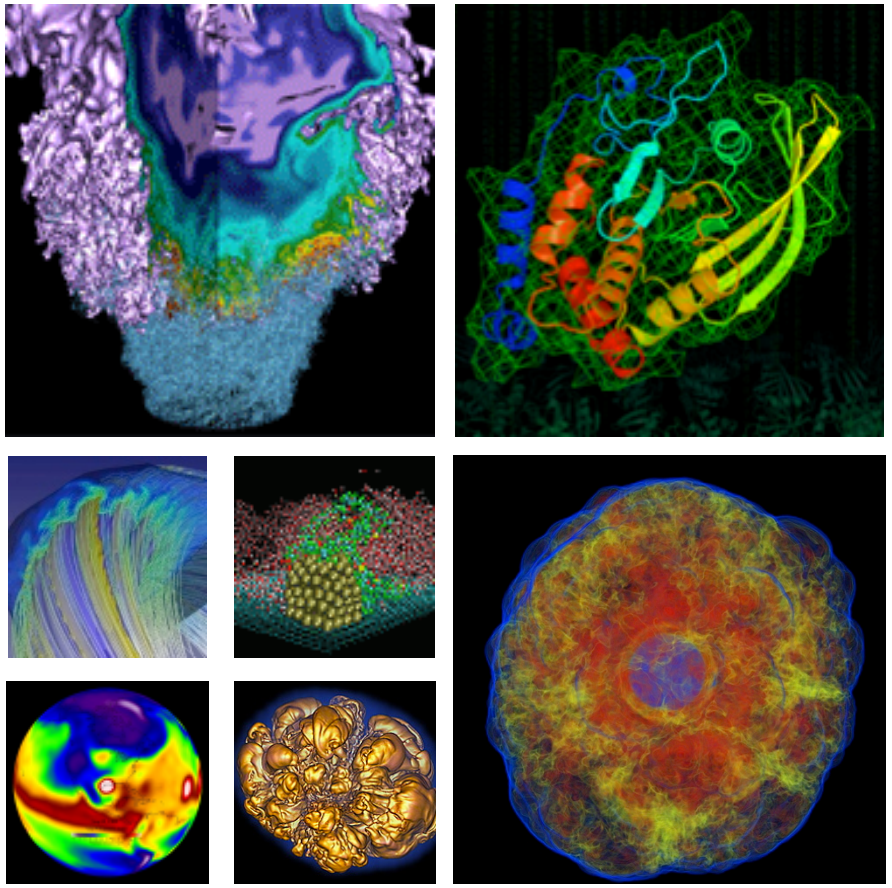
Introduction to High Performance Parallel I/O



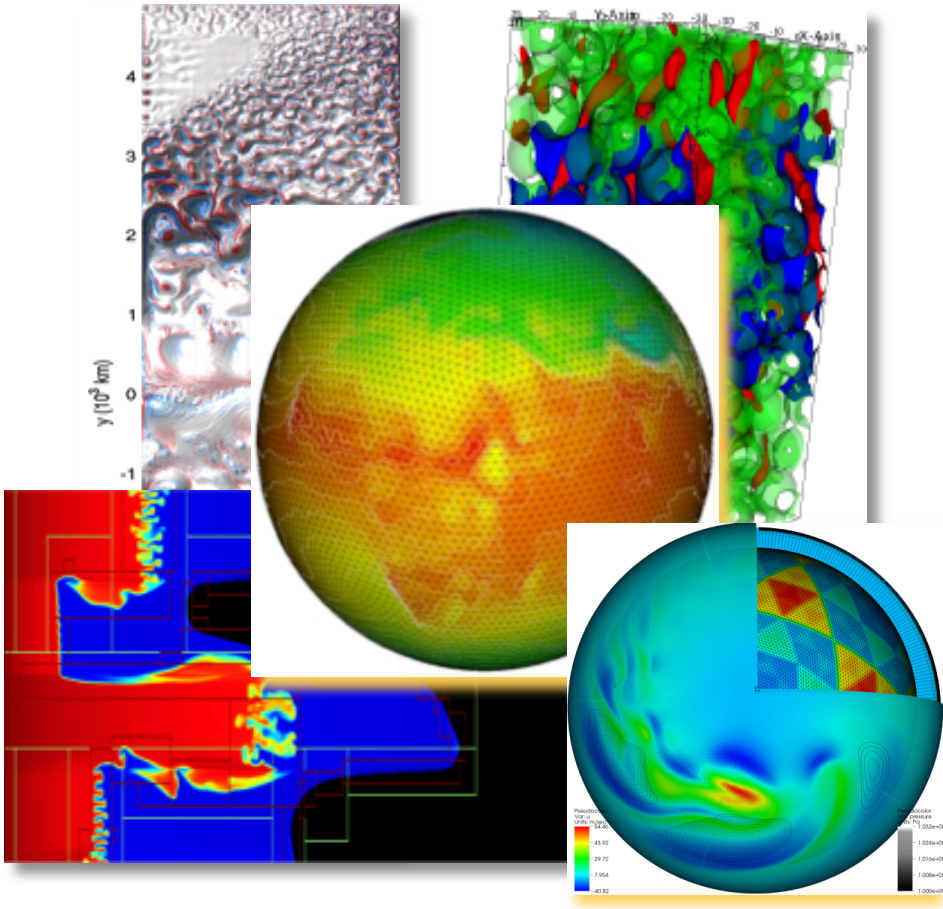
Richard Gerber
Deputy Group Lead
NERSC User Services

August 6, 2013

Some slides from Katie Antypas

I/O Needs Getting Bigger All the Time

- I/O needs growing each year in scientific community
- I/O is a bottleneck for many
- Data sizes growing with system memory
- For large users I/O parallelism is mandatory
- Titan has 10 PB of disk and Blue Waters 22 PB!!!



