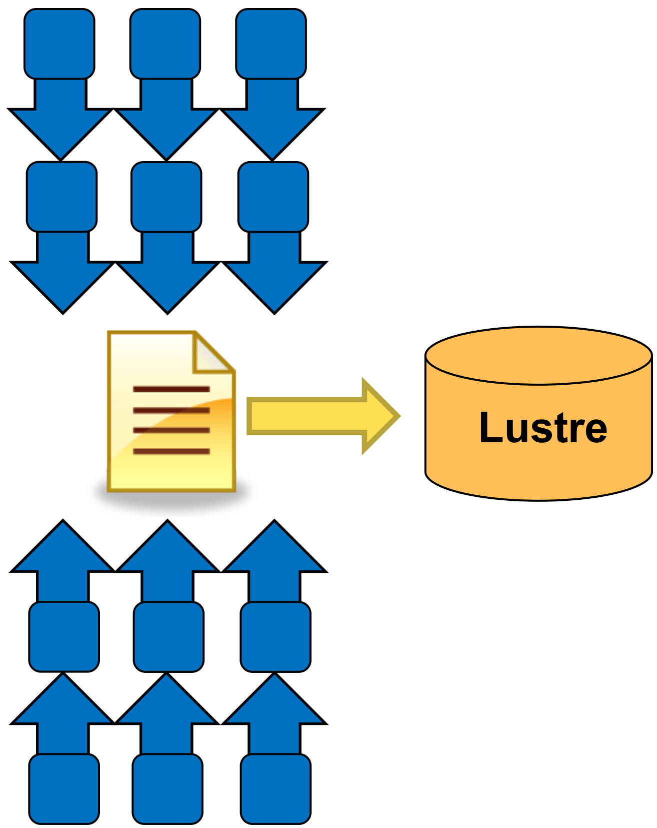
Stripe size and I/O Operation size

- **Single OST, 256 MB File Size**
  - **Performance can be limited by the process (transfer size) or file system (stripe size)**





