HDF5 Community BOF - Agenda

Topic	Presenter
HDF5 update	Elena Pourmal, The HDF Group
Virtual Object Layer (VOL) and connectors	Quincey Koziol, NERSC / LBNL
DAOS VOL, Subfiling, Querying	Scot Breitenfeld, The HDF Group
Applications - CGNS, E3SM, HACC	Scot Breitenfeld, The HDF Group
Applications - EQSIM, AMReX (Nyx and Castro)	Houjun Tang, LBNL
Q & A	

HDF5 Update

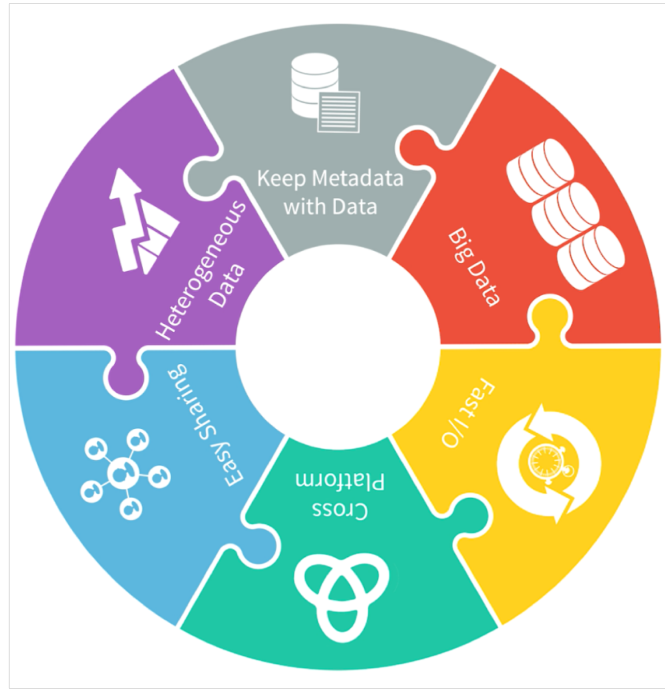
March 30, 2021

ECP HDF5 BOF

The HDF Group and ExaIO team @LBNL

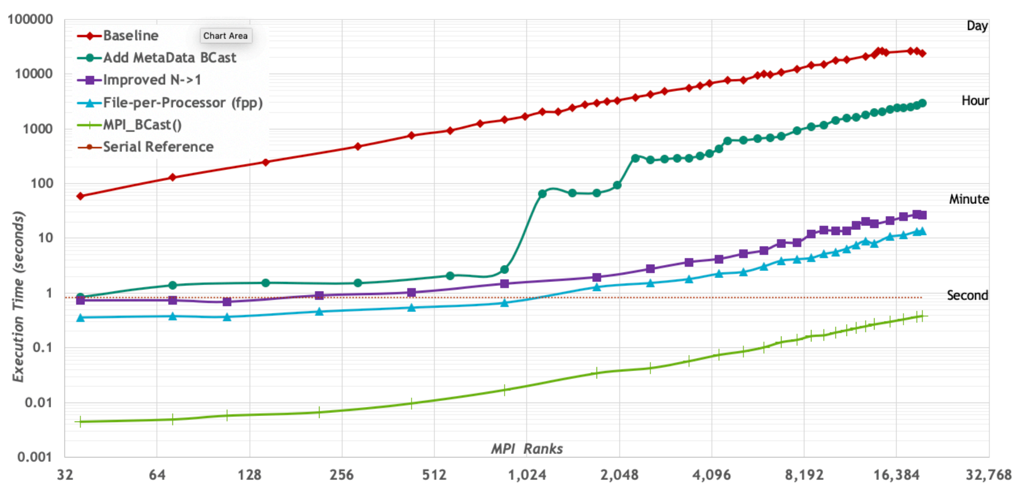
HDF5 in a nutshell

- HDF5 is a data model, I/O library and binary format for storing and managing data
- One of the most used I/O libraries and file formats across DOE
- Maintained and developed by The HDF Group in collaboration with the ExaIO ECP team
- Originally designed for storing data on POSIX FS; extended to other storage



Community involvement and outreach

- HDF5 is on GitHub https://github.com/hdfgroup/hdf5
- The HDF Group holds
  - Bi-monthly Webinars/Tutorials and weekly face-to-face teleconferences with HDF5 users
  - HDF User Group (HUG) Meetings (2019, 2020, and 2021 is in planning stage)
- Performance improvements and contributions to other software (netCDF-4, CGNS, h5py)



Courtesy Greg Sjaardema, Sandia National Labs

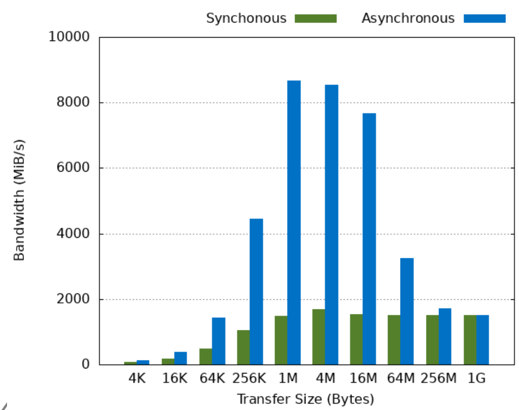
Before HDF5 optimization (Day)

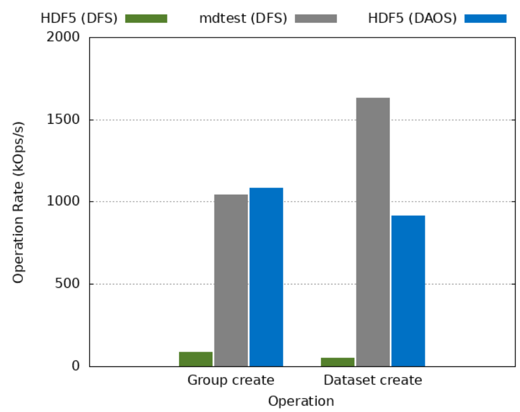
After HDF5 optimization (minute)

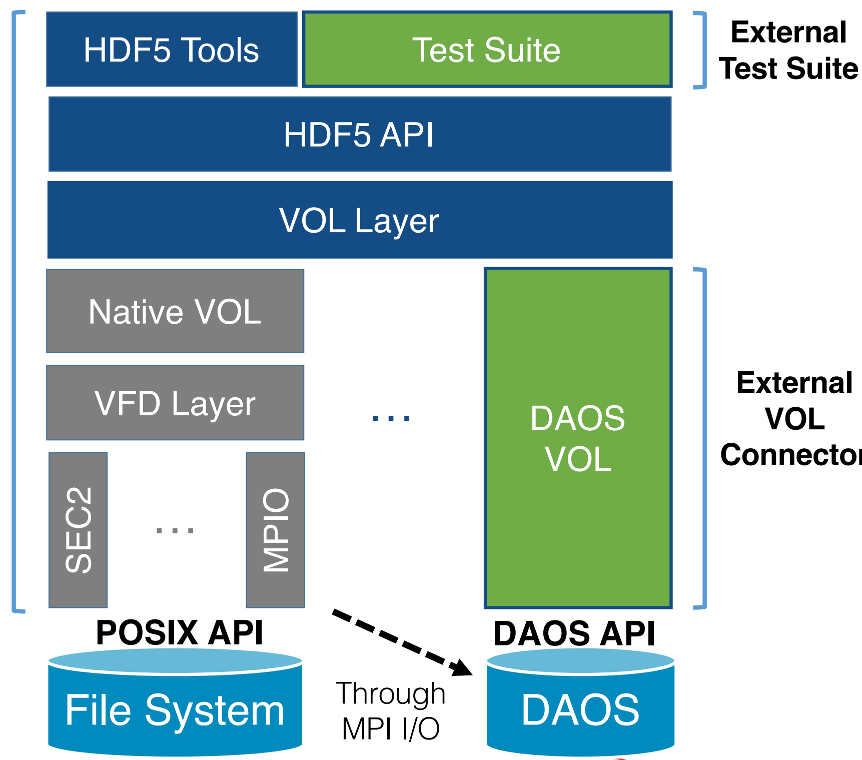
Baseline - File per MPI rank

Accessing data on "non-Posix" storage

- HDF5 VOL connectors (HDF5 VOLs)
  - Cloud, Object Store
  - Example: DAOS VOL connector
    - Introduces new features to HDF5
      - Asynchronous I/O
      - Independent HDF5 metadata updates
      - New HDF5 object - maps







HDF5 development to improve I/O performance

- ECP features (details later today)
  - HDF5 VOLs (Async, Cache VOLs, HDF5 GPU VFD)
  - *Sub-filing* - a compromise between file-per-process and a single shared file; implemented as HDF5 Virtual File Driver (VFD)
- Performance Tools Enhancements
  - Multi-level I/O tracing tool Recorder is now in Spack
  - Jupyter notebook tutorial for working with Darshan HDF5 output; GitHub source
- Performance study of ECP applications (FLASH, NWChem, Chombo, QMCPack and HACC)
  - Publish findings and recommendations in a white paper.
  - HACC with HDF5 delivers *comparable performance* with pure MPI-IO implementation by tuning stripe settings on Lustre and the HDF5 alignment parameter or metadata block sizes.

HDF5 Benchmark

- hdf5-iotest benchmark
  - Exercises different organization of data in HDF5 files using different HDF5 features (chunking, collective and independent I/O modes, datasets of different dimensionality, alignment, alignment threshold, and metadata block size; each configuration writes 80MBs per time stamp); available in Spack

A Simple Problem

Writing multiple 2D array variables over time:

```
ACROSS P processes arranged in a R x C process grid
    FOREACH step 1 .. S
        FOREACH count 1 .. A
            CREATE a double ARRAY of size [X,Y] | [R*X,C*Y] (strong | weak)
            (WRITE | READ) the ARRAY (to | from) an HDF5 file
        END
    END
END
```

S(teps) = 20, A(rrays) = 500, X = 100, Y = 200 (See adios_iotest)

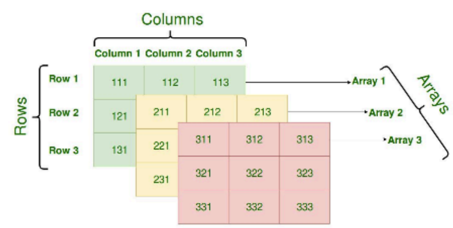
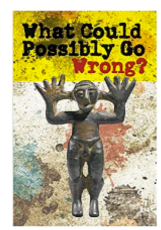


Figure: GeeksForGeeks



HDF5 and Spack

- The HDF Group is now an official maintainer of HDF5 in Spack
  - **GNU Autotools** builds and testing
  - Defaults to HDF5 1.10.7 parallel
    - Command "`spack install hdf5`"
    - Maintenance releases 1.12.0, 1.10.0-1.10.7, 1.8.10 – 1.8.22 are also available
- Imminent change (in review by the Spack team)
  - **CMake** builds and testing
  - Szip compression (licensed) is replaced with its OS version (libaec)
  - Added three maintenance branches 1_12, 1_10, 1_8
    - Command "`spack install hdf5@develop-1.12`"
  - Added HDFView

HDF5 and Spack

- Additional variants (prototypes) in progress
  - Additional compression plugins (registered with The HDF Group)
    - BITGROOM, BLOSC, BSHUF, BZIP2, JPEG, LZ4, LZF, MAFISC, ZFP, SZ, and ZSTD
  - ExaIO HDF5 VOL connectors (Async, Cache, external pass-through)
  - Example: command "**`spack install hdf5~zfp~mafisc+szip~zstd~blosc~bshuf~bitgroom+av~pv~cv+mpi+threadsafe`**" disables everything except szip, mpi, and threadsafe. The **`+av`** means to build Async VOL.
- Additional HDF5 releases in progress
  - HDF5 1.13.* (from develop branch) for the early releases of ECP productized features
    - Async, Cache, Pass-through VOLs
    - DAOS VOL
    - GPU VFD
    - Datalib VOL
    - ADIOS VOL
    - GPU VOLs
  - VOL connectors have to pass external VOL test suite

HDF5 Resources

- Check documentation on https://portal.hdfgroup.org
- Send email to help@hdfgroup.org
- Join https://forum.hdfgroup.org/
- Attend THG Webinars and Tutorials
  - Announced on HDF-FORUM, ECP Training Events page and ECP Training Newsletter
- **New**: Call the Doctor - The Weekly HDF clinic (on Tuesdays at 8:30 am or 1:00 pm Central)

Thank you!

Questions?

HDF5: Virtual Object Layer

ECP HDF5 Birds-of-a-Feather
March 30, 2021
Quincey Koziol
koziol@lbl.gov

Many Team Members and Contributors

- LBNL: Suren Byna, Houjun Tang, Tony Li, Bin Dong
- ANL: Venkat Vishwanath, Huihuo Zheng, Paul Coffman
- The HDF Group: Scot Breitenfeld, Elena Pourmal, John Mainzer, Richard Warren, Dana Robinson, Neil Fortner, Jerome Soumagne, Jordan Henderson, Neelam Bagha, …
- Northwestern University: Kai-yuan Hou
- North Carolina State University: John Ravi

Overview

- HDF5 Virtual Object Layer (VOL) Introduction
- ECP VOL Connectors
  - Asynchronous I/O
  - Node-local Caching
- GPU-IO
  - GPU Direct Storage (GSD) HDF5 Virtual File Driver

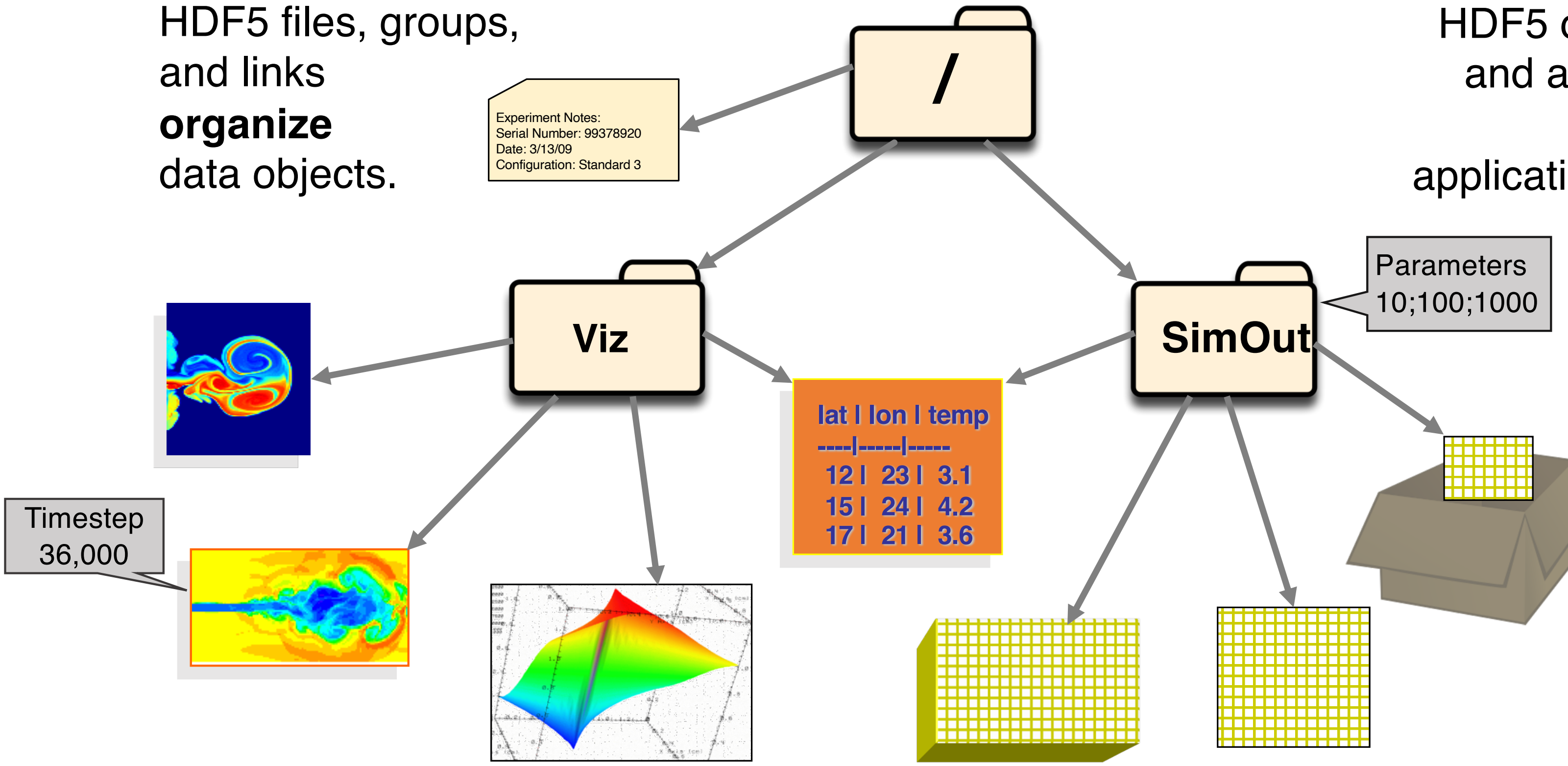
HDF5 Virtual Object Layer (VOL)

