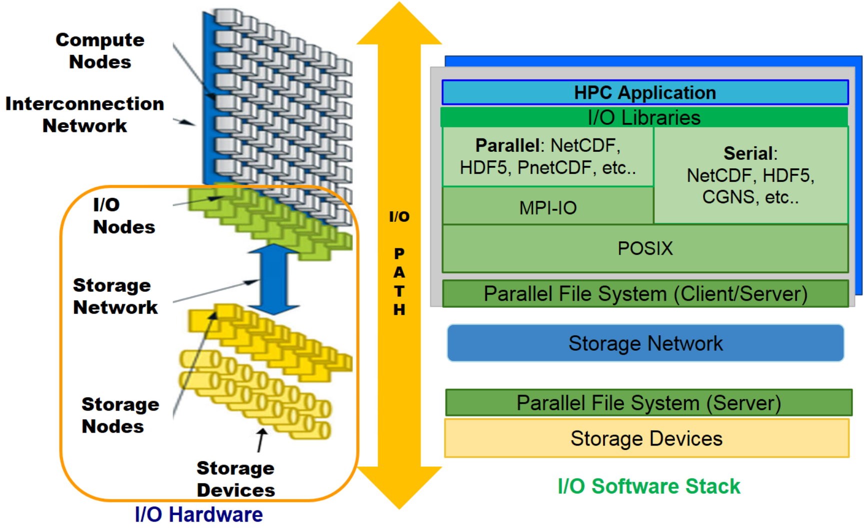
Leibniz Supercomputing Centre
of the Bavarian Academy of Sciences and Humanities

PRACE PATC Course: Advanced Topics in HPC
Topic: Tuning I/O on LRZ's HPC systems

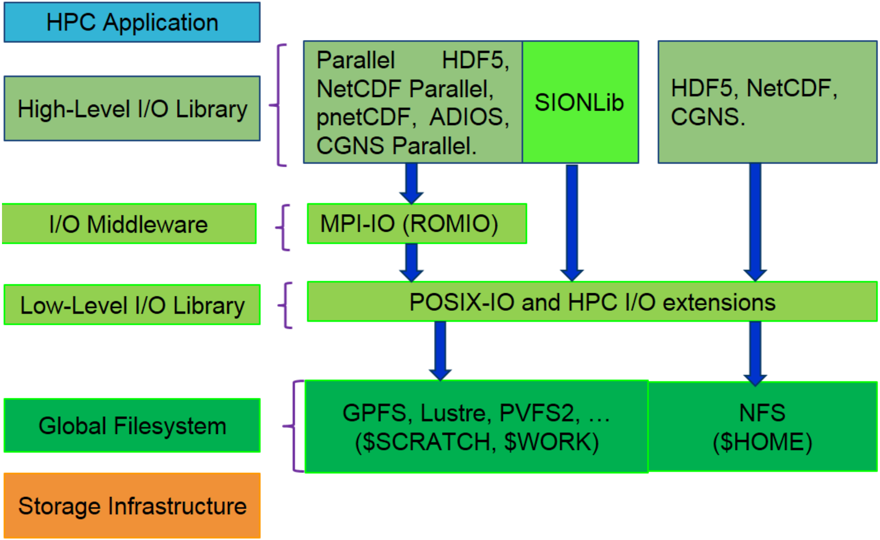
Sandra Mendez, PhD. - HPC Group, LRZ.
sandra.mendez@lrz.de

Outline

The HPC I/O system



I/O Software Stack



Stack Software

POSIX to maintain compatibility with the broadest base of applications while enabling high performance.

Parallel file systems across HPC provides parallel access while retaining POSIX semantics.

The portability and optimization needed for parallel I/O cannot be achieved with the POSIX interface.

MPI-IO provides a high-level interface supporting partitioning of file data among processes and a collective interface supporting complete transfers of global data structures between process memories and files.

Many applications make use of **higher-level I/O libraries** such as the Hierarchical Data Form (HDF), the Network Common Data Format (NetCDF).

