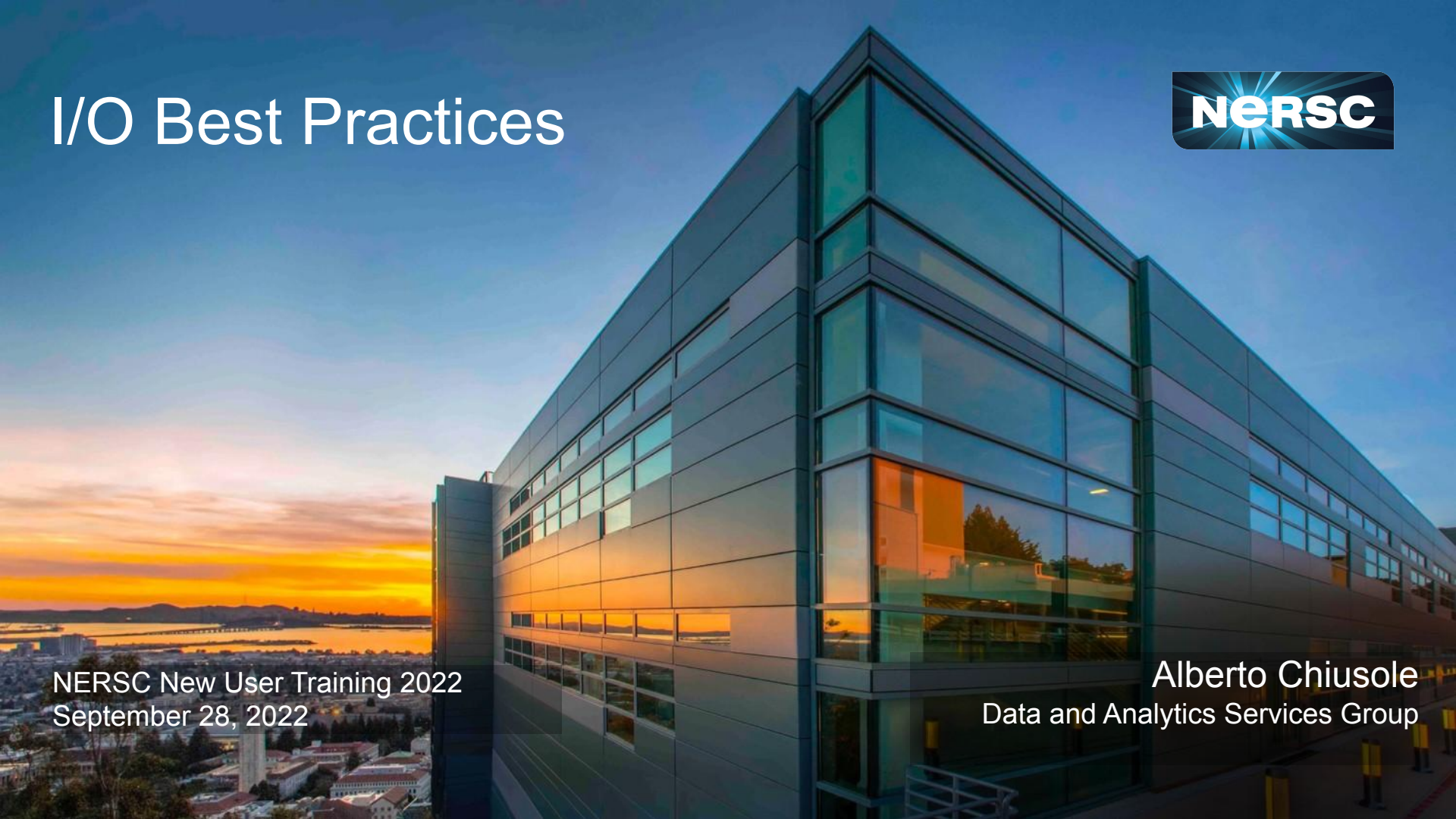
I/O Best Practices



NERSC New User Training 2022
September 28, 2022

Alberto Chiusole
Data and Analytics Services Group

Outline

- Parallel I/O
  - I/O stack overview
  - I/O Libraries: MPI-IO, HDF5, h5py
  - Independent and Collective Parallel I/O
  - I/O Pattern Analysis: Contiguous, Non-contiguous, Random, etc.
  - h5py and MPI-IO tips and tricks
  - I/O Profiling: Darshan
- NERSC Storage Systems
- Lustre striping guidelines
- I/O and metadata best practices