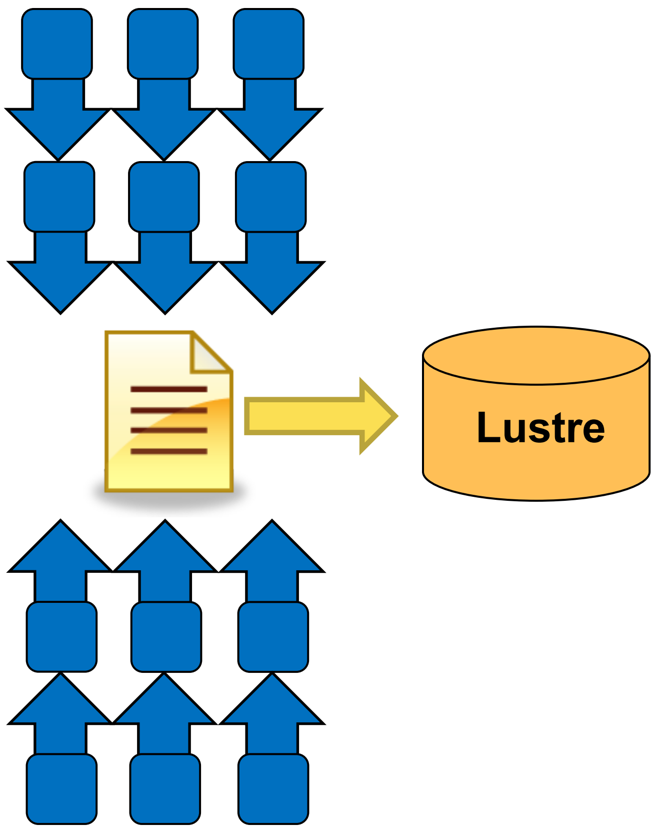
Single Shared Files and Lustre Stripes

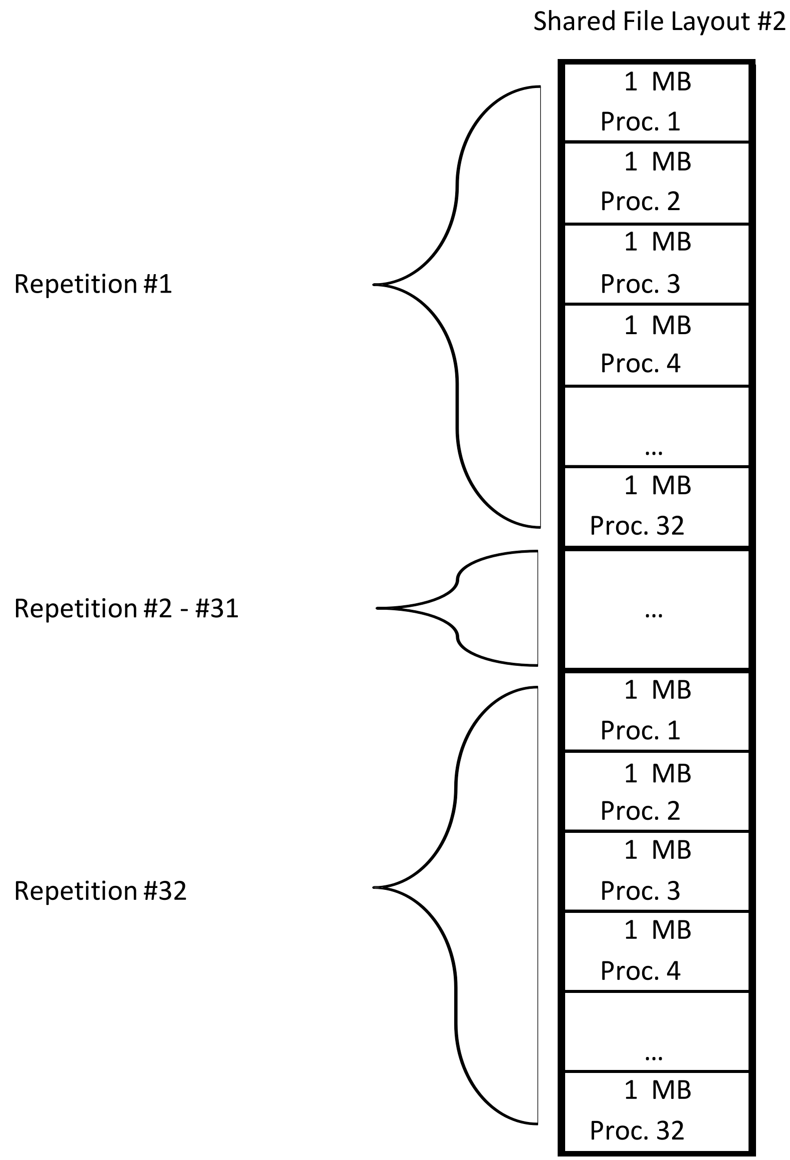