File Systems in HPC

File Systems

File systems have two key roles

- Organizing and maintaining the file name space
- Storing contents of files and their attributes

Networked file systems must solve two new problems

- File servers coordinate sharing of their data by many clients
- Scale-out storage systems coordinate actions of many servers

Parallel file systems support parallel applications

- A special kind of networked file system that provides high-performance I/O when multiple clients share the file system
- The ability to scale capacity and performance is an important characteristic of a parallel file system implementation

Parallel File Systems

Striping is the basic mechanism used in Parallel File Systems for improving performance.

- File data is split up and written across multiple I/O servers

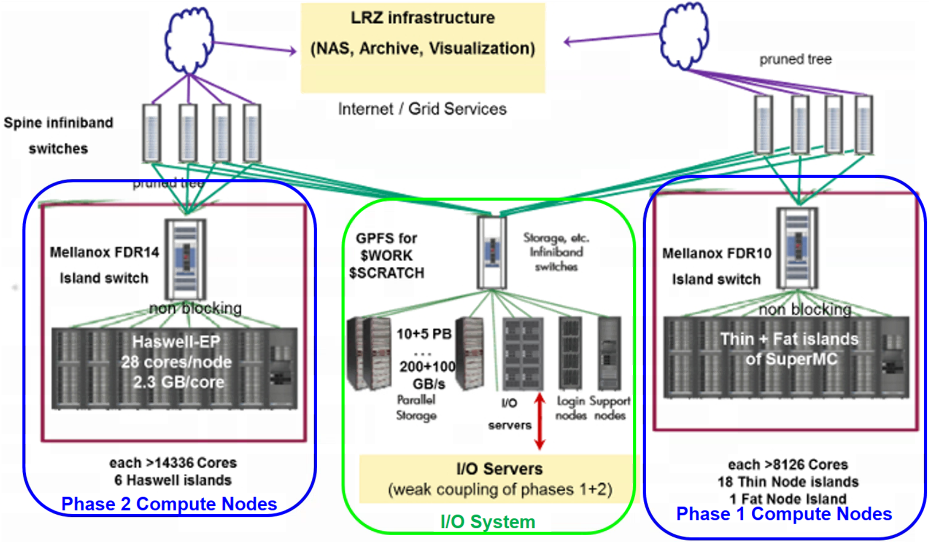
Primarily striping allows multiple servers, disks, network links to be leveraged during concurrent I/O operations

- Eliminates bottlenecks
- Can also improve serial performance over a single, local disk

Coordinating access can re-introduce bottlenecks

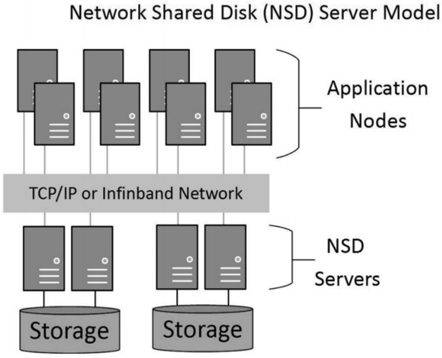
- It is necessary for coherence

SuperMUC Supercomputer



GPFS

Built from a collection of arrays that contain the file system data and metadata.



Picture Source, High Performance Parallel I/O Book, Chapter 9. Editors Prabhat, Quincey Koziol. October 2014.

Characteristics:

- Striping data (in block size) across multiple disks attached to multiple nodes.
- High performance metadata (inode) scans.
- Supporting a wide range of file system block sizes to match I/O requirements.
- Using block level locking based on a very sophisticated scalable token management system to provide data consistency while allowing multiple application nodes concurrent access to the files.