I/O Stack: Moving Data To Disk

Productivity Interface is a thin layer on top of existing high performance I/O libraries, for productive big data analytics

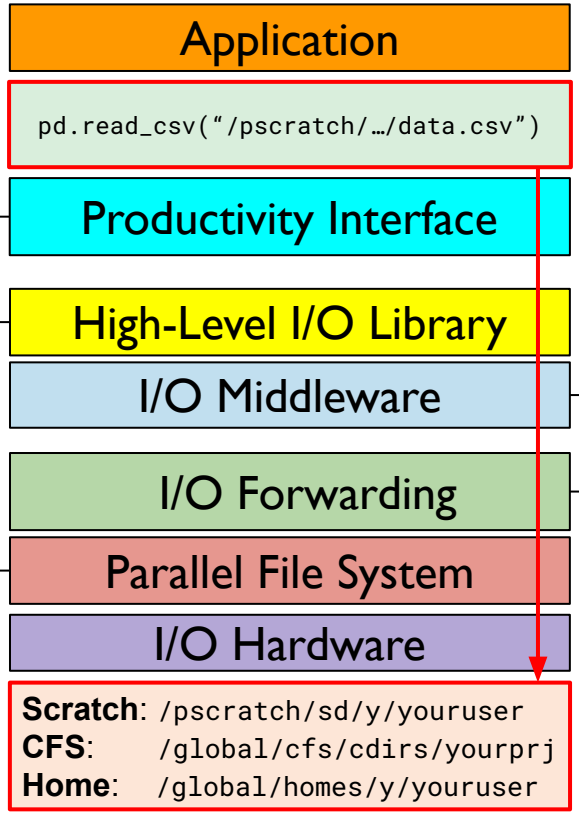
Python, Spark, TensorFlow

High Level I/O Libraries map application abstractions onto storage abstractions and provide data portability.

HDF5, Parallel netCDF, ADIOS

Parallel file systems maintain a logical file model and provide efficient access to data.

Lustre, GPFS, PVFS, PanFS



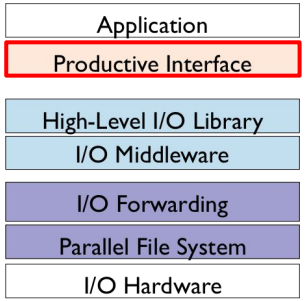
I/O Middleware organizes accesses from many processes, especially those using collective I/O.

MPI-IO, GLEAN, PLFS

I/O Forwarding transforms I/O from many clients into fewer, larger requests; reduces lock contention; and bridges between the HPC system and external storage.

Cray DVS, IBM CIOD, IOFSL

Productive I/O Interface: h5py



- Parallel h5py example:

```
from mpi4py import MPI
import h5py
fx=h5py.File('output.h5', 'w', driver='mpio', comm=MPI.COMM_WORLD)
```

```
dset[start:end,:]=temp
```