Shared File Layout #1

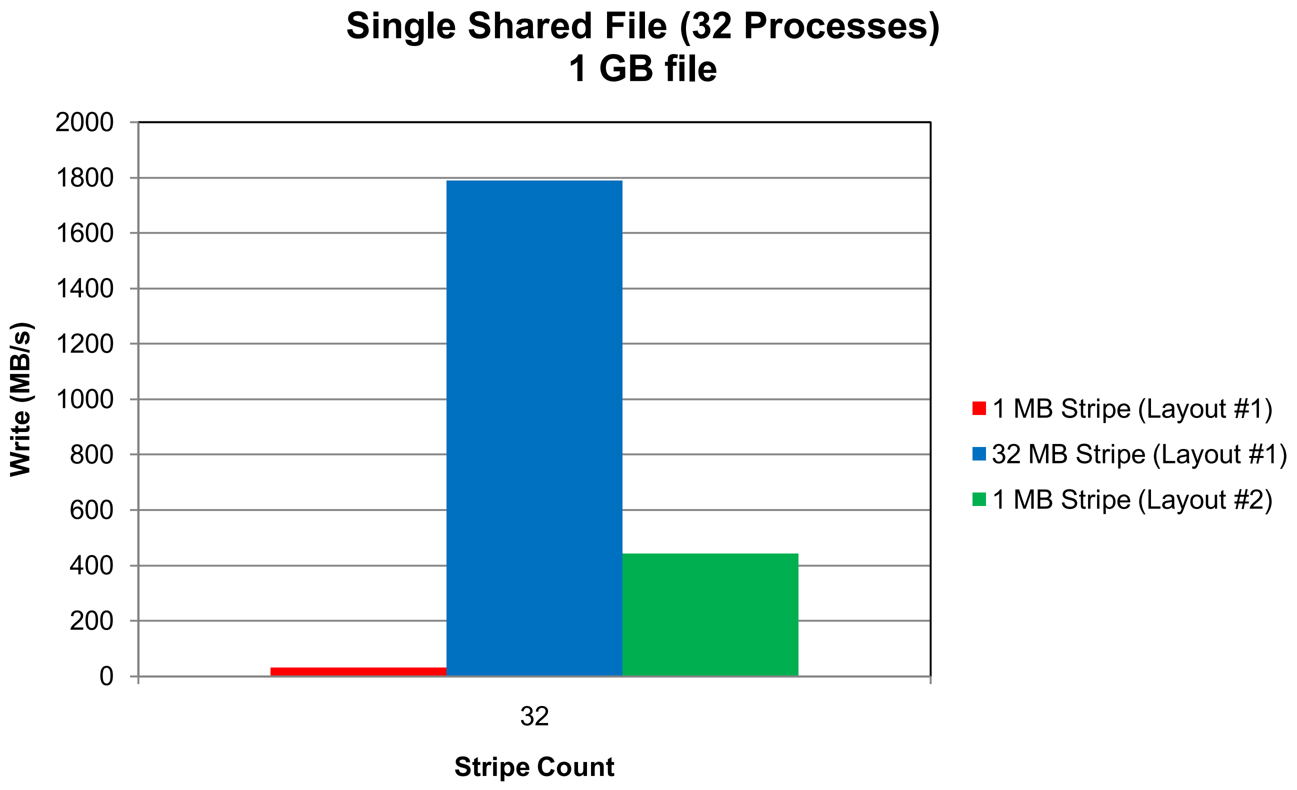
32 MB Proc. 1
32 MB Proc. 2
32 MB Proc. 3
32 MB Proc. 4
…
32 MB Proc. 32