Images from David Randall, Paola Cessi, John Bell, T Scheibe

Outline

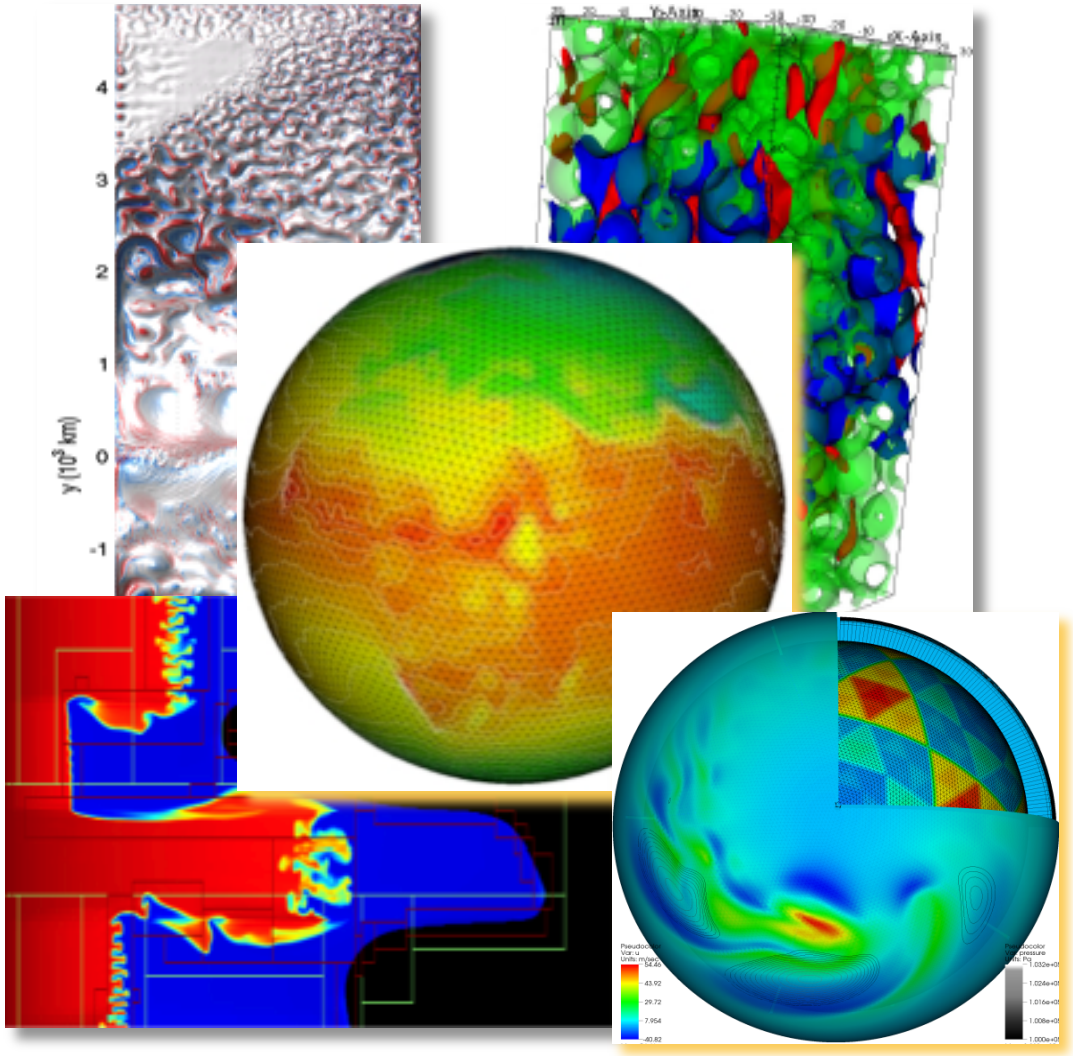
- Storage Architectures
- File Systems
- I/O Strategies
- MPI I/O
- Parallel I/O Libraries

Is it I/O? IO? I-O?

I say, who cares?

Why is Parallel I/O for science applications difficult?

- Scientists think about data in terms of how a system is represented in the code: as grid cells, particles, …
- Ultimately, data is stored on a physical device
- Layers in between the application and the device are complex and varied
- I/O interfaces are often arcane and complicated



Images from David Randall, Paola Cessi, John Bell, T Scheibe

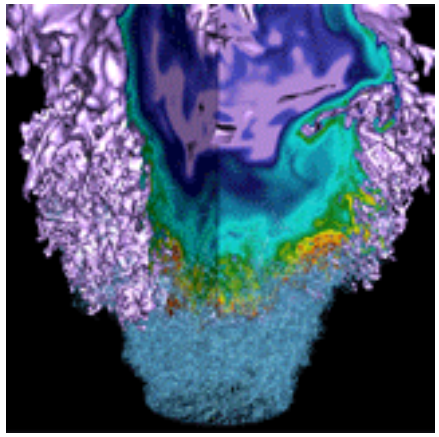
System I/O Architecture

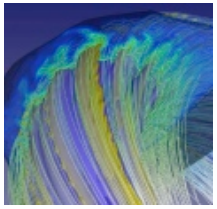
- **Should an application scientist/programmer care about what the I/O system looks like?**
  - Yes! It would be nice not to have to, but performance and perhaps functionality depend on it.
  - You may be able to make simple changes to the code or runtime environment that make a big difference.
  - **Inconvenient Truth:** *Scientists need to understand their I/O in order to get good performance*

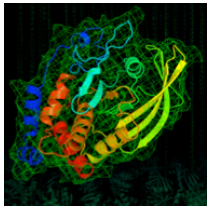
or acceptable

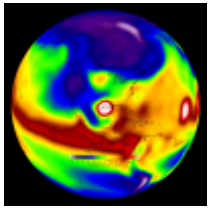
This may be mitigated by using I/O libraries.