Independent I/O

```
with dset.collective:
  dset[start:end,:]=temp
```

Collective I/O

Coding Effort: h5py vs C HDF5

```python
from mpi4py import MPI
import numpy as np
import h5py
import time
import sys
comm =MPI.COMM_WORLD
nproc = comm.Get_size()
comm.Barrier()
timefstart=MPI.Wtime()
f = h5py.File(filename, 'w', driver='mpio', comm=MPI.COMM_WORLD)
dset = f.create_dataset('test', (length_x,length_y), dtype='f8')
comm.Barrier()
timefend=MPI.Wtime()
f.atomic = False
length_rank=length_x / nproc
length_last_rank=length_x -length_rank*(nproc-1)
comm.Barrier()
timestart=MPI.Wtime()
start=rank*length_rank
end=start+length_rankL
if rank==nproc-1: #last rank
    end=start+length_last_rank
temp=np.random.random((end-start,length_y))
comm.Barrier()
timemiddle=MPI.Wtime()
if colw==1:
        with dset.collective:
                dset[start:end,:] = temp
        dset[start:end,:] = temp
comm.Barrier()
timeend=MPI.Wtime()
f.close()
```

```c
#include "stdlib.h"
dataspace_id2 = H5Screate_simple(2, dims2, NULL);
dset_id2 = H5Dcreate(file_id2,dataset, H5T_NATIVE_DOUBLE, dataspace_id2, ...)
H5Sclose(dataspace_id2);
MPI_Barrier(comm);
double t00 = MPI_Wtime();
result_offset[1] = 0;
result_offset[0] =  (dims_x / mpi_size) * mpi_rank;
result_count[0] = dims_x / mpi_size;
result_count[1] = dims_y;
if(mpi_rank==mpi_size-1)
result_count[0] = dims_x / mpi_size + dims_x % mpi_size;
result_space = H5Dget_space(dset_id2);
H5Sselect_hyperslab(result_space, H5S_SELECT_SET, result_offset, ...);
result_memspace_size[0] = result_count[0];
result_memspace_size[1] = result_count[1];
result_memspace_id = H5Screate_simple(2, result_memspace_size, NULL);
  else{
   H5Dwrite(dset_id2, H5T_NATIVE_DOUBLE, result_memspace_id,...);
  }
  MPI_Barrier(comm);

  double t1 = MPI_Wtime()-t0;
  free(data_t);
  double tclose=MPI_Wtime();
  H5Sclose(result_space);
  H5Sclose(result_memspace_id);
  H5Dclose(dset_id2);
  H5Fclose(file_id2);
  tclose=MPI_Wtime()-tclose;
  MPI_Finalize();
}
```

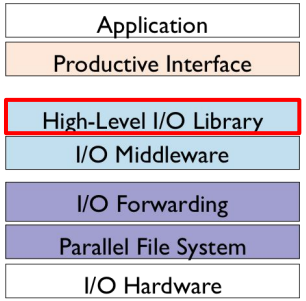
Python vs C Performance: h5py vs HDF5 C

Question: When you improve productivity, how much performance do you lose?

		Single Node	Multi-nodes
Metadata	1k File Creation	63.8%	
	1k Object Scanning	60.0%	
Independent I/O	Weak Scaling	97.8%	100%
	Strong Scaling	100%	97.1%
Collective I/O	Weak Scaling	100%	90%
	Strong Scaling	98.6%	87%

HDF5 vs. h5py: https://www.nersc.gov/assets/Uploads/H5py-2017-May26-public.pdf

High-level I/O Libraries



- Provide high-performance parallel I/O, while reducing complexity
  - Object-oriented data model that allows users to specify complex data relationships and dependencies
  - Have self-describing, machine-independent data formats that are suitable for array-oriented scientific data

- Examples:
  - HDF5:
    - HDF Group / LBNL, started in 1997
    - Very popular: in top 5 libraries at NERSC
  - Parallel netCDF:
    - Unidata / NWU / ANL, started in 2001
  - ADIOS:
    - ORNL / SNL, started in 2009

High-level I/O Libraries: HDF5

- Serial HDF5 example:

```
fapl_id = H5Pcreate(H5P_FILE_ACCESS);

file_id = H5Fcreate(FNAME,…, fapl_id);
space_id = H5Screate_simple(…);
dset_id = H5Dcreate(file_id, DNAME, H5T_NATIVE_INT, space_id,…);


status = H5Dwrite(dset_id, H5T_NATIVE_INT, …, xf_id);
```

High-level I/O Libraries: HDF5

- Parallel HDF5 example:

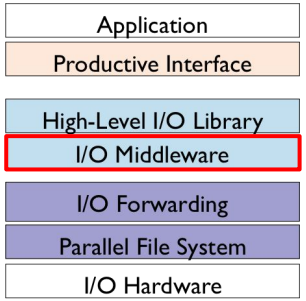
```
MPI_Init(&argc, &argv);
…
fapl_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(fapl_id, comm, info);
file_id = H5Fcreate(FNAME,…, fapl_id);
space_id = H5Screate_simple(…);
dset_id = H5Dcreate(file_id, DNAME, H5T_NATIVE_INT, space_id,…);
xf_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(xf_id, H5FD_MPIO_COLLECTIVE);
status = H5Dwrite(dset_id, H5T_NATIVE_INT, …, xf_id);
…
MPI_Finalize();
```

High-level I/O Libraries: HDF5

Parallel HDF5 Tutorials

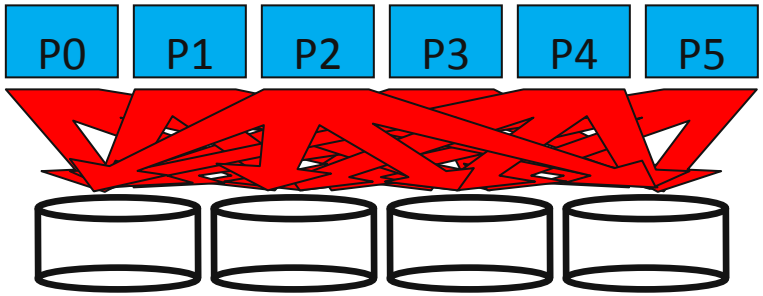
- NERSC:
  - https://docs.nersc.gov/development/libraries/hdf5/
  - https://www.nersc.gov/users/training/online-tutorials/introduction-to-scientific-i-o/
- Argonne National Lab:
  - Introduction to HDF5 for HPC Data Models, Analysis, and Performance (https://www.alcf.anl.gov/events/introduction-hdf5-hpc-data-models-analysis-and-performance), Scot Breitenfeld, HDF group
  - ATPESC: https://extremecomputingtraining.anl.gov/files/2019/08/ATPESC_2019_Track-3_6_8-2_130pm_Koziol-Scalable_HDF5.pdf
- The HDF Group:
  - https://portal.hdfgroup.org/display/HDF5/Parallel+HDF5

I/O Middleware



- More I/O Software! Why?
  - I/O middleware: performance portability between parallel file systems
  - Reduces or eliminates optimization in application code
- MPI-IO
  - Standardized I/O Interface specification for MPI applications
  - Same access model as POSIX: byte-stream in file
  - Features:
    - Collective I/O operations
    - Non-contiguous I/O w/MPI datatypes & file views
    - Non-blocking I/O
    - FORTRAN (and other) language bindings
    - Method of encoding files in a portable format (external32)

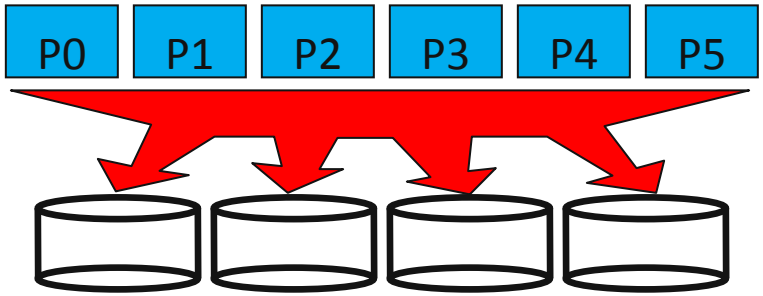
Independent and Collective Parallel I/O



Independent I/O

- Independent I/O operations specify only what a single MPI process will do
  - Independent I/O calls do no pass on relationships between I/O to other processes
- Why use independent I/O?
  - Sometimes the synchronization of collective calls is not natural
  - Sometimes the overhead of collective calls outweighs their benefits
    - Example: Very small metadata I/O operations