- **<u>VOL Framework</u> is an abstraction layer within HDF5 Library**
  - Redirects I/O operations into VOL "connector", immediately after an API routine is invoked
  - Non-I/O operations handled with library "infrastructure"
- **<u>VOL Connectors</u>**
  - Implement storage for HDF5 objects, and "methods" on those objects
    - Dataset create, write / read selection, query metadata, close, …
  - Can be transparently invoked from a dynamically loaded library, without modifying application source code
    - Or even rebuilding the app binary!
  - Can be stacked, allowing many types of connectors
    - "Pass-through" and "Terminal" connector types

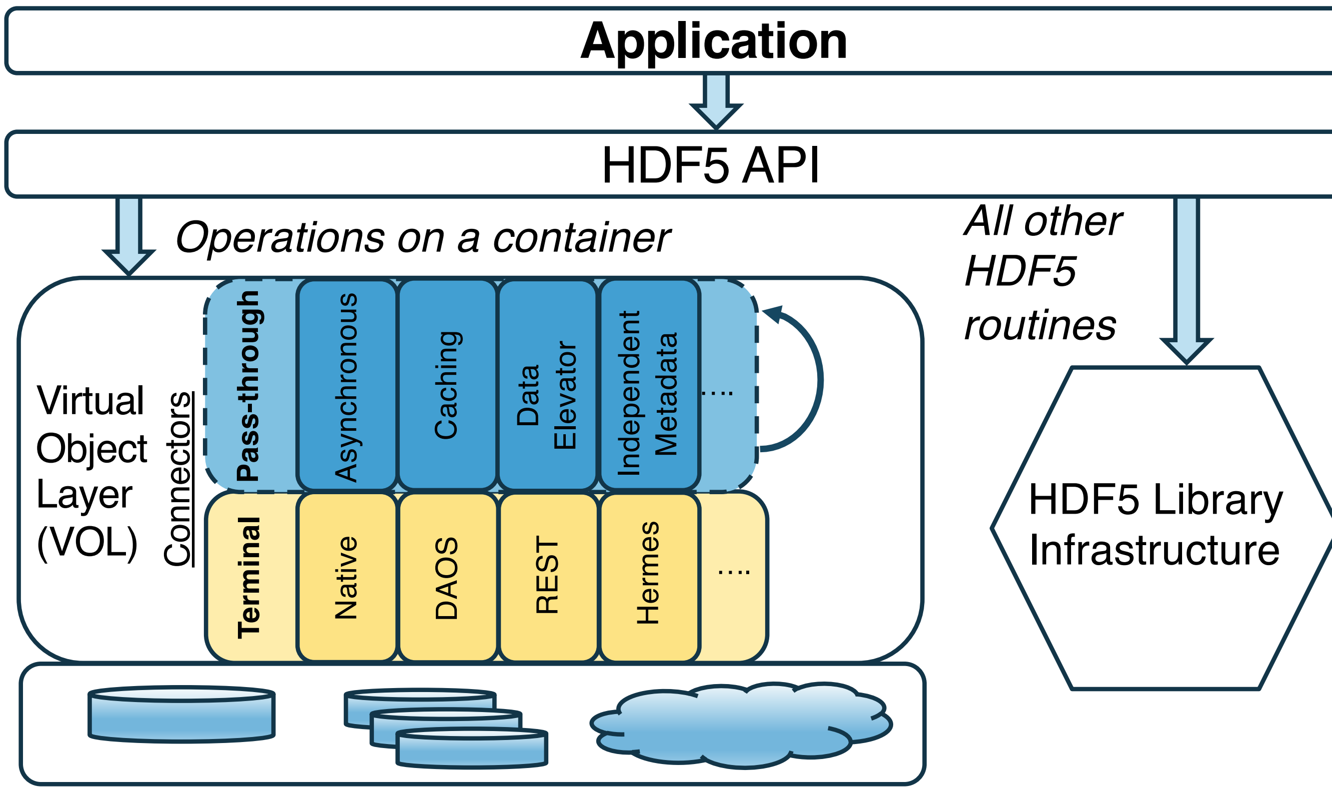
HDF5 Containers (Files)

HDF5 files, groups, and links **organize** data objects.

HDF5 datasets and attributes **store** application data.



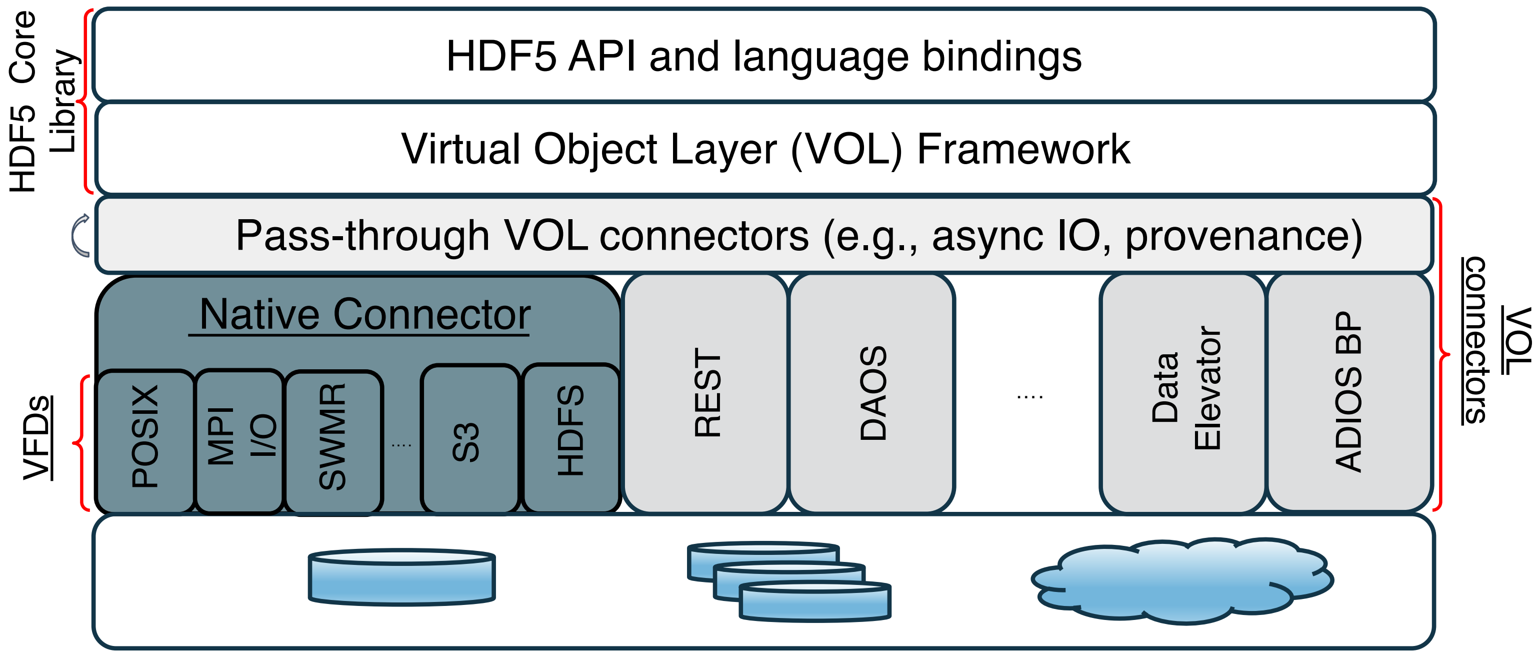
VOL: High-Level Overview



Virtual Object Layer (VOL) Connectors

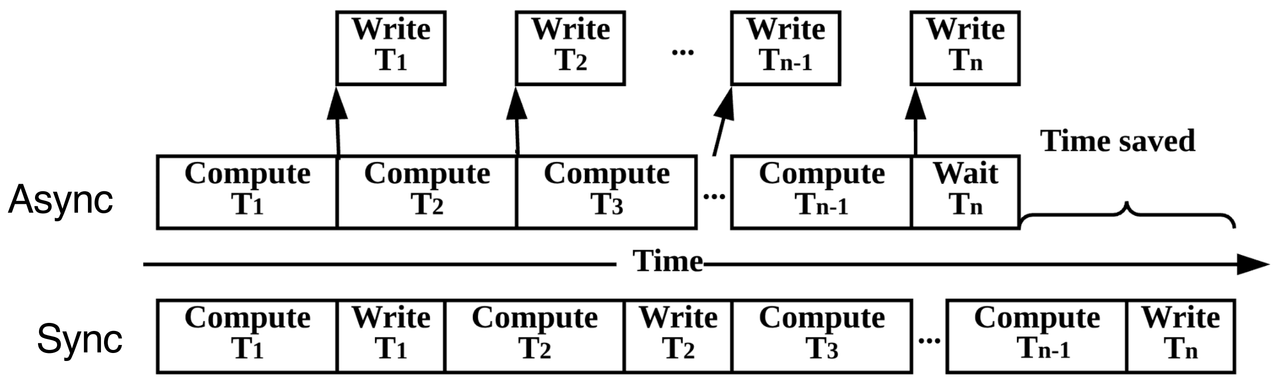
- Implement callbacks for HDF5 data model operations
- "Terminates" call by performing action directly, or "passes operation through" by invoking VOL API connector interface:
  - Pass-through - can be stacked, must eventually have terminal connector
    - Examples:
      - Provenance tracking
      - Asynchronous I/O
      - Caching
  - Terminal - non-stackable, final connector
    - Examples:
      - Remote access (e.g. cloud, streaming, etc)
      - Non-HDF5 file access (e.g. ADIOS BP, netCDF "classic", etc)
      - Object stores (e.g. DAOS, S3, etc)

VOL: Connector Architecture



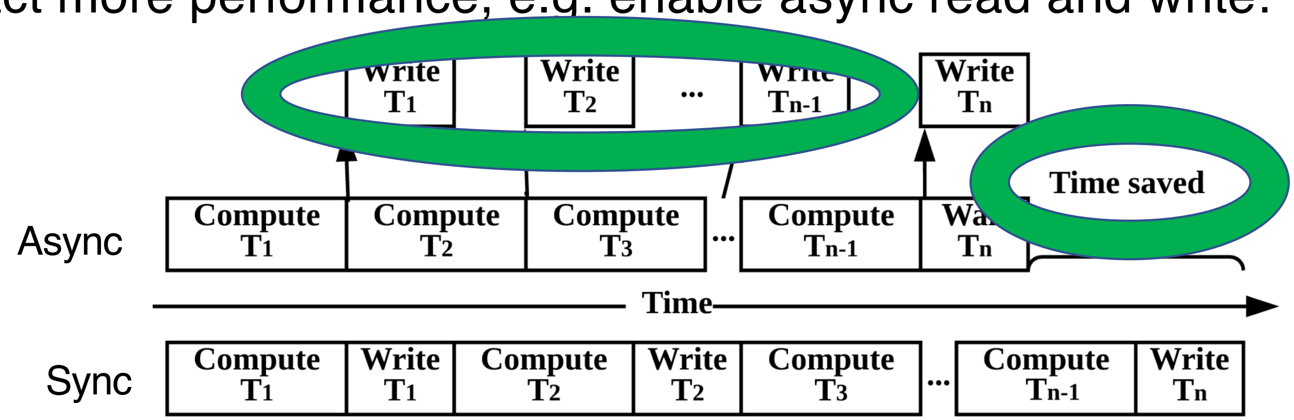
Async VOL Connector

- Pass-through VOL connector
  - Can be stacked on any other connector, to provide asynchronous operations to it
- Uses an "event set" to manage async operations
  - Can extract more performance, e.g. enable async read and write:



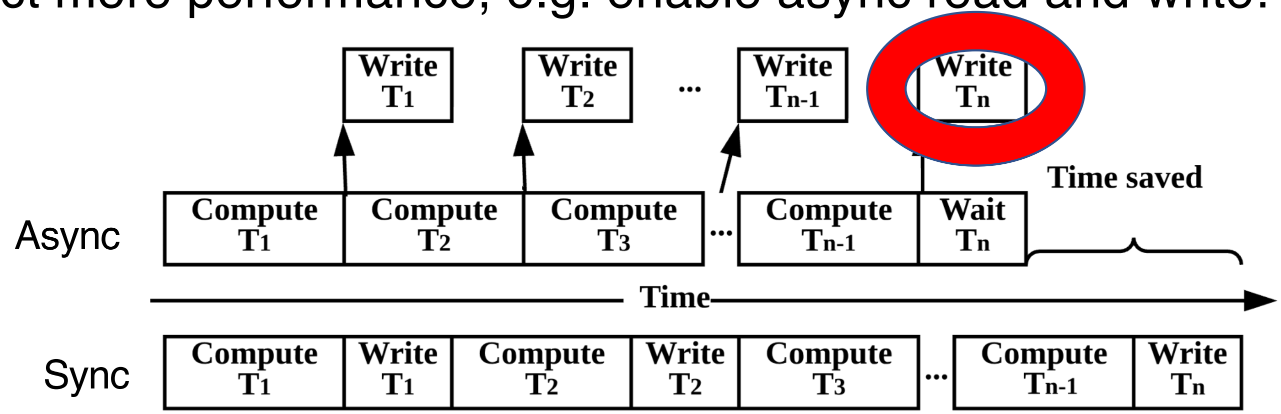
Async VOL Connector

- Pass-through VOL connector
  - Can be stacked on any other connector, to provide asynchronous operations to it
- Uses an “event set” to manage async operations
  - Can extract more performance, e.g. enable async read and write:

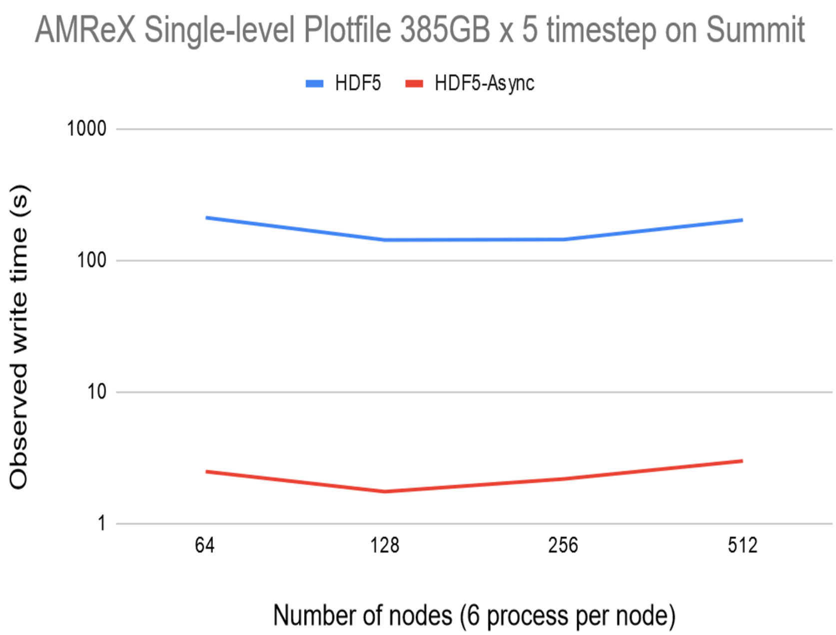


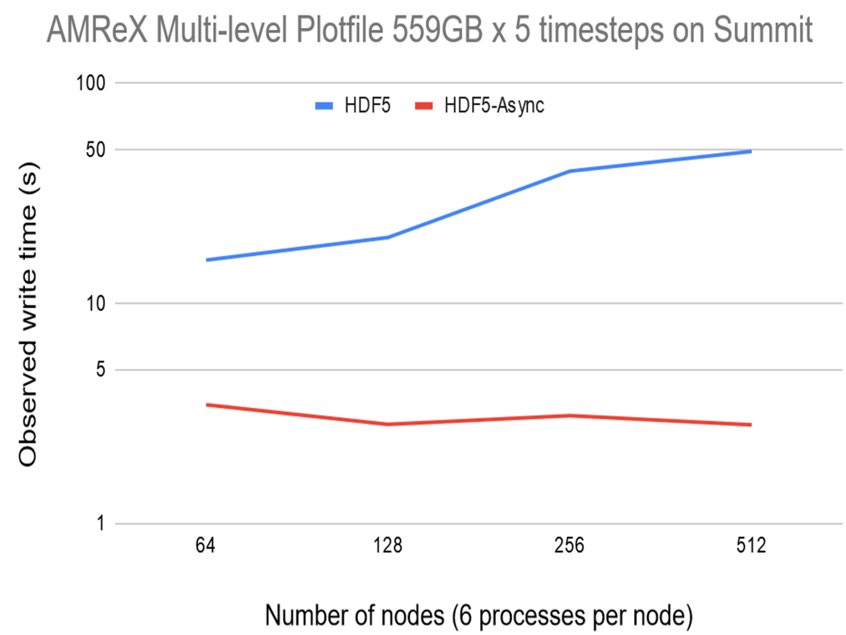
Async VOL Connector

- Pass-through VOL connector
  - Can be stacked on any other connector, to provide asynchronous operations to it
- Uses an “event set” to manage async operations
  - Can extract more performance, e.g. enable async read and write:



Async VOL Connector – Benefits





Async VOL Connector – Programming Example

```
fid = H5Fopen(..);
gid = H5Gopen(fid, ..);
did = H5Dopen(gid, ..);
status = H5Dwrite(did, ..);

status = H5Dwrite(did, ..);

...
<other user code>
...
```

Async VOL Connector – Programming Example

```
es_id = H5EScreate();                      // Create event set for tracking async operations
fid = H5Fopen_async(.., es_id);            // Asynchronous, can start immediately
gid = H5Gopen_async(fid, .., es_id);       // Asynchronous, starts when H5Fopen completes
did = H5Dopen_async(gid, .., es_id);       // Asynchronous, starts when H5Gopen completes
status = H5Dwrite_async(did, .., es_id);   // Asynchronous, starts when H5Dopen completes,
                                           //        may run concurrently with other H5Dwrite in event set
status = H5Dwrite_async(did, .., es_id);   // Asynchronous, starts when H5Dopen completes,
                                           //        may run concurrently with other H5Dwrite in event set
...
<other user code>
...
H5ESwait(es_id);                           // Wait for operations in event set to complete, buffers
                                           //   used for H5Dwrite must only be changed after wait
```

Async VOL Connector – Programming Example

```
es_id = H5EScreate();                        // Create event set for tracking async operations
fid = H5Fopen_async(.., es_id);              // Asynchronous, can start immediately
gid = H5Gopen_async(fid, .., es_id);         // Asynchronous, starts when H5Fopen completes
did = H5Dopen_async(gid, .., es_id);         // Asynchronous, starts when H5Gopen completes
status = H5Dwrite_async(did, .., es_id);     // Asynchronous, starts when H5Dopen completes,
                                             //        may run concurrently with other H5Dwrite in event set
status = H5Dwrite_async(did, .., es_id);     // Asynchronous, starts when H5Dopen completes,
                                             //        may run concurrently with other H5Dwrite in event set
...
<other user code>
...
H5ESwait(es_id);                             // Wait for operations in event set to complete, buffers
                                             //   used for H5Dwrite must only be changed after wait
```

Async VOL Connector – Programming Example

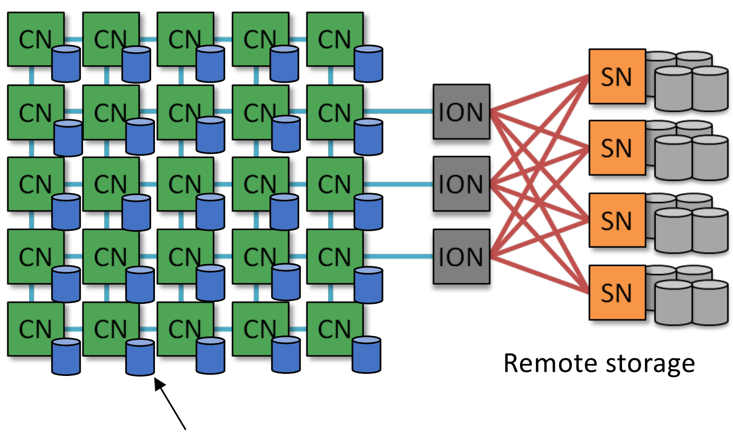
```
es_id = H5EScreate();                       // Create event set for tracking async operations
fid = H5Fopen_async(.., es_id);             // Asynchronous, can start immediately
gid = H5Gopen_async(fid, .., es_id);        // Asynchronous, starts when H5Fopen completes
did = H5Dopen_async(gid, .., es_id);        // Asynchronous, starts when H5Gopen completes
status = H5Dwrite_async(did, .., es_id);    // Asynchronous, starts when H5Dopen completes,
                                            //        may run concurrently with other H5Dwrite in event set
status = H5Dwrite_async(did, .., es_id);    // Asynchronous, starts when H5Dopen completes,
                                            //        may run concurrently with other H5Dwrite in event set

<other user code>

H5ESwait(es_id);                            // Wait for operations in event set to complete, buffers
                                            //   used for H5Dwrite must only be changed after wait
```

Async VOL Connector

- Available now: https://github.com/hpc-io/vol-async
- Future work:
  - Switch to TaskWorks thread engine
    - A portable, high-level, task engine designed for HPC workloads
    - Task dependency management, background thread execution.
  - Merge compatible VOL operations
    - If two async dataset write operations are putting data into same dataset, can merge into only one call to underlying VOL connector
    - Turn multiple ‘normal’ group create operations into a single ‘multi’ group create operation
  - Use multiple background threads
    - Needs HDF5 library thread-safety work, to drop global mutex

Cache VOL Connector - Integrating node-local storage into parallel I/O

Typical HPC storage hierarchy



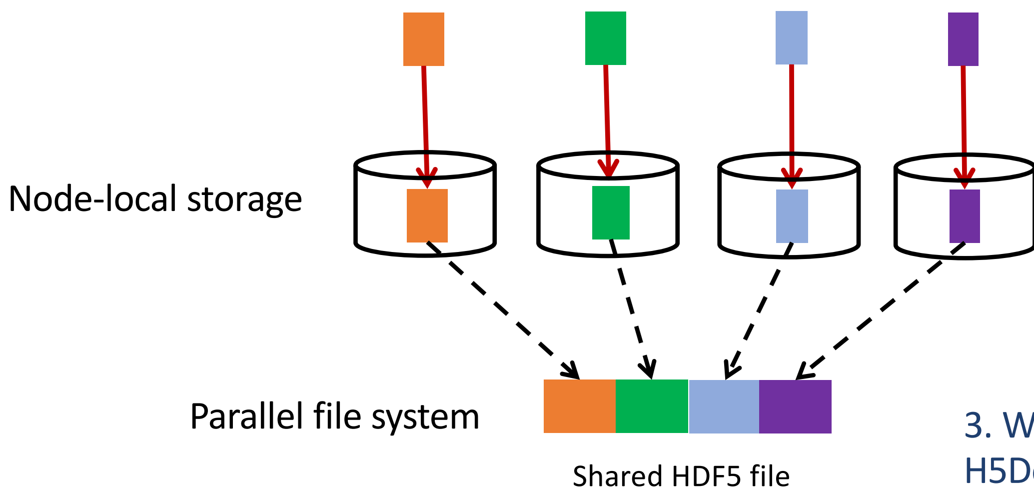
Theta @ ALCF: Lustre + SSD (128 GB / node),
ThetaGPU (DGX-3) @ ALCF: NVMe (15.4 TB / node)
Summit @ OLCF: GPFS + NVMe (1.6 TB / node)

Cache VOL

- Using node-local storage for caching / staging data for fast and scalable I/O.
- Data migration to and from the remote storage is performed in the background.
- Managing data movement in multi-tiered memory / storage through stacking multiple connectors
- All complexity is hidden from the users

Repo: https://github.com/hpc-io/vol-cache.git

Parallel Write (H5Dwrite)



1. Data is synchronously copied from the memory buffer to memory mapped files on the node-local storage using POSIX I/O.

2. Move data from memory mapped file to the parallel file system asynchronously by calling the dataset write function from the *Async VOL* stacked below the Cache VOL

3. Wait for all the tasks to finish in H5Dclose() / H5Fclose()

w/o caching	Compute	I/O (RAM→PFS)	Compute
w/ caching	Compute	RAM->NLS	Compute
			I/O: NLS->PFS

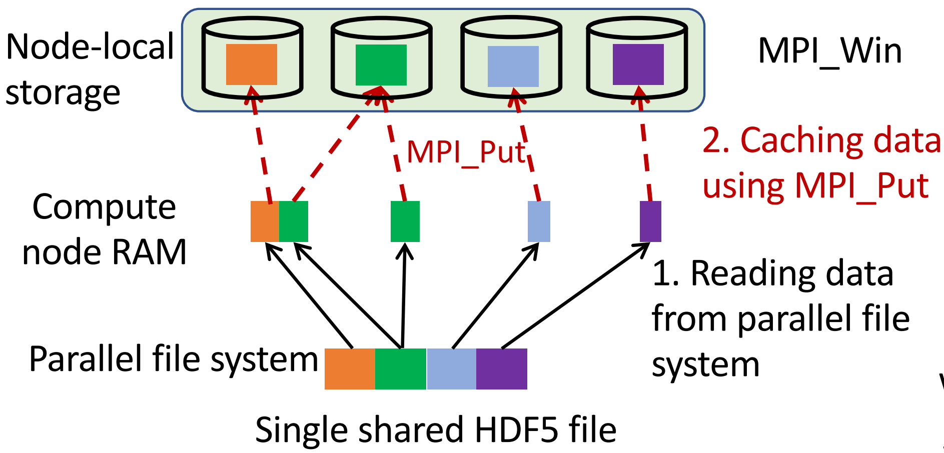
Partial overlap of compute with I/O

Details are hidden from the application developers.

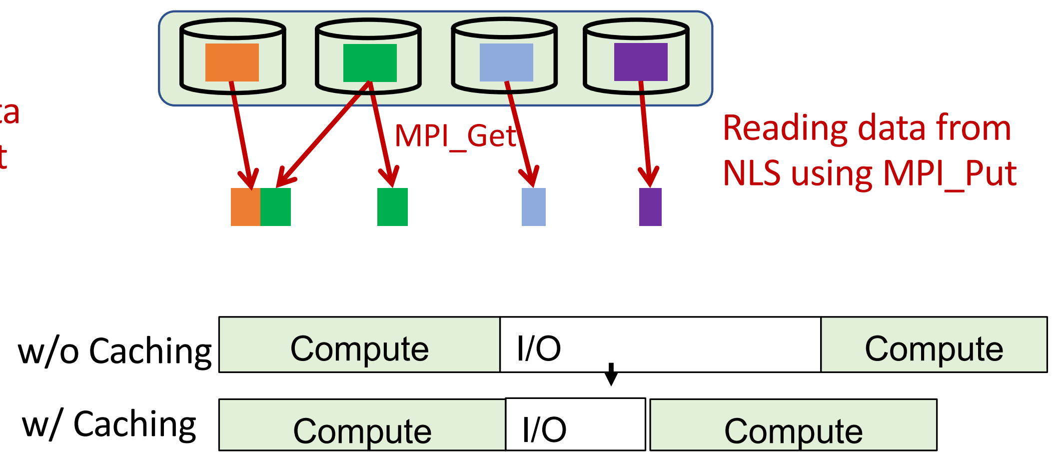
Parallel Read (H5Dread)

One-sided communication for accessing remote node storage.
- Each process exposes a part of its memory to other processes (MPI Window)
- Other processes can directly read from or write to this memory, without requiring that the remote process synchronize (MPI_Put, MPI_Get)

Create memory mapped files and attached them to a MPI_Win for one-sided remote access

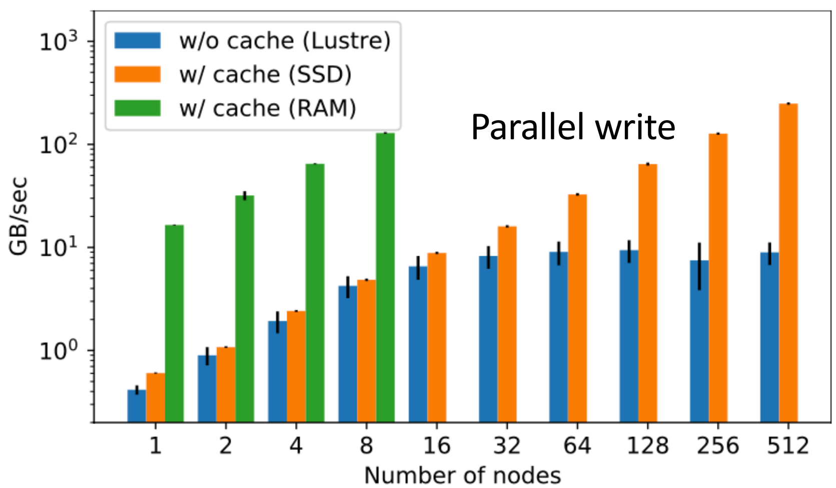


First time reading the data

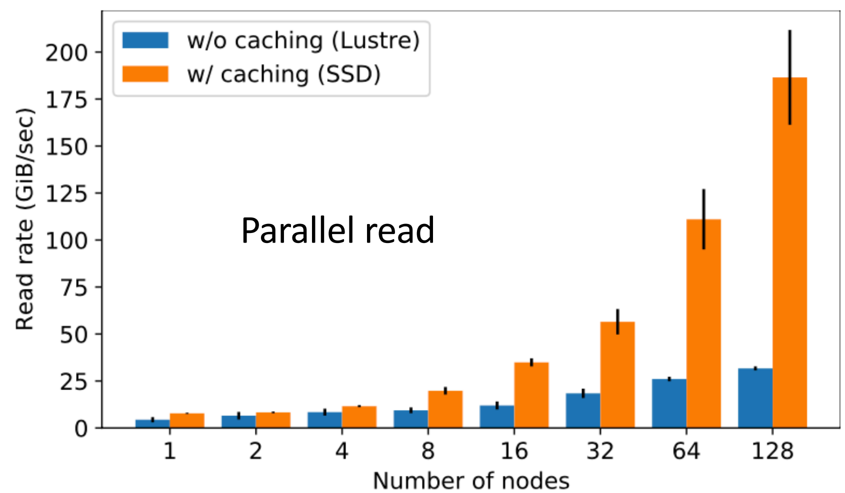


Reading the data directly from node-local storage

Performance evaluation on Theta @ ALCF



Parallel write performance on Theta w/ and w/o caching data on RAM or node-local SSDs. (Lustre stripe count is 48, and Lustre stripe size is 16MB). Each processor writes 16 MB data to a shared file.