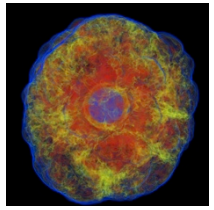
Storage Architectures

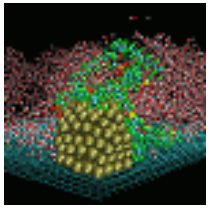




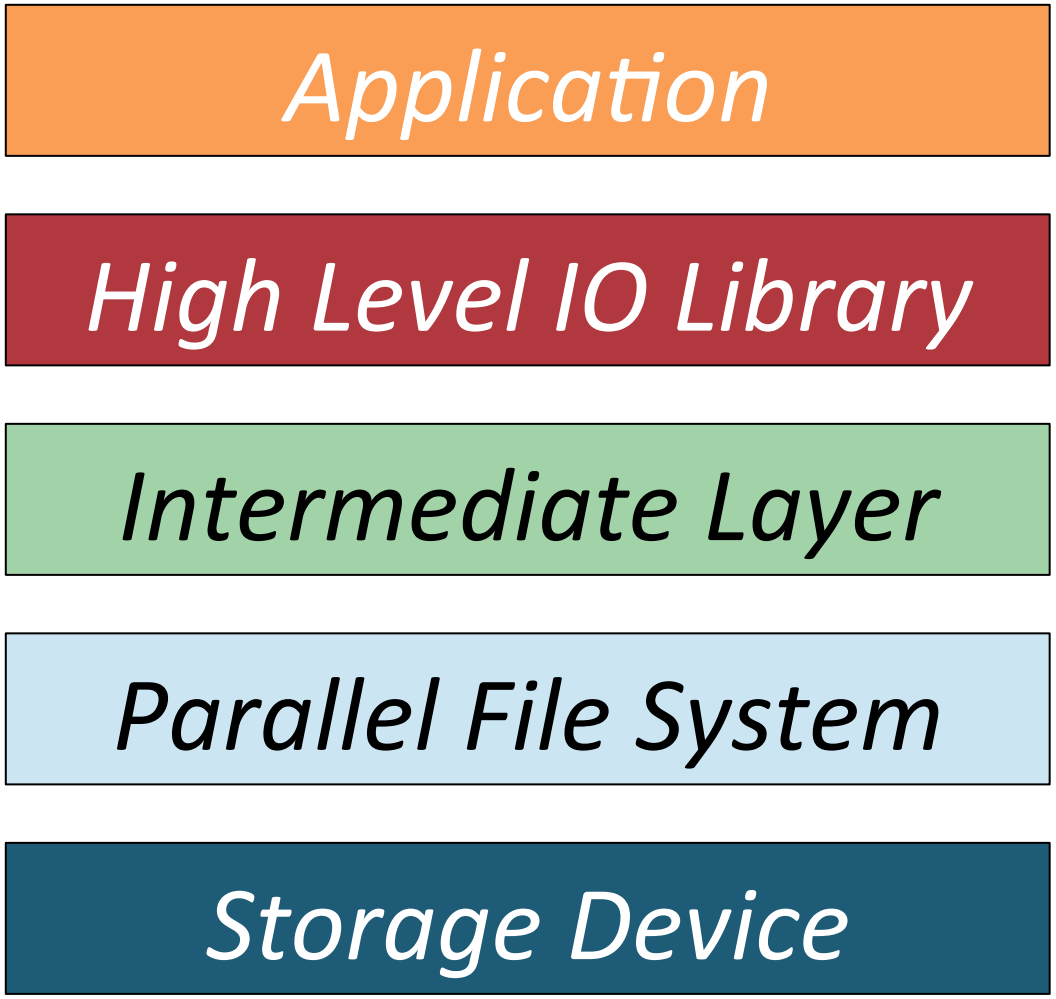








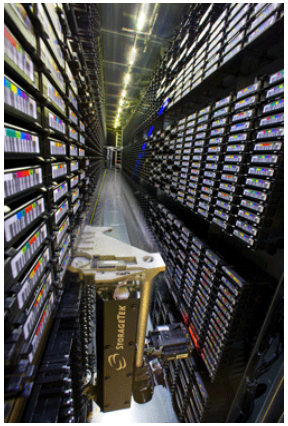
Simplified I/O Hierarchy



May be MPI IO

Storage Devices

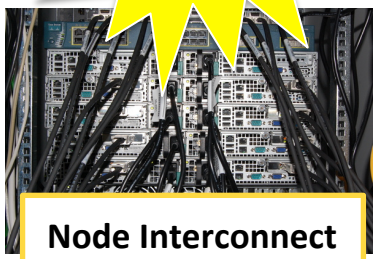
- **Usually we'll be talking about arrays of hard disks**
- **FLASH "drives" are being used as fast "disks," but are expensive**
  - "Burst buffers" coming soon
- **Magnetic tapes are cheap, but slow and probably don't appear as standard file systems**



Some Definitions

- **Capacity (in MB, GB, TB, PB)**
  - Depends on area available on storage device and the density data can be written
- **Transfer Rate (bandwidth) – MB/sec or GB/sec**
  - Rate at which a device reads or writes data
  - Depends on many factors: network interfaces, disk speed, etc.
  - Be careful with parallel BW numbers: aggregate? per what?
- **Access Time (latency)**
  - Delay before the first byte is read
- **Metadata**
  - A description of where and how a file or directory is stored on physical media
  - Is itself data that has to be read/written
  - Excessive metadata access can limit performance

Latencies



1,000 ns

Node Interconnect

1,000,000 ns

Disk Storage

100 ns

Main Memory (Instructions & Data)

10 ns

Memory Interface

L2 Cache

L1 Data Cache

L1 Instr. Cache

1 ns

ST

LD

INT Registers

Shift

INT

FP Registers

FMA

FP add

FP mult

CPU

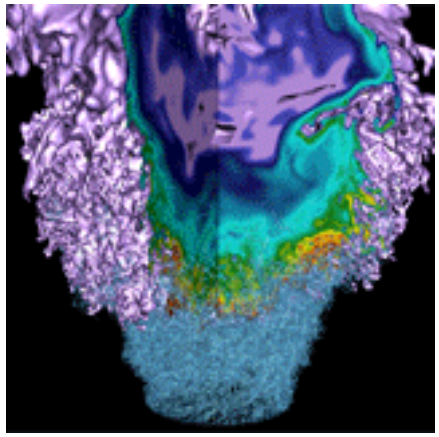
Bandwidths

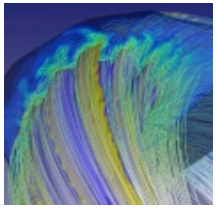
- **How fast can you stream data from your application to/ from disk?**
- **Once you pay the latency penalty, HPC system BWs are large.**
- **~ 10s to now 100s GB/sec**
- **To take advantage of this, read/write large chunks (MBs) of data**
- **Serial bandwidths < 1 GB/sec**
  - Limited by interfaces and/or physical media
- **You need parallelism to achieve high aggregate bandwidth**

