ExaIO HDF5 features and application use cases

Suren Byna
Lawrence Berkeley National Laboratory

ATPESC 2021- Aug 6th, 2021
ExaIO - Many Team Members and Contributors

- **LBNL**: Quincey Koziol, Houjun Tang, Tony Li, Bin Dong, Alex Sim, Junmin Gu
- **ANL**: Venkat Vishwanath, Huihuo Zheng, Rick Zamora, Paul Coffman
- **The HDF Group**: Scot Breitenfeld, Elena Pourmal, John Mainzer, Richard Warren, Dana Robinson, Neil Fortner, Jerome Soumagne, Jordan Henderson, Neelam Bagha
- **North Carolina State University**: John Ravi, Michela Becchi
Overview

• Features
  • HDF5 Virtual Object Layer (VOL) Introduction
  • ECP VOL Connectors
    • Asynchronous I/O
    • Node-local Caching
  • Subfiling and querying

• ECP HDF5 Applications and benchmarks
  • EQSIM
  • AMReX - Nyx and Castro
  • Chombo-IO
  • h5bench
HDF5 Virtual Object Layer (VOL)

- **VOL Framework** 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”

- **VOL Connectors**
  - 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
VOL: High-Level Overview

Application

HDF5 API

Operations on a container

All other HDF5 routines

Virtual Object Layer (VOL) Connectors

Pass-through

Asynchronous

Caching

Data Elevator

Provenance

Terminal

Native

DAOS

REST

Arrow

PDC

HDF5 Library Infrastructure
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 ([https://github.com/hpc-io/vol-provenance](https://github.com/hpc-io/vol-provenance))
      • Asynchronous I/O ([https://github.com/hpc-io/vol-async](https://github.com/hpc-io/vol-async))
      • Caching ([https://github.com/hpc-io/vol-cache](https://github.com/hpc-io/vol-cache))
  • **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 ([https://github.com/HDFGroup/vol-daos](https://github.com/HDFGroup/vol-daos)), S3, Apache Arrow, etc.)
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:

Main developer: Houjun Tang
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

AMReX Single-level Plotfile 385GB x 5 timestep on Summit

AMReX Multi-level Plotfile 559GB x 5 timesteps on Summit
Async VOL Connector – Programming Example

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

status = H5Dwrite(did, ..);

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

https://github.com/hpc-io/vol-async
Async VOL Connector – Programming Example

```c
es_id = H5EScreate();
fid = H5Fopen_async(.., es_id);
gid = H5Gopen_async(fid, .., es_id);
did = H5Dopen_async(gid, .., es_id);
status = H5Dwrite_async(did, .., es_id);

status = H5Dwrite_async(did, .., es_id);
...
<other user code>
...
H5ESwait(es_id);
```

// Create event set for tracking async operations
// Asynchronous, can start immediately
// Asynchronous, starts when H5Fopen completes
// Asynchronous, starts when H5Gopen completes
// Asynchronous, starts when H5Dopen completes,
//   may run concurrently with other H5Dwrite in event set
// Asynchronous, starts when H5Dopen completes,
//   may run concurrently with other H5Dwrite in event set
// Wait for operations in event set to complete, buffers
//   used for H5Dwrite must only be changed after wait

https://github.com/hpc-io/vol-async
Async VOL Connector – Programming Example

```c
es_id = H5EScreate();
fid = H5Fopen_async(..., es_id);
gid = H5Gopen_async(fid, ..., es_id);
did = H5Dopen_async(gid, ..., es_id);
status = H5Dwrite_async(did, ..., es_id);

status = H5Dwrite_async(did, ..., es_id);

...<other user code>
...
H5ESwait(es_id);
```

// Create event set for tracking async operations
// Asynchronous, can start immediately
// Asynchronous, starts when H5Fopen completes
// Asynchronous, starts when H5Gopen completes
// Asynchronous, starts when H5Dopen completes,
// may run concurrently with other H5Dwrite in event set
// Asynchronous, starts when H5Dopen completes,
// may run concurrently with other H5Dwrite in event set

// Wait for operations in event set to complete, buffers
// used for H5Dwrite must only be changed after wait

https://github.com/hpc-io/vol-async
Async VOL Connector

- Available now:
  - Source: [https://github.com/hpc-io/vol-async](https://github.com/hpc-io/vol-async)

- Future work:
  - 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
  - Switch to TaskWorks thread engine
    - A portable, high-level, task engine designed for HPC workloads
    - Task dependency management, background thread execution.
Cache VOL Connector - Integrating node-local storage into parallel I/O

Main developer: Huihuo Zheng

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 VOL connectors (async -> cache -> async)
- All complexity is hidden from the users

Repo: [https://github.com/hpc-io/vol-cache.git](https://github.com/hpc-io/vol-cache.git)

Typical HPC storage hierarchy

Node-local storage (SSD, NVMe, etc)

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

<table>
<thead>
<tr>
<th>w/o caching</th>
<th>Compute</th>
<th>I/O (RAM→PFS)</th>
<th>Compute</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/ caching</td>
<td>Compute</td>
<td>RAM→NLS</td>
<td>Compute</td>
</tr>
</tbody>
</table>

Partial overlap of compute with I/O

I/O: NLS→PFS

Details are hidden from the application developers.
Parallel Read (H5Dread)

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

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).