GPFS in SuperMUC (LRZ)

- To display the amount of available disk space for each filesystem:

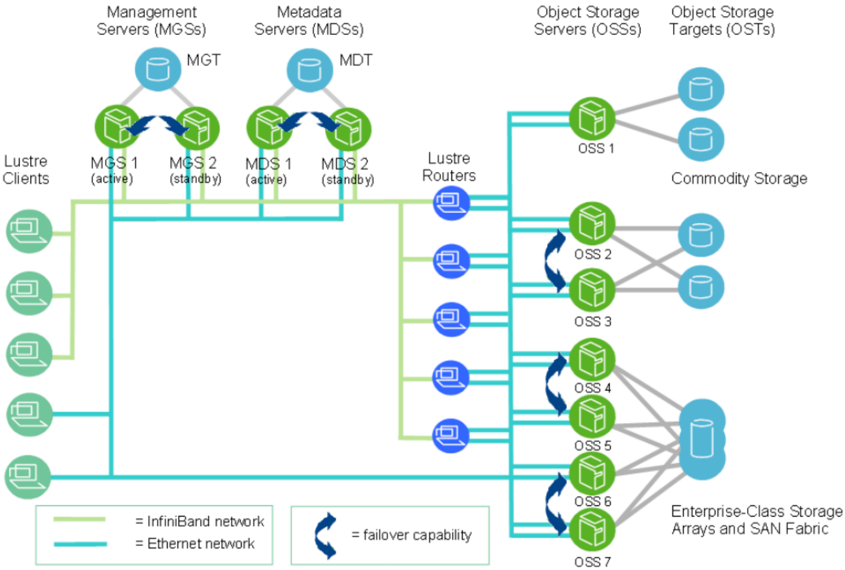
```
di98het@login05:~> df -Th
Filesystem     Type    Size  Used  Avail Use% Mounted on
/dev/fs1       gpfs     12P  8.8P  3.0P  75% /gpfs
/dev/fs2       gpfs    5.2P  3.8P  1.4P  73% /gss/scratch
```

- To display the properties of GPFS on SuperMUC, you must load the appropriate module:

```
di98het@login05:~> module load gpfs
di98het@login05:~> mmlsfs fs1
flag        value        description
------------------------------------------------------
 -f         262144       Minimum fragment size in bytes
...
 -B        8388608       Block size
...
```

The I/O operation size should be a multiple of the block size.

Lustre


Management Servers (MGSs)
Metadata Servers (MDSs)
Object Storage Servers (OSSs)
Object Storage Targets (OSTs)
MGT
MDT
MGS 1 (active)
MGS 2 (standby)
MDS 1 (active)
MDS 2 (standby)
Lustre Clients
Lustre Routers
OSS 1
OSS 2
OSS 3
OSS 4
OSS 5
OSS 6
OSS 7
Commodity Storage
Enterprise-Class Storage Arrays and SAN Fabric
= InfiniBand network
= Ethernet network
= failover capability

Lustre Components

- Metadata Servers (MDS): The MDS makes metadata stored in one or more MDTs available to Lustre clients.
- Metadata Targets (MDT):The MDT stores metadata (such as filenames, directories, permissions and file layout) on storage attached to an MDS.
- Object Storage Servers (OSS): The OSS provides file I/O service and network request handling for one or more local OSTs.
- Object Storage Target (OST): User file data is stored in one or more objects, each object on a separate OST in a Lustre file system. The number of objects per file is configurable by the user and can be tuned to optimize performance for a given workload.
- Lustre clients: Lustre clients are computational, visualization or desktop nodes that are running Lustre client software, allowing them to mount the Lustre file system.

Lustre File Layout (Striping)

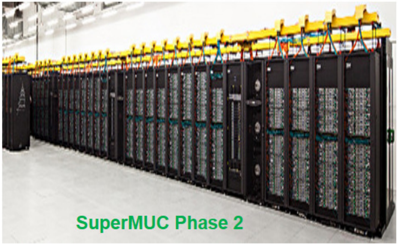
```
> df -Th
mds6802-ib0@o2ib:mds6801-ib0@o2ib:/scratch
Filesystem     Type    Size  Used  Avail Use% Mounted on
mds6802-ib0@o2ib:mds6801-ib0@o2ib:/scratch
               lustre  695T  470T  226T  68% /mnt/lustre/scratch
```

