Parallel I/O with HDF5: Overview, Tuning, and New Features

HPC Best Practices Webinar

Quincey Koziol & Suren Byna
Lawrence Berkeley National Laboratory (LBNL)

March 13, 2019
ExaHDF5 Team Members

• LBNL
  – Suren Byna, Quincey Koziol, Houjun Tang, Bin Dong, Junmin Gu, Jialin Liu, Alex Sim

• ANL
  – Venkat Vishwanath, Rick Zamora, Paul Coffman, Todd Munson

• The HDF Group
  – Scot Breitenfeld, John Mainzer, Dana Robinson, Jerome Soumagne, Richard Warren, Neelam Bagha, Elena Pourmal
WHAT IS HDF5?
HDF5 is like …
HDF5 is designed …

• for high volume and/or complex data

• for every size and type of system – from cell phones to supercomputers

• for flexible, efficient storage and I/O

• to enable applications to evolve in their use of HDF5 and to accommodate new models

• to support long-term data preservation
What is HDF5?

• HDF5 ➞ Hierarchical Data Format, v5

• Open file format
  – Designed for high volume and complex data

• Open source software
  – Works with data in the format

• An extensible data model
  – Structures for data organization and specification
HDF5 DATA MODEL
An HDF5 file is a **container** that holds data objects.
HDF5 Data Model

Dataset

Group

Attribute

File

Link

Datatype

Dataspace

HDF5 Objects
HDF5 Dataset

• HDF5 datasets organize and contain data elements.
  • HDF5 datatype describes individual data elements.
  • HDF5 dataspace describes the logical layout of the data elements.
HDF5 Dataspace

• Describes the logical layout of the elements in an HDF5 dataset
  – NULL
    • no elements
  – Scalar
    • single element
  – Simple array (*most common*)
    • multiple elements organized in a rectangular array
      – rank = number of dimensions
      – dimension sizes = number of elements in each dimension
      – maximum number of elements in each dimension
        » may be fixed or unlimited
HDF5 Dataspace

Two roles:

Spatial information for Datasets and Attributes
  – Rank and dimensions
  – Permanent part of object definition

Partial I/O: Dataspace and selection describe application’s data buffer and data elements participating in I/O

Rank = 2
Dimensions = 4x6

Rank = 1
Dimension = 10

Start = 5
Count = 3
HDF5 Datatypes

• Describe individual data elements in an HDF5 dataset

• Wide range of datatypes supported
  – Integer
  – Float
  – Enum
  – Array
  – User-defined (e.g., 13-bit integer)
  – Variable-length types (e.g., strings, vectors)
  – Compound (similar to C structs)
  – More …
HDF5 Datatypes

• Describe individual data elements in an HDF5 dataset

• Wide range of datatypes supported
  – Integer
  – Float
  – Enum
  – Array (similar to matrix)
  – User-defined (e.g., 12-bit integer, 16-bit float)
  – Variable-length types (e.g., strings, vectors)
  – Compound (similar to C structs)
  – More …
HDF5 Dataset

Datatype: 32-bit Integer

Dataspace: Rank = 2
Dimensions = 5 x 3
HDF5 Dataset with Compound Datatype

Compound Datatype:
- `uint16`
- `char`
- `int32`
- `2x3x2 array of float32`

Dataspace: Rank = 2
Dimensions = 5 x 3
How are data elements stored? (1/2)

- **Contiguous (default)**
  - Data elements stored physically adjacent to each other

- **Chunked**
  - Better access time for subsets; extendible

- **Chunked & Compressed**
  - Improves storage efficiency, transmission speed
How are data elements stored? (2/2)

**Compact**
- Data elements stored directly within object's metadata
- Diagram: Data in the file (Dataset Object Header)

**External**
- Data elements stored outside the HDF5 file, possibly in another file format
- Diagram: Data in the file (Dataset Object Header)

**Virtual**
- Data elements actually stored in “source datasets”, using selections to map
- Diagram: Data in the file (Dataset Object Header)
HDF5 Groups and Links

HDF5 groups and links organize data objects.