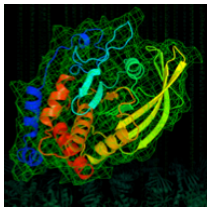
File Buffering and Caching

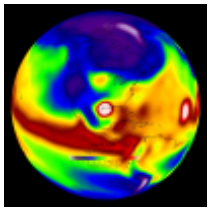
- **Buffering**
  - Used to improve performance
    - File system collects full blocks of data before transferring data to disk
    - For large writes, can transfer many blocks at once
- **Caching**
  - File system retrieves an entire block of data, even if all data was not requested, data remains in the cache
- **Can happen in many places, compute node, I/O server, disk**
- **Not the same on all platforms**
- **Important to study your own application's performance rather than look at peak numbers**

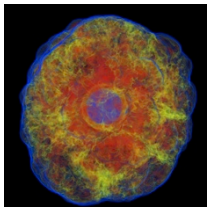
File Systems

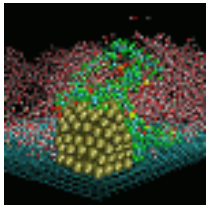












NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

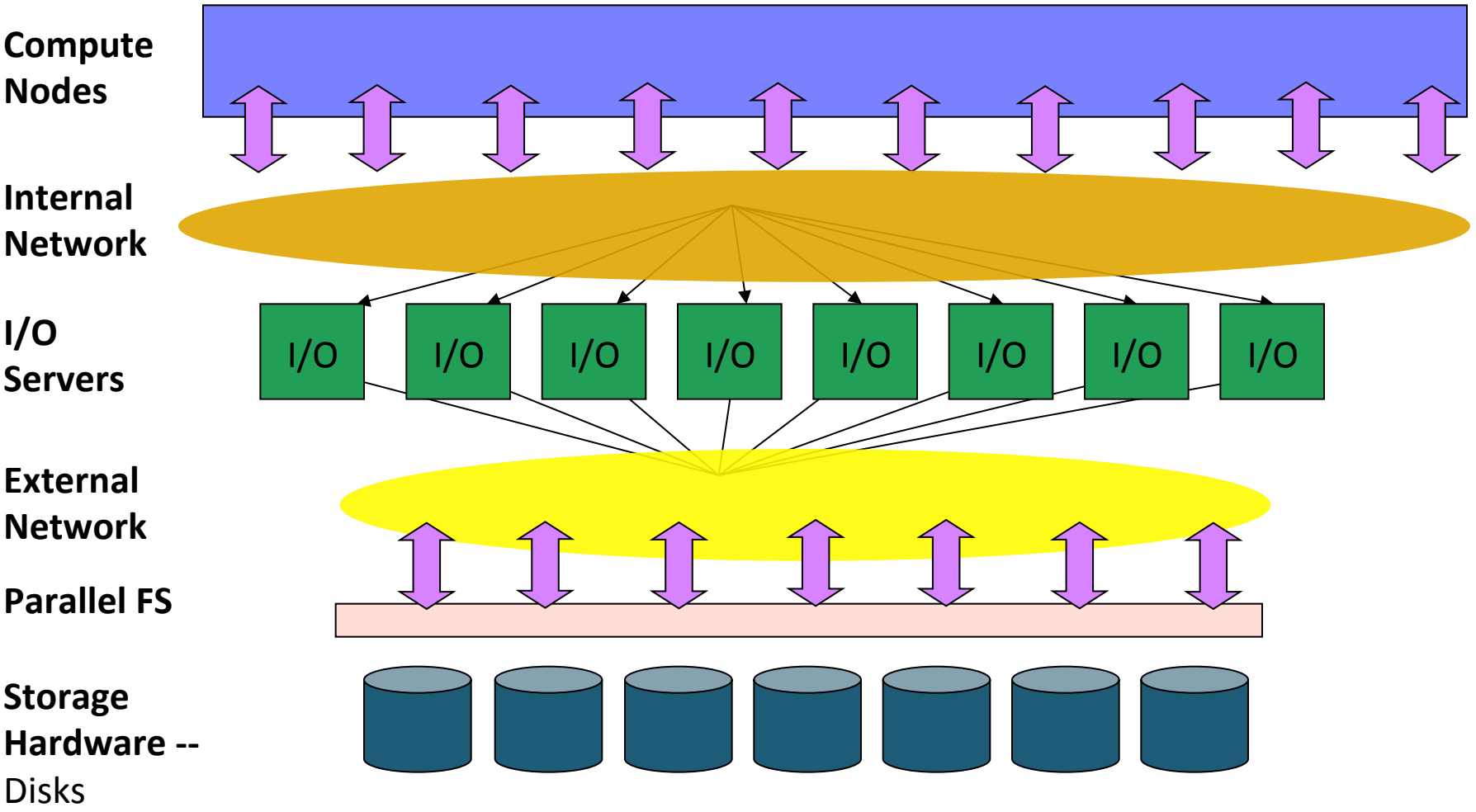
Local vs. Global File Systems

- **"On-board" (the old "local")**
  - Directly attached to motherboard via some interface
  - Few HPC systems have disks directly attached to a node
- **"Local" in HPC: Access from one system**
  - Network attached PB+ file systems
    - Via high-speed internal network (e.g. IB,Gemini, Airies)
    - Direct from node via high-speed custom network (e.g. IB,FibreChannel)
    - Ethernet
  - Contention among jobs on system
- **"Global": Access from multiple systems**
  - Networked file system
  - Activity on other systems can impact performance
  - Useful for avoiding data replication, movement among systems

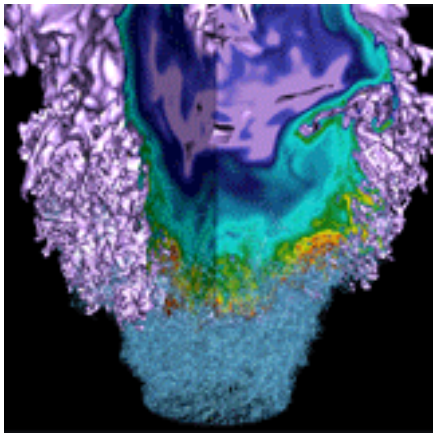
Top Parallel File Systems Used in HPC

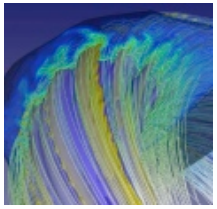
GPFS

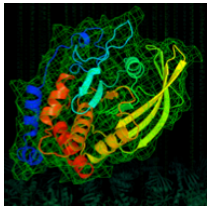
Generic Parallel File System Architecture