- The stripe_size indicates how much data to write to one OST before moving to the next OST.
- The stripe_count indicates how many OSTs to use. The default stripe_count value is 1.

```
>lfs getstripe reads2.fastq
reads2.fastq
lmm_stripe_count:    1
lmm_stripe_size:     1048576
lmm_pattern:         1
lmm_layout_gen:      0
lmm_stripe_offset:   42
  obdidx     objid      objid       group
   42       37138823   0x236b187     0
```

```
>lfs setstripe -s 4M file-Stripe-4MB.txt
>lfs getstripe file-Stripe-4MB.txt
file-Stripe-4MB.txt
lmm_stripe_count:    1
lmm_stripe_size:     4194304
lmm_pattern:         1
lmm_layout_gen:      0
lmm_stripe_offset:   31
  obdidx     objid      objid       group
   31       40653404   0x26c525c     0
```

LRZ's HPC Systems



Linux-Cluster

Usually Global File Space:

- Home ($HOME)
- Project ($WORK)
- Pseudo-Temporary ($SCRATCH)

Filesystems at SuperMUC

File system	Environment Variable for Access	Purpose	Implementation Overall size Bandwidth	Backup and Snapshots	Intended Lifetime and Cleanup Strategy	Quota size (per project)
/home/hpc	$HOME	Store the user's source, input data, and small and important result files. Globally accessible from login and compute nodes.	NAS-Filer 1.5 PB 10 GB/s	YES	Project duration	default: 100 GB per project
/gss/scratch	$SCRATCH	Temporary huge files (restart files, files to be pre-/post processed). Globally accessible from login and compute nodes.	GPFS 5.2 PB <= 100 GB/s on Phase 1 <= 150GB on Phase2.	NO	Automatic deletion after approx. 2 weeks	no quota, but high water mark deletion if necessary
/gpfs/work	$WORK	Huge result files. Globally accessible from login and compute nodes.	GPFS 12 PB <= 200 GB/s on Phase 1. <= 150 GB/s on Phase 2.	NO	Project duration	default: 1 TB per project.

Filesystems at Linux-Clusters

Purpose	Segment of the Linux Cluster	File system type and full name	How the user should access the files	Space Available	Approx. aggregated bandwidth
Globally accessible Home and Project Directories					
User's Home Directories	all	NFS /home/hpc/<group>/<user>	$HOME	100 GByte by default **per project**	up to a few 100 MB/s
Project file system	all	NFS /naslx/projects/<group>/<user>	$WORK	up to 5 TByte **per project** **available on request**	up to 1 GB/s
Temporary/scratch File Systems					
Scratch file system	all **except** CooLMUC2	NFS /naslx/ptmp/<vol>/<user> (<vol> is a one or two digit number uniquely determined from the user ID)	$SCRATCH	several TByte	up to 1 GB/s
Scratch file system	CooLMUC2, CooLMUC3	GPFS /gpfs/scratch/<group>/<user>	$SCRATCH	1,400 TByte	up to ~30 GB/s on CooLMUC2 (aggregate) up to ~8 GB/s on CooLMUC3 (aggregate)
Node-local File Systems (please do not use!)					
Node-local temporary user data	all	local disks, if available /tmp		8-200 GByte	approx. 30 MB/s for diskfull nodes

FS at SuperMUC-NG (1)