Every HDF5 file has a root group

Experiment Notes:
Serial Number: 99378920
Date: 3/13/09
Configuration: Standard 3

Parameters
10;100;1000

Timestep
36,000

lat | lon | temp
---- | ---- | ----
12   | 23   | 3.1
15   | 24   | 4.2
17   | 21   | 3.6

Viz
SimOut

HDF5 groups and links organize data objects.
HDF5 Attributes

• Attributes “decorate” HDF5 objects
• Typically contain user metadata
• Similar to Key-Values:
  – Have a unique name (for that object) and a value
• Analogous to a dataset
  – “Value” is an array described by a datatype and a dataspace
  – Do not support partial I/O operations; nor can they be compressed or extended
HDF5 SOFTWARE
HDF5 Home Page

- Latest release: HDF5 1.10.5 (1.10.6 coming soon)

HDF5 source code:
- Written in C, and includes optional C++, Fortran, and Java APIs
  - Along with “High Level” APIs
- Contains command-line utilities (h5dump, h5repack, h5diff, ..) and compile scripts

HDF5 pre-built binaries:
- When possible, include C, C++, Fortran, Java and High Level libraries.
  - Check ./lib/libhdf5.settings file.
- Built with and require the SZIP and ZLIB external libraries
Useful Tools For New Users

**h5dump:**
Tool to “dump” or display contents of HDF5 files

**h5cc, h5c++, h5fc:**
Scripts to compile applications (like mpicc, …)

**HDFView:** Java browser to view HDF5 files
http://support.hdfgroup.org/products/java/hdfview/

**HDF5 Examples (C, Fortran, Java, Python, Matlab, …)**
http://support.hdfgroup.org/HDF5/examples/
HDF5 PROGRAMMING MODEL AND API
HDF5 Software Layers & Storage

HDF5 Library
- Internals
  - Memory Mgmt
  - Datatype Conversion
  - Filters
  - Chunked Storage
  - Version Compatibility
- Virtual File Layer
  - I/O Drivers
    - Posix I/O
    - Split Files
    - MPI I/O
    - Custom
- Other

HDF5 File Format
- File
- Split Files
- File on Parallel Filesystem

HDF5 Data Model Objects
- Groups, Datasets, Attributes, ...

Tunable Properties
- Chunk Size, I/O Driver, ...

Language Interfaces
- C, Fortran, C++

Apps
- High Level APIs
- H5Part
- netCDF-4
- h5dump
- HDFview
- VPIC

Internals
- Memory Mgmt
- Datatype Conversion
- Filters
- Chunked Storage
- Version Compatibility
- and so on...
The General HDF5 API

• C, Fortran, Java, C++, and .NET bindings
  – Also: IDL, MATLAB, Python (H5Py, PyTables), Perl, ADA, Ruby, …
• C routines begin with prefix: H5?
  ? is a character corresponding to the type of object the function acts on

Example Functions:

- **H5D**: Dataset interface  \(\text{e.g., H5Dread}\)
- **H5F**: File interface  \(\text{e.g., H5Fopen}\)
- **H5S**: dataSpace interface  \(\text{e.g., H5Sclose}\)
The HDF5 API

• For flexibility, the API is extensive
  ✓ 300+ functions

• This can be daunting… but there is hope
  ✓ A few functions can do a lot
  ✓ Start simple
  ✓ Build up knowledge as more features are needed
General Programming Paradigm

• Object is opened or created
• Object is accessed, possibly many times
• Object is closed

• Properties of object are optionally defined
  ✓ Creation properties (e.g., use chunking storage)
  ✓ Access properties
Basic Functions