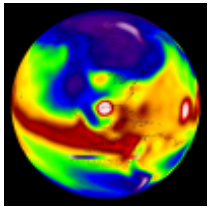


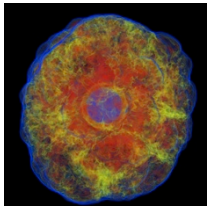
I/O Strategies

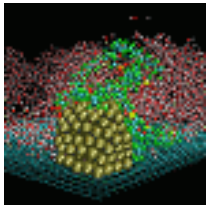












Application I/O

- All significant I/O performed by your job should use the file system designed for HPC applications.
- Home directories are not optimized for large I/O performance.
- Consult your center's documentation.

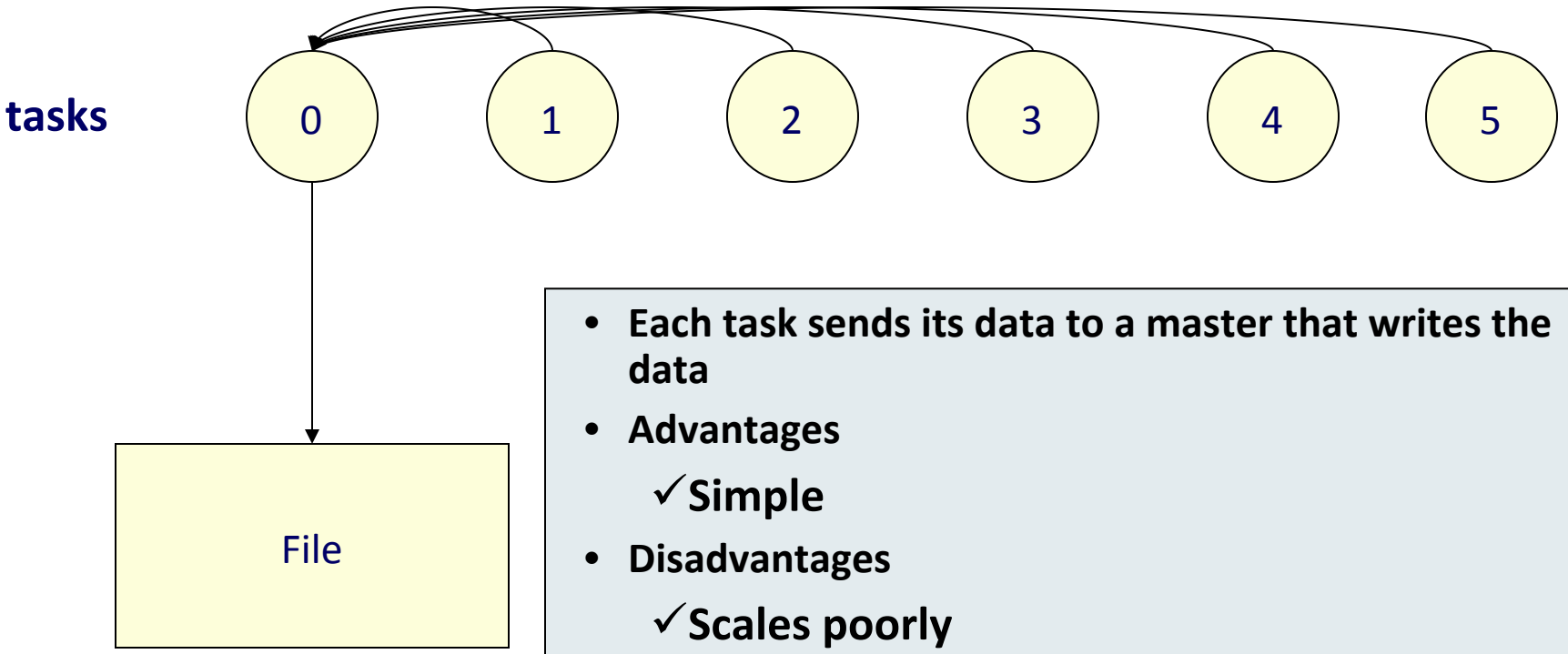
Parallel I/O: A User Wish List

- Easy to program
- Get acceptable performance
  - Users tell us that I/O should be on the order of 10% of run time or less
- Have data files portable among systems
- Write data from many processors into a single file
- Read data from any number of tasks (i.e., you want to see the logical data layout… not the physical layout)
- Be able to easily use M tasks to read a data file written using *N* tasks

High Level I/O Strategies

- Single task does all I/O
- Each task writes to its own file
- All tasks write to single shared file
- n<N tasks write to a single file
- n_1<N tasks write to n_2<N files

