

Understanding High-performance I/O on BlueGene

Rob Latham

Mathematics and Computer Science Division

Argonne National Laboratory

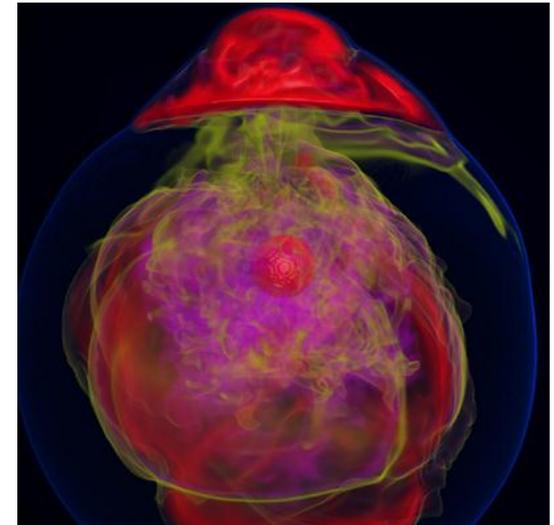
robl@mcs.anl.gov

Computational Science

- Use of computer simulation as a tool for greater understanding of the real world
 - Complements experimentation and theory
- Problems are increasingly computationally challenging
 - Large parallel machines needed to perform calculations
 - Critical to leverage parallelism in all phases
- Data access is a huge challenge
 - Using parallelism to obtain performance
 - Finding usable, efficient, portable interfaces
 - Understanding and tuning I/O



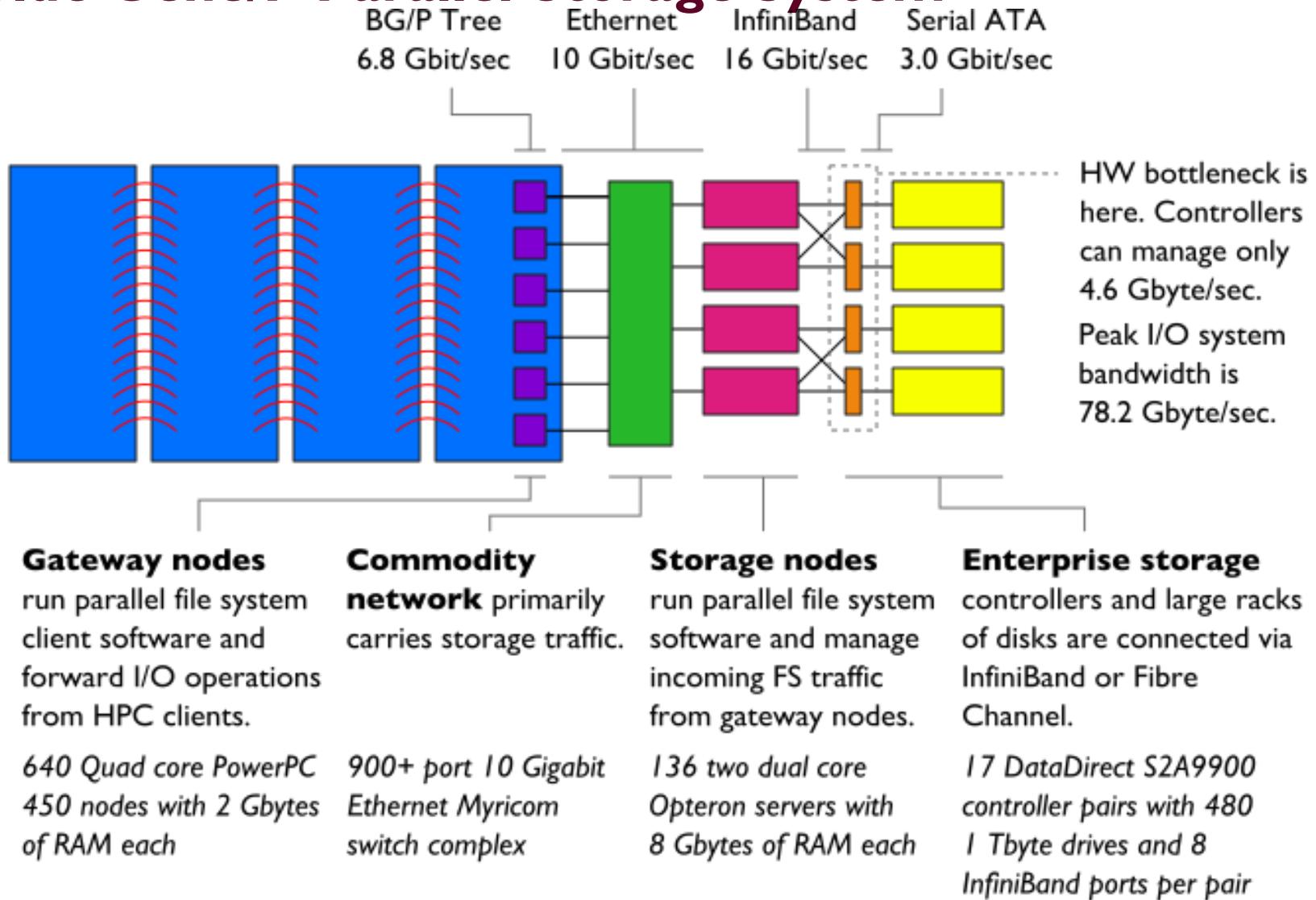
IBM Blue Gene/P system at Argonne National Laboratory.



Visualization of entropy in Terascale Supernova Initiative application. Image from Kwan-Liu Ma's visualization team at UC Davis.



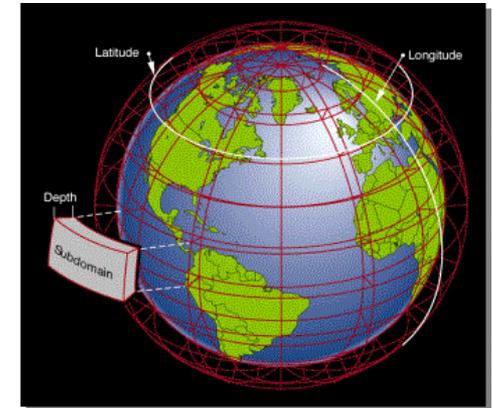
Blue Gene/P Parallel Storage System



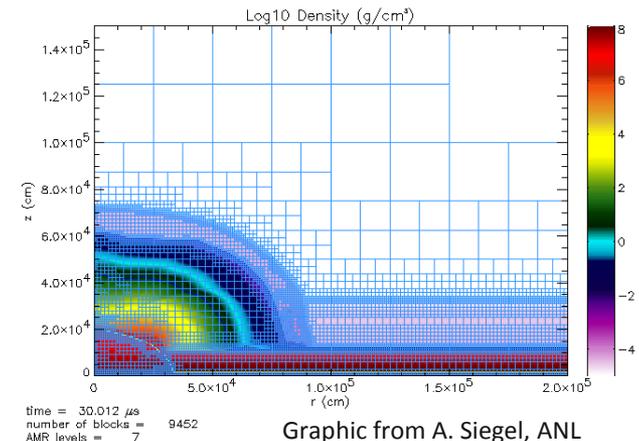
Architectural diagram of the 557 TFlop IBM Blue Gene/P system at the Argonne Leadership Computing Facility.

Applications, Data Models, and I/O

- Applications have data models appropriate to domain
 - Multidimensional typed arrays, images composed of scan lines, variable length records
 - Headers, attributes on data
- I/O systems have very simple data models
 - Tree-based hierarchy of containers
 - Some containers have streams of bytes (files)
 - Others hold collections of other containers (directories or folders)
- Someone has to map from one to the other!

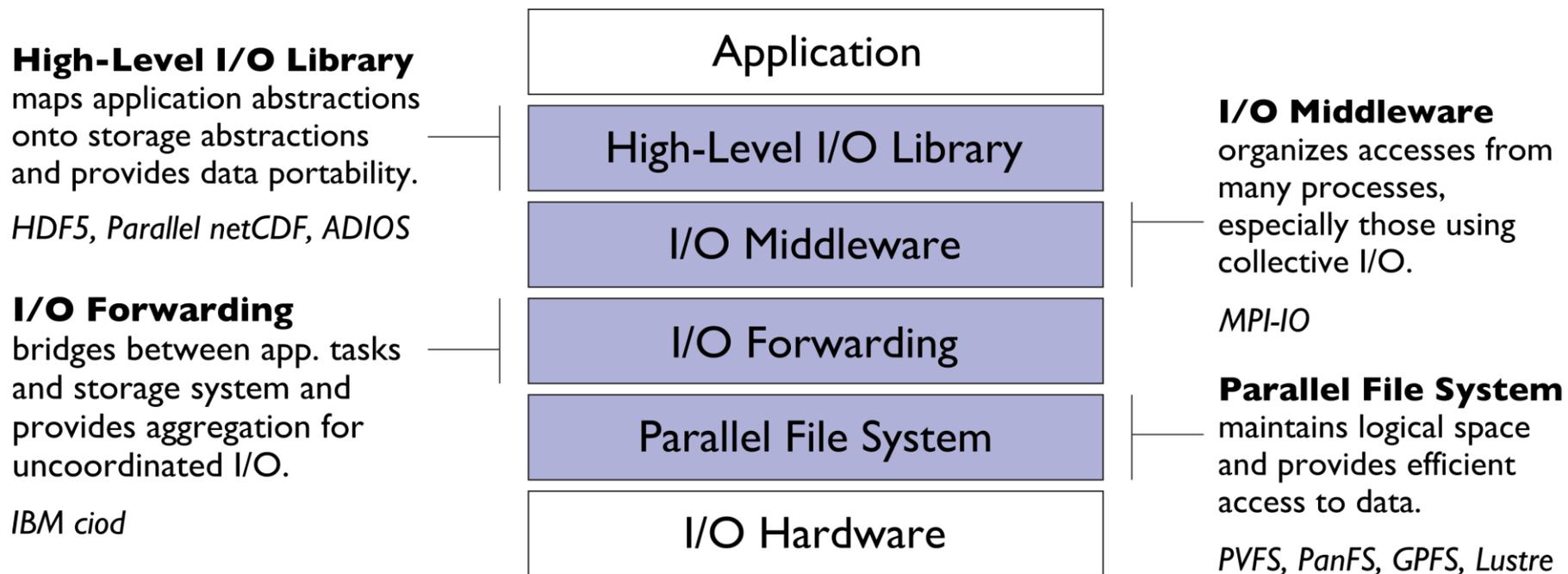


Graphic from J. Tannahill, LLNL



Graphic from A. Siegel, ANL

I/O for Computational Science



Additional I/O software provides improved performance and usability over directly accessing the parallel file system. Reduces or (ideally) eliminates need for optimization in application codes.