- **H5F** create (H5F open) → create (open) File
- **H5S** create_simple/H5S create → create dataSpace
  - **H5D** create (H5D open) → create (open) Dataset
    - **H5D** read, **H5D** write → access Dataset
    - **H5D** close → close Dataset
- **H5S** close → close dataSpace
- **H5F** close → close File
Other Common Functions

Data Spaces:
- H5Sselect_hyperslab (Partial I/O)
- H5Sselect_elements (Partial I/O)
- H5Dget_space

Data Types:
- H5Tcreate, H5Tcommit, H5Tclose
- H5Tequal, H5Tget_native_type

Groups:
- H5Gcreate, H5Gopen, H5Gclose

Attributes:
- H5Acreate, H5Aopen_name, H5Aclose
- H5Aread, H5Awrite

Property lists:
- H5Pcreate, H5Pclose
- H5Pset_chunk, H5Pset_deflate
PARALLEL HDF5
(MPI-)Parallel vs. Serial HDF5

- PHDF5 allows multiple MPI processes in an MPI application to perform I/O to a single HDF5 file
- Uses a standard parallel I/O interface (MPI-IO)
- Portable to different platforms
- PHDF5 files ARE HDF5 files conforming to the HDF5 file format specification
- The PHDF5 API consists of:
  - The standard HDF5 API
  - A few extra knobs and calls
  - A parallel “etiquette”
Standard HDF5 “Skeleton”

- `H5Fcreate (H5Fopen)`  
  - create (open) File

- `H5Screate_simple/H5Screate`  
  - create dataSpace

- `H5Dcreate (H5Dopen)`  
  - create (open) Dataset

- `H5Dread, H5Dwrite`  
  - access Dataset

- `H5Dclose`  
  - close Dataset

- `H5Sclose`  
  - close dataSpace

- `H5Fclose`  
  - close File
Example of a PHDF5 C Program

A parallel HDF5 program has a few extra calls

```c
file_id = H5Fcreate(FNAME, ..., H5P_DEFAULT);
space_id = H5Screate_simple(...);
dset_id = H5Dcreate(file_id, DNAME, H5T_NATIVE_INT, space_id, ...);

status = H5Dwrite(dset_id, H5T_NATIVE_INT, ..., H5P_DEFAULT, ...);
```

...
Example of a PHDF5 C Program

A parallel HDF5 program has a few extra calls

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();
PHDF5 Implementation Layers

- Science Application
- Compute node
- Comput node
- Comput node
- HDF5 Library
- MPI Library
- HDF5 file on Parallel File System
- Interconnect network + I/O servers
- Disk architecture and layout of data on disk
PHDF5 Etiquette

• PHDF5 opens a shared file with an MPI communicator
  – Returns a file handle
  – All future access to the file via that file handle
• All processes must participate in collective PHDF5 APIs
• Different files can be opened via different communicators
• **All** HDF5 APIs that modify file structure are collective!
  – Object create / delete, attribute and group changes, etc.
Collective vs. Independent I/O

• Collective I/O attempts to combine multiple smaller independent I/O ops into fewer larger ops.
  – Neither mode is preferable \textit{a priori}

• MPI definition of collective calls:
  – All processes of the communicator must participate in calls in the same order:

    | Process 1          | Process 2                  |
    |--------------------|---------------------------|
    | call A(); \rightarrow call B(); | call A(); \rightarrow call B(); **right** |
    | call A(); \rightarrow call B(); | call B(); \rightarrow call A(); **wrong** |

  – Independent calls are not collective 😊
  – Collective calls are not necessarily synchronous, nor must they require communication
    • It could be that only internal state for the communicator changes
Parallel HDF5 tutorial examples

• For examples how to write different data patterns see:

http://support.hdfgroup.org/HDF5/Tutor/parallel.html
Tools

DIAGNOSTICS AND INSTRUMENTATION
## Data and Metadata I/O

### Data
- Problem-sized
- I/O can be independent or collective
- Improvement targets:
  - Avoid unnecessary I/O
  - I/O frequency
  - Layout on disk
    - Different I/O strategies for chunked layout
  - Aggregation and balancing
  - Alignment