Independent and Collective Parallel I/O



Collective I/O

- Collective I/O operations are coordinated access by a group of MPI processes
  - Collective I/O routines must be called by all processes that opened the file
- Why use collective I/O?
  - Allows I/O middleware to get a global perspective on entire access from all processes, providing more opportunities for optimization in lower software layers
  - When used for non-contiguous access patterns, collective I/O typically yields the best performance

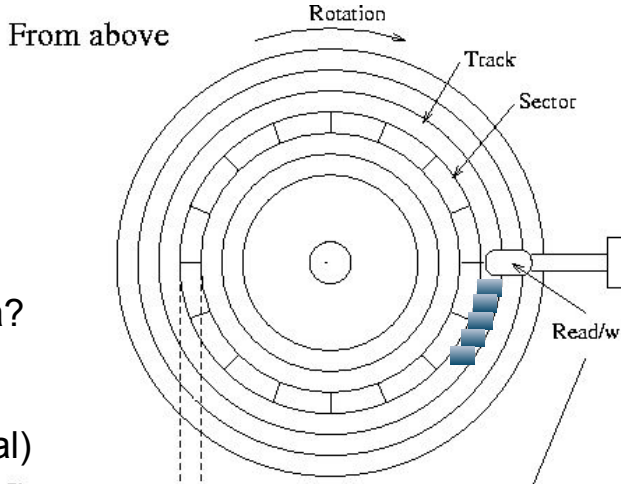
I/O Middleware

- MPI-IO Tutorials:
  - NERSC:
    - https://docs.nersc.gov/performance/io/library/
  - Argonne National Lab:
    - https://extremecomputingtraining.anl.gov/files/2019/08/ATPESC_2019_Track-3_4_8-2_1030am_Latham-Introduction_to_MPI_IO.pdf
  - NCSA:
    - http://wgropp.cs.illinois.edu/courses/cs598-s16/lectures/lecture32.pdf

I/O Pattern Analysis

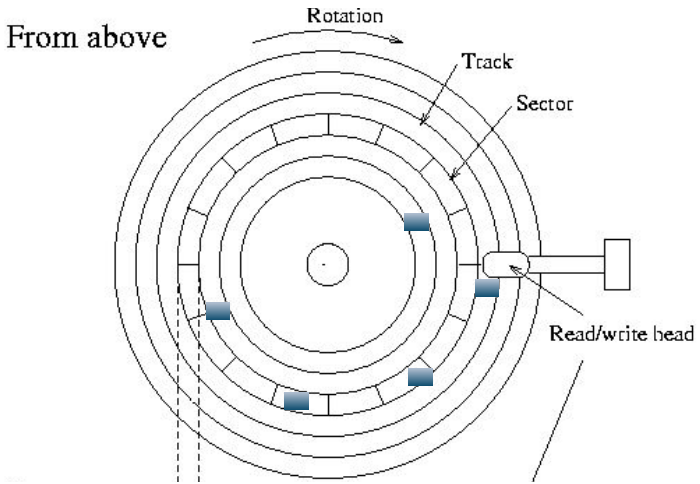
How to describe your I/O

- Number of Processes
- Number of Files
- Size per file
- Frequency of I/O
- Size per I/O
- Read, Write, Metadata?
- Shared File or not
- I/O Libraries
- Contiguous (Sequential) vs Non-contiguous (Random) access pattern
- Data alignment
- ...