Area	Purpose	Total Capacity	Aggregate Bandwidth	Linux Ownership
Home	Storage for user's source, input data, and small and important result files. Globally accessible from login and compute nodes.	256 TiB	~25 GiB/s (SSD Tier) ~6 GiB/s (HDD Tier)	User account
Work	Large datasets that need to be kept on-disk medium or long term. Globally accessible from login and compute nodes.	34 PiB	~300 GiB/s	Project manager
Scratch	Temporary storage for large datasets (usually restart files, files to be pre-/postprocessed). Globally accessible from login and compute nodes.	16 PiB	~200 GiB/s	User account
DSS	Data Science Storage. Long term storage for project's purposes and/or the science community. World wide access/transfer for this data via high performance WAN optimized transfer protocols, using a simple Graphical User Interface in the Web. Share data like LRZ Sync+Share, Dropbox or Google Drive.	20 PiB	~70 GiB/s	Data project manager
Node-local	**/tmp partition** on login and compute nodes. Resides in memory on compute nodes. Locally accessible only. Please **do not use** paths to this area explicitly (e.g. in scripts). TMPDIR (see below) can be used and will automatically be set to an appropriate value.	Small. A completely filled /tmp causes the node to become **unusable**. Therefore, lifetimes are short (per-job, or a few days).	varies	root, with read/write rights for all users.

FS at SuperMUC-NG (2)

Area	Environment Variable	Path pattern	Quota	Lifetime of Data	Data Safety/Integrity Measures
Home	HOME	/dss/home/<hash>/<user>	100 GB/user	Expiration of all projects an account is associated with	Replication to secondary storage plus weekly backup to tape
Work	WORK_<project>	$WORK_<project>	In accordance with project grant[1].	End of specified project	None. See section below on archiving important data.
Scratch	SCRATCH	$SCRATCH	1 PB/user (safety measure)	Usually 3-4 weeks. Execution of deletion procedure depends on file system filling.	None. See section below on archiving important data.
DSS	-	/dss/<data-project>/<container>	Per data-project and container[1]	End of data project	per-container policy. Regarding backup to tape archive: NONE, BACKUP_WEEKLY, BACKUP_DAILY
temporary	TMPDIR	depends on availability of file systems, usually a subfolder of SCRATCH. /tmp is only used as a last measure cop-out.	depends on target file system	depends on target file system.	depends on target file system.

Select the suitable file system

- Put really important data into your $HOME file system.
- Perform your own tape archiving on $SCRATCH and $WORK file systems.
- Use the $SCRATCH environment variable, which always references the performance-optimal temporary file system.
- Avoid using your $HOME directory for temporary files.
- Avoid using too many files (>1000 per directory).
- Avoid putting any data you cannot easily recompute into a pseudo-temporary file system (unless you reliably do your own tape archiving).

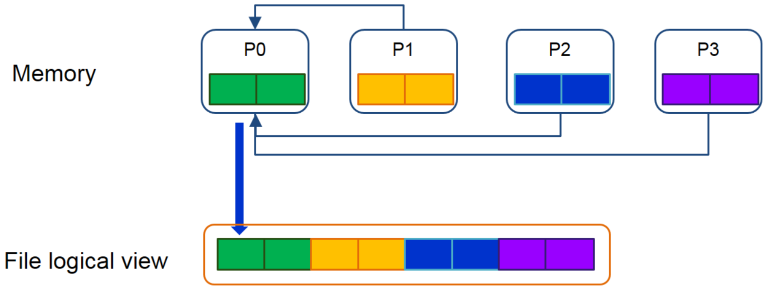
HPC I/O Patterns

Typical I/O in HPC(1)

Parallel applications perform I/O in serial and parallel.

Serial I/O

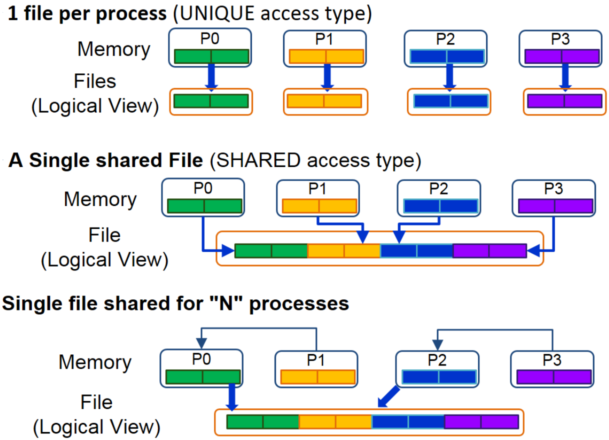
A process writes or reads to/from a file. Usually an I/O process receives the data through communication events and writes the data to a file.