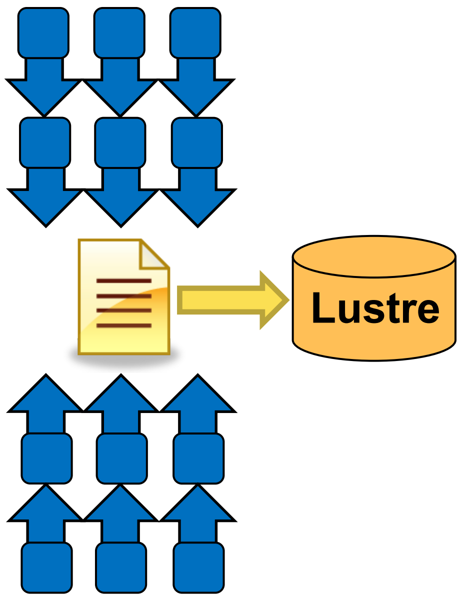
Single Shared Files and Lustre Stripes





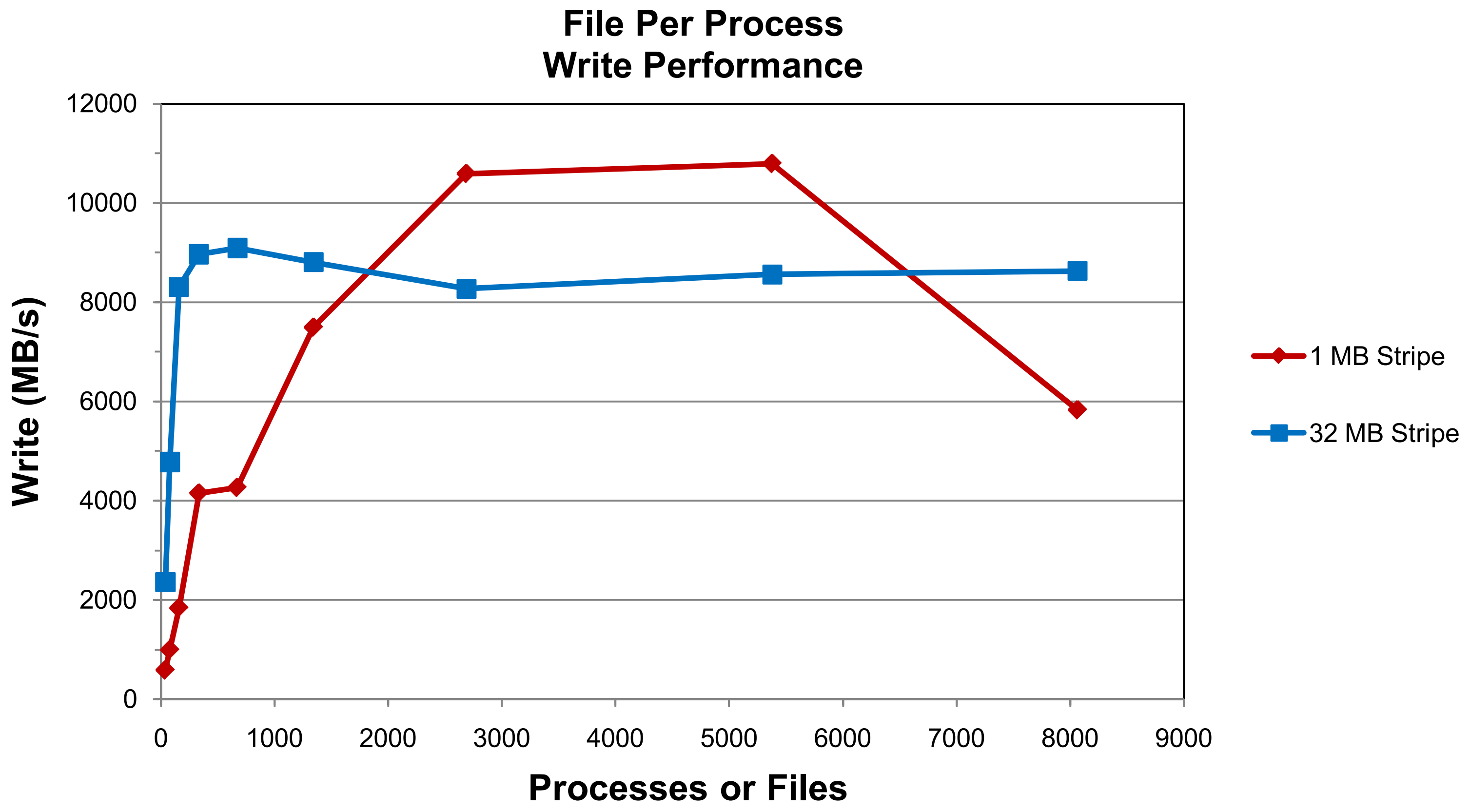
File Layout and Lustre Stripe Pattern





Scalability: File Per Process

- **128 MB per file and a 32 MB Transfer size**



NICS

Summary

- **Lustre**
  - Minimize contention for file system resources.
  - A process should not access more than one or two OSTs.
- **Performance**
  - Performance is limited for single process I/O.
  - Parallel I/O utilizing a file-per-process or a single shared file is limited at large scales.
  - Potential solution is to utilize multiple shared file or a subset of processes which perform I/O.

I/O Libraries (MPI-IO)

- Many I/O libraries such as HDF5 and Parallel NetCDF are built atop MPI-IO.
- Such libraries are abstractions from MPI-IO.

- Such implementations allow for higher information propagation to MPI-IO (without user intervention).
- Understand information flow through MPI-IO and how this may affect performance.

NICS

MPI I/O: Opening a File

- **int MPI_File_open (MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh)**
  - **Fortran: Subroutine with additional argument (integer ierr). MPI_File, MPI_Info, and MPI_Comm data types are integers in Fortran.**
  - **File is opened for each member of MPI_comm comm. MPI_COMM_SELF may be used for a private file.**
  - **int amode allows the file to be opened Read or Write only.**
  - **MPI_INFO_NULL may be used for MPI_Info info. May set hints specific to this file. See MPICH_MPIIO_HINTS.**