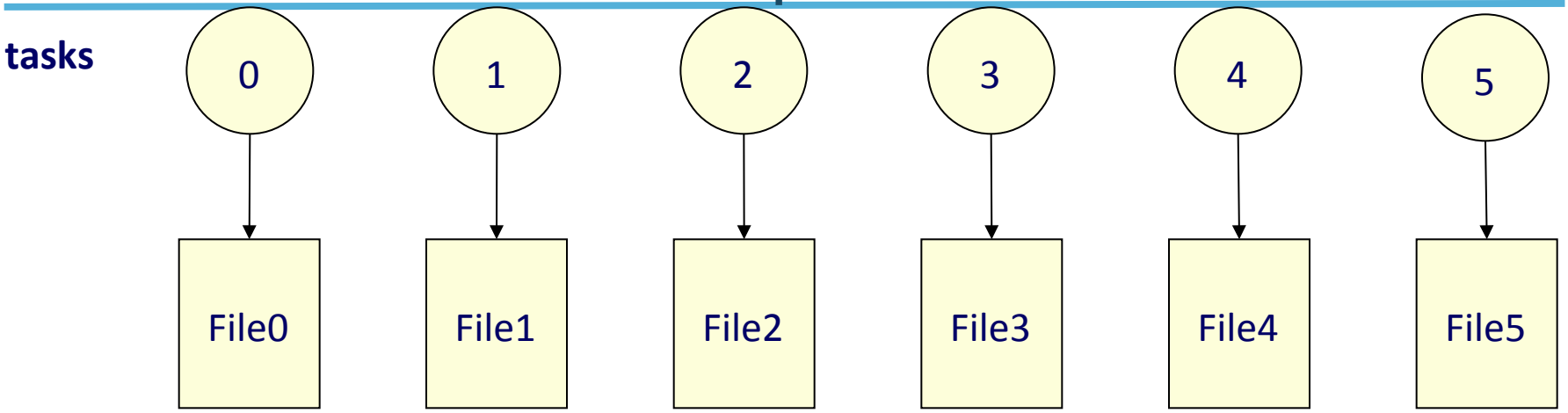
Serial I/O



- Each task sends its data to a master that writes the data
- Advantages
  - ✓Simple
- Disadvantages
  - ✓Scales poorly
  - ✓May not fit into memory on task 0
  - ✓Bandwidth from 1 task is very limited

Parallel I/O Multi-file

Each Processors Writes Its Data to Separate File



Advantages

Easy to program

Can be fast
(up to a point)

Disadvantages

Many files can cause serious performance problems

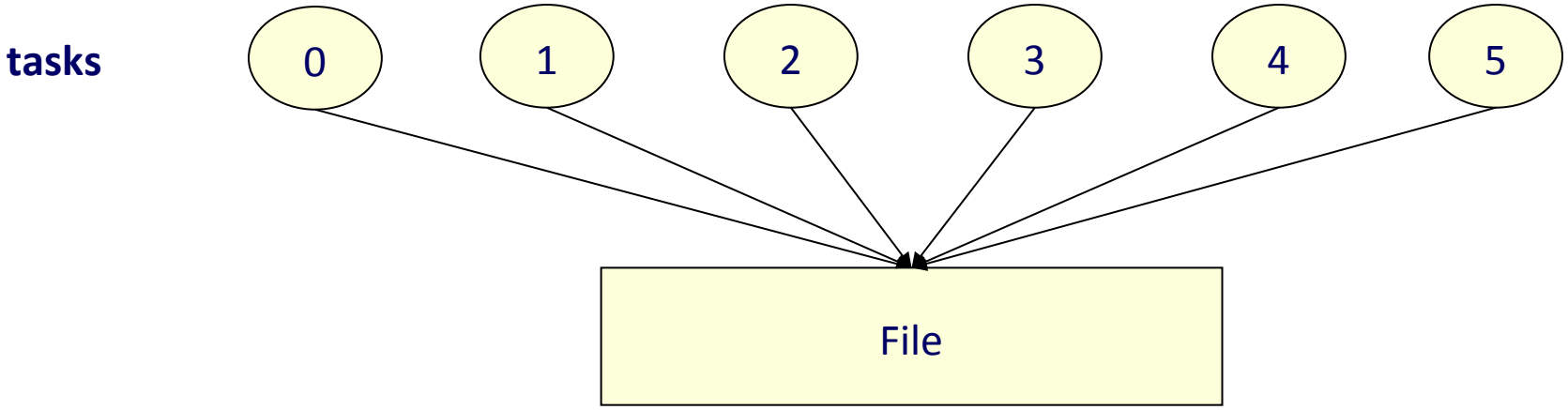
Hard for you to manage 10K, 100K or 1M files

Flash Center IO Nightmare…

- 32,000 processor run on LLNL BG/L
- Parallel IO libraries not yet available
- Every task wrote
  - Checkpoint files: .7 TB, every 4 hours, 200 total
  - Plot files: 20GB each, 700 files
  - Particle files: 470 MB each, 1,400 files
- Used 154 TB total
- Created 74 million files!
- UNIX utility problems (e.g., mv, ls, cp)
- It took 2 years to sift though data, sew files together

Parallel I/O Single-File

All Tasks to Single File



Advantages

Single file makes data manageable

No system problems with excessive metadata

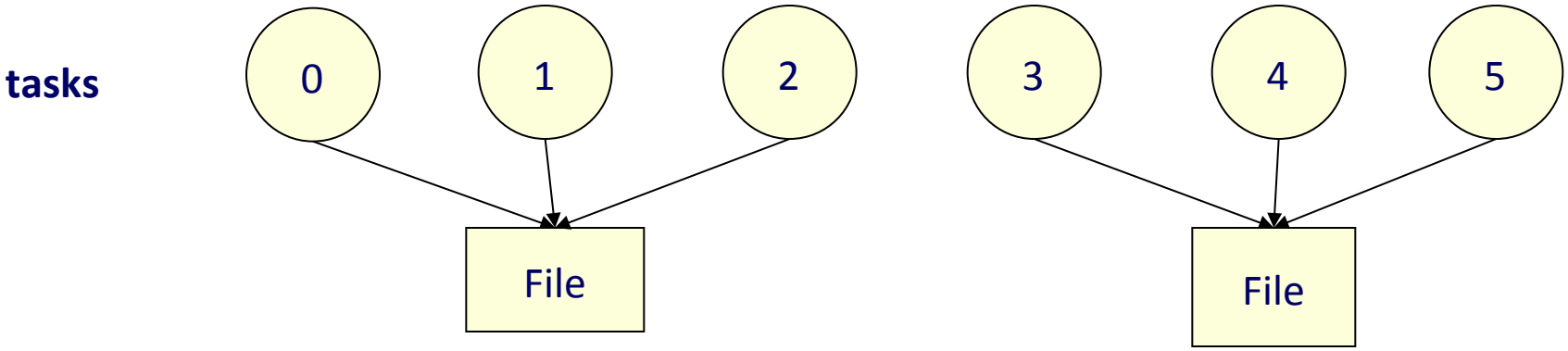
Disadvantages

Can be more difficult to program (use libs)