Typical I/O in HPC(2)

Parallel I/O

Several processes perform I/O within a single file or multiple files.



Typical I/O in HPC(3)

1 file per process (UNIQUE access type)

Limited by file system and it does not scale for large count of processes. Number of files creates bottleneck with metadata operations and a number of simultaneous disk accesses can create contention for file system resources.

A Single shared File (SHARED access type)

Data layout within the shared file must be defined appropriately to avoid the contention due to concurrent accesses.

Single file shared for "N" processes

The number of shared files increases and it decreases the number of processes per file. In this way, it is possible reduce the metadata operations and the concurrent accesses to shared files.

I/O Access Patterns

I/O Operation
- Read only
- Write only

Request size
- Fixed / Variable
- Small / Medium / Large

Spatial Patterns


- Contiguous


- Non-contiguous: Strided
  Fixed strided / 2D-strided / Negative strided /Random strided/ kD-strided


- Combination of contiguous and non-contiguous patterns

Temporal intervals
- Fixed
- Random

Repetition
- Single Occurrence
- Multiple Ocurrence

MPI-IO: An I/O Middleware

MPI-IO

A parallel I/O system for distributed memory architectures will need

- collective operations
- user-defined datatypes to describe both memory and file layout
- communicators to separate application-level message passing from I/O-related message passing
- non-blocking operations

Reading and writing in parallel is like receiving and sending messages.

An implementation of MPI-IO is typically layered on top of a parallel file system that supports the notion of a single, common file shared by multiple processes.

MPI-IO Libraries at LRZ

PE IBM

IBM Parallel Environment (MPI-IO implementation a ROMIO version)

- Default MPI distribution in SuperMUC Phase 1 and Phase 2

Intel MPI

Intel MPI (MPI-IO implementation a ROMIO version) Set the I_MPI_EXTRA_FILESYSTEM environment variable to on to enable parallel file system support. Set the I_MPI_EXTRA_FILESYSTEM_LIST environment variable to request native support for the specific file system Version:

- Intel MPI 2017, Intel MPI 2018
- Intel MPI 5.1

OpenMPI

OpenMPI is used for research and experimental purposes.

File Info

```
MPI_Info info
```

- Parameter in MPI_FILE_OPEN, MPI_FILE_SET_VIEW, MPI_FILE_SET_INFO

Generate new, empty Info object:

```
MPI_INFO_CREATE(info,ierror)
```

Add entry to existing Info object:

```
MPI_INFO_SET(info, key, value, ierror)
```

MPI-IO Hints (1)

Data Sieving:

- ind_rd_buffer_size Controls the size (in bytes) of the intermediate buffer used when performing data sieving during read operations.
- ind_wr_buffer_size Controls the size (in bytes) of the intermediate buffer when performing data sieving during write operations.
- romio_ds_read Determines when ROMIO will choose to perform data sieving for read. Valid values are enable, disable, or automatic.
- romio_ds_write Determines when ROMIO will choose to perform data sieving for write. Valid values are enable, disable, or automatic.

MPI-IO Hints (2)

Collective buffering (Two-Phase I/O)

- cb_buffer_size Controls the size (in bytes) of the intermediate buffer used in two-phase collective I/O.
- cb_nodes Controls the maximum number of aggregators to be used.
- romio_cb_read Controls when collective buffering is applied to collective read operations. Valid values are enable, disable, and automatic.
- romio_cb_write Controls when collective buffering is applied to collective write operations.
- romio_no_indep_rw This hint controls when “deferred open” is used.
- cb_config_list Provides explicit control over aggregators. *:1 One process per hostname (i.e., one process per node).