### Metadata
- Small
- Reads can be independent or collective
- All modifying I/O must be collective
- Improvement targets:
  - Metadata design
  - Use the latest library version, if possible
  - Metadata cache
    - In desperate cases, take control of evictions
Don’t Forget: It’s a Multi-layer Problem

Application

HDF5
(Disable truncate in H5Fclose)

MPI-IO
(Number of collective buffer nodes, Collective buffer size)

Lustre Parallel File System
(Stripe factor and Stripe size)

Storage Hardware
A Textbook Example

User reported:

• Independent data transfer mode is much slower than the collective data transfer mode

• Data array is tall and thin: 230,000 rows by 4 columns

230,000 rows
Symptoms

Writing to one dataset
- 4 MPI processes → 4 columns
- Datatype is 8-byte floats (doubles)
- 4 processes x 1000 rows x 8 bytes = 32,000 bytes

% mpirun -np 4 ./a.out 1000
   ➢ Execution time: 1.783798 s.

% mpirun -np 4 ./a.out 2000
   ➢ Execution time: 3.838858 s. (linear scaling) 😞)

• 2 sec. extra for 1000 more rows = 32,000 bytes.
  16KB/sec → Way too slow!!!
“Poor Man’s Debugging”

• Build a version of PHDF5 with
  – ./configure --enable-debug --enable-parallel ...

• This allows the tracing of MPIO I/O calls in the HDF5 library such as
  MPI_File_read_xx and MPI_File_write_xx

• Don’t forget to:
  – % setenv H5FD_mpio_Debug “rw”

• You’ll get something like this…
Independent and Contiguous

% setenv H5FD_mpio_Debug “rw”

% mpirun -np 4 ./a.out 1000 # Indep.; contiguous.
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=0     size_i=96
in H5FD_mpio_write  mpi_off=2056  size_i=8
in H5FD_mpio_write  mpi_off=2048  size_i=8
in H5FD_mpio_write  mpi_off=2072  size_i=8
in H5FD_mpio_write  mpi_off=2064  size_i=8
in H5FD_mpio_write  mpi_off=2088  size_i=8
in H5FD_mpio_write  mpi_off=2080  size_i=8

... 
- A total of 4000 of these 8 bytes writes == 32,000 bytes.
Plenty of Independent and Small Calls

Diagnosis:

- Each process writes one element of one row, skips to next row, writes one element, and so on.

- Each process issues 230,000 writes of 8 bytes each.
Chunked by Column

% setenv H5FD_mpio_Debug "rw"
% mpirun -np 4 ./a.out 1000  # Indep., Chunked by column.
in H5FD_mpio_write  mpi_off=0   size_i=96
in H5FD_mpio_write  mpi_off=0   size_i=96
in H5FD_mpio_write  mpi_off=0   size_i=96
in H5FD_mpio_write  mpi_off=0   size_i=96
in H5FD_mpio_write  mpi_off=3688 size_i=8000
in H5FD_mpio_write  mpi_off=11688 size_i=8000
in H5FD_mpio_write  mpi_off=27688 size_i=8000
in H5FD_mpio_write  mpi_off=19688 size_i=8000
in H5FD_mpio_write  mpi_off=96    size_i=40
in H5FD_mpio_write  mpi_off=136   size_i=544
in H5FD_mpio_write  mpi_off=680   size_i=120
in H5FD_mpio_write  mpi_off=800   size_i=272

• Execution time: 0.011599 s.
Use Collective Mode or Chunked Storage

Remedy:

• Collective I/O will combine many small independent calls into few but bigger calls

• Chunks of columns speeds up too
Collective vs. independent write

- **Independent write**
- **Collective write**

Data size in MBs | Seconds to write
--- | ---
0.25 | 0
0.5 | 0
1.0 | 100
1.88 | 200
2.29 | 300
2.75 | 400
Other Helpful Tools...