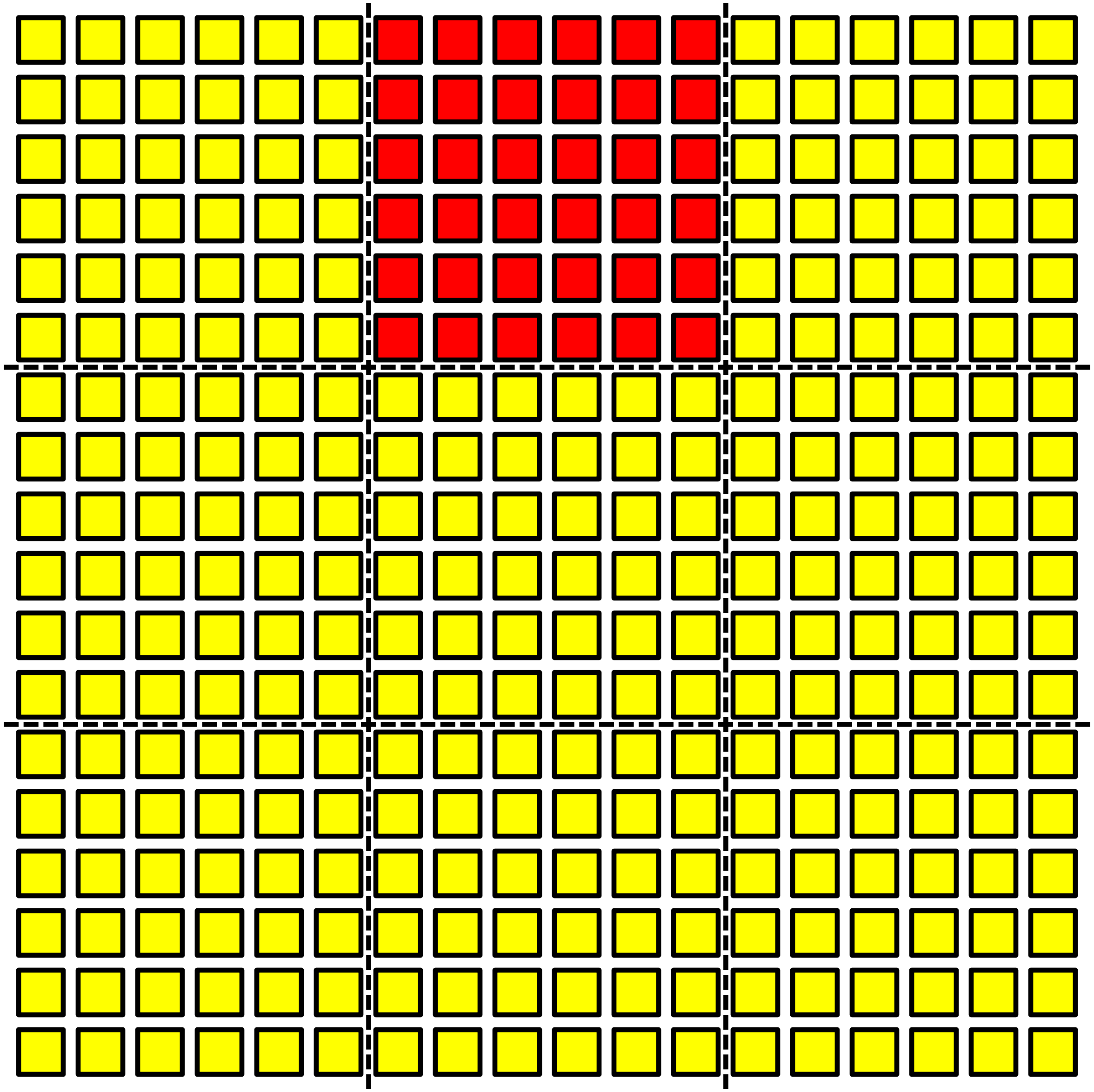
Describing the file: MPI_File_set_view

- **int MPI_File_set_view (MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char *datarep, MPI_Info_info)**
  - **Fortran: Subroutine with additional argument (integer ierr). MPI_File, MPI_Info, MPI_Offset, and MPI_Datatype data types are integers in Fortran.**
  - **etype is a data type which forms the basis of file access. Offset is in terms of etype.**
  - **Filetype is a data type which describes the portions of the file for which data will be written.**
  - **datarep may be 'NATIVE' for machine dependent binary.**
  - **MPI_INFO_NULL may be used for MPI_Info info. May set hints specific to this file. See MPICH_MPIIO_HINTS.**

MPI Derived Data Types

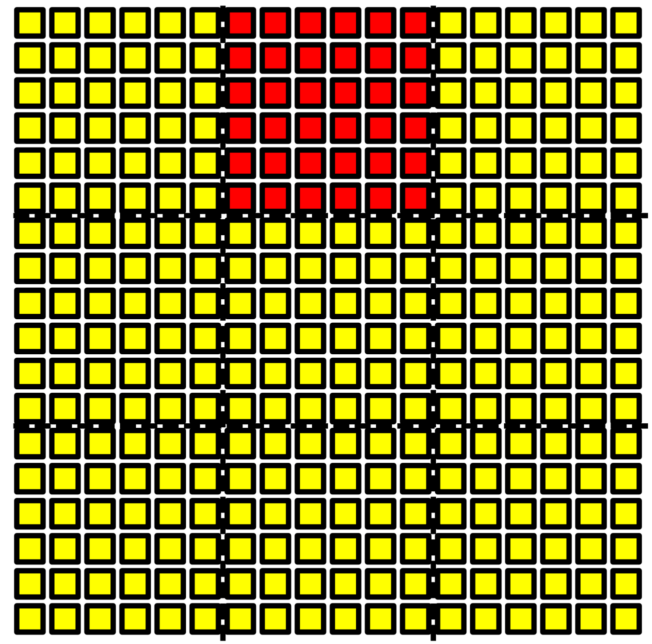
- **User defined data types which are made up of elementary data types such as MPI_DOUBLE or MPI_INTEGER.**
- **Derived data types can contain “holes” which are used to read or write noncontiguous data.**
- **Derived data types pass information to the MPIIO implementation which allows for better performance.**