Contiguous I/O

- Read time: 0.1ms



Noncontiguous I/O

- Seek time: 4ms
- Rotation time: 3ms
- Read time: 0.1 ms
- Total time = 7.1ms

h5py and MPI-IO tips and tricks

Latest HDF5 file format can speed up execution time:

```
f = h5py.File('name.h5', libver='earliest') # most compatible, default
f = h5py.File('name.h5', libver='latest')   # most modern
```

Modern HDF5 file format performed 2.5x faster in h5py than backward compatible format when creating 1 file w/8k objects inside, on 1 processor

MPI-IO hints can control collective I/O internal algorithms

```
export MPICH_MPIIO_HINTS="*:romio_cb_write=enable:romio_ds_write=disable"
```

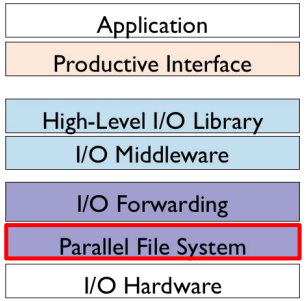
https://docs.nersc.gov/performance/io/

https://docs.nersc.gov/performance/io/library/

I/O Profiling: Darshan

- Darshan (https://docs.nersc.gov/tools/performance/darshan/)
  - Lightweight HDF5/MPI-IO/POSIX I/O profiling tool, developed by ANL
  - Loaded by default at NERSC: currently version 3.3.1 (soon 3.4.0)
    - `module av -S darshan → darshan/3.3.1  darshan/3.3.1-hdf5`
- Darshan Logs
  - Location: **/pscratch/darshanlogs/<year>/<month>/<day>/**
    - More than 1000 logs/day (more expected when system in production)
  - Filename format: username_jobname_jobid_date_uniqueid_timing.darshan
    - zisheng_vasp_id31418939_23851_9-20-57716-7672284139867200748_1.darshan
- Darshan Scripts (`darshan-parser`, `darshan-job-summary.pl`, etc)
- [NEW] PyDarshan (https://www.mcs.anl.gov/research/projects/darshan/docs/pydarshan/)
  - `pip install darshan`

NERSC Storage Systems



NERSC File Systems: https://docs.nersc.gov/filesystems/

Location	File System	Visibility	Access	Backups	Snapshots	Purged
PM Scratch	Lustre	Local (system)	User	No	No	Yes (8 weeks)
Community	GPFS	Global	Project	No	Yes	No
Common	GPFS	Global	Project	No	No	No
Home	GPFS	Global	User	Yes	Yes	No
Archive	HPSS	Global	User	No	No	No

Perlmutter Scratch: Overview

Lustre is a high-performance parallel file system

- POSIX File System
  - Directories & files, ownership, permissions
- Each file's data can be striped over multiple storage servers ("OSTs")
  - Default is a "stripe count" of 1, i.e. not striped
  - "Stripe size" is amount of data in each stripe (default 1 MB)
  - Data "round robin-ed" over # of OSTs for file
- Files inherit striping configuration of directory where they are created
- More details at: https://docs.nersc.gov/performance/io/lustre/

Lustre striping guidelines

- When to use striping
  - On medium-to-large files when doing Collective I/O
  - On large files when doing Independent I/O (one-file-per-process)
  - **`mv`** doesn't restripe existing files
- Example
  - ~24 GB HDF5 input files?

```
$ mkdir inputdir_striped/
$ stripe_medium inputdir_striped/
$ cp inputdir/* inputdir_striped/
```

	Single Shared-File I/O	File per Process
File size	Command	Command
< 1 GB	keep default striping	keep default striping
1 - 10 GB	`stripe_small`	keep default striping
10 - 100 GB	`stripe_medium`	keep default striping
100 GB - 1 TB	`stripe_large`	keep default striping
> 1 TB	`stripe_large`	`stripe_large`

I/O and metadata best practices

- Data alignment, especially for Independent I/O
  - I/O should match file system granularity
  - 1 MB stripe width on Perlmutter Scratch, configurable with `lfs`
- Combine operations using Collective I/O
- Profile your application for I/O: Darshan
- Aggregate small files into larger files whenever possible
  - Reduces metadata overhead. Read-ahead optimizations
- Do not write thousands of files in a single directory
  - Has performance impact on your application and others'
- [HDF5] **compact storage** when working with small data
  - <64KB data stored in the metadata layer, can be cached


Welcome to
NERSC!
Questions?
NERSC