Upcoming New HDF5 Features

Progress on Multi-thread, Sparse Data Storage, and Encryption in HDF5

John Mainzer john.mainzer@lifeboat.llc

Elena Pourmal elena.pourmal@lifeboat.llc

Lifeboat

Outline

- Intro to Lifeboat, LLC
- Multi-threaded access to data in HDF5
  - Approach
  - Progress
  - Bypass VOL
- Sparse data in HDF5
- Integrity of data in HDF5 (HDF5 encryption)

Lifeboat, LLC

U.S. DEPARTMENT OF ENERGY AWARDEE™

We don't make HDF5… we make HDF5 *better*

- Goal: Sustain and enhance open source HDF5
  - Founded in August 2021; located in Champaign, IL and Laramie, WY
  - www.lifeboat.llc
  - info@lifeboat.llc
- Funded by DOE SBIR/STTR Program
  - Phase II: "*Toward multi-threaded concurrency in HDF5*" (started in April 2023)
  - Phase II: "*Supporting sparse data in HDF5*" (started in April 2024)
  - Phase I: *"Protecting the confidentiality and integrity of data stored in HDF5" (aka"HDF5 Encryption")* (started in February 2024)

Lifeboat, LLC (cont'd)

U.S. DEPARTMENT OF ENERGY AWARDEE™

We don't make HDF5… we make HDF5 *better*

- H5+ product
  - In near term - Collection of pluggable connectors and tools to enhance functionality of open-source HDF5 and tools
    - Encryption, multi-threaded access to data, full implementation of single writer/multiple reader access mode, data recovery tool
  - In long term - Better engineered multi-threaded HDF5 library with full set of features along with a rich collection of connectors to enhance functionality and to access data on all kinds of storage systems (e.g., traditional FS, Cloud, Object Store)

Multi-threaded access to data in HDF5

Approach

Enabling multi-threaded VOL connectors with open-source HDF5

HDF5 at Present

- Fundamentally a single thread library.
- Thread safety supported via a global mutex – only one thread active in the library at a time.
- This constraint is imposed on VOL connectors, even if they can support multi-thread operation.

Multi-Thread HDF5

- True multi-thread support has been requested for a long time.
  - Retrofitting multi-thread support to an existing large, and largely un-documented code base is a daunting task.
  - Thus little tangible progress beyond the global mutex – allowing thread safety but not multi-thread execution.

- The VOL layer changes the picture.
  - Pushing the global mutex down a bit, would allow multiple threads of execution into VOL connectors that support it.
  - Only need to retrofit multi-thread support onto a small number of HDF5 packages to do this – H5E (error reporting), H5I (index), H5P (property lists), H5CX (context), and H5VL (VOL).
    - Multi-thread versions of the H5S (selections) and H5FD (file driver) packages are desirable, but not necessary for the initial prototype.

Towards Multi-Thread VOL Support

Lifeboat is pursuing this strategy -- objectives are to:

- Retrofit multi-thread support on the required HDF5 packages
- Push the HDF5 global mutex down to allow multiple threads into VOL connectors that support it.
- Develop the Bypass VOL to allow limited multi-thread I/O on HDF5 files.

All modifications to the HDF5 library and related documentation to be contributed to the HDF5 open source project.

Multi-threaded access to data in HDF5

Progress

Enabling multi-threaded VOL connectors with open-source HDF5

Current Status

- Working prototypes for multi-thread error reporting (H5E) and index (H5I).
- Design work for property lists (H5P) and the VOL layer (H5VL) at or near completion – implementation to start soon.
- Context (H5CX) initial review and design work complete, further work on hold pending completion of working prototypes of H5P and H5VL due to dependencies on these modules.

Lessons Learned

- Plan on multiple passes to adjust for issues and interactions that are missed, or whose implications are not immediately obvious.

- Going lock free to the extent practical makes this easier.

- Expect some existing internal and external APIs to be problematic with multi-thread. Best to bypass in the prototype where possible, and then negotiate re-designs. Sometimes, just minor semantic changes are sufficient.

- Maintain separation of concerns, and simplify where possible.

- Complex API's make multi-thread testing much harder. To partially mitigate this:
  - Collect extensive statistics.
  - Throw assertion failures as soon as errors are detected.