Subarray Data Type



- Parameters
  - Global (18 x 18)
  - Subarray (6 x 6)
  - Index = {0, 6}
  - Extent of data type is 324 elements.
- Subarray contains the data. Remaining portions of the global array are "holes".
- Must define how global array is laid out in memory (column or row major, i.e. Fortran or C)

Subarray Data Type (Linearized)



- **Column Major (Fortran Ordering)**

- **Row Major (C Ordering)**

Vector Data Type

- Parameters
  - 6 Blocks (One for each row or column, are contiguous)
  - Blocksize = 6 elements
  - Stride = 18 (Elements between the beginning of each block)
  - Extent of data type is 96 elements.
- Blocks contain data
- Elements not within blocks are “holes” in the data type.

MPI Data type syntax

- **int MPI_Type_vector (int count, int blocklen, int stride, MPI_DataType oldtype, MPI_Datatype *newtype)**
- **int MPI_Type_create_subarray (int ndims, int *array_of_sizes, int *array_of_subsizes, int *array_of_starts, int order, MPI_Datatype oldtype, MPI_Datatype *newtype)**
  - **Fortran: These are subroutines with an additional argument at the end (integer ierr). The MPI_Datatype C data types are integers in Fortran.**
  - **Data types must be committed before use via:**
    - **int MPI_Type_commit (MPI_Datatype *datatype)**

Information in file reads/writes.

- **Explicit Read/Write**

  - **MPI_File_set_view (Offset = 108)**
  - **MPI_File_write (6 elements)**
  - **MPI_File_seek (12 elements)**

  - **MPI_File_write_at (Uses explicit offsets, combines write and seek)**

Information in file reads/writes.

- **Using Derived Data Types**
  - **MPI_Type_vector**

  - **MPI_Type_create_subarray**

  - **MPI_File_set_view (Offset = 108 or Offset = 0, filetype = vector or filetype = subarray)**
  - **MPI_File_write_at (36 elements)**

Collective I/O

- **The use of MPI_File_write [read]_at_all or MPI_File_write [read]_all allows for collective I/O using shared file pointers.**
- **Information can be given to MPI-IO via MPI derived data types. However, additional information can be given to MPI-IO (between MPI ranks) by using collective I/O.**
- **Minimizes the number of independent file accesses. Additionally allows collective mechanisms such as collective buffering and data sieving to be used.**

Read/Write Syntax

- **int MPI_File_write [read]_at_all (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)**
  - **Fortran: These are subroutines with an additional argument at the end (integer ierr). The MPI_Datatype, MPI_Offset, and MPI_Status C data types are integers in Fortran.**
  - **Difference between MPI_File_write [read] is the MPI_Offset offset argument. MPI_File_write [read]_at has the same arguments.**
  - **MPI_STATUS_IGNORE can be used for MPI_Status *status**

Closing Files and Freeing Memory

- **int MPI_File_close (MPI_File *fh)**

- **int MPI_Type_free (MPI_Datatype *datatype)**

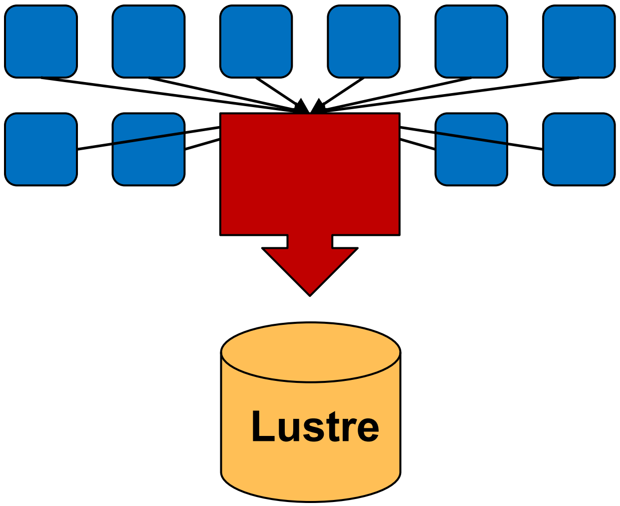
  – **Fortran: These are subroutines with an additional argument at the end (integer ierr). The MPI_Datatype and MPI_File C data types are integers in Fortran.**