- Two kinds of tools:
  - I/O benchmarks for measuring a system’s I/O capabilities
  - I/O profilers for characterizing applications’ I/O behavior

- Two examples:
  - h5perf (in the HDF5 source code distro)
  - Darshan (from Argonne National Laboratory)

- Profilers have to compromise between
  - A lot of detail $\Rightarrow$ large trace files and overhead
  - Aggregation $\Rightarrow$ loss of detail, but low overhead
I/O Patterns

Contiguous

Memory

File

Contiguous in memory, not in file

Memory

File

Contiguous in file, not in memory

Mem

File

Dis-contiguous

Mem

File
h5perf(_serial)

• Measures performance of a filesystem for different I/O patterns and APIs

• Three File I/O APIs for the price of one!
  – POSIX I/O (open/write/read/close…)
  – MPI-I/O (MPI_File_{open,write,read,close})
  – HDF5 (H5Fopen/H5Dwrite/H5Dread/H5Fclose)

• An indication of I/O speed ranges and HDF5 overheads

• Expectation management…
A Serial Run

h5perf_serial, 3 iterations, 1 GB dataset, 1 MB transfer buffer, HDF5 dataset contiguous storage, HDF5 SVN trunk, NCSA BW
A Parallel Run

h5perf, 3 MPI processes, 3 iterations, 3 GB dataset (total),
1 GB per process, 1 GB transfer buffer,
HDF5 dataset contiguous storage, HDF5 SVN trunk, NCSA BW
Darshan (ANL)

• Design goals:
  – Transparent integration with user environment
  – Negligible impact on application performance

• Provides aggregate figures for:
  – Operation counts (POSIX, MPI-IO, HDF5, PnetCDF)
  – Datatypes and hint usage
  – Access patterns: alignments, sequentiality, access size
  – Cumulative I/O time, intervals of I/O activity

• Does not provide I/O behavior over time

• Excellent starting point, maybe not your final stop
Darshan Sample Output

Source: NERSC
PARALLEL HDF5 TUNING
Metadata Read Storm Problem (I)

• All metadata “write” operations are required to be collective:

```c
if(0 == rank)
    H5Dcreate("dataset1");
else if(1 == rank)
    H5Dcreate("dataset2");

/* All ranks have to call */
H5Dcreate("dataset1");
H5Dcreate("dataset2");
```

• Metadata read operations are not required to be collective:

```c
if(0 == rank)
    H5Dopen("dataset1");
else if(1 == rank)
    H5Dopen("dataset2");

/* All ranks have to call */
H5Dopen("dataset1");
H5Dopen("dataset2");
```
Metadata Read Storm Problem (II)

• Metadata read operations are treated by the library as independent read operations.

• Consider a very large MPI job size where all processes want to open a dataset that already exists in the file.

• All processes
  – Call H5Dopen("/G1/G2/D1");
  – Read the same metadata to get to the dataset and the metadata of the dataset itself
    • IF metadata not in cache, THEN read it from disk.
  – Might issue read requests to the file system for the same small metadata.

• ➔ READ STORM
Avoiding a Read Storm

• Application sets hint that metadata access is done collectively
  – A property on an access property list: H5Pset_all_coll_metadata_ops
  – If set on the file access property list, then all metadata read operations will be required to be collective

• Can be set on individual object property list or on a file level

• If set, MPI rank 0 will issue the read for a metadata entry to the file system and broadcast to all other ranks
Successful Collective Dataset I/O

• Request Collective Dataset I/O:

... 

xf_id = H5Pcreate(H5P_DATASET_XFER); 
H5Pset_dxpl_mpio(xf_id, H5FD_MPIO_COLLECTIVE); 
H5Dwrite(dset_id, H5T_NATIVE_INT, ..., xf_id, ...);