Node-local storage

Compute node RAM

Parallel file system

Single shared HDF5 file

MPI_Win

1. Reading data from parallel file system

2. Caching data using MPI_Put

Reading data from NLS using MPI_Put

First time reading the data

Reading the data directly from node-local storage

w/o Caching
Compute I/O Compute

w/ Caching
Compute I/O Compute
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
Subfiling

- Subfiling: a compromise between file-per-process (fpp) and a single shared file (ssf)
  - Multiple files are 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 - Initial results

(h5bench – write benchmark)

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

Main developer: THG devs

HDF5 github repo containing the querying and indexing source code: https://github.com/HDFGroup/hdf5/tree/feature/indexing
Parallel scaling of index generation and query resolution is evidenced even for small-scale experiments.
ECP HDF5 Applications
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.
• Seismologists sets up an earthquake event for simulation.  
  \textit{Various input data}

• SW4 generates and outputs ground motions for specified locations.  
  \textit{1D, 2D, 3D, 4D output data}

• Analysis codes (OpenSees, ESSI) produces building response.  
  \textit{Visualization and analysis data}
SW4 I/O with HDF5 integration

• Input
  • Material model and topography: \textit{sfile}: \(\frac{1}{2}\) size, 3x faster, new curvilinear grid.
  • Forcing function: \textbf{SRF-HDF5}: 1/3 size, 5x faster.
  • Station location: \textit{inputHDF5}: single file.

• Output
  • Time-series
    • Station output: \textbf{SAC-HDF5}: 1/5 USGS, same as SAC, \textit{single} file
    • Subsurface output: \textbf{SSI}, with ZFP compression (155GB / 38TB), 3x faster
  • Image: \textit{imgHDF5}, same as native, easy to access
  • Checkpoint: \textit{chkHDF5} with ZFP compression - 4x to 6x less data (optimization 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.
Results on Summit

Single-level (Nyx) Workload

Multiple-level (Castro) Workload
h5bench - A suite of HDF5 benchmarks

- Captures various I/O patterns
  - Locality in memory and in files
    - Contiguous, strided, compound data types
  - Array dimensionality - 1D, 2D, and 3D
- I/O modes
  - Synchronous
  - Asynchronous - Implicit and explicit
- Processor type - CPUs and GPUs
- MPI-IO modes
  - Collective buffering on or off
- File system configuration
  - Alignment and striping

https://github.com/hpc-io/h5bench

<table>
<thead>
<tr>
<th>In memory representation</th>
<th>In HDF5 file representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contiguous in memory and contiguous in file</td>
<td>array A</td>
</tr>
<tr>
<td></td>
<td>array B</td>
</tr>
<tr>
<td>Contiguous in memory and compound in file</td>
<td>array A</td>
</tr>
<tr>
<td></td>
<td>array B</td>
</tr>
<tr>
<td>Compound structure in memory and contiguous in file</td>
<td>array AB</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Compound structure in memory and compound in file</td>
<td>array AB</td>
</tr>
</tbody>
</table>
Conclusions

• Testing of stacking of asynchronous I/O and cache VOL is in progress
• Try h5bench for verifying HDF5 performance at scale
  • Feedback and adding more I/O patterns are welcome
• Subfiling development is in progress
• Contact us if you have any querying use cases
• Contact us with any HDF5 performance or functionality problems
  • The HDF Group: Helpdesk at help@hdfgroup.org
  • HDF5 resources: https://www.hdfgroup.org/
  • ECP ExaIO: SByna@lbl.gov
Useful links and info

• HDF5 tutorials
  • https://github.com/HDFGroup/Tutorial
  • Parallel HDF5 hands-on tutorial examples
    • https://github.com/HDFGroup/Tutorial/tree/main/Parallel-hands-on-tutorial

• HUG 2021 (HDF5 User Group) meeting
  • https://www.hdfgroup.org/hug/hug21
  • Contact: hug@hdfgroup.org

• HPC Data Management Systems Postdoctoral Scholar position available at LBNL
  • https://tinyurl.com/2021-sdm-postdoc