I/O Hardware and Software on Blue Gene/P

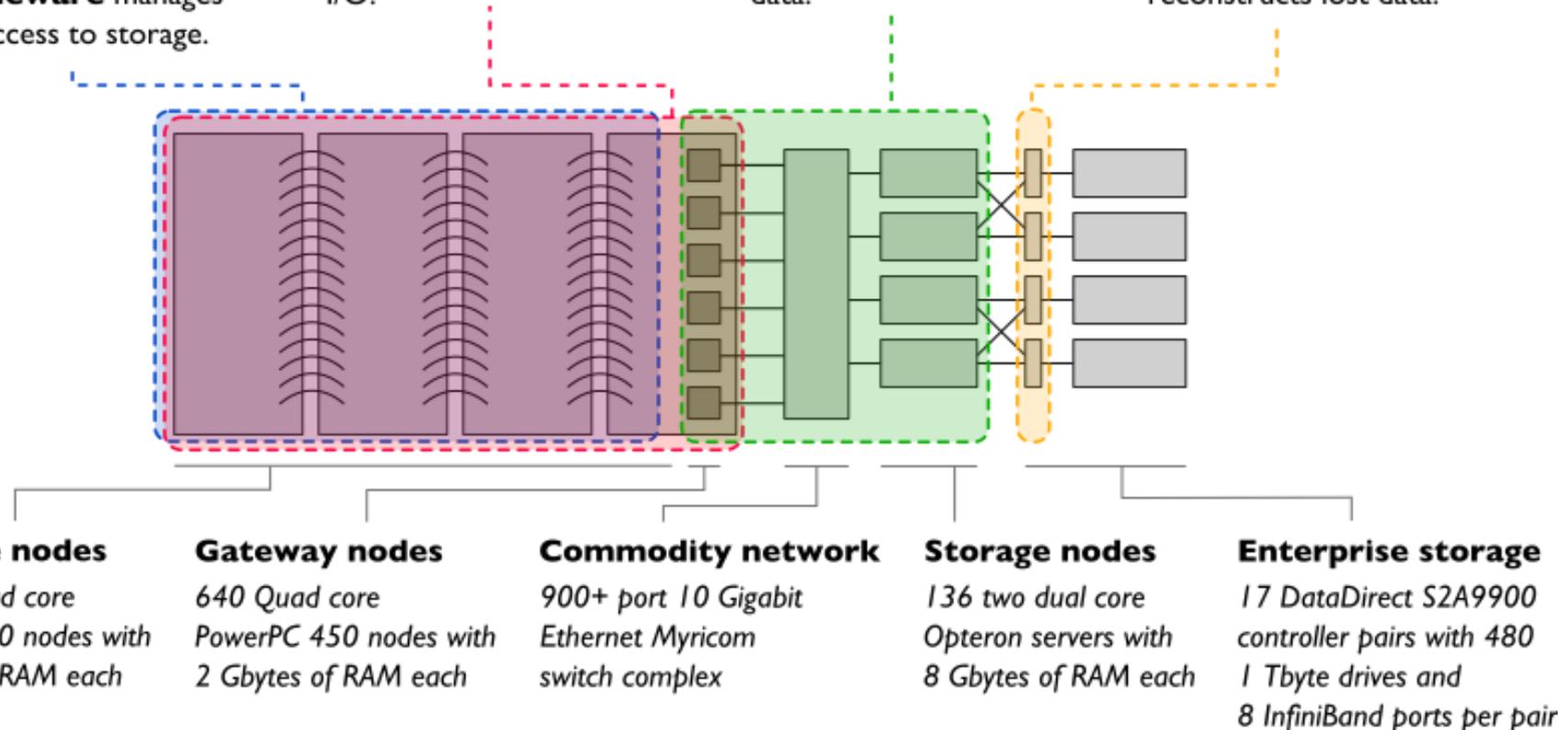
High-level I/O libraries execute on compute nodes, mapping application abstractions into flat files, and encoding data in portable formats.

I/O middleware manages collective access to storage.

I/O forwarding software runs on compute and gateway nodes, bridges networks, and provides aggregation of independent I/O.

Parallel file system code runs on gateway and storage nodes, maintains logical storage space and enables efficient access to data.

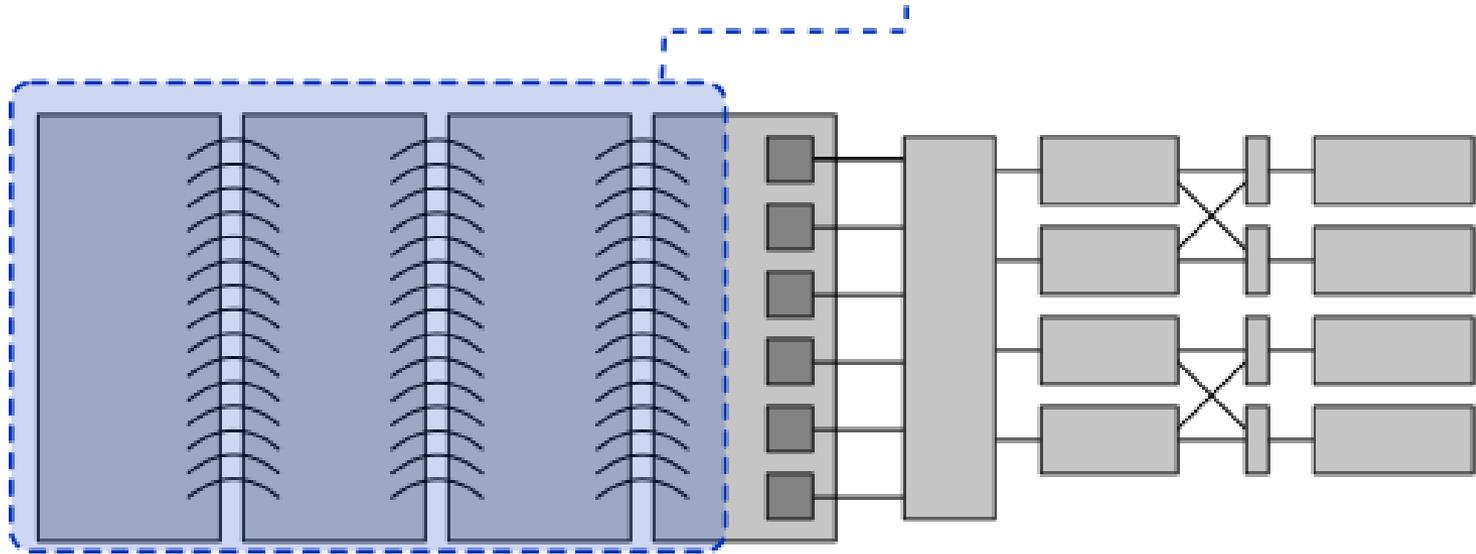
Drive management software or firmware executes on storage controllers, organizes individual drives, detects drive failures, and reconstructs lost data.



Architectural diagram of the 557 TFlop IBM Blue Gene/P system at the Argonne Leadership Computing Facility.

High-level Libraries and MPI-IO Software

High-level I/O libraries and **MPI-IO** execute on compute nodes and organize accesses before the I/O system sees them.