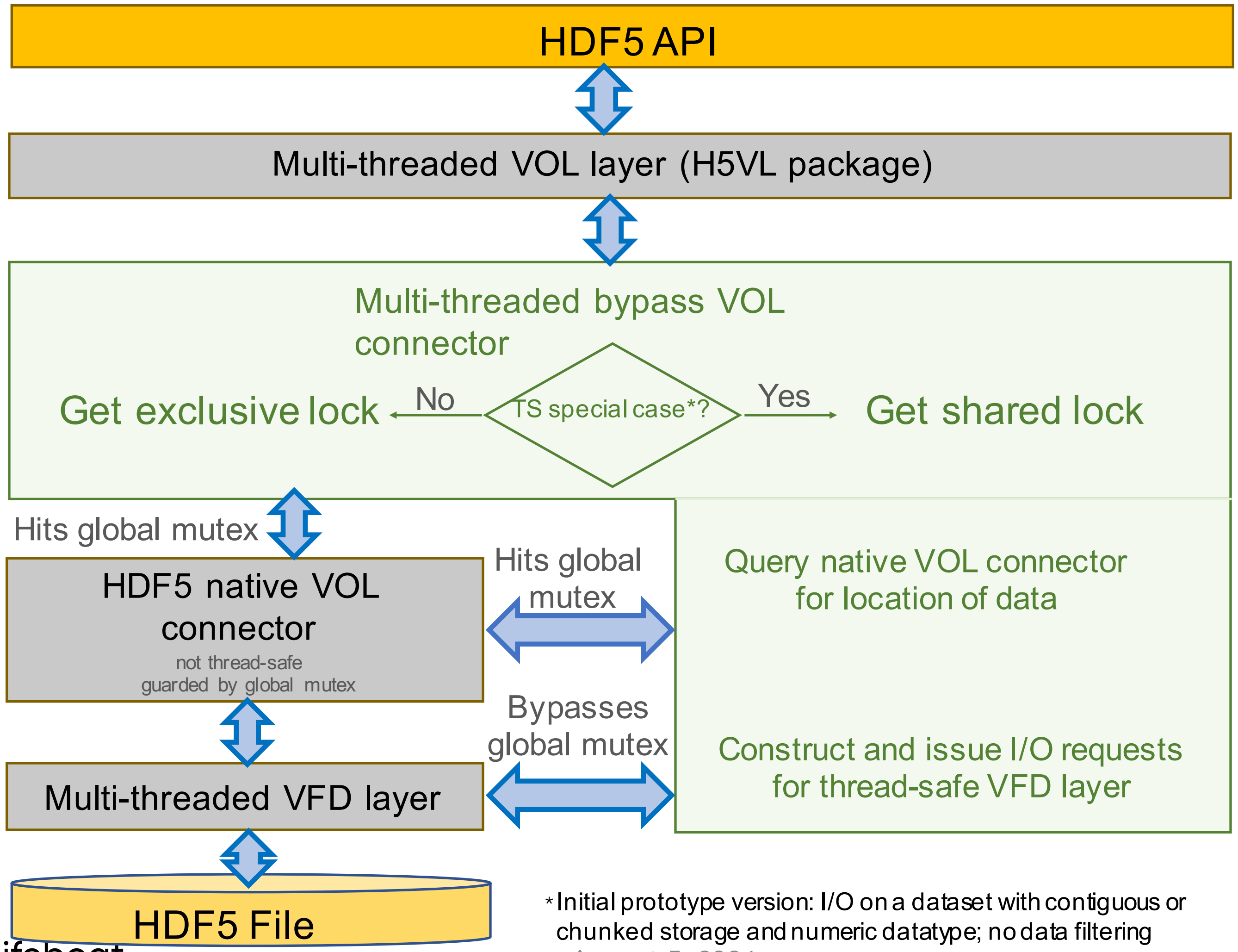
Multi-threaded access to data in HDF5

Bypass VOL

Enabling multi-threaded VOL connectors with open-source HDF5

Bypass VOL Objectives

- Offer significant I/O performance improvements relative to straight HDF5.
- Serve as an initial use and test case for multi-thread VOL support.



HDF5 API

Multi-threaded VOL layer (H5VL package)

Multi-threaded bypass VOL connector

Get exclusive lock

No

TS special case*?