Setting Info object

Using info object in the program

```
integer info, ierr
call MPI_Info_create(info, ierror)
call MPI_Info_set(info, 'romio_cb_read', 'disable', ierr)
call MPI_Info_set(info, 'romio_cb_write', 'disable', ierr)
...
call MPI_File_open(comm, filename, amode, info, fh, ierror)
```

User can define a list of hints in a single file which are going to be set up at execution time for his parallel application.

```
>cat $HOME/romio-hints
romio_cb_read disable
romio_cb_write disable
```

Setting for ROMIO HINTS:

```
export ROMIO_HINTS=$HOME/romio-hints
```

MPI-IO in SuperMUC (1)

Logical View of a File – **Strided** Access Pattern

P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P3	P4

Logical View of a File – **Sequential** Access Pattern

P1	P1	P1	P1	P2	P2	P2	P2	P3	P3	P3	P3	P4	P4	P4	P4

- Request = 1MiB and 256MiB
- Access Pattern = Strided and Sequential
- Sixteen I/O processes were allocated for each compute node.
- The amount of data written and read is 64GiB per CN.
- To set up the hints along the execution, the variable ROMIO HINTS is used:

  Hints file $HOME/romio-hints:

  ```
  romio_cb_read disable
  romio_cb_write disable
  ```

  Setting for ROMIO_HINTS:

  ```
  export ROMIO_HINTS=$HOME/romio-hints
  ```

MPI-IO in SuperMUC(2)

Exp.	MPI Processes	Compute Nodes	Access Pattern	Request Size	Hints	Transfer Rate(GiB/sec)
1	512	32	Sequential	1 MiB	romio_cb_read = automatic romio_cb_write = automatic	write = 25.92 read = 23.80
					romio_cb_read = disable romio_cb_write = disable	write = 75.34 read = 67.58
					independent I/O	**write = 80.39** **read = 69.62**
2	512	32	Strided	1 MiB	romio_cb_read = automatic romio_cb_write = automatic	write = 1.63 read = 17.74
					romio_cb_read = enable romio_cb_write = enable	**write = 25.49** **read = 26.10**
					independent I/O	write = 5.15 read = 12.60
3	512	32	Sequential	256 MiB	romio_cb_read = automatic romio_cb_write = automatic	write = 74.48 read = 46.67
					romio_cb_read = disable romio_cb_write = disable	**write = 83.37** **read = 65.88**
					independent I/O	write = 82.29 read = 64.22
4	512	32	Strided	256 MiB	romio_cb_read = automatic romio_cb_write = automatic	write = 71.12 read = 41.80
					romio_cb_read = disable romio_cb_write = disable	**write = 77.22** **read = 70.21**
					romio_cb_read = enable romio_cb_write = enable	write = 24.81 read = 24.90
					independent I/O	write = 71.25 read = 67.71

Summary: A General I/O Guide

General guides for the I/O (1)

- Avoid unnecessary I/O. For example: switch off debug output for production runs.
- Perform I/O in few and large chunks. In parallel file systems, the chunk size should be multiple of the block size or stripe size.
- Prefer binary/unformatted I/O instead of formatted data.
- Avoid unnecessary/large-scale open/close statements. Remember that metadata operations are latency bound.
- Use appropriate file system. Parallel file systems may be bad or not scale well for metadata operations, but provide high/scalable bandwidth. NFS-based file systems may show the reversed behaviour.

General guides for the I/O (2)

- Avoid explicit flushes of data to disk, except when needed for consistency reasons.
- Use specialized I/O libraries based on the I/O requirements of your applications. These provide more portable way of writing data and may reduce metadata load when properly used.
- Convert to target / visualization format in memory if possible.
- For parallel programs, the strategy a file per process could provide highest throughput, but usually this needs post-processing. Strategies shared files are recommendable for an I/O intensive parallel applications. MPI-IO is more suitable for large scale systems.

General guides for the I/O (3)

- May need to use library/compiler support for conversion. If you need to transfer the binary files between different architectures, consider the little vs. big-endian byte order. Limitations may apply on file sizes and data types.
- When accesses are non-contiguous, users must create derived datatypes, define file views, and use the collective I/O functions
- Use of MPI I/O is often limited to parallel file systems; do not expect to obtain good performance using it in $HOME (NFS).