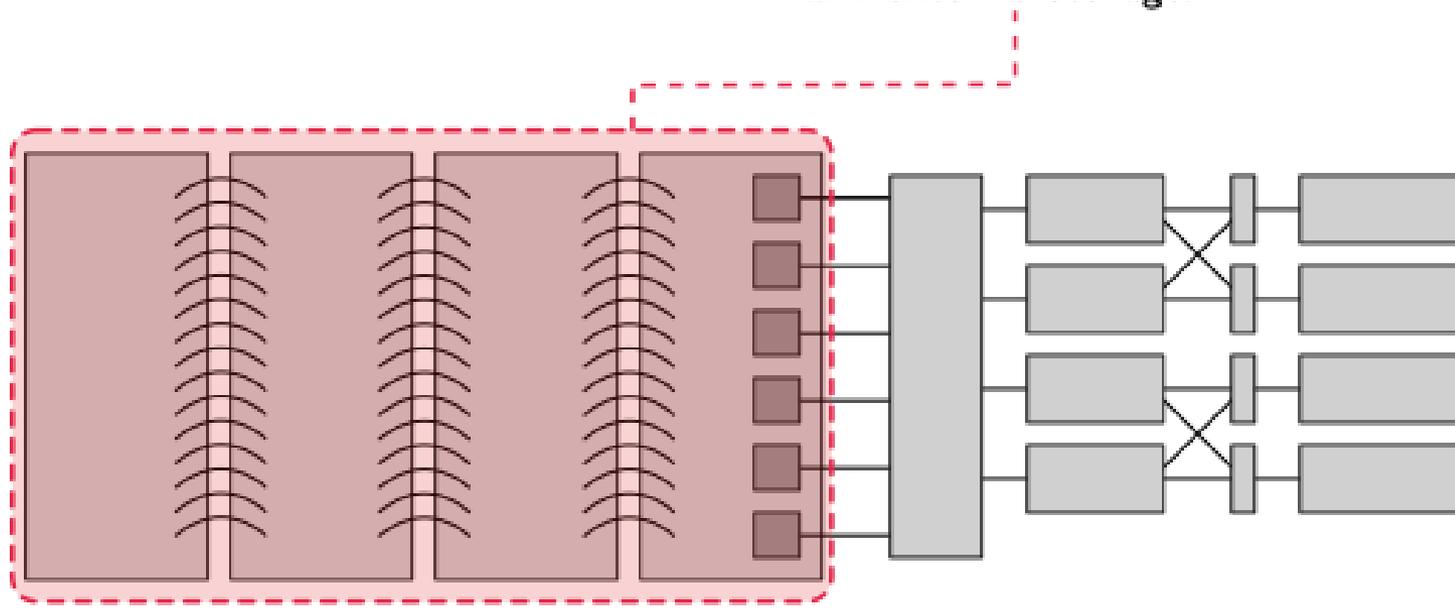
Compute nodes

run application codes with high-level I/O libraries and MPI-IO. I/O libraries make I/O calls to I/O forwarding system



I/O Forwarding Software

I/O forwarding software runs on compute and gateway nodes and bridges between the compute nodes and external storage.



Compute nodes

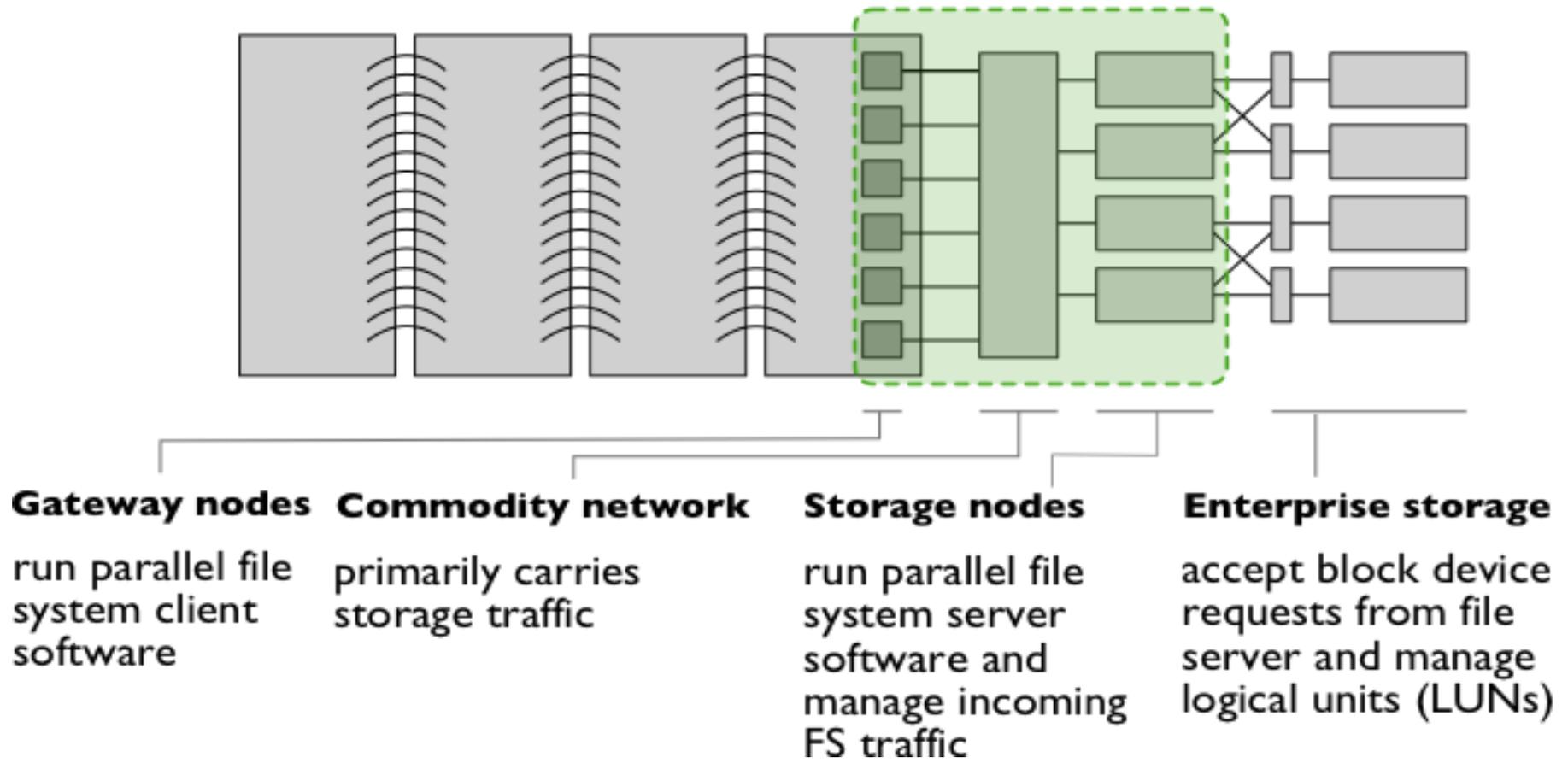
run I/O forwarding software intercepting I/O calls from application and forwarding to gateway nodes

Gateway nodes

run I/O forwarding software accepting I/O requests from compute nodes and forward to parallel file system

Parallel File System Software

PVFS code runs on gateway and storage nodes, maintains logical storage space, and enables efficient access to data.



The MPI-IO Interface

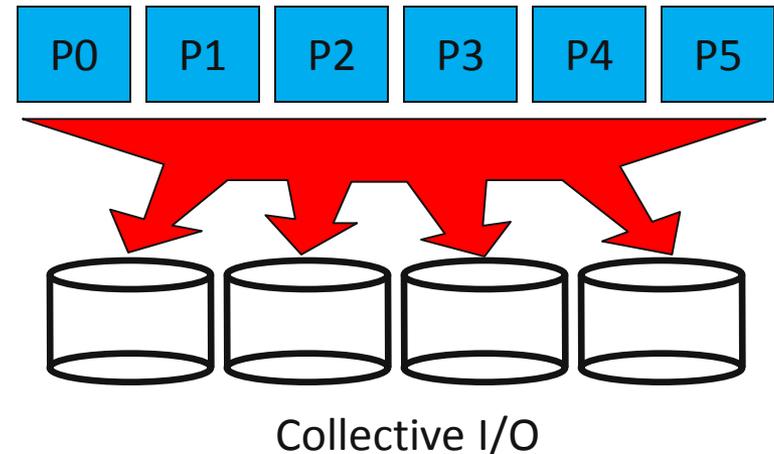
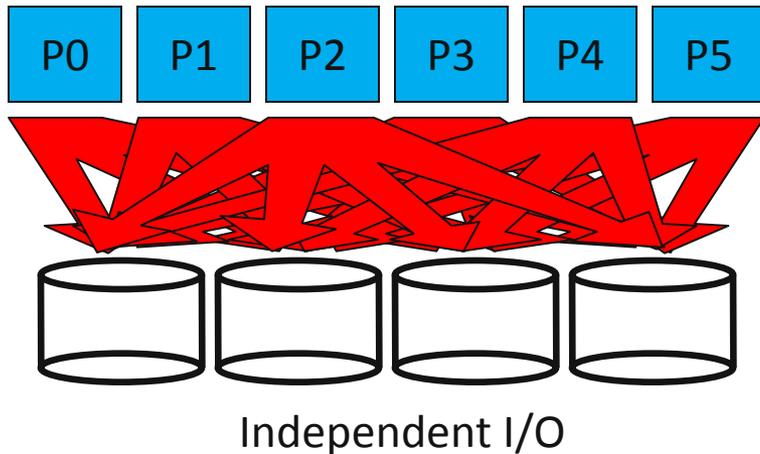


MPI-IO

- I/O interface **specification** for use in MPI apps
- Data model is same as POSIX
 - Stream of bytes in a file
- Features:
 - Collective I/O
 - Noncontiguous I/O with MPI datatypes and file views
 - Nonblocking I/O
 - Fortran bindings (and additional languages)
 - System for encoding files in a portable format (external32)
 - Not self-describing - just a well-defined encoding of types
- Implementations available on most platforms (more later)