Performance may be less

Hybrid Model I

Groups of Tasks Access Different Files



Advantages

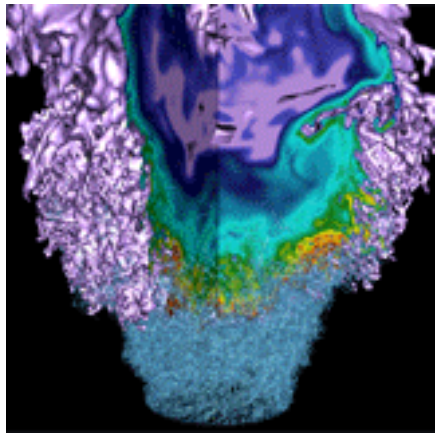
Fewer files than 1➔1

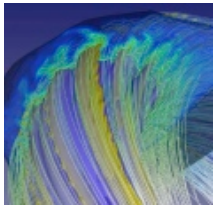
Better performance than All➔1

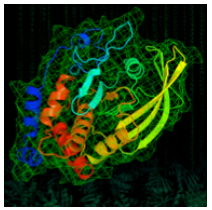
Disadvantages

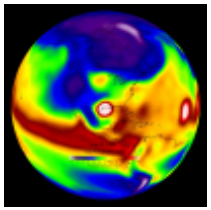
Algorithmically complex

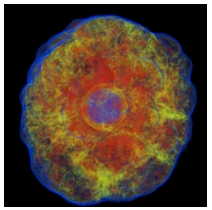
MPI-IO

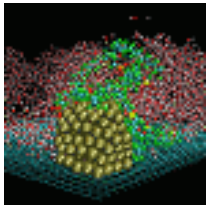












NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

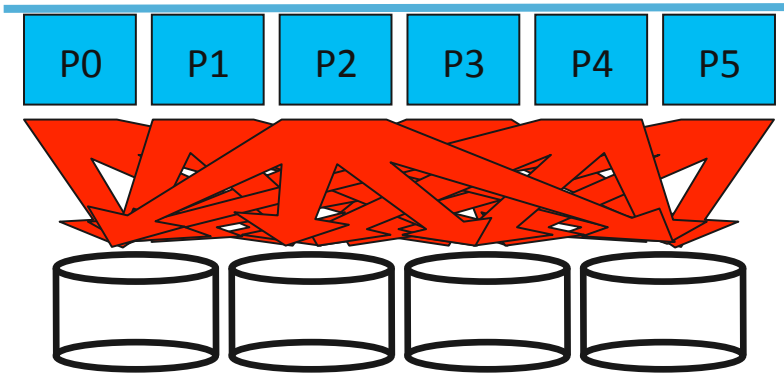
What is MPI-IO?

- **Parallel I/O interface for MPI programs**
- **Allows access to shared files using a standard API that is optimized and safe**
- **Key concepts:**
  - MPI communicators
    - open()s and close()s are collective within communicator
    - Only tasks in communicator can access
  - Derived data types
    - All operations (e.g. read()) have an associated MPI data type
  - Collective I/O for optimizations

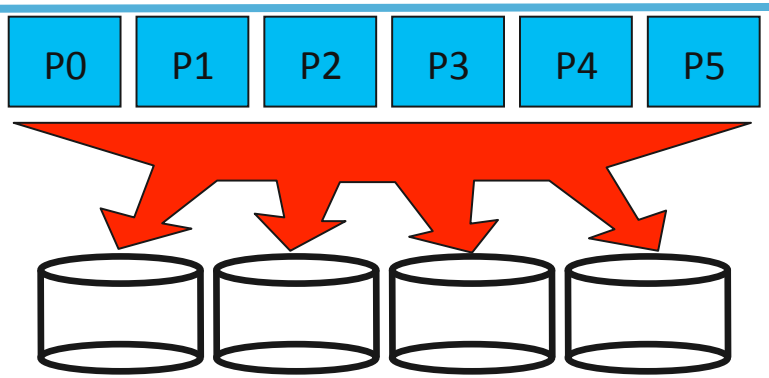
Basic MPI IO Routines

- MPI_File_open() – associate a file with a file handle.
- MPI_File_seek() – move the current file position to a given location in the file.
- MPI_File_read() – read some fixed amount of data out of the file beginning at the current file position.
- MPI_File_write() – write some fixed amount of data into the file beginning at the current file position.
- MPI_File_sync() -- flush any caches associated with the file handle.
- MPI_File_close() – close the file handle.

Independent and Collective I/O



Independent I/O



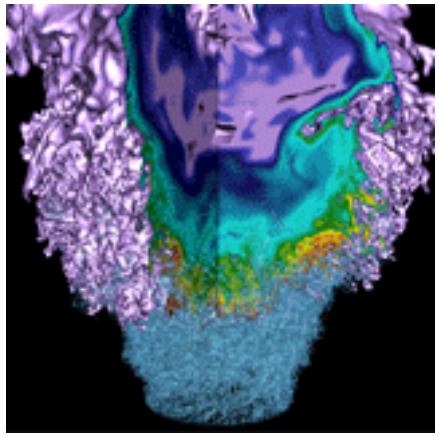
Collective I/O

- **Independent I/O operations specify only what a single process will do**
  - Independent I/O calls obscure relationships between I/O on other processes
- **Many applications have phases of computation and I/O**
  - During I/O phases, all processes read/write data
- **Collective I/O is coordinated access to storage by a group of processes**
  - Collective I/O functions are called by all processes participating in I/O
  - Allows I/O layers to know more about access as a whole, more opportunities for optimization in lower software layers, better performance

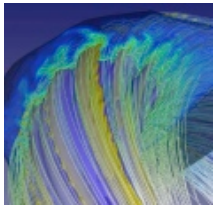
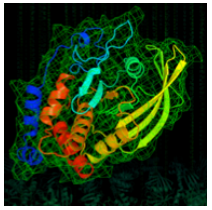
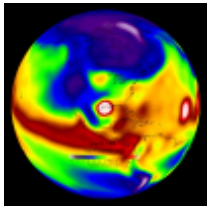
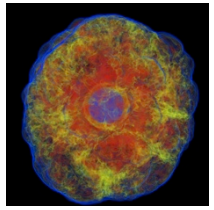
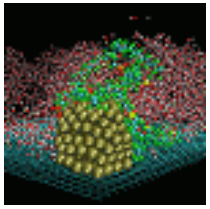
MPI-IO Summary

- Provides optimizations for typically low performing I/O patterns (non-contiguous I/O and small block I/O)
- You could use MPI-IO directly, but better to use a high level I/O library
- MPI-IO works well in the middle of the I/O stack, letting high-level library authors write to the MPI-IO API