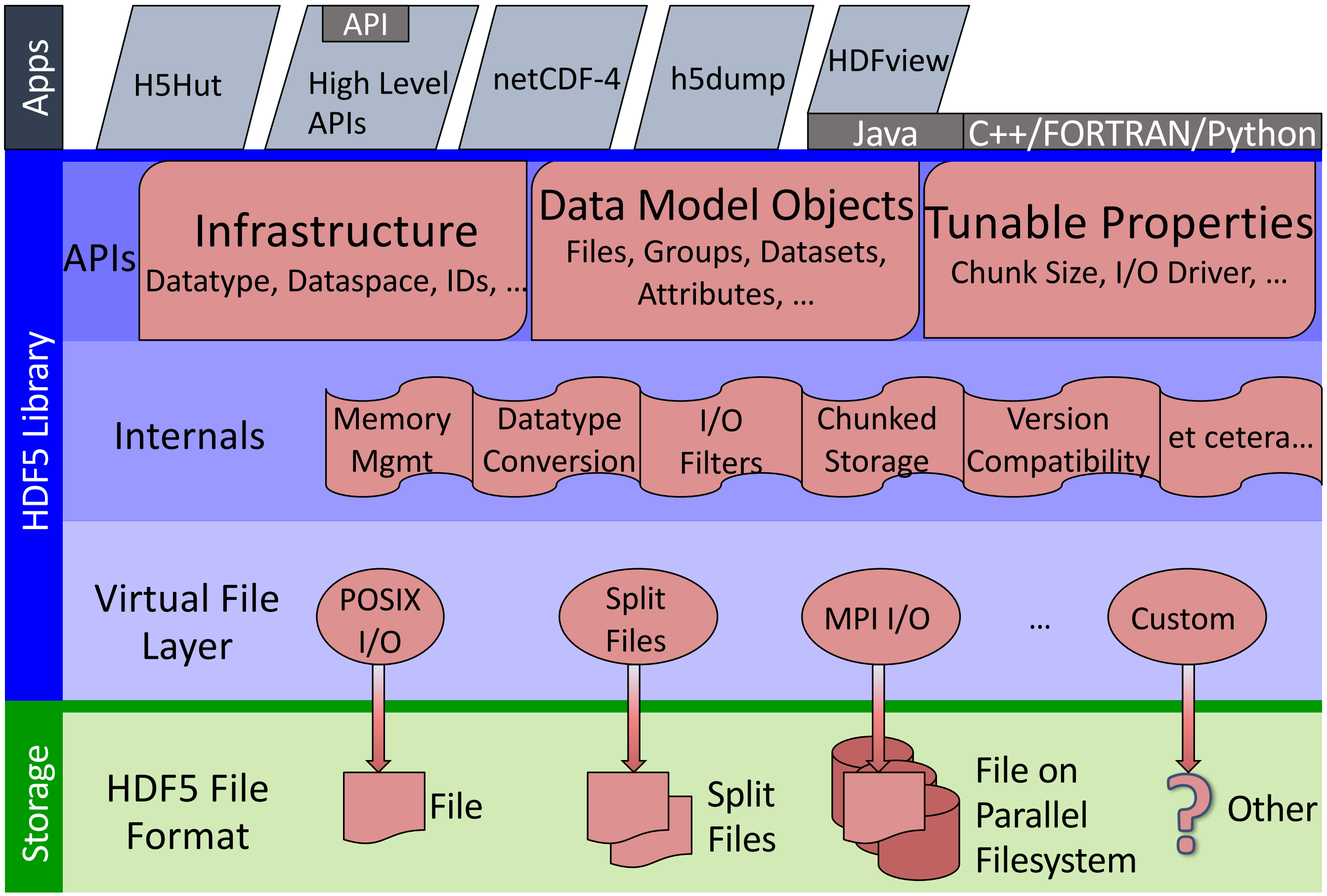


Parallel read performance on Theta. At each step, each processor reads a random batch (32) of samples (224×224×3) from a shared HDF5 file. All the processors together read the entire dataset in one iteration. The read performance is measured after the first iteration finishes.

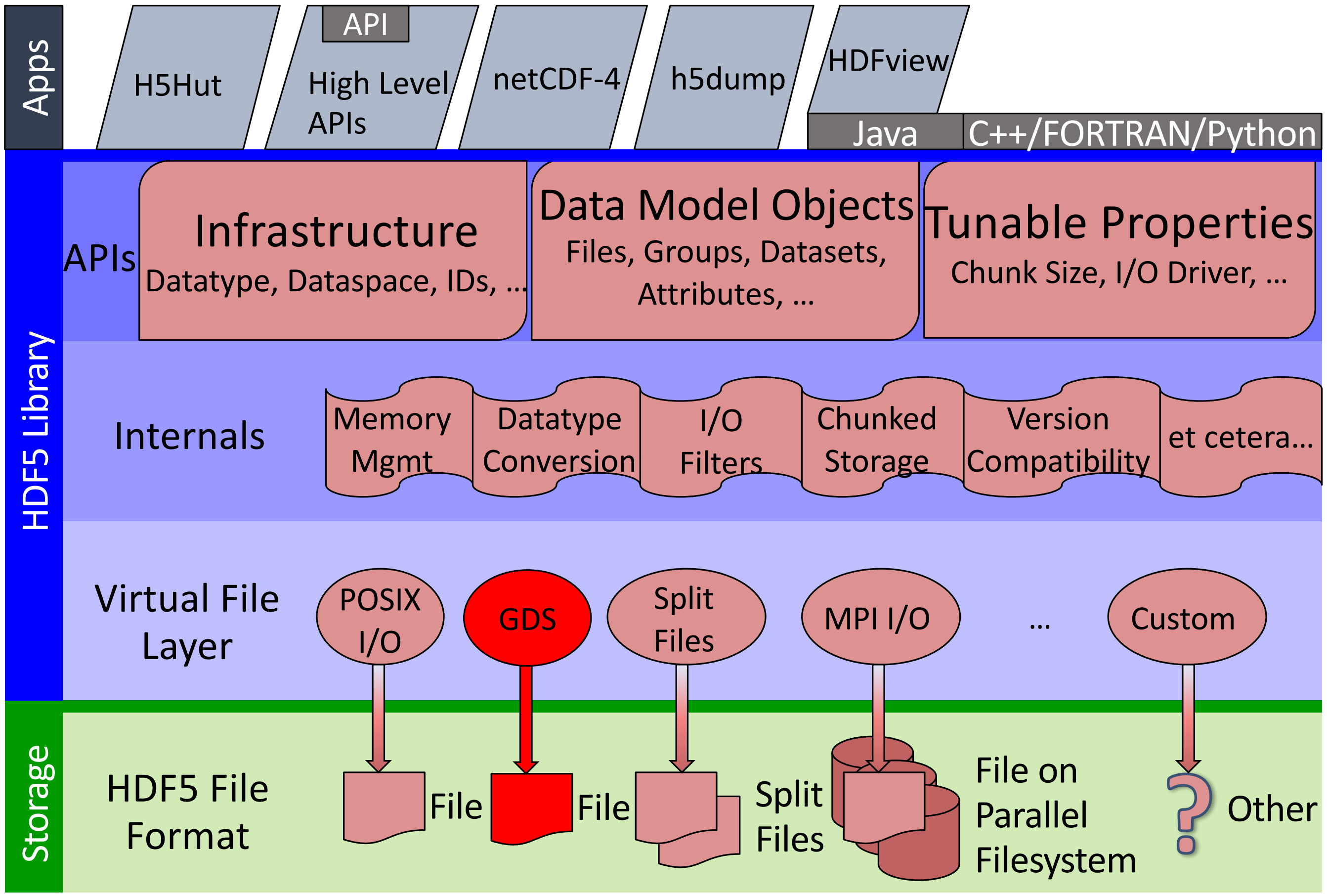
VCD100: VOL Connector Development 100

- **Subscribe to the hdf5vol mailing list:**
  - Email hdf5vol-subscribe@hdfgroup.org with "subscribe" as subject
- **Clone the "external pass-through" example VOL connector**
  - An "external" VOL connector that has all VOL callbacks implemented as transparent "no-ops", just invoking the underlying VOL connector
    - External VOL connectors can be loaded with environment variables
  - https://bitbucket.hdfgroup.org/projects/HDF5VOL/repos/external_pass_through/browse
- **Build the external pass-through connector with logging enabled:**
  - Follow instructions in README in the git repo
- **Modify to your purposes**

GPU-I/O – Fast data access from GPU Memory



GPU-I/O – Fast data access from GPU Memory

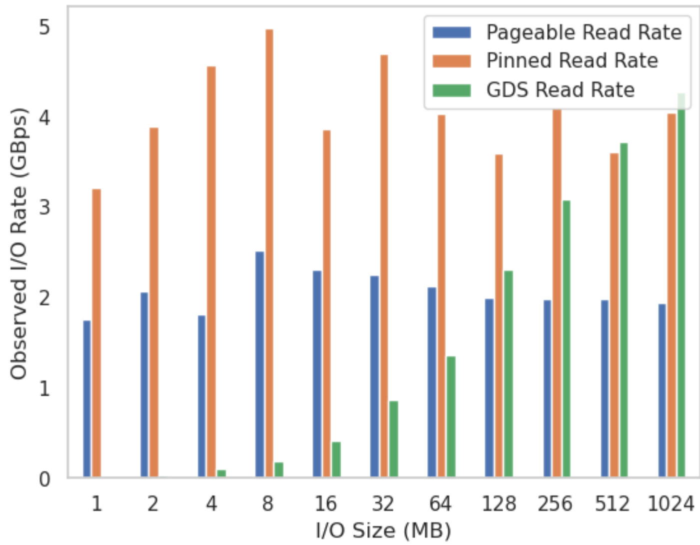


HDF5 GDS Virtual File Driver (VFD)

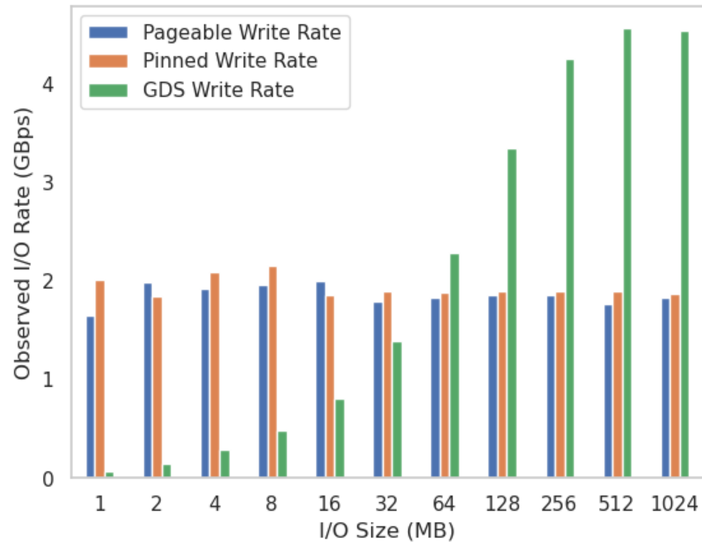
- Drop-in replacement for POSIX I/O VFD
  - Therefore: serial I/O only, currently
- Single API call to enable from applications:
  - H5Pset_fapl_gds()
- Ready for beta testers:
  - Passing all the HDF5 regression tests:
  - Available on the ‘cu_dev’ branch of HDF5 git repo:
    - https://github.com/hpc-io/hdf5/tree/cu_dev

```
bash-3.2$ (cd test; ./testhdf5)

For help use: ./testhdf5 -help
Linked with hdf5 version 1.13 release 0
Testing  -- Configure definitions (config)
Testing  -- Encoding/decoding metadata (metadata)
Testing  -- Checksum algorithm (checksum)
Testing  -- Ternary Search Trees (tst)
Testing  -- Memory Heaps (heap)
Testing  -- Skip Lists (skiplist)
Testing  -- Reference Counted Strings (refstr)
Testing  -- Low-Level File I/O (file)
Testing  -- Generic Object Functions (objects)
Testing  -- Dataspaces (h5s)
Testing  -- Dataspace coordinates (coords)
Testing  -- Shared Object Header Messages (sohm)
Testing  -- Attributes (attr)
Testing  -- Selections (select)
Testing  -- Time Datatypes (time)
Testing  -- Deprecated References (ref_deprec)
Testing  -- References (ref)
Testing  -- Variable-Length Datatypes (vltypes)
Testing  -- Variable-Length Strings (vlstrings)
Testing  -- Group & Attribute Iteration (iterate)
Testing  -- Array Datatypes (array)
Testing  -- Generic Properties (genprop)
Testing  -- UTF-8 Encoding (unicode)
Testing  -- User-Created Identifiers (id)
Testing  -- Miscellaneous (misc)

All tests were successful.
```