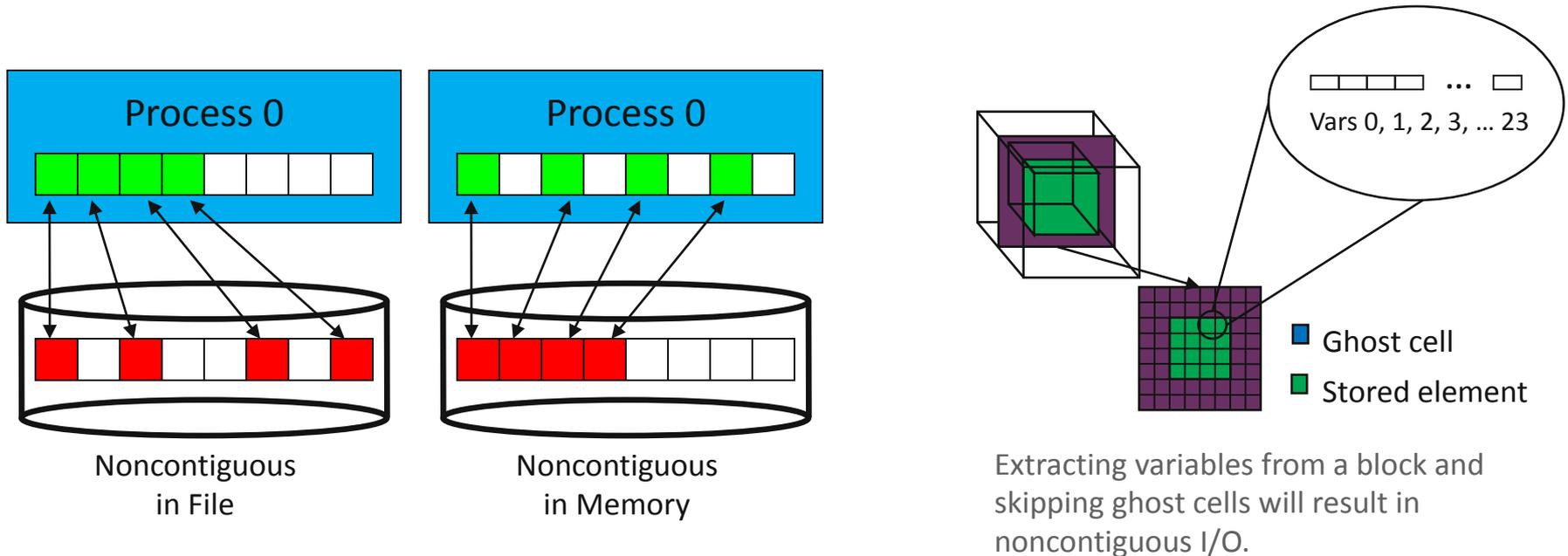


Independent and Collective I/O



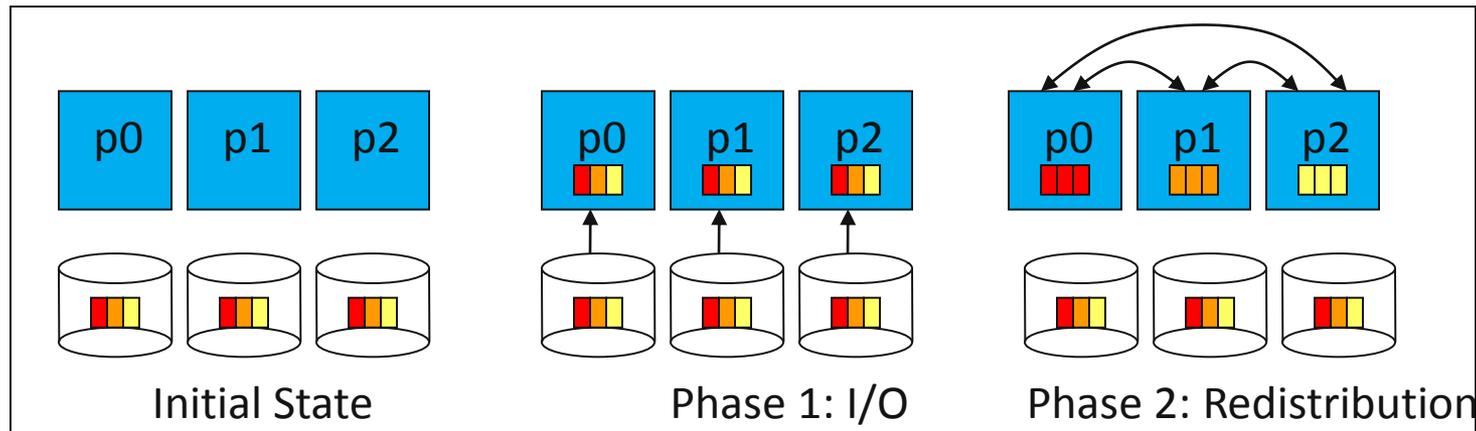
- **Independent** I/O operations specify only what a single process will do
 - Independent I/O calls do not pass on relationships between I/O on other processes
- Many applications have phases of computation and I/O
 - During I/O phases, all processes read/write data
 - We can say they are **collectively** accessing storage
- 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**

Contiguous and Noncontiguous I/O



- **Contiguous I/O** moves data from a single memory block into a single file region
- **Noncontiguous I/O** has three forms:
 - Noncontiguous in memory, noncontiguous in file, or noncontiguous in both
- Structured data leads naturally to noncontiguous I/O (e.g. block decomposition)
- **Describing noncontiguous accesses with a single operation passes more knowledge to I/O system**

Collective I/O and Two-Phase I/O

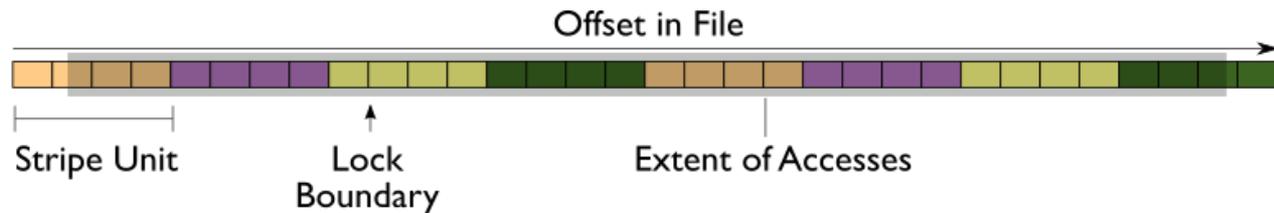


Two-Phase Read Algorithm

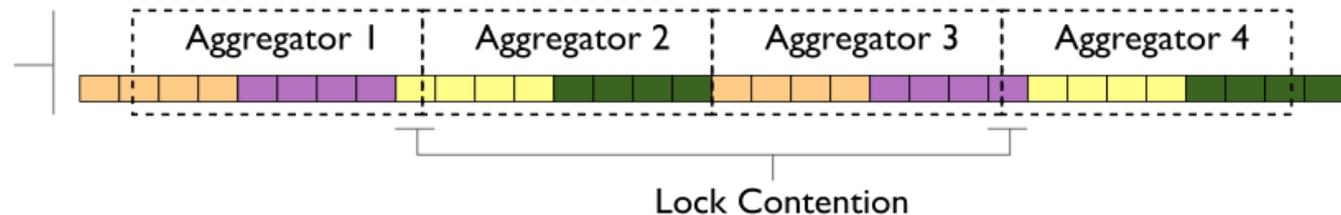
- Problems with independent, noncontiguous access
 - Lots of small accesses
 - Independent data sieving reads lots of extra data, can exhibit false sharing
- Idea: Reorganize access to match layout on disks
 - Single processes use data sieving to get data for many
 - Often reduces total I/O through sharing of common blocks
- Second “phase” redistributes data to final destinations
- Two-phase writes operate in reverse (redistribute then I/O)
 - Typically read/modify/write (like data sieving)
 - Overhead is lower than independent access because there is little or no false sharing
- Note that two-phase is usually applied to file regions, not to actual blocks

Two-Phase I/O Algorithms

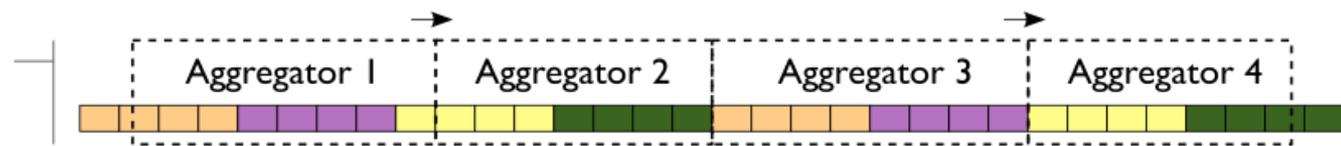
Imagine a collective I/O access using four aggregators to a file striped over four file servers (indicated by colors):



One approach is to evenly divide the region accessed across aggregators.



Aligning regions with lock boundaries eliminates lock contention.