• However, collective I/O can be changed to independent I/O within HDF5:
  – Datatype conversion, data transform, layout isn’t contiguous or chunked
Successful Collective Dataset I/O

• Check for Collective I/O and why I/O was performed Independently:

```c
xf_id = H5Pcreate(H5P_DATASET_XFER);
H5Pset_dxpl_mpio(xf_id, H5FD_MPIO_COLLECTIVE);
H5Dwrite(dset_id, H5T_NATIVE_INT, ..., xf_id, ...);
H5Pget_mpio_actual_io_mode(xf_id, &io_mode);
if(io_mode == H5D_MPIO_NO_COLLECTIVE)
    H5Pget_mpio_no_collective_cause(xf_id, &local_cause, &global_cause);
```
ECP EXAHDF5
ECP ExaHDF5 project mission

• Work with ECP applications and facilities to meet their needs
• Productize HDF5 features
• Support, maintain, package, and release HDF5
• Research toward future architectures and incoming requests from ECP teams
# ECP Engagement – AD and Co-design teams

<table>
<thead>
<tr>
<th>ECP AD team</th>
<th>Type of engagement</th>
<th>Status</th>
<th>ExaHDF5 POC(s) / ECP team POC(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subsurface simulation – Chombo I/O</td>
<td>I/O performance tuning</td>
<td>Improved performance</td>
<td>Suren Byna, Quincey / Brian van Straalen</td>
</tr>
<tr>
<td>QMCPACK</td>
<td>File close performance issue</td>
<td>Improved performance</td>
<td>Rick Zamora / Ye Luo</td>
</tr>
<tr>
<td>EQSim – SW4</td>
<td>HDF5 I/O implementation</td>
<td>Benchmark developed</td>
<td>Suren Byna / Hans Johansen</td>
</tr>
<tr>
<td>WarpX</td>
<td>Performance issue</td>
<td>Improved performance</td>
<td>Alex Sim / Jean-Luc Vay</td>
</tr>
<tr>
<td>ExaFEL</td>
<td>SWMR enhancements</td>
<td>Prototype in testing</td>
<td>Quincey Koziol / Amedeo Perazzo</td>
</tr>
<tr>
<td>ExaSky – HACC</td>
<td>I/O performance tuning</td>
<td>Tuning performance</td>
<td>Scot Breitenfeld / Hal Finkel, Salman Habib</td>
</tr>
<tr>
<td>ExaSky – Nyx</td>
<td>I/O performance tuning</td>
<td>Tuned performance for the AMReX I/O benchmark</td>
<td>Suren Byna / Ann Almgren, Zarija Lukic</td>
</tr>
<tr>
<td>AMReX co-design center</td>
<td>AMReX I/O performance tuning</td>
<td>Tuned performance for the AMReX I/O benchmark</td>
<td>Suren Byna / Ann Almgren, Andrew Myers</td>
</tr>
<tr>
<td>E3SM-MMF</td>
<td>Testing Data Elevator</td>
<td>The AD team tested the feature</td>
<td>Suren Byna / Jayesh Krishna</td>
</tr>
<tr>
<td>Lattice QCD, NWChemEx, CANDLE</td>
<td>I/O using HDF5</td>
<td>Initial communications w/ the AD teams</td>
<td>Suren Byna, Venkat Vishwanath / Chulwoo (LQCD), Ray Bair (NWChem), Venkat (CANDLE)</td>
</tr>
<tr>
<td>ExaLearn</td>
<td>I/O for ML applications</td>
<td>Initial communications</td>
<td>Quincey Koziol / Quincey Koziol</td>
</tr>
</tbody>
</table>

More details: [https://confluence.exascaleproject.org/display/STDM10](https://confluence.exascaleproject.org/display/STDM10)
# ECP & ASCR Engagement – ST teams