HDF5 GDS VFD – Early Performance Results



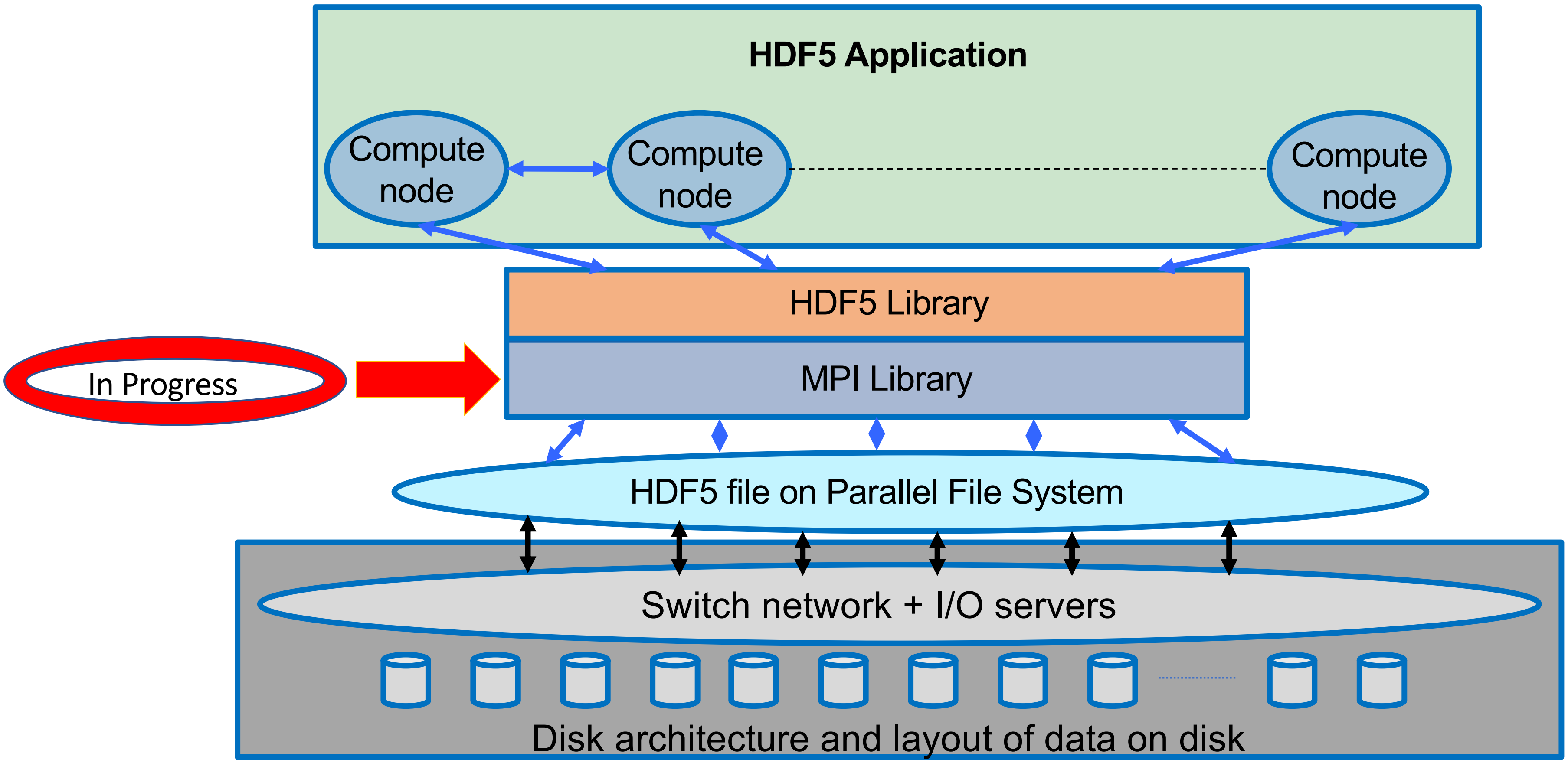
HDF5 GDS Read



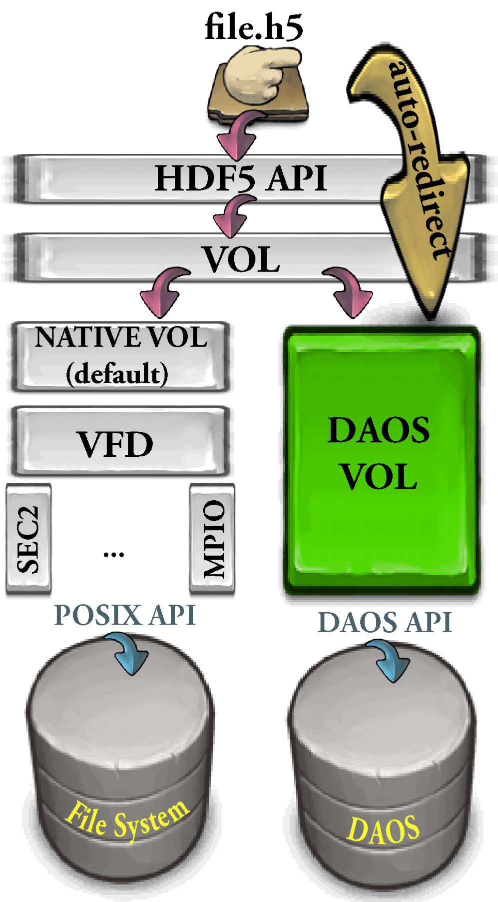
HDF5 GDS Write

(Single thread, one GPU I/O to a single NVME drive)

HDF5 GDS VFD – Parallel I/O



DAOS VOL Connector



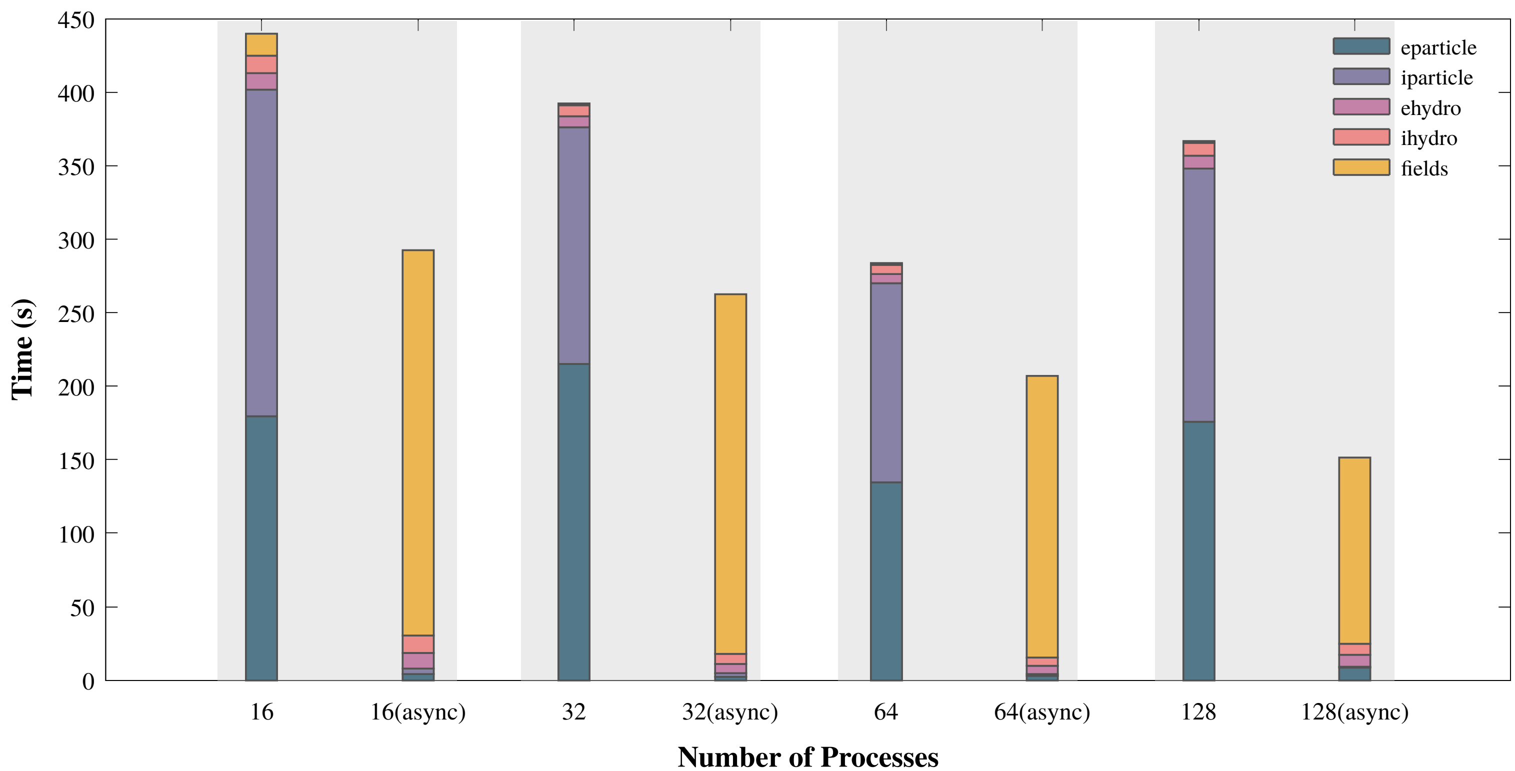
- HDF5 VOL connector for I/O to Distributed Asynchronous Object Storage (DAOS)

  https://github.com/HDFGroup/vol-daos

- Minimal code changes needed to use, enable via environment variables or through HDF5 APIs.
- HDF5 Tools are supported
  - h5dump, h5ls, h5diff, h5repack, h5copy, etc
- Supports async I/O

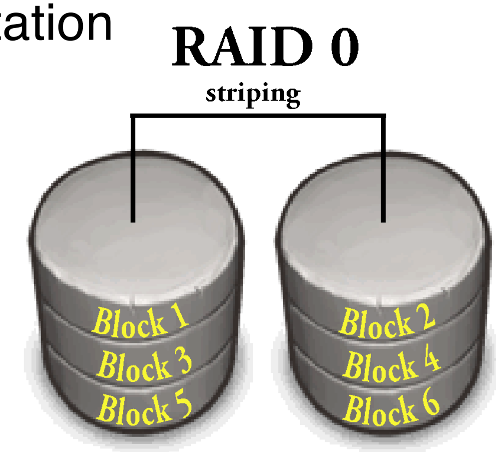
BERKELEY LAB
The HDF Group
Argonne NATIONAL LABORATORY
ECP EXASCALE COMPUTING PROJECT

VPIC – explicit async (ANL testbed)



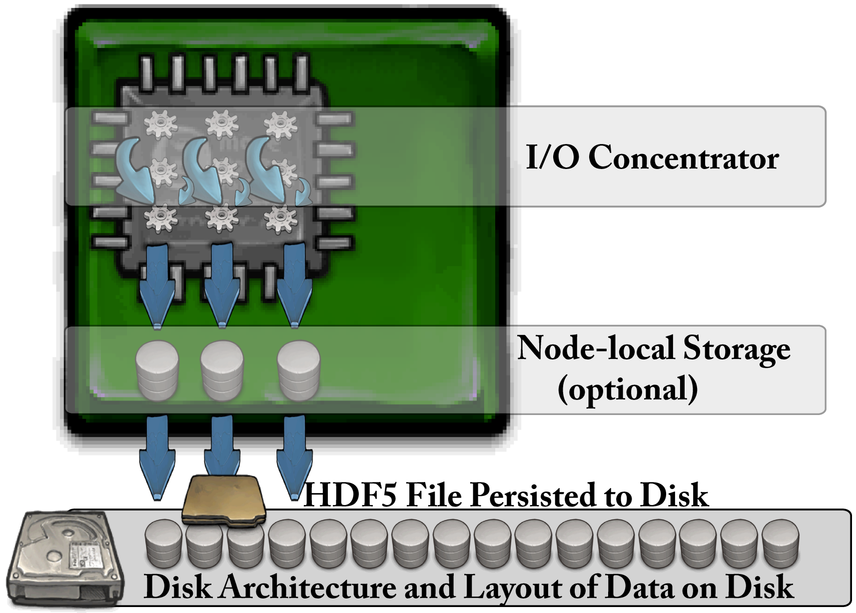
Subfiling

- Subfiling is a compromise between file-per-process (*fpp*) and a single shared file (*ssf*)
  - Multiple files organized as a Software RAID-0 Implementation
    - i. Configurable “stripe-depth” and “stripe-set size”
    - ii. A default “stripe-set” is created by using 1 file per node
    - iii. A default “stripe-depth” is 32MB
  - One metadata (.h5) file *stitching* the small files together



- Benefits
  - Better use of parallel I/O subsystem
  - Reduces the complexity of *fpp*
  - Reduced locking and contention issues to improve performance at larger processor counts over *sff*

Subfiling



For Subfiling, the HDF5 content is separated into two components:

1. **The Metadata –** written to a regular HDF5 file
    1. Final implementation has metadata embedded in subfiles
2. **The RAW data –** written logically to a RAID-0 file, and is spread over a number of individual files, each managed by an I/O concentrator.

The resulting collection can be read using Subfiling or eventually coalesced via a post-processing step into a single HDF5 file.

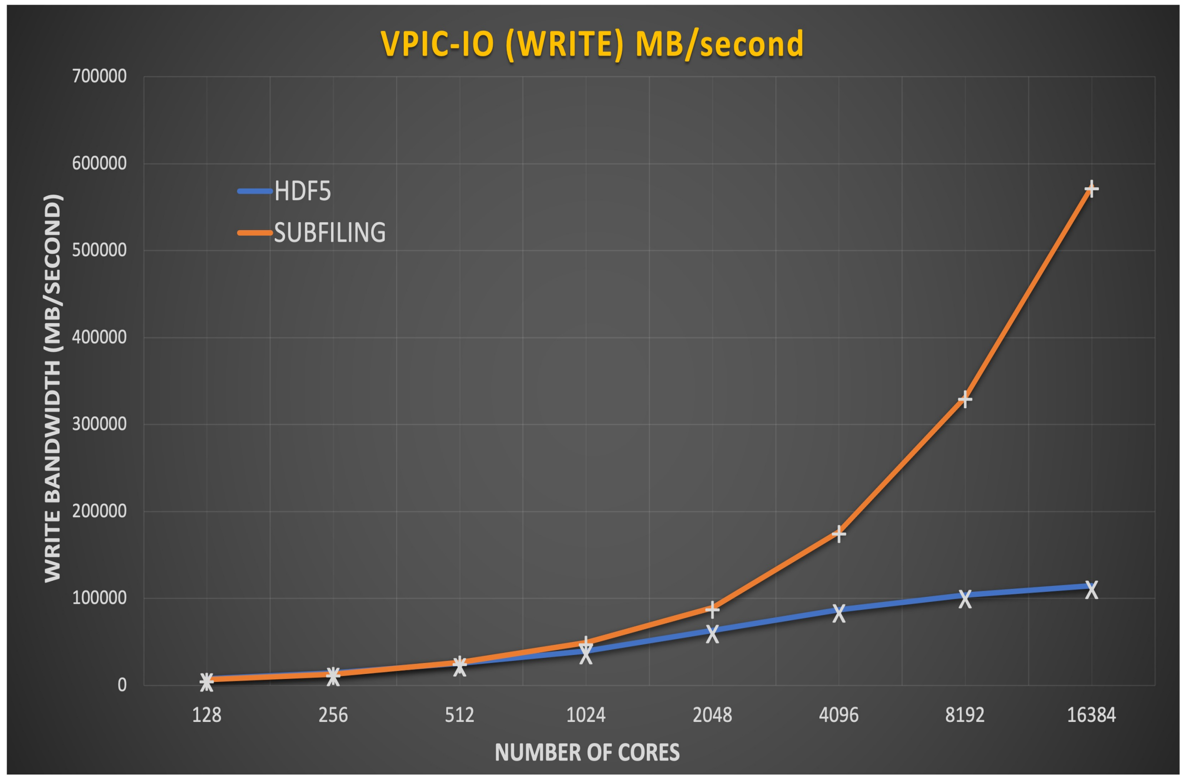
a. I/O Concentrators are implemented as independent threads attached to a normal HDF5 process.
b. MPI is utilized for communicating between HDF5 processes and the set of I/O Concentrators.
c. Because of (b), applications need to use MPI_Init_thread to initialize the MPI library.

Subfiling

Initial Results

(**h5bench – vpicio**)

- Parallel runs on *SUMMIT* showing results from 256 to 16384 cores.
- The number of *Subfiles* utilized range from **6** (for a 256 MPI rank application run) to **391** (for the 16K MPI rank application); based on 42 cores per node.