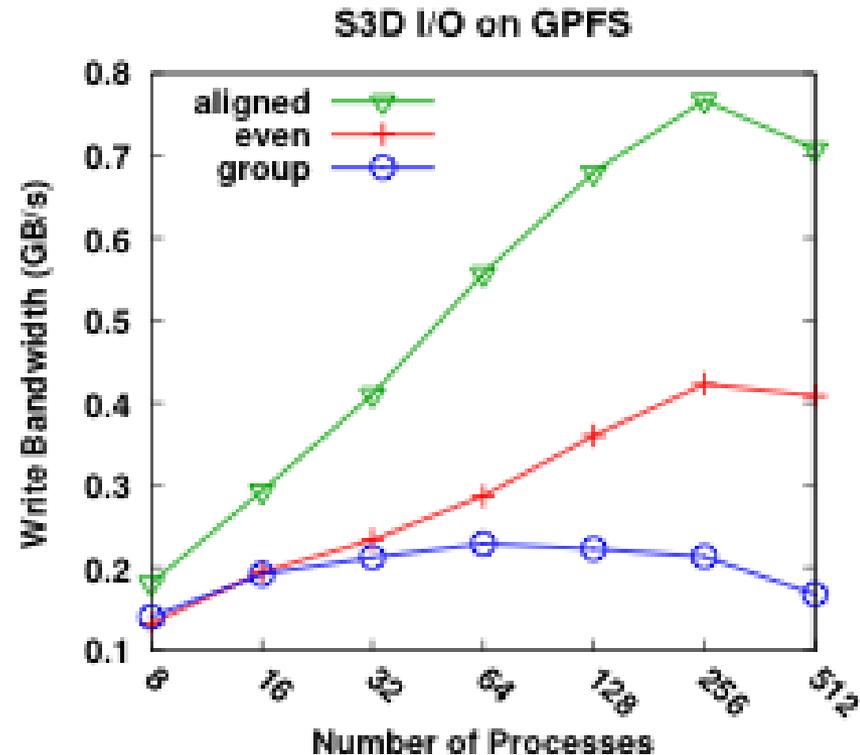
Mapping aggregators to servers reduces the number of concurrent operations on a single server and can be helpful when locks are handed out on a per-server basis (e.g., Lustre).



For more information, see W.K. Liao and A. Choudhary, "Dynamically Adapting File Domain Partitioning Methods for Collective I/O Based on Underlying Parallel File System Locking Protocols," SC2008, November, 2008.

Impact of Two-Phase I/O Algorithms

- This graph shows the performance for the S3D combustion code, writing to a single file.
- Aligning with lock boundaries doubles performance over default “even” algorithm.
- “Group” algorithm similar to server-aligned algorithm on last slide.
- Testing on Mercury, an IBM IA64 system at NCSA, with 54 servers and 512KB stripe size.



W.K. Liao and A. Choudhary, “Dynamically Adapting File Domain Partitioning Methods for Collective I/O Based on Underlying Parallel File System Locking Protocols,” SC2008, November, 2008.

The Parallel netCDF Interface and File Format

Thanks to Wei-Keng Liao, Alok Choudhary, and Kui Gao (NWU) for their help in the development of PnetCDF.



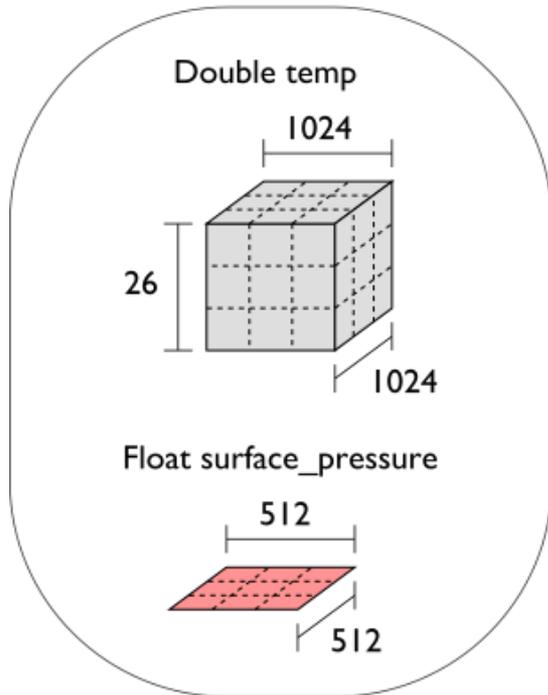
Parallel netCDF (PnetCDF)

- Based on original “Network Common Data Format” (netCDF) work from Unidata
 - Derived from their source code
- Data Model:
 - Collection of variables in single file
 - Typed, multidimensional array variables
 - Attributes on file and variables
- Features:
 - C and Fortran interfaces
 - Portable data format (identical to netCDF)
 - Noncontiguous I/O in memory using MPI datatypes
 - Noncontiguous I/O in file using sub-arrays
 - Collective I/O
 - Non-blocking I/O
- Unrelated to netCDF-4 work

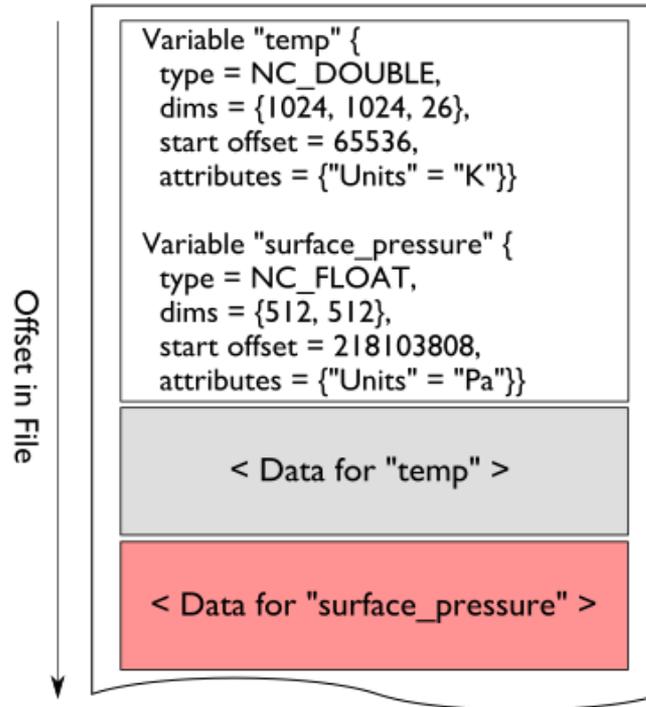


Data Layout in netCDF Files

Application Data Structures



netCDF File "checkpoint07.nc"

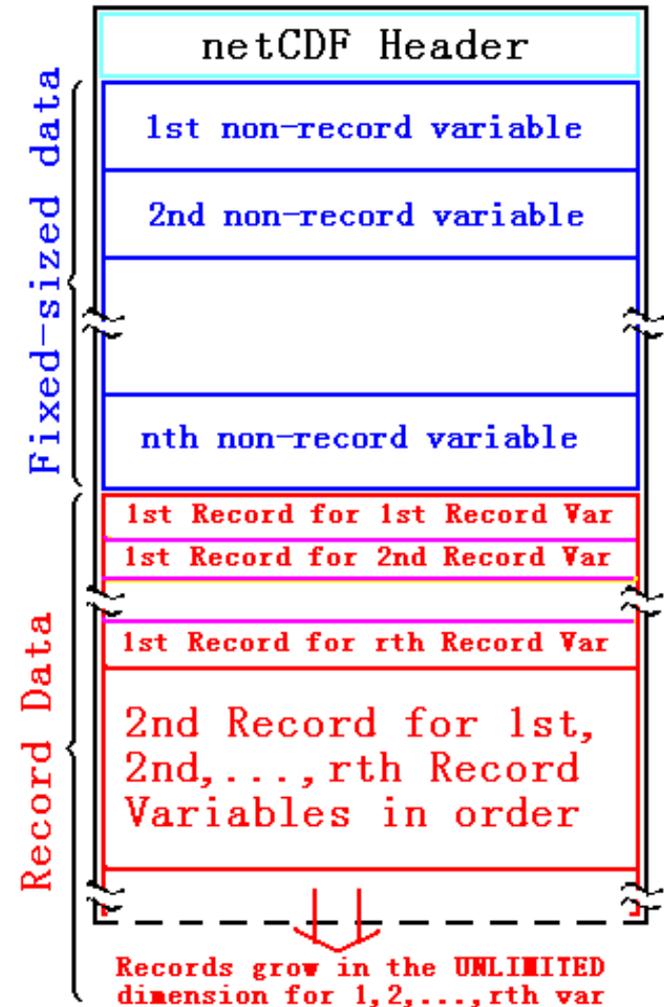


netCDF header describes the contents of the file: typed, multi-dimensional variables and attributes on variables or the dataset itself.

Data for variables is stored in contiguous blocks, encoded in a portable binary format according to the variable's type.

Record Variables in netCDF

- Record variables are defined to have a single “unlimited” dimension
 - Convenient when a dimension size is unknown at time of variable creation
- Record variables are stored after all the other variables in an interleaved format
 - Using more than one in a file is likely to result in poor performance due to number of noncontiguous accesses



Storing Data in PnetCDF

- Create a **dataset** (file)
 - Puts dataset in define mode
 - Allows us to describe the contents
 - Define **dimensions** for variables
 - Define **variables** using dimensions
 - Store **attributes** if desired (for variable or dataset)
- Switch from define mode to data mode to write variables
- Store variable data
- Close the dataset

Other High-Level I/O libraries

- NetCDF-4: <http://www.unidata.ucar.edu/software/netcdf/netcdf-4/>
 - netCDF API with HDF5 back-end
- ADIOS: <http://adiosapi.org>
 - Configurable (xml) I/O approaches
- SILO: <https://wci.llnl.gov/codes/silo/>
 - A mesh and field library on top of HDF5 (and others)
- H5part: <http://vis.lbl.gov/Research/AcceleratorSAPP/>
 - simplified HDF5 API for particle simulations
- GIO: <https://svn.pnl.gov/gcrm>
 - Targeting geodesic grids as part of GCRM
- PIO:
 - climate-oriented I/O library; supports raw binary, parallel-netcdf, or serial-netcdf (from master)
- ... Many more: my point: it's ok to make your own.