<table>
<thead>
<tr>
<th>ECP ST team</th>
<th>Type of engagement</th>
<th>Status</th>
<th>ExaHDF5 POC(s) / ECP team POC(s)</th>
</tr>
</thead>
</table>
| ADIOS       | Interoperability of HDF5 and other file formats | • Developing VOL to read ADIOS data  
• ADIOS R/W of HDF5 data | Suren Byna, Quincey / Junmin Gu, John Wu |
| DataLib     | Interoperability of HDF5 and other file formats | • VOL to read netCDF data – To Do  
• HDF5 relies on MPI-IO | Venkat Vishwanath / Rob Ross, Rob Latham |
| UnifyCR     | Data Elevator can use a unified node-local storage namespace | Discussion with the UnifyCR teams on the API, data movement strategy, etc. | Suren Byna / Kathryn Mohror |
| EZ          | Compression in HDF5 | EZ team developing parallel filter with HDF5 | Scot Breitenfeld / Franck Cappello, Sheng Di |
| ZFP         | Compression in HDF5 | Initial communications | Suren Byna / Peter Lindstorm |
| ALPINE      | VTK / HDF5 mapping | Initial communications | Suren Byna, Scot Breitenfeld / Jim Ahrens |
Productizing HDF5 features

• Virtual Object Layer (VOL)
• Indexing and querying raw data
Virtual Object Layer (VOL)

• Goal: Provide an application with the HDF5 data model and API, but allow different underlying storage mechanisms

• Enables developers to easily use HDF5 on novel current and future storage systems
  – Prototype plugins for using burst buffer storage transparently and for accessing DAOS are available
  – VOL plugins for reading netCDF and ADIOS data are in development

• Integrated into the HDF5 trunk
Indexing and querying datasets

- HDF5 *index* objects and API routines allow the creation of indexes on the contents of HDF5 containers, to improve query performance

- HDF5 *query* objects and API routines enable the construction of query requests for execution on HDF5 containers
  - H5Qcreate
  - H5Qcombine
  - H5Qapply
  - H5Qclose

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

  - HDF5 Bitbucket repo containing the “topic-parallel-indexing” source code: [https://bitbucket.hdfgroup.org/projects/HDFFV/repos/hdf5](https://bitbucket.hdfgroup.org/projects/HDFFV/repos/hdf5)
Support and maintenance

• HDF5 home page: http://hdfgroup.org/HDF5/
  – Latest release: HDF5 1.10.5 (1.10.6, 1.12.0 Spring 2019)

• Documentation is available https://portal.hdfgroup.org/display/HDF5/HDF5

• Support: HDF Helpdesk help@hdfgroup.org

• HDF-FORUM https://forum.hdfgroup.org/

• For ECP teams: Contact the ExaHDF5 POCs for existing collaborations and the PIs for new collaborations
New features for exascale architectures

• Data Elevator to take advantage of burst buffers
• Topology-aware I/O
• Full Single writer multiple readers (SWMR) functionality
• Asynchronous I/O
• Querying metadata
• Interoperability with other file formats
Data Elevator for write and read caching using burst buffers

- Data Elevator write caching prototype
  - Transparent data movement in storage hierarchy
  - In situ data analysis capability using burst buffers
- Tested with a PIC code and Chombo-IO benchmark
- Applications evaluating Data Elevator
  - E3SM-MMF and Sandia ATDM project is evaluating performance
  - Other candidates: EQSim, AMReX
- Installed on NERSC’s Cori system (module load data-elevator)
Topology-aware I/O

- Taking advantage of the topology of compute and I/O nodes and network among them improves overall I/O performance
- Developing topology-aware data-movement algorithms and collective I/O optimizations within a new HDF5 virtual file driver (VFD)

Performance comparison of the new HDF5 VFD, using one-sided aggregation, with the default binding to Cray MPICH MPI-IO. Data was collected on Theta using an I/O benchmarking tool (the HDF5 Exerciser).