Feature: Querying datasets ‡

Objective

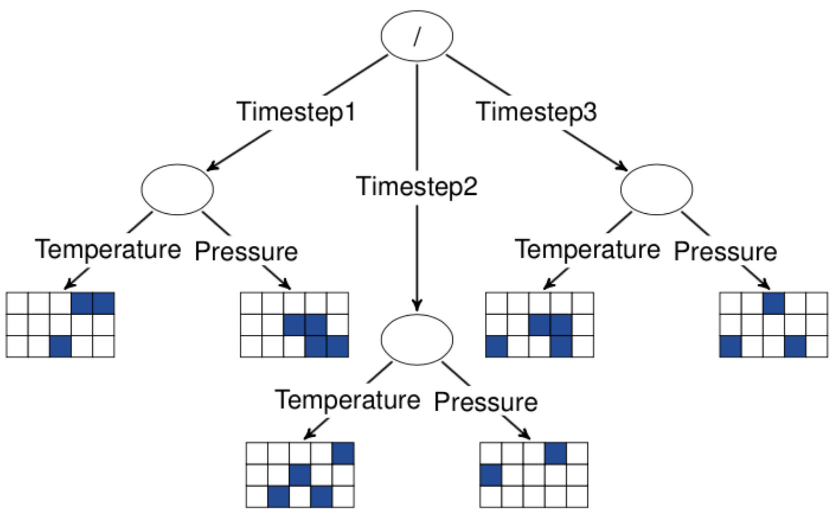
- Create complex queries on both metadata and data elements within a HDF5 container
- Retrieve the results of applying those query operations.

Solution

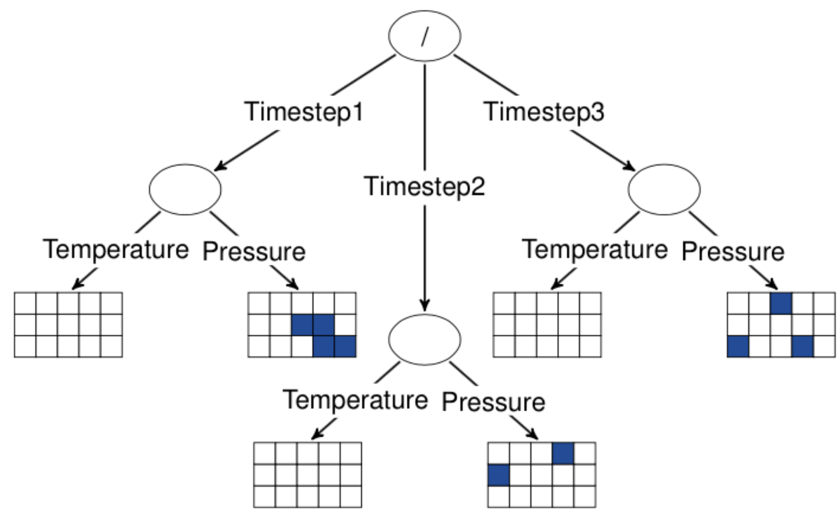
- HDF5 *index* API routines allow the creation of indexes on the contents of HDF5 objects, to improve query performance
- HDF5 *query* API routines enable the construction of query requests for execution on HDF5 containers
  - H5Qcreate
  - H5Qcombine
  - H5Qapply
  - H5Qclose

‡ HDF5 github repo containing the *querying and indexing* source code: https://github.com/HDFGroup/hdf5/tree/feature/indexing

Querying and Indexing

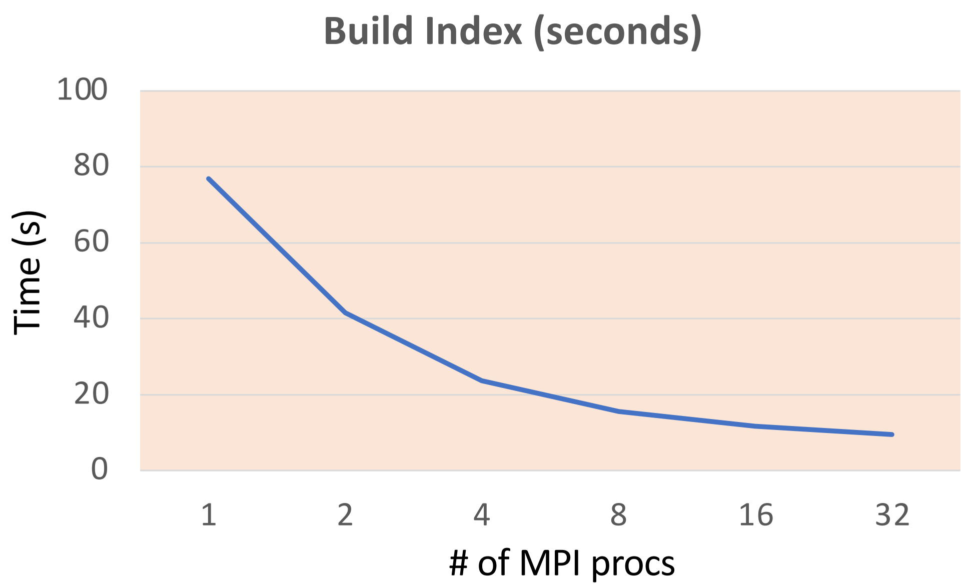


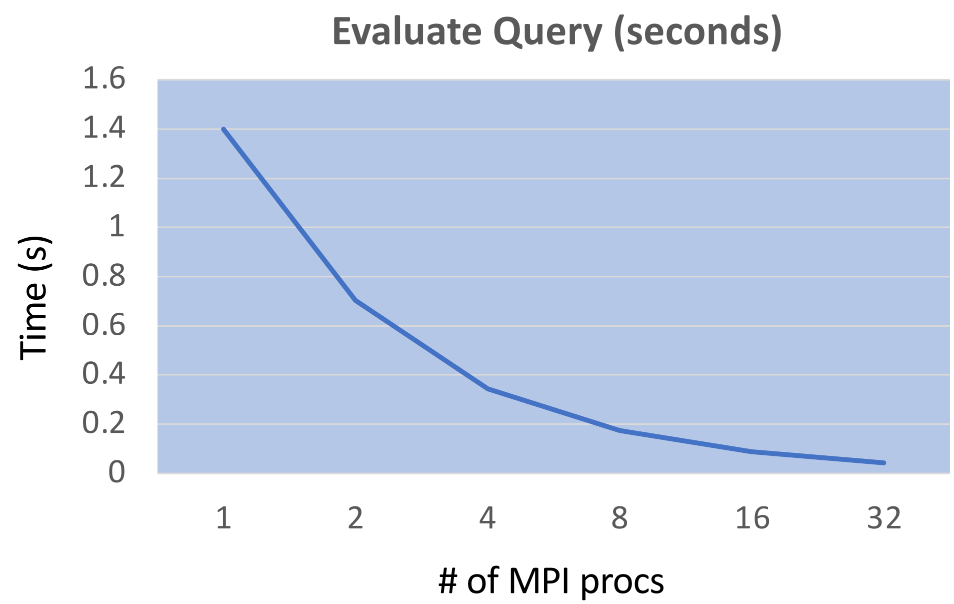
(a) Container with data element query applied



(b): HDF5 container with combine query applied

Querying and Indexing



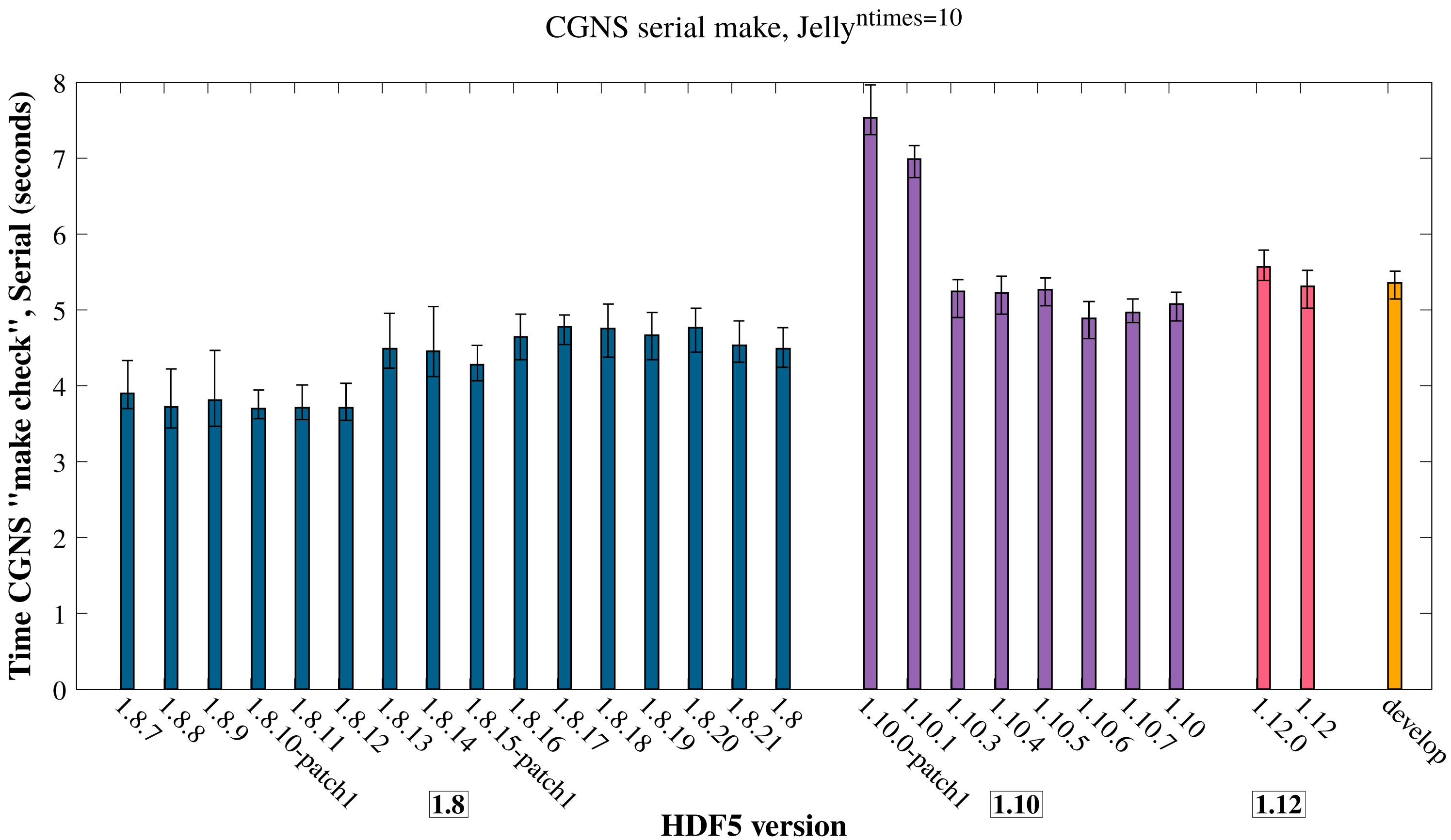


Parallel scaling of index generation and query resolution is evidenced even for small-scale experiments.

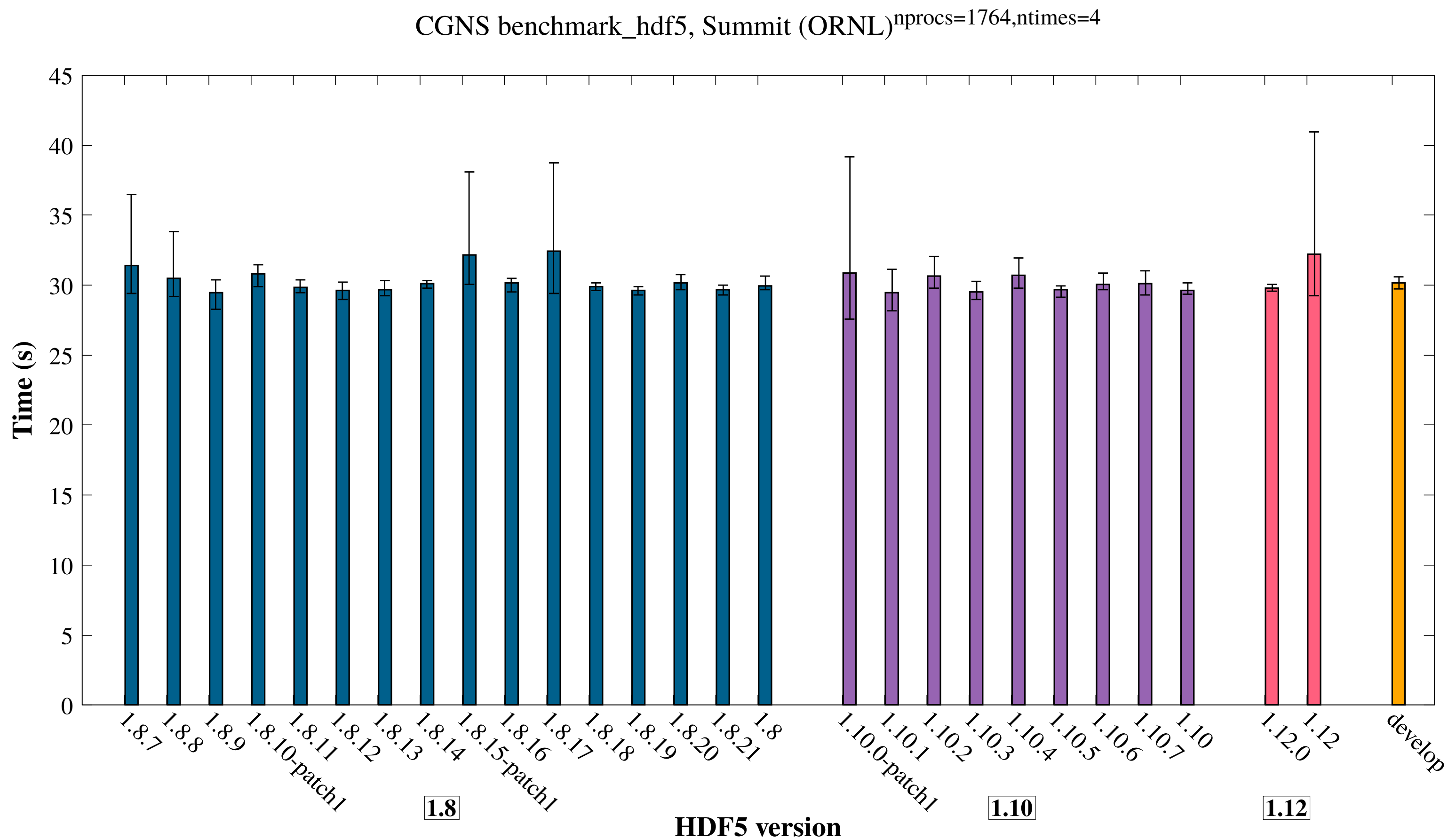
APPS

- CGNS = Computational Fluid Dynamics (CFD) General Notation System
- An effort to standardize CFD input and output data including:
  - Grid (both structured and unstructured), flow solution
  - Connectivity, boundary conditions, auxiliary information.
- Two parts:
  - A standard format for recording the data
  - Software that reads, writes, and modifies data in that format.
- An American Institute of Aeronautics and Astronautics Recommended Practice