NICS

MPI-IO_HINTS

- **MPI-IO are generally implementation specific. Below are options from the Cray XT5. (partial)**
  - **striping_factor (Lustre stripe count)**
  - **striping_unit (Lustre stripe size)**
  - **cb_buffer_size (Size of Collective buffering buffer)**
  - **cb_nodes (Number of aggregators for Collective buffering)**
  - **ind_rd_buffer_size (Size of Read buffer for Data sieving)**
  - **ind_wr_buffer_size (Size of Write buffer for Data sieving)**
- **export MPICH_MPIIO_HINTS = ‘ pathname pattern : key=value : key2=value2 : …’**

Collective Buffering and Data Sieving



- **Collective Buffering**
  - Aggregates I/O to a process (buffer)
  - This buffer is then written to disk.
- **Data Sieving**
  - More data than needed is written/read (buffer).
  - The needed information is obtained from the buffer.

NICS

Summary

- **Three Levels of I/O possible within MPI-IO.**
  - **Explicit Read/Write**
  - **Use of MPI Derived Data types (non-contiguous data)**
  - **Collective I/O (parallel I/O to a shared file)**

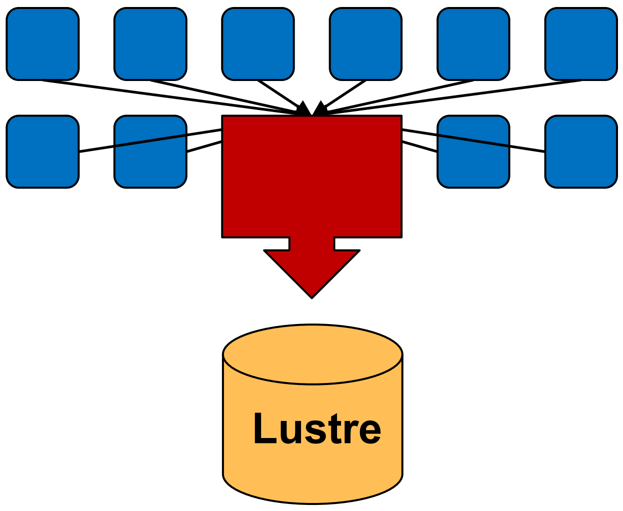
- **MPI-IO Hints can be given to improve performance by supplying more information to the library. This information can provide the link between application and file system.**

Common I/O Considerations

- **Standard Input/Output**
- **Buffered I/O**
- **Binary Files and Endianess**
- **Subsetting I/O**
  - **Aggregation**
  - **Turnstile**
  - **Multiple Shared Files**