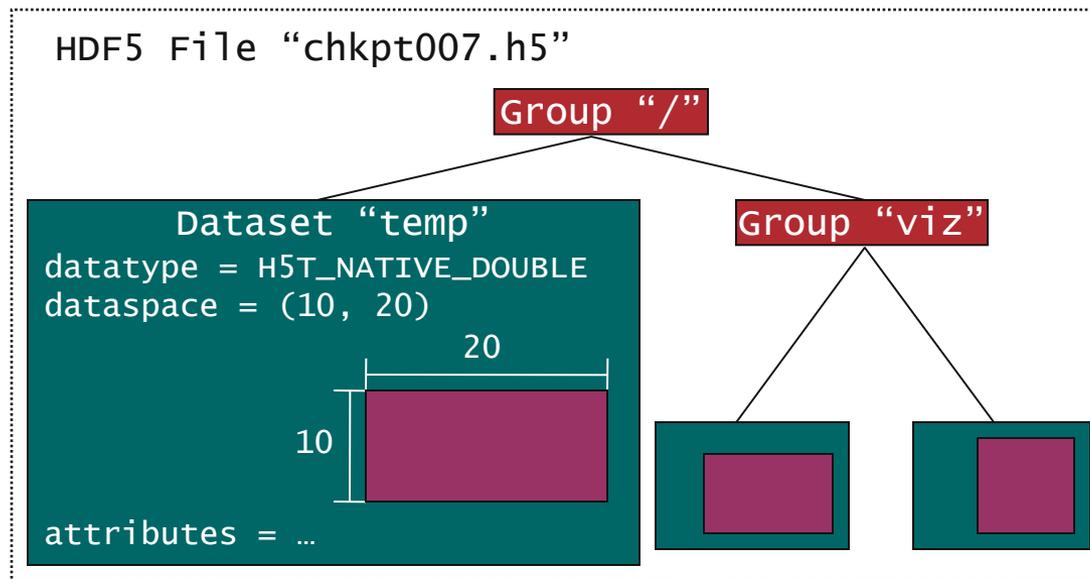


HDF5

- Hierarchical Data Format, from the HDF Group (formerly of NCSA)
- Data Model:
 - Hierarchical data organization in single file
 - Typed, multidimensional array storage
 - Attributes on dataset, data
- Features:
 - C, C++, and Fortran interfaces
 - Portable data format
 - Optional compression (not in parallel I/O mode)
 - Data reordering (chunking)
 - Noncontiguous I/O (memory and file) with hyperslabs



HDF5 Files



- HDF5 files consist of groups, datasets, and attributes
 - **Groups** are like directories, holding other groups and datasets
 - **Datasets** hold an array of typed data
 - A **datatype** describes the type (not an MPI datatype)
 - A **dataspace** gives the dimensions of the array
 - **Attributes** are small datasets associated with the file, a group, or another dataset
 - Also have a datatype and dataspace
 - May only be accessed as a unit

Enabling High-performance I/O with HDF5

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

/* Create a new file collectively. */
file_id = H5Fcreate(filename, H5F_ACC_TRUNC,
                    H5P_DEFAULT, plist_id);
H5Pclose(plist_id);
/* ... omitted data decomposition for brevity */
/* Set up data transfer property list w/ collective MPI-IO */
plist_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