High Level Parallel I/O Libraries



NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

Common Storage Formats

- **ASCII:**
  - Very slow
  - Takes a lot of space!
  - Inaccurate
- **Binary**
  - Non-portable (eg. byte ordering and types sizes)
  - Not future proof
- **Self-Describing formats**
  - NetCDF/HDF4, HDF5, Parallel NetCDF
- **Community File Formats**
  - FITS, HDF-EOS, SAF, PDB, Plot3D
  - Modern Implementations built on top of HDF, NetCDF, or other self-describing object-model API

Many users at this level. We would like to encourage you to transition to a higher IO library

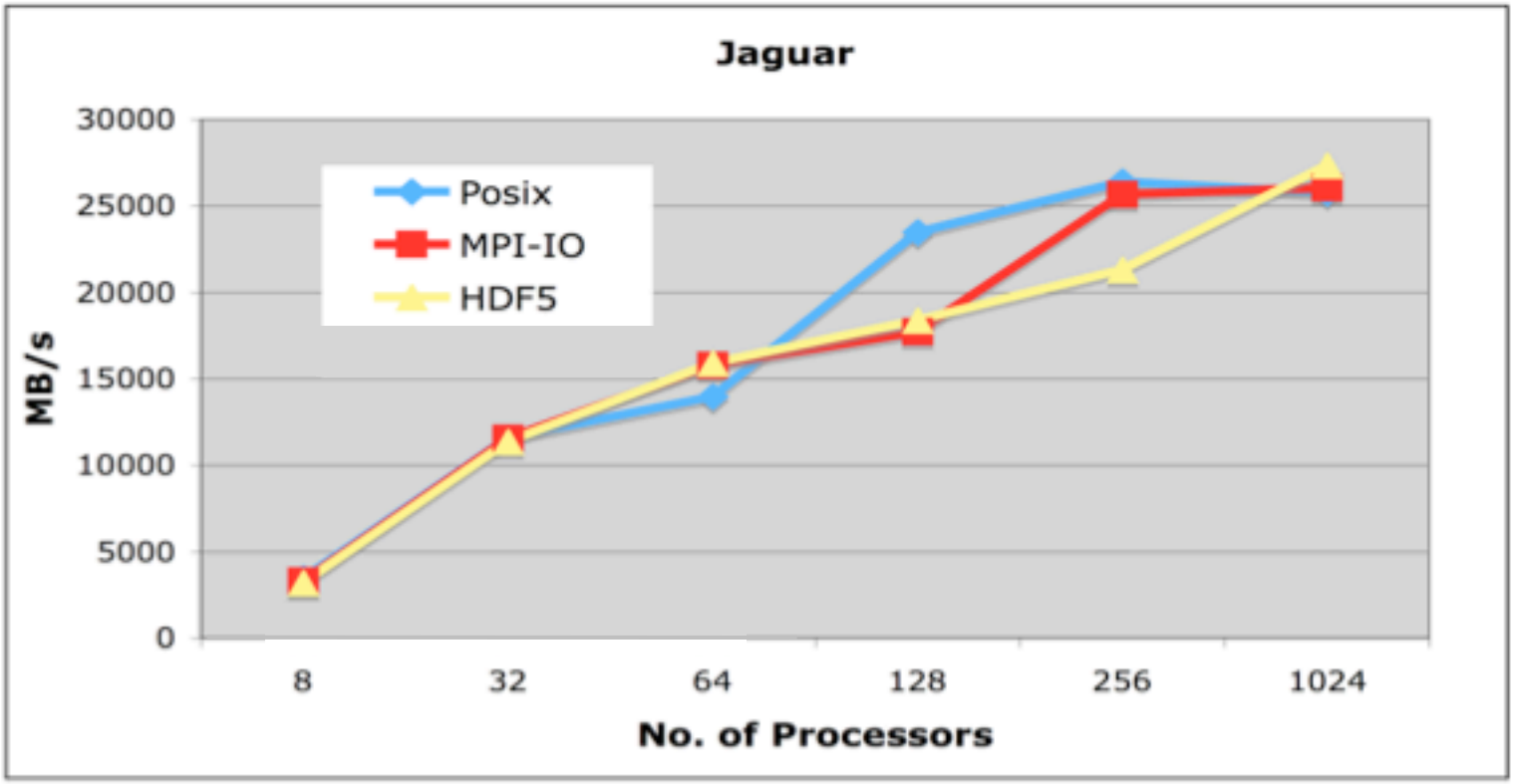
What is a High Level Parallel I/O Library?

- **An API which helps to express scientific simulation data in a more natural way**
  - Multi-dimensional data, labels and tags, non-contiguous data, typed data
- **Typically sits on top of MPI-IO layer and can use MPI-IO optimizations**
- **A library offers**
  - Portable formats - can write on one machine and read from another
  - Longevity - output will last and be accessible with library tools and no need to remember version number of code

What about performance?

- Hand tuned I/O for a particular application and architecture may perform better, but …
- … maybe not. A library can be optimized for a given architecture by the library developers
- Performance is not just GB/s, but more importantly, productivity
- If you use your own binary file format, it forces you to understand layers below the application and preserve your I/O routines if you want to read later
- Every time code is ported to a new machine or underlying file system is changed or upgraded, you are required to make changes to maintain performance

IO Library Overhead



Very little, if any overhead from HDF5 for one file per processor IO compared to Posix and MPI-IO

Data from Hongzhang Shan

ADIOS

ADIOS
easy-to-use, fast, scalable, and portable I/O

ADIOS provides a code API and external XML descriptor file that lets you process data in different ways by changing the XML file and rerunning your code.

ADIOS can use different back-end file storage formats (e.g. HDF5, netCDF)

Recommendations

- **Do large I/O: write fewer big chunks of data (1MB+) rather than small bursty I/O**
- **Do parallel I/O**
  - Serial I/O (single writer) can not take advantage of the system's parallel capabilities.
- **Use a single, shared file instead of 1 file per writer, esp. at high parallel concurrency**
- **Avoid excessive metadata activity (e.g., avoid keeping millions of small files in a single directory)**
- **Use an I/O library API and write flexible, portable programs**

National Energy Research Scientific Computing Center