Full functionality of Single Writer, Multiple Readers (SWMR)

- SWMR enables a single writing process to update an HDF5 file, while multiple reading processes access the file in a concurrent, lock-free manner.
- Previously limited to the narrow use-case of appending new elements to HDF5 datasets.
- Full SWMR extends existing SWMR to support all metadata operations, such as object creation and deletion, attribute updates.

In ECP, ExaFEL project requires this feature. In general, Full SWMR is useful for managing experimental and observational data.

Features in development

• Asynchronous I/O
  – Store data in an intermediate faster memory or storage location and move data asynchronously to storage
  – Prototype of the async I/O VOL with Argobots in progress

• Interoperability with other file formats
  – Capability to read netCDF and ADIOS files
  – Developed a VOL to read ADIOS files and netCDF read VOL dev in progress

• Indexing and querying metadata
  – HDF5 file metadata querying – design is in progress

• Breaking collective dependency in updating metadata
  – Metadata updates are collective operations, which may have high overhead
  – Developing independent updates, inspired by blockchain technology
Roadmap of current project

**Q1 2017**
- LBL: Develop and test Data Elevator (DE) write caching
- ANL: Refine
- THG: Prepare Release

**Q2 2017**
- LBL: Data Elevator (DE) read caching
- ANL: Develop
- THG: Prepare Release

**Q3 2017**
- LBL: Evaluate and design topology-aware API
- ANL: Refine
- THG: Prepare Release

**Q4 2017**
- LBL: Develop topology-aware API
- ANL: Refine
- THG: Develop

**Q1 2018**
- LBL: Integrate VOL branch into HDF5
- ANL: Develop
- THG: Prepare Release

**Q2 2018**
- LBL: Design
- ANL: Develop
- THG: Prepare Release

**Q3 2018**
- LBL: Development
- ANL: Develop
- THG: Prepare Release

**Q4 2018**
- LBL: Prepare Release
- ANL: Design
- THG: Develop

**Q1 2019**
- LBL: Design
- ANL: Develop
- THG: Prepare Release

**Q2 2019**
- LBL: Develop and test
- ANL: Prepare Release
- THG: Release

**Q3 2019**
- LBL: Track application, ST, HT requirements and changes
- ANL: Support, performance improvements
- THG: Support, performance improvements

**Q4 2019**
- LBL: Release
- ANL: Release
- THG: Release
FY20 – 22 Plans

- HDF5 I/O to handle extreme scalability
  - Performance enhancement features to use memory and storage hierarchy of exascale systems
- Productization of features that have been prototyped
  - Various features have been developed recently to improve data access performance and data reduction, and to open the HDF5 API
- Support for ECP apps
  - I/O performance tuning for ECP apps
- Support for DOE exascale facilities
  - Deployment and performance tuning
- Software Maintenance
  - Integration of features, bug fixes, releases of ECP features
Experimental & Observational Data (EOD) Management Requirements

- Experimental and observational science (EOS) facilities have data management requirements beyond existing HDF5 features
- Targeted science drivers
  - LCLS / LCLS-II, NCEM, ALS, NIF, LSST
- Requirements
  - Multiple producers and multiple consumers of data
  - Remote streaming synchronization
  - Handling changes in data, data types, and data schema
  - Search metadata and provenance directly in HDF5 files
  - Support for different forms of data - Streaming, sparse, KV, etc.
  - Optimal data placement
EOD-HDF5 - Features

- Multi-modal access and distributed data in workflows
  - Multiple-Writers / Multiple-Readers (“MWMR” or “Multi-SWMR”)
  - Distribution of local changes to remote locations
    - “rsync for HDF5 data”
- Data model extensions
  - Storing new forms of data (Key-Value pairs, Sparse data, Column-oriented data)
  - Addressing science data schema variation
  - Managing collections of containers
- Metadata and Provenance management
  - Capturing and storing rich metadata contents and provenance
  - Searching metadata and provenance
  - Optimal data placement based on data analysis patterns
Thank you!