status = H5Dwrite(dset_id, H5T_NATIVE_INT,
                 memspace, filespace, plist_id, data);
```

Inside HDF5

- `MPI_File_open` used to open file
- Because there is no “define” mode, file layout is determined at write time
- In `H5Dwrite`:
 - Processes communicate to determine file layout
 - Process 0 performs metadata updates after write
 - Call `MPI_File_set_view`
 - Call `MPI_File_write_all` to collectively write
 - Only if enabled via property list
- Memory hyperslab could have been used to define noncontiguous region in memory
- In FLASH application, data is kept in native format and converted at read time (defers overhead)
 - Could store in some other format if desired
- At the MPI-IO layer:
 - Metadata updates at every write are a bit of a bottleneck
 - MPI-IO from process 0 introduces some skew

HDF5 Wrap-up

- Tremendous flexibility: 300+ routines
- H5Lite high level routines for common cases
- Tuning via property lists
 - “use MPI-IO to access this file”
 - “read this data collectively”
- Extensive on-line documentation, tutorials (see “On Line Resources” slide)
- New efforts:
 - Journaling: make datasets more robust in face of crashes (Sandia)
 - Fast appends (finance motivated)
 - Single-writer, Multiple-reader semantics
 - Aligning data structures to underlying file system



Lightweight Application Characterization with Darshan

Thanks to Phil Carns (carns@mcs.anl.gov) for providing background material on Darshan.

Characterizing Application I/O

How are applications using the I/O system, and how successful are they at attaining high performance?

Darshan (Sanskrit for “sight”) is a tool we developed for I/O characterization at extreme scale:

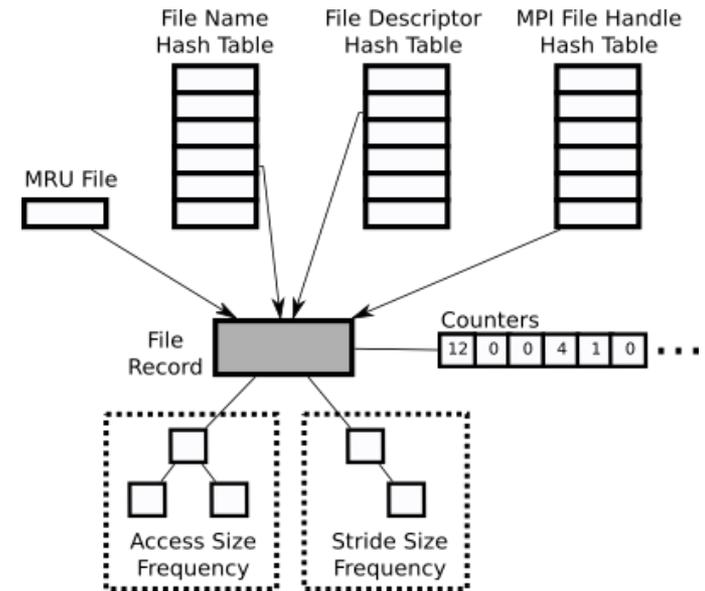
- No code changes, small and tunable memory footprint (~2MB default)
- Characterization data aggregated and compressed prior to writing
- Captures:
 - Counters for POSIX and MPI-IO operations
 - Counters for unaligned, sequential, consecutive, and strided access
 - Timing of opens, closes, first and last reads and writes
 - Cumulative data read and written
 - Histograms of access, stride, datatype, and extent sizes

<http://www.mcs.anl.gov/darshan/>

P. Carns et al, “24/7 Characterization of Petascale I/O Workloads,” IASDS Workshop, held in conjunction with IEEE Cluster 2009, September 2009.

Darshan Internals

- Characterization centers around per-file records
 - Multiple hash tables allow relating accesses to one another
 - Falls back to aggregate (across files) mode if file limit is exceeded
- At output time, processes further reduce output size
 - Communicate to combine data on identical files accessed by all processes
 - Independently compress (gzip) remaining data
 - 32K processes writing a shared file leads to 203 bytes of compressed output
 - 32K processes writing a total of 262,144 files leads to 13.3MB of output



Multiple tables allow efficient location of file records by name, file descriptor, or MPI File handle.

The Darshan Approach

- Use PMPI and Id wrappers to intercept I/O functions
 - Requires re-linking, but no code modification
 - Can be transparently included in mpicc
 - Compatible with a variety of compilers
- Record statistics independently at each process
 - Compact summary rather than verbatim record
 - Independent data for each file
- Collect, compress, and store results at shutdown time
 - Aggregate shared file data using custom MPI reduction operator
 - Compress remaining data in parallel with zlib
 - Write results with collective MPI-IO
 - Result is a single gzip-compatible file containing characterization information



Example Statistics (per file)

- Counters:
 - POSIX open, read, write, seek, stat, etc.
 - MPI-IO nonblocking, collective, independent, etc.
 - Unaligned, sequential, consecutive, strided access
 - MPI-IO datatypes and hints
- Histograms:
 - access, stride, datatype, and extent sizes
- Timestamps:
 - open, close, first I/O, last I/O
- Cumulative bytes read and written
- Cumulative time spent in I/O and metadata operations
- Most frequent access sizes and strides
- Darshan records 150 integer or floating point parameters per file, plus job level information such as command line, execution time, and number of processes.



sequential



consecutive

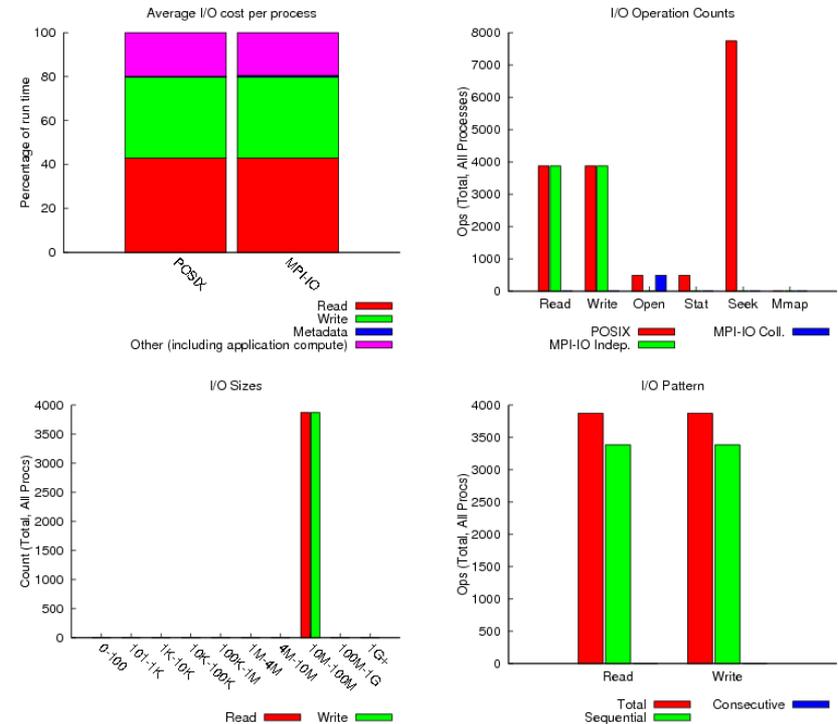


strided

Job Summary

- Job summary tool shows characteristics “at a glance”
- MADBench2 example
- Shows time spent in read, write, and metadata
- Operation counts, access size histogram, and access pattern
- Early indication of I/O behavior and where to explore in further

uid: 4279 nprocs: 484 runtime: 255 seconds



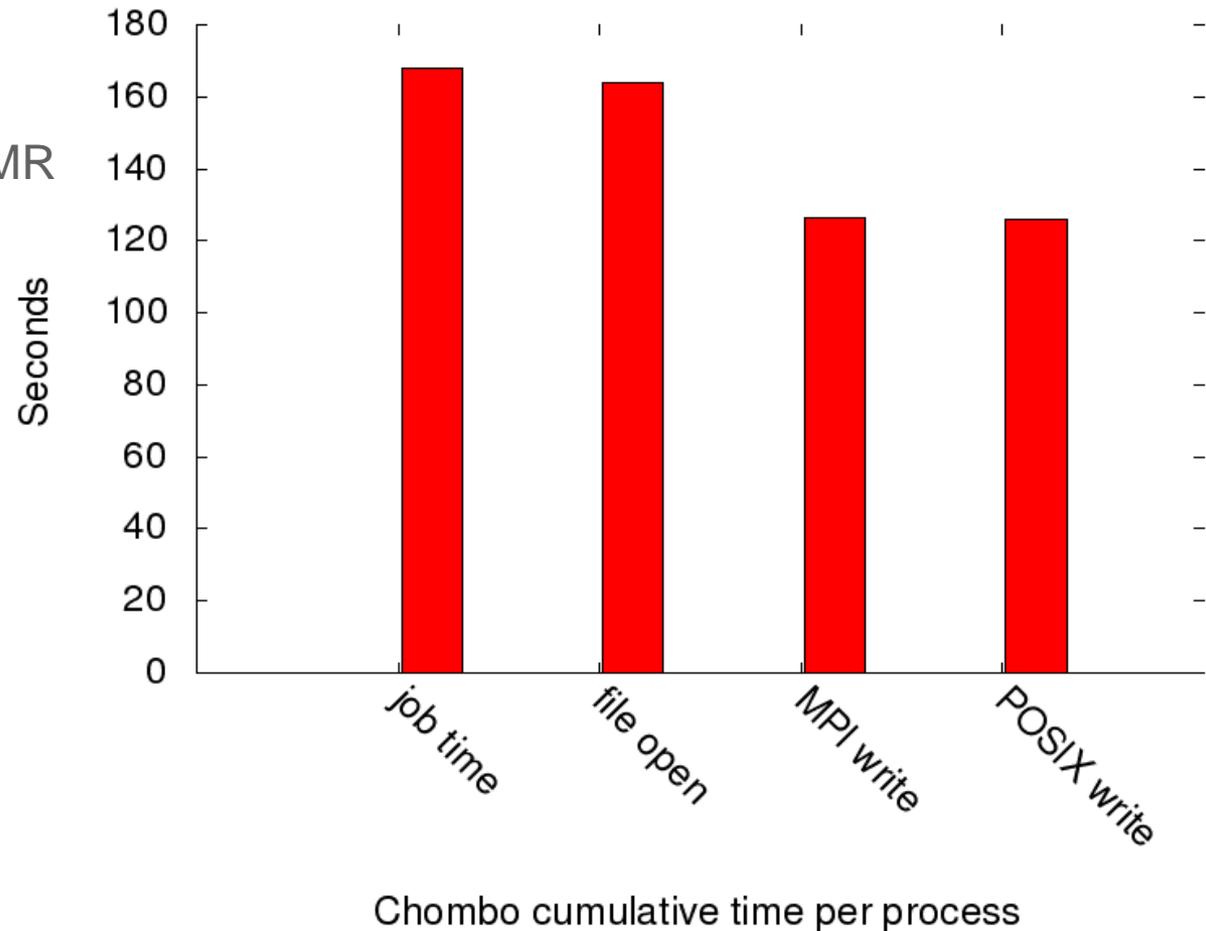
Top 4 Access Sizes

access size	count
19568768	7744
0	0
0	0
0	0

/home/carns/tar/MADBench2/MADBench2 34408 4 1 1564 4194304 1 1

Chombo I/O Benchmark