Yes

Get shared lock

Hits global mutex

HDF5 native VOL connector

not thread-safe
guarded by global mutex

Hits global mutex

Query native VOL connector for location of data

Bypasses global mutex

Construct and issue I/O requests for thread-safe VFD layer

Multi-threaded VFD layer

HDF5 File

*Initial prototype version: I/O on a dataset with contiguous or chunked storage and numeric datatype; no data filtering

Bypass VOL Concept

- Query HDF5 library for the location of raw data
- Execute raw data I/O in parallel in multiple threads

Basic concept has been implemented outside the HDF5 library with good results

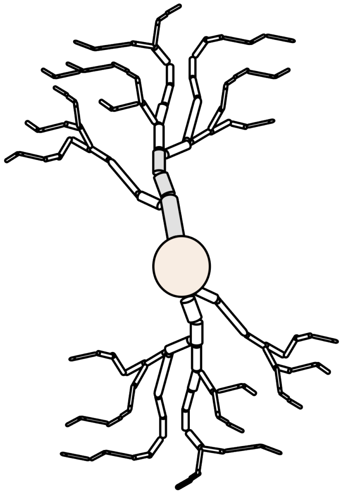
Can't implement fully until support for multi-thread VOLS is available, but a prototype single thread version with thread pool to accelerate large reads has been implemented.

Plan to develop this version as far as practical, and then convert to multi-thread as an initial test case for multi-thread VOL support

Proof of concept: Digitally Reconstructed Neurons

Blue Brain Project https://www.epfl.ch/research/domains/bluebrain/

1k - 100M neurons



```
{
  "0000": {
    "points”: np.empty((9610, 3), np.float32),
    “offsets": np.empty(21, np.uint64)
  },
  "0001": {
    "points": np.empty((14983, 3), np.float32),
    "offsets": np.empty(48, np.uint64)
  },
  ...
}
```

System:

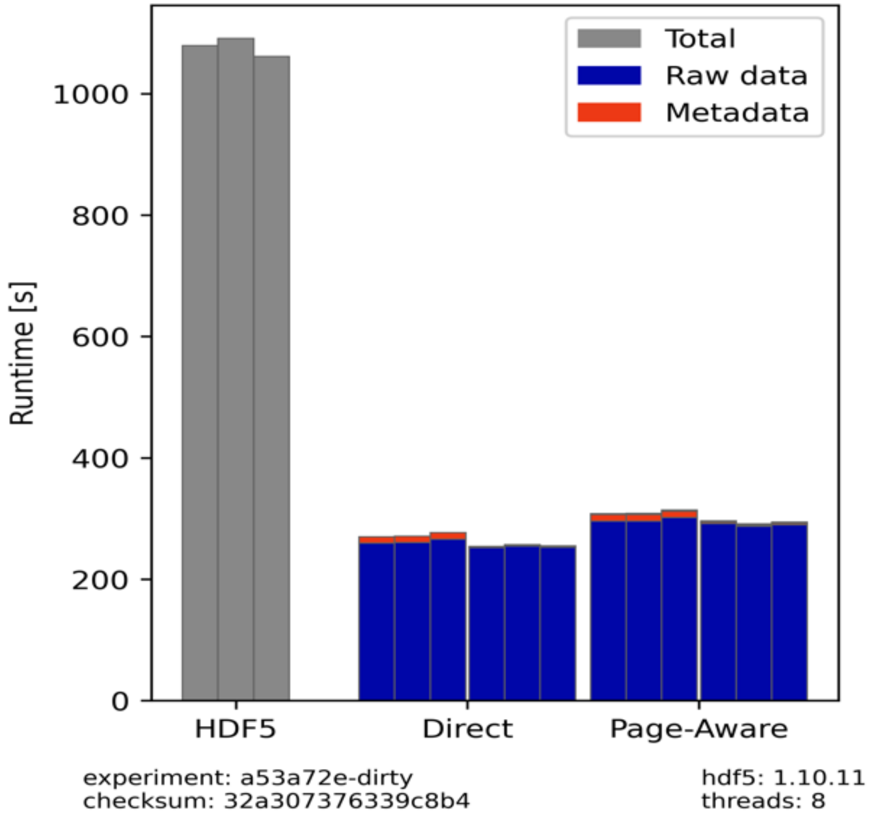
- Cray EX (Perlmutter @NERSC)
  - 512 GB memory per node
  - 2 x AMD EPYC 7763 CPUs per node
  - 64 cores per CPU

Synthetic Data Presented:

Datasets: 500 000
Total size: 640 GB
File Space Strategy: Paged allocation
Page size: 64 kB (not a default value!)