Useful for monitoring HDF5 Performance



Useful for monitoring HDF5 Performance



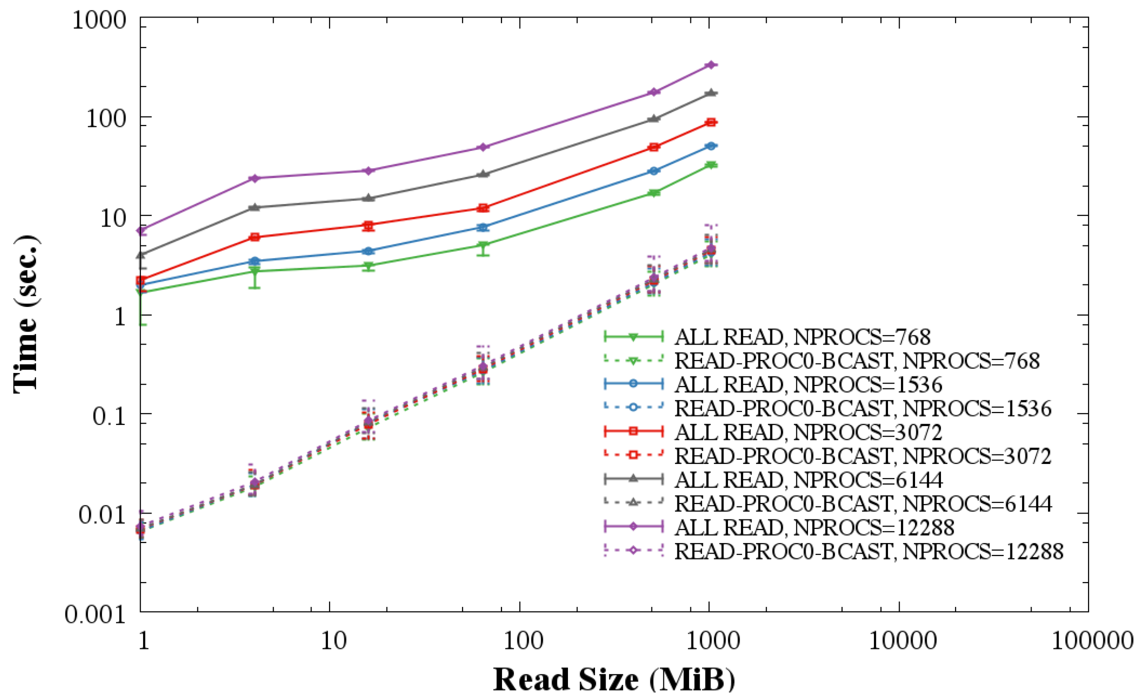
Improve the performance of reading/writing H5S_all selected datasets

(1) New in HDF5 1.10.5

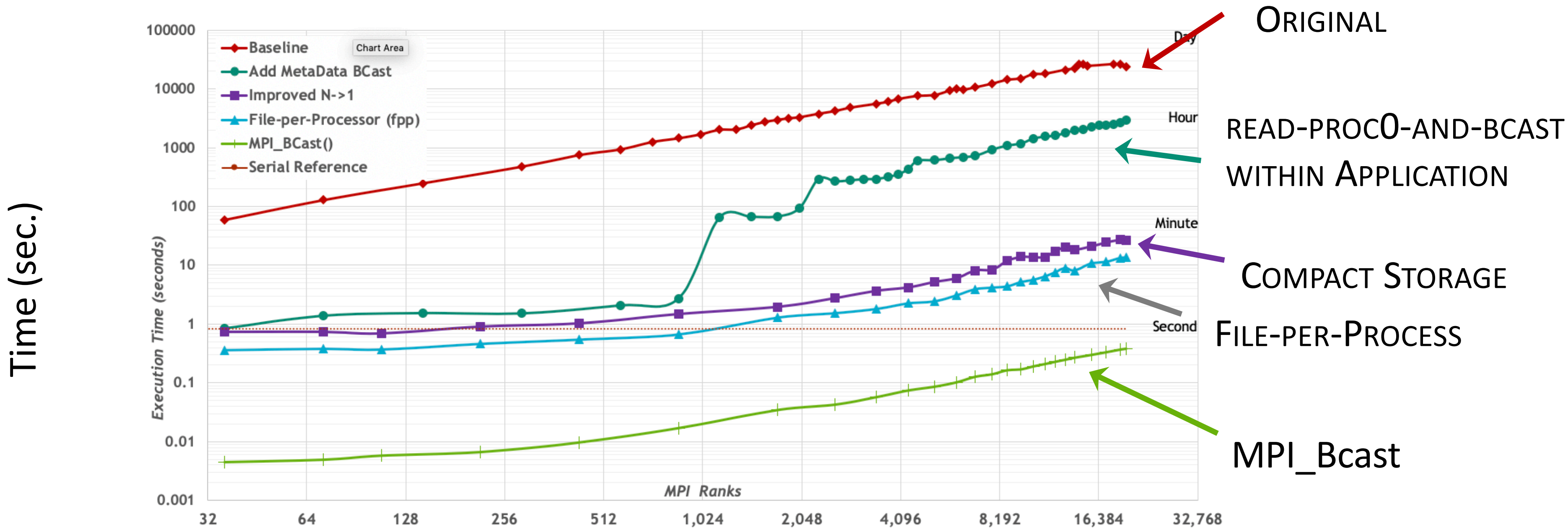
- If:
  - All the processes are reading/writing the same data
  - And the dataset is less than 2GB
- Then
  - The lowest process id in the communicator will read and broadcast the data or will write the data.

(2) Use of compact storage, or

- For compact storage, this same algorithm gets used.



SCALING OPTIMIZATIONS



Greg Sjaardema, Sandia National Labs

Challenging HDF5 Use Cases

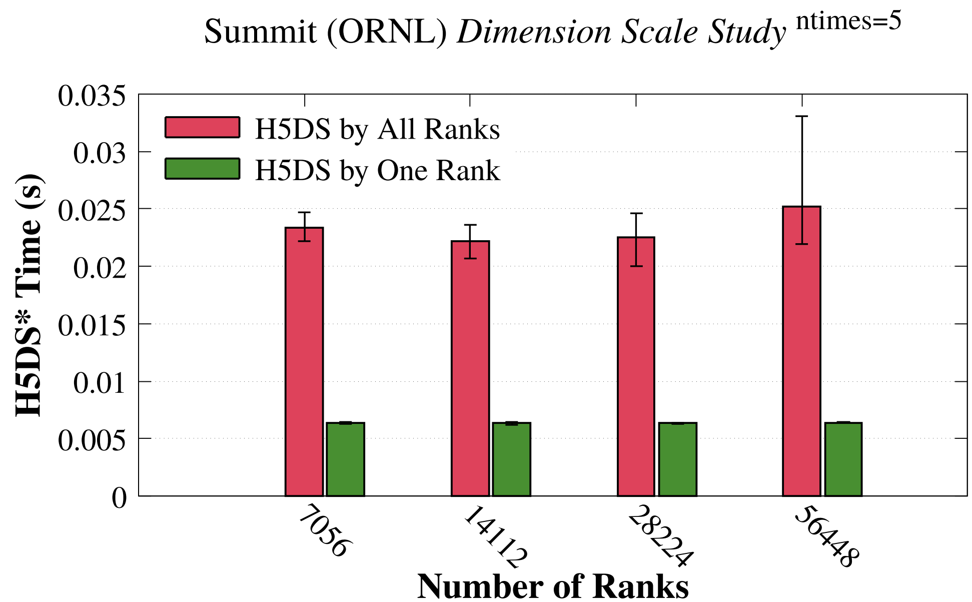
- Ideally, HDF5 parallel performance should be comparable (or better) to raw binary I/O.
- Issues with third-party libraries (netCDF, CGNS…) using HDF5:
  - Can be metadata heavy due to the need to conform to a standard format.
  - The standard's format may dictate raw data output pattern.
    - May lead to optimal write performance but poor read performance, or vice-versa.
- Mitigating performance issues
  - Implement new features in HDF5 to address metadata performance
    - Collective metadata, using the core file driver for metadata creation, etc…
  - Work with third-party libraries to use parallel file system friendly HDF5 schemes.

E3SM

E3SM: Earth system model development and simulation project

Levels of library usage:

- **Scorpio**: A high-level Parallel I/O Library for structured grid application.
  - **NetCDF**: software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.
    - **HDF5**



E3SM – E3SM-IO[1]

Issue

- E3SM-IO writes hundreds of variables, contributing to small portions of spatial-temporal values for each of the variables. Note they can be out of order (in each dimension).
- When flattened to file views, data from a process can be highly non-contiguous.

Investigation

- Alternate implementation using only HDF5 (or PnetCDF), no third-party libraries
  - A variable is expressed by an HDF5 dataset.
  - Rank 0 initializes metadata for the final HDF5 file.
  - HDF5 hyperslab is used to merge 2D/3D requests for a dataset into a single data space.
  - Memory space is sorted accordingly to align with the newly created dataspace.
  - Multi-dataset implementation can merge the collective write for all datasets into a single one.

Performance (Cori: 338 nodes, 21362 processes, 14.7GiB)

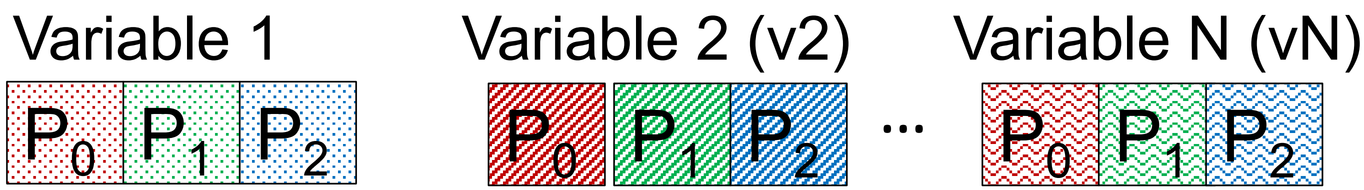
- HDF5 write performance (data only): 19.4s
- PnetCDF write performance (data only): 19.0s
- Tuning metadata write performance is in-progress

[1] https://github.com/QiaoK/E3SM-IO

HACC/GenericIO Study [1]

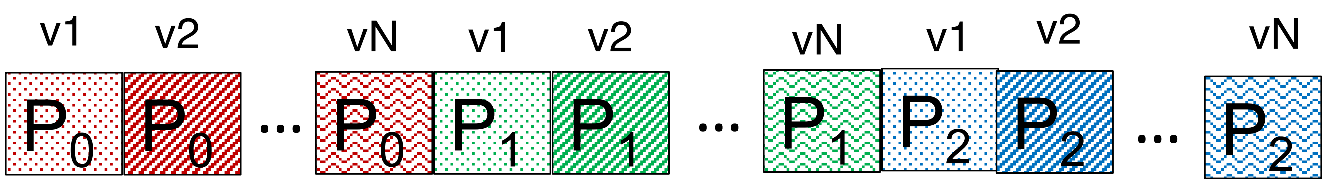
Write Pattern Effects – Data location in the file

Pattern 1 – HDF5 pattern



Variables are contiguously stored in the file

Pattern 2 – MPI-IO pattern (or HDF5 compound datatype)

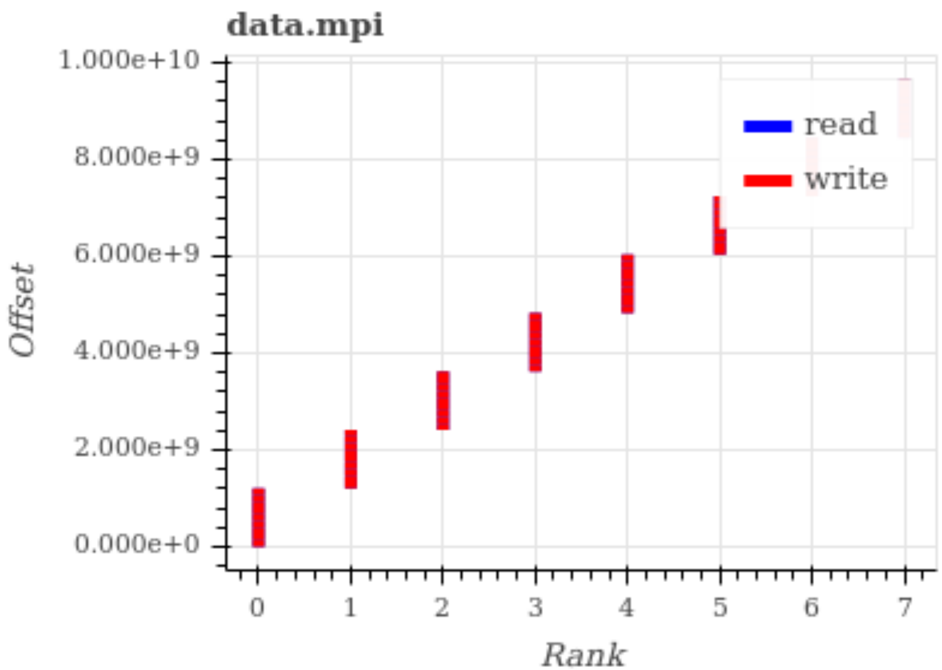


Variables are interleaved in the file

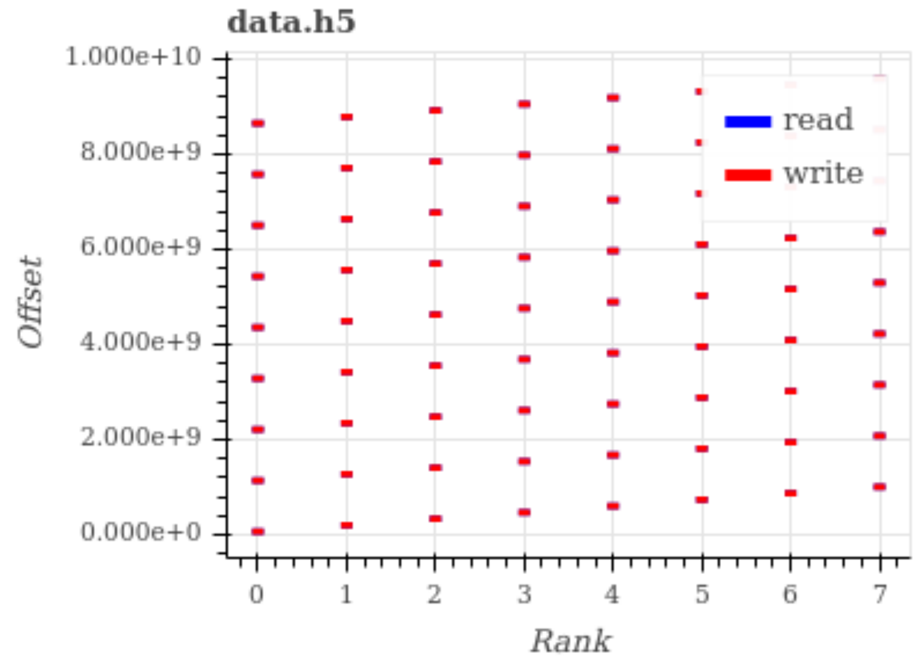
[1] https://portal.hdfgroup.org/display/HDF5/Parallel+HDF5?preview=%2F50904591%2F62458303%2FAn_IO_Study_of_ECP_Applications-2020-10-19.pdf

HACC-IO: MPI vs HDF5, why HDF5 is slow?

Example of access patterns with 8 ranks writing 9GB.