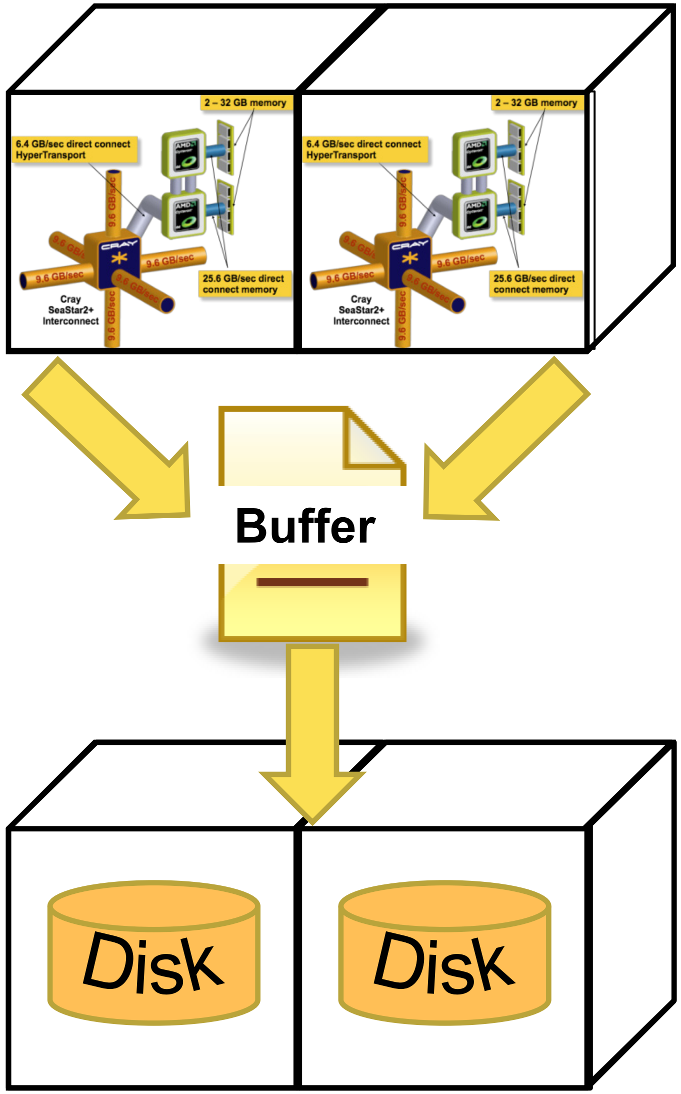
Standard Output and Error

- **Standard Output and Error streams are effectively serial I/O.**
- **Generally, the MPI launcher will aggregate these requests. (Example: mpirun, mpiexec, aprun, ibrun, etc..)**
- **Disable debugging messages when running in production mode.**
  - **"Hello, I'm task 32000!"**
  - **"Task 64000, made it through loop."**



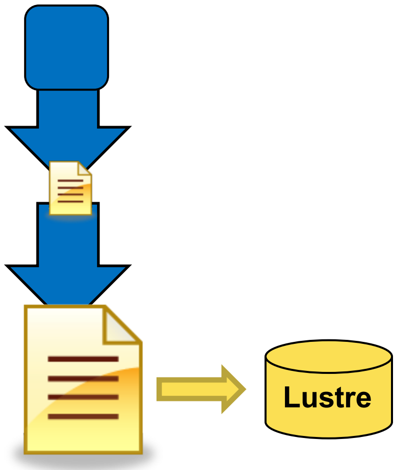
Buffered I/O

- **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.
- **Caution**
  - Frequent buffer flushes can adversely affect performance.



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
```

Binary Files and Endianess

- **Writing a big-endian binary file with compiler flag byteswapio**

File “XXXXXX"

	Calls	Megabytes	Avg Size
Open	1		
Write	5918150	23071.28062	4088
Close	1		
Total	5918152	23071.28062	4088

- **Writing a little-endian binary**

File “XXXXXX"

	Calls	Megabytes	Avg Size
Open	1		
Write	350	23071.28062	69120000
Close	1		
Total	352	23071.28062	69120000

Subsetting I/O

- **At large core counts, I/O performance can be hindered**
  - **by the collection of metadata operations (File-per-process) or**
  - **by file system contention (Single-shared-file).**
- **One solution is to use a subset of application processes to perform I/O. This limits**
  - **the number of files (File-per-process) or**
  - **the number of processes accessing file system resources (Single-shared-file).**
- **If you can not implement a subsetting approach, try to limit the number of synchronous file opens to reduce the number of requests simultaneously hitting the metadata server.**

Further Information

- **Lustre Operations Manual**
  - http://dlc.sun.com/pdf/821-0035-11/821-0035-11.pdf
- **GPFS: Concepts, Planning, and Installation Guide**
  - http://publib.boulder.ibm.com/epubs/pdf/a7604133.pdf
- **HDF5 User Guide**
  - http://www.hdfgroup.org/HDF5/doc/PSandPDF/HDF5_UG_r183.pdf
- **The NetCDF Tutorial**
  - http://www.unidata.ucar.edu/software/netcdf/docs/netcdf-tutorial.pdf

Further Information MPI-IO

- Rajeev Thakur, William Gropp, and Ewing Lusk, "A Case for Using MPI's Derived Datatypes to Improve I/O Performance," in *Proc. of SC98: High Performance Networking and Computing*, November 1998.
  - http://www.mcs.anl.gov/~thakur/dtype
- Rajeev Thakur, William Gropp, and Ewing Lusk, "Data Sieving and Collective I/O in ROMIO," in *Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation*, February 1999, pp. 182-189.
  - http://www.mcs.anl.gov/~thakur/papers/romio-coll.pdf
- Getting Started on MPI I/O, Cray Doc S–2490–40, December 2009.
  - http://docs.cray.com/books/S-2490-40/S-2490-40.pdf