Slide courtesy of Luc Grosheintz, Blue Brain Project, EPFL

Benchmark results

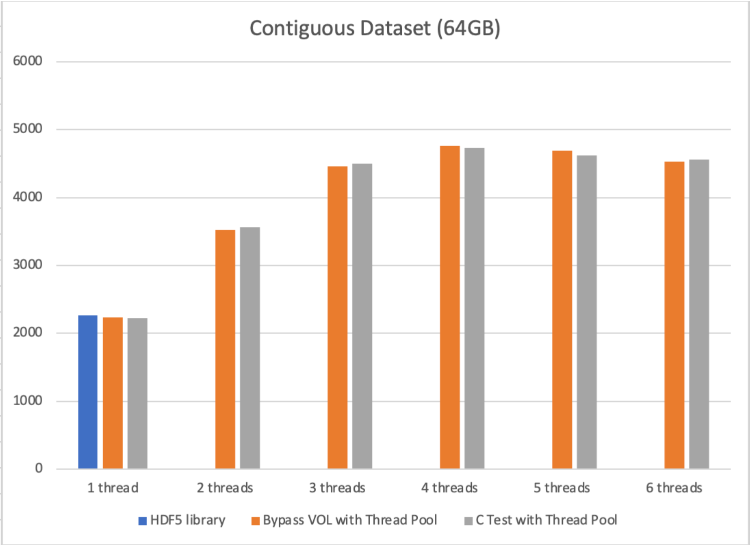


- **Presented experimental setup:**
  - 8 Threads
  - 3 measurements for each run
  - HDF5 1.10.1
- **HDF5**: Plain HDF5 with 512 MB page buffer, 75% reserved for raw data; paged allocation 64KB pages
- **Direct / Page-Aware**: The two variants of the prototype:
  - **Left:** Read metadata for the file using HDF5
  - **Right:** Read metadata from pre-computed JSON file
  - **Page-Aware**: Pages of HDF5 file are brought into memory and threads read raw data from memory
  - We see ~5x speedup when using 8 threads; 2x speedup when using 2 threads

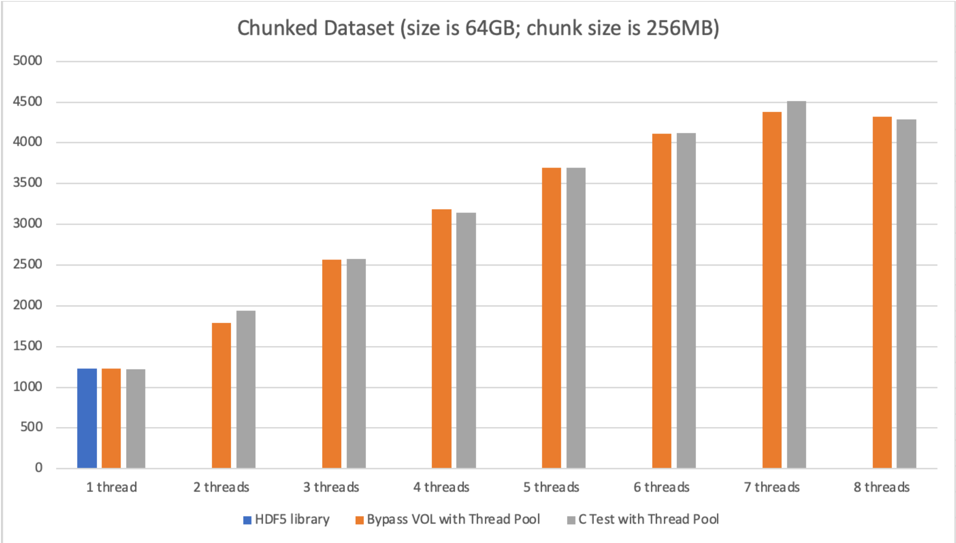
Using bypass VOL connector with single-threaded application (preliminary results)

- Benchmark:
  - Read contiguous or chunked dataset by 4 hyperslabs with
    - Thread-safe HDF5 library
    - Thread-safe HDF5 library with bypass VOL connector **and thread thread pool**; each thread reads 1MB of data
    - C program using Pthreads; each thread reads 1MB of data
    - Compare performance
- Systems
  - Linux box
  - Cray EX node (Perlmutter @NERSC)
  - macOS
- 1.6x to 3x scaling is achieved on Linux and Cray systems
  - Note: no scaling on macOS; need to investigate and document
- *We are looking for applications to test the connector!*

Linux box results (64GB dataset)