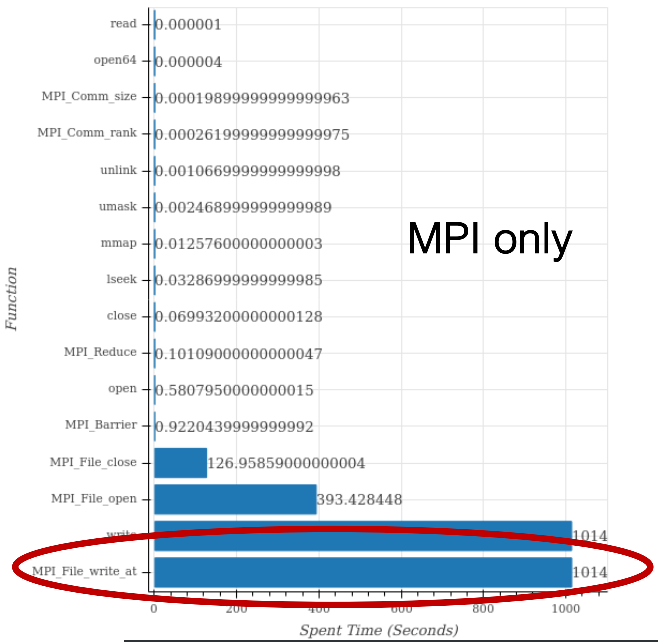
MPI-IO Access Pattern

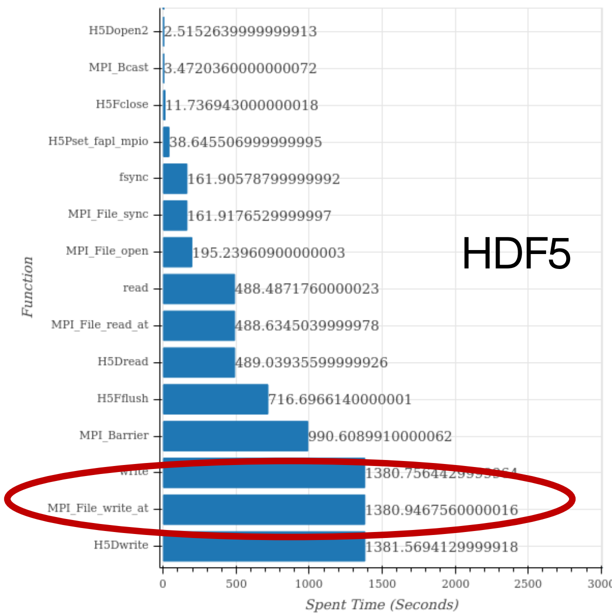


HDF5 with individual dataset

HACC-IO: MPI vs HDF5

- Same access pattern, but why MPI is faster?





MPI_File_write_at is slower in HDF5?

```
7093442 7093810 0 H5Dopen2 ['dset_id,', 'id', '0']
7093820 7093856 0 H5Sselect_hyperslab ['file_space_id', '0', '0x7fffffffb280', '(nil)', '0x7fffffffb288', '(nil)']
7093859 7093860 0 H5Sselect_hyperslab ['mem_space_id', '0', '0x7fffffffb290', '(nil)', '0x7fffffffb298', '(nil)']
7093864 7147935 0 H5Dwrite ['dset_id', 'H5T_NATIVE_DOUBLE,', 'mem_space_id', 'file_space_id', '0', '0x2aaacae4b010']
7094119 7147912 0 MPI_File_write_at ['0x8a6c58', '2048', '0x2aaacae4b010', '8388608', 'MPI_BYTE', '0x7ffffff93c0']
7094136 7094142 0 lseek ['8', '2048', '0']
7094144 7147900 0 write ['8', '0x2aaacae4b010', '8388608']
7147940 7148015 0 H5Dclose ['dset_id']
```

- HDF5 writes 2048 bytes metadata at the beginning of the file.
- This causes the alignment issue for the data writes.

HACC-IO: MPI vs HDF5

Study Summary

- HDF5 can use a different data layout to achieve similar MPI-IO access patterns.
- Stripe settings of the parallell system significantly effects write performance.
- The default metadata header can greatly slow down the write performance.
  - Proper alignment or metadata data blocksize can deliver similar HDF5 performance as a pure MPI-IO implementation

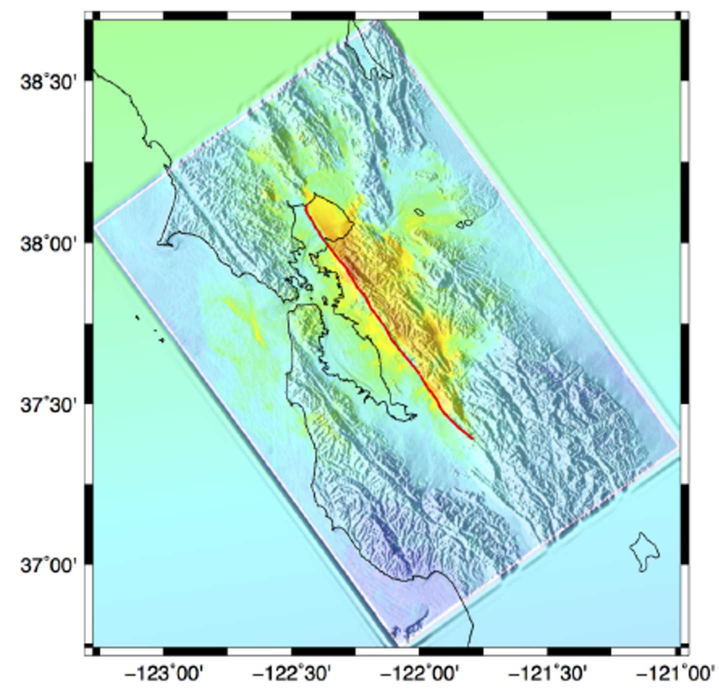
HDF5 Application Use Cases

EQSIM, Castro, Nyx

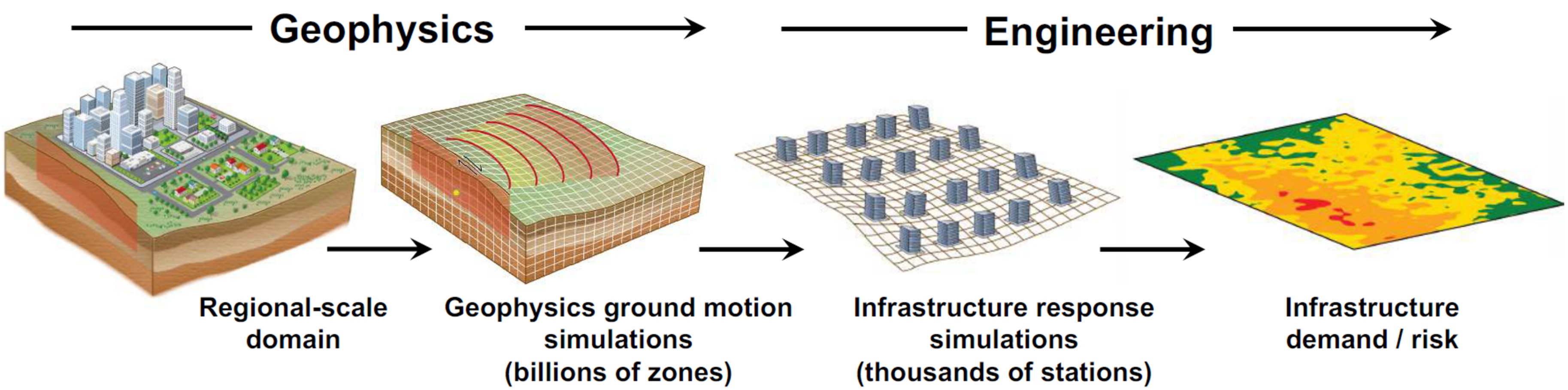
Houjun Tang, Berkeley Lab

EQSIM

- High-Performance, Multidisciplinary Simulation for Regional-Scale Earthquake Hazard and Risk Assessments
- Provide the first **strong coupling** and **linkage** between simulations of earthquake ***hazards*** (ground motions) and ***risk*** (structural system demands).
- **SW4**, main code to simulate seismic wave propagation.



EQSIM Workflow



- Seismologists sets up an earthquake event for simulation.

  Various input data

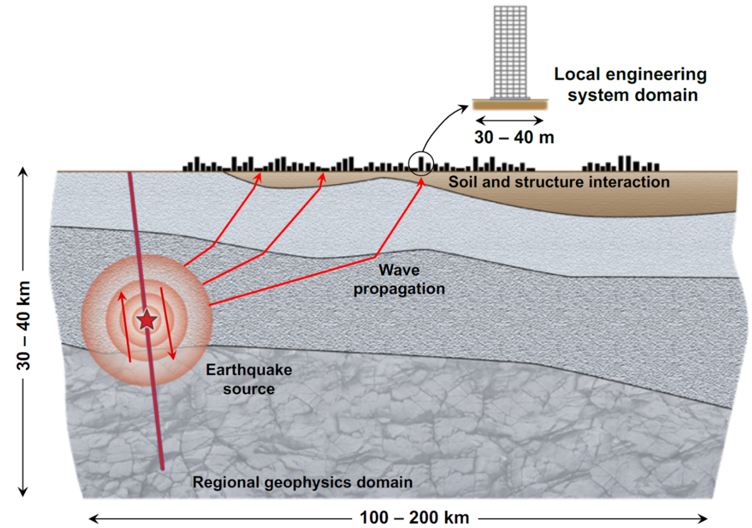
- SW4 generates and outputs ground motions for specified locations.

  1D, 2D, 3D, 4D output data

- Analysis codes (OpenSees, ESSI) produces building response.

  Visualization and analysis data

SW4 I/O pre HDF5 integration



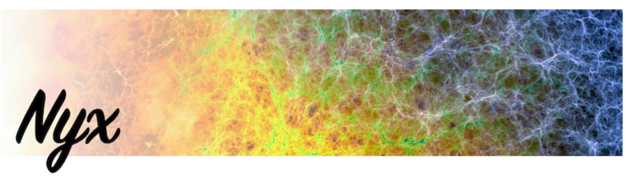
- **Input**
  - Material model and topography: **rfile** (binary).
  - Forcing function: **SRF** (ASCII).
  - Station location: input file (ASCII).
- **Output**
  - Time-series
    - Station output: USGS (ASCII), or SAC (binary), **10k+** files, a few **MB** each.
    - Subsurface output: N/A, 4D, **30+ TB**.
  - Image: sw4img (binary), 2D or 3D, **MB to GB**.
  - Checkpoint: sw4chk (binary), 3D, **40+ TB**.

SW4 I/O with HDF5 integration

- **Input**
  - Material model and topography: **sfile**: ½ size, 3x faster, new curvilinear grid.
  - Forcing function: **SRF-HDF5**: 1/3 size, 5x faster.
  - Station location: **inputHDF5**: single file.
- **Output**
  - Time-series
    - Station output: **SAC-HDF5**: 1/5 USGS, same as SAC, ***single*** file.
    - Subsurface output: **SSI**, with ZFP compression (**155GB** / **38TB**), 3x faster.
  - Image: **imgHDF5**, same as native, easy to access.
  - Checkpoint: **chkHDF5** with ZFP compression (WIP).

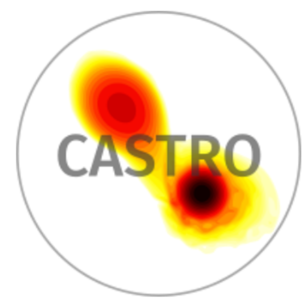
AMReX Applications

- AMReX is a software framework for massively parallel, block-structured adaptive mesh refinement (AMR) applications.
- HDF5 output format is supported for writing plotfiles and particle data, asynchronous I/O can also be enabled.



Nyx is an adaptive mesh, massively-parallel, cosmological simulation code.

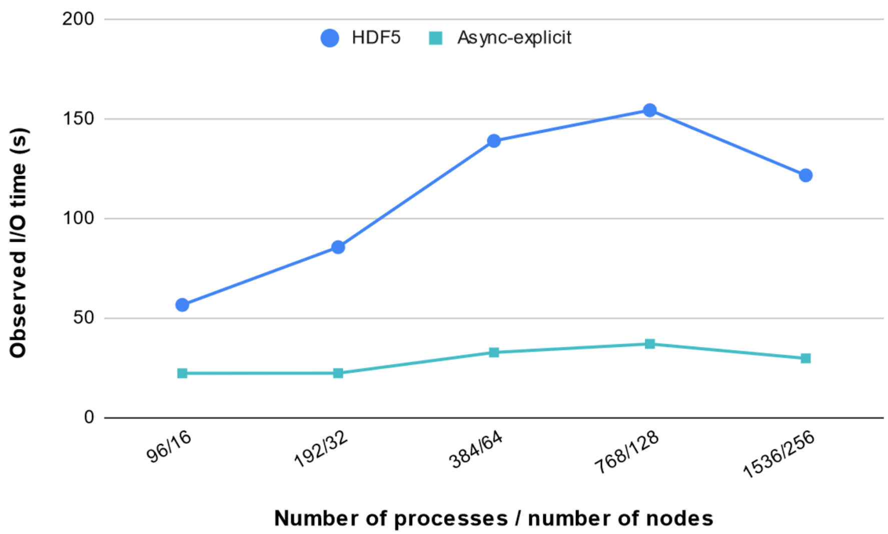
Castro is an adaptive-mesh compressible radiation / MHD / hydrodynamics code for astrophysical flows.



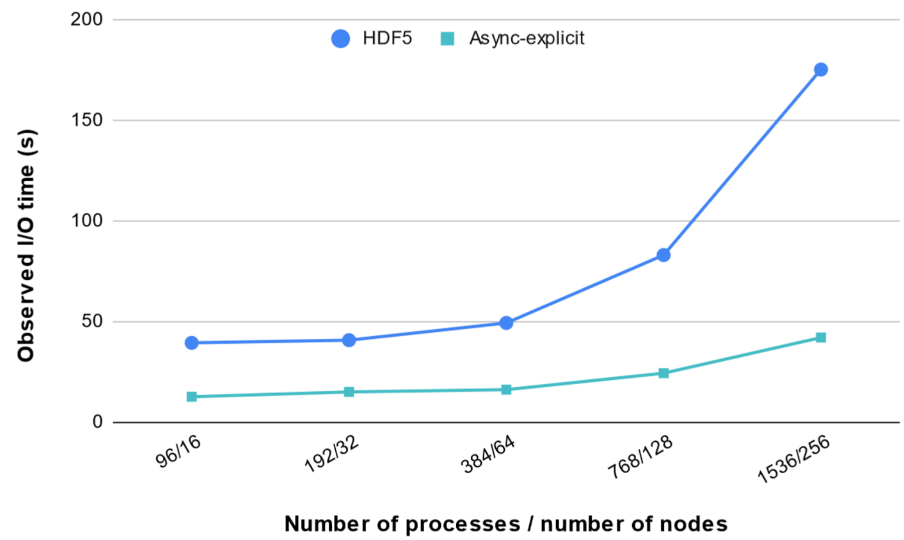
BERKELEY LAB The HDF Group Argonne NATIONAL LABORATORY

ECP EXASCALE COMPUTING PROJECT

Results on Summit



Single-level (Nyx) Workload



Multiple-level (Castro) Workload

HDF5 Tutorial at the ECP Annual Meeting 2021

April 16th, 10:00 am - 1:30 pm ET

https://ecpannualmeeting.com

HDF5 User Group meeting (HUG 2021)
October 12-15, 2021

Call for papers and presentations
https://www.hdfgroup.org/hug/hug21

Thanks and contact info

- **Contacts**
  - Suren Byna (LBNL) SByna@lbl.gov
  - Scot Breitenfeld (The HDF Group) brtnfld@hdfgroup.org
  - Quincey Koziol (LBNL - NERSC) koziol@lbl.gov
  - Elena Pourmal epourmal@hdfgroup.org

HDF5 User Support:
HDF Helpdesk: help@hdfgroup.org
HDF Forum: https://forum.hdfgroup.org/