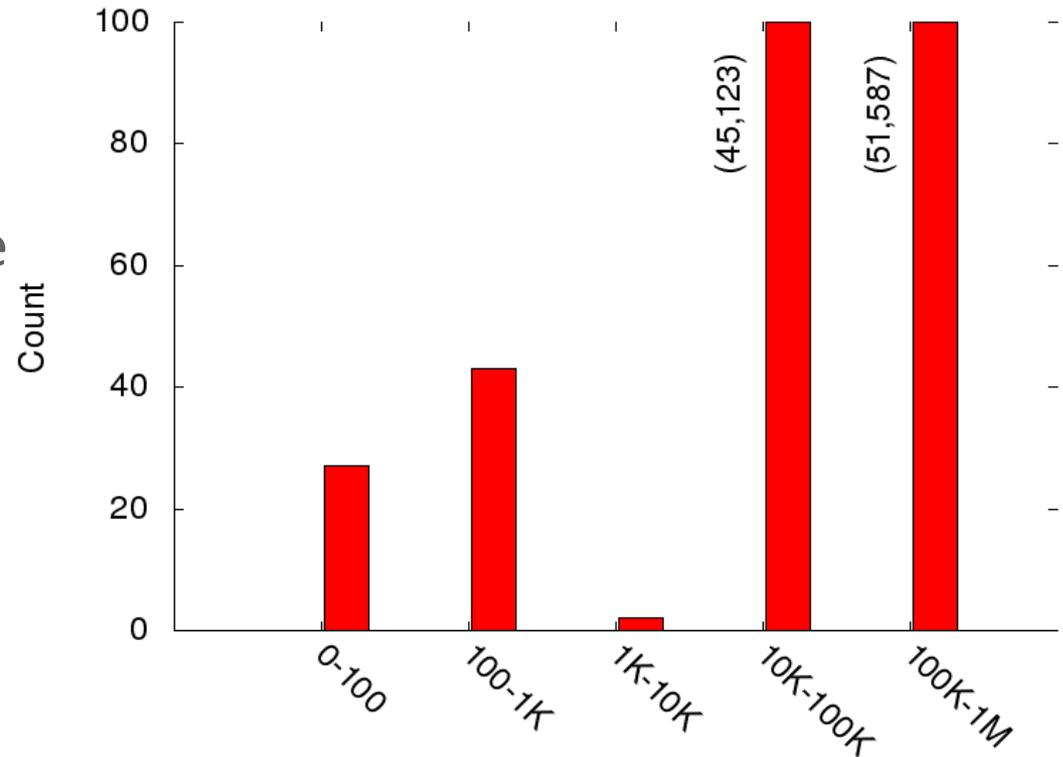
- Checkpoint writes from AMR framework
- Uses HDF5 for I/O
- Code base is complex
- 512 processes
- 18.24 GB output file



- Why does the I/O take so long in this case?
- Why isn't it busy writing data the whole time?

Chombo I/O Benchmark

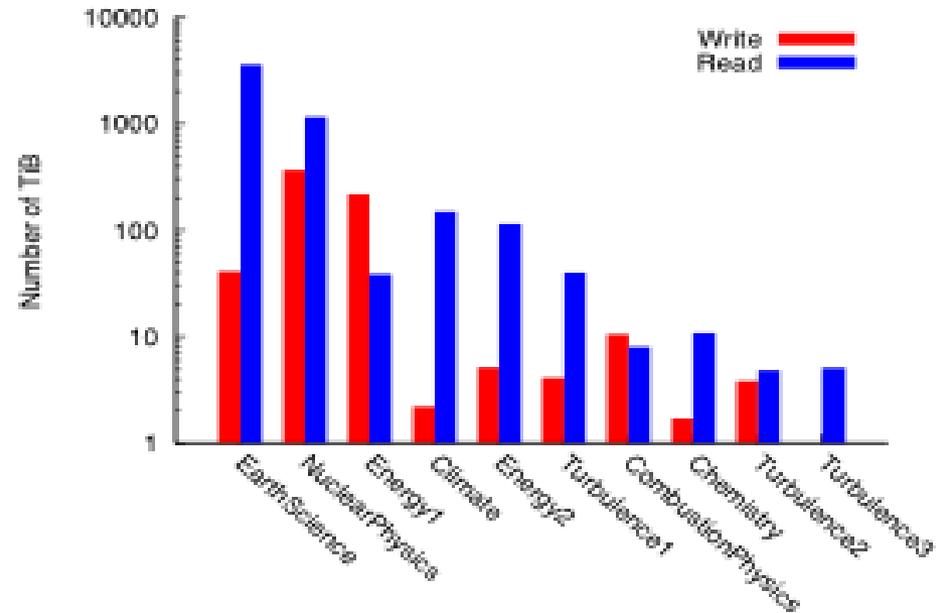
- Many write operations, with none over 1 MB in size
- Most common access size is 28,800 (occurs 15622 times)
- No MPI datatypes or collectives
- All processes frequently seek forward between writes
 - Consecutive: 49.25%
 - Sequential: 99.98%
 - Unaligned in file: 99.99%
 - Several recurring regular stride patterns



Chombo write size histogram, 512 procs

Two Months of Application I/O on ALCF Blue Gene/P

- After additional testing and hardening, Darshan installed on Intrepid
- By default, all applications compiling with MPI compilers are instrumented
- Data captured from late January through late March of 2010
- Darshan captured data on 6,480 jobs (27%) from 39 projects (59%)
- Simultaneously captured data on servers related to storage utilization



Top 10 data producers and/or consumers shown. Surprisingly, most “big I/O” users read more data during simulations than they wrote.

P. Carns et al, “Storage Access Characteristics of Computational Science Applications,” forthcoming.

Application I/O on ALCF Blue Gene/P

Application	Mbytes/sec/CN*	Cum. MD	Files/Proc	Creates/Proc	Seq. I/O	Mbytes/Proc
EarthScience	0.69	95%	140.67	98.87	65%	1779.48
NuclearPhysics	1.53	55%	1.72	0.63	100%	234.57
Energy1	0.77	31%	0.26	0.16	87%	66.35
Climate	0.31	82%	3.17	2.44	97%	1034.92
Energy2	0.44	3%	0.02	0.01	86%	24.49
Turbulence1	0.54	64%	0.26	0.13	77%	117.92
CombustionPhysics	1.34	67%	6.74	2.73	100%	657.37
Chemistry	0.86	21%	0.20	0.18	42%	321.36
Turbulence2	1.16	81%	0.53	0.03	67%	37.36
Turbulence3	0.58	1%	0.03	0.01	100%	40.40

* Synthetic I/O benchmarks (e.g., IOR) attain 3.93 - 5.75 Mbytes/sec/CN for modest job sizes, down to approximately 1.59 Mbytes/sec/CN for full-scale runs.

P. Carns et al, "Storage Access Characteristics of Computational Science Applications," forthcoming.



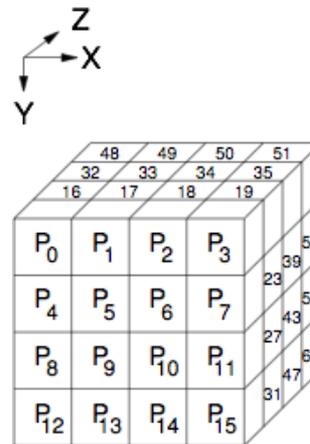
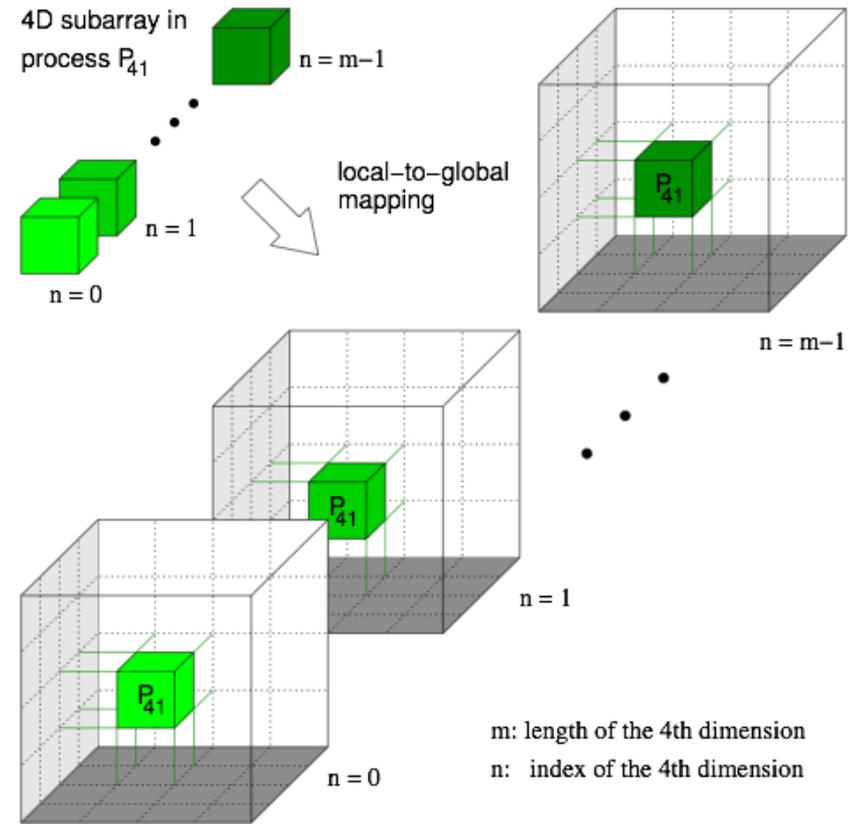
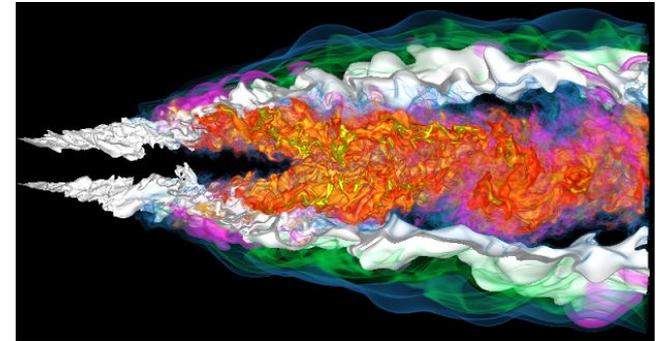
Darshan Summary

- Scalable tools like Darshan can yield useful insight
 - Identify characteristics that make applications successful
 - Identify problems to address through I/O research
- Petascale performance tools require special considerations
 - Target the problem domain carefully to minimize amount of data
 - Avoid shared resources
 - Use collectives where possible
- For more information:
<http://www.mcs.anl.gov/research/projects/darshan>



S3D Turbulent Combustion Code

- S3D is a turbulent combustion application using a direct numerical simulation solver from Sandia National Laboratory
- Checkpoints consist of four global arrays
 - 2 3-dimensional
 - 2 4-dimensional
 - 50x50x50 fixed subarrays



Thanks to Jackie Chen (SNL), Ray Grout (SNL), and Wei-Keng Liao (NWU) for providing the S3D I/O benchmark, Wei-Keng Liao for providing this diagram, C.Wang, H.Yu, and K.-L. Ma of UC Davis for image.

Impact of Optimizations on S3D I/O

- Testing with PnetCDF output to single file, three configurations, 16 processes
 - All MPI-IO optimizations (collective buffering and data sieving) disabled
 - Independent I/O optimization (data sieving) enabled
 - Collective I/O optimization (collective buffering, a.k.a. two-phase I/O) enabled

	Coll. Buffering and Data Sieving Disabled	Data Sieving Enabled	Coll. Buffering Enabled (incl. Aggregation)
POSIX writes	102,401	81	5
POSIX reads	0	80	0
MPI-IO writes	64	64	64
Unaligned in file	102,399	80	4
Total written (MB)	6.25	87.11	6.25
Runtime (sec)	1443	11	6.0
Avg. MPI-IO time per proc (sec)	1426.47	4.82	0.60



Wrapping Up

- We've covered a lot of ground in a short time
 - Very low-level, serial interfaces
 - High-level, hierarchical file formats
- Storage is a complex hardware/software system
- There is no magic in high performance I/O
 - Lots of software is available to support computational science workloads at scale
 - Knowing how things work will lead you to better performance
- Using this software (correctly) can dramatically improve performance (execution time) and productivity (development time)



On-Line References

- netCDF and netCDF-4
 - <http://www.unidata.ucar.edu/packages/netcdf/>
- PnetCDF
 - <http://www.mcs.anl.gov/parallel-netcdf/>
- ROMIO MPI-IO
 - <http://www.mcs.anl.gov/romio/>
- HDF5 and HDF5 Tutorial
 - <http://www.hdfgroup.org/>
 - <http://www.hdfgroup.org/HDF5/>
 - <http://www.hdfgroup.org/HDF5/Tutor>
- Darshan I/O Characterization Tool
 - <http://www.mcs.anl.gov/research/projects/darshan>
- Assorted ALCF-Specific suggestions:
 - https://wiki.alcf.anl.gov/index.php/I_O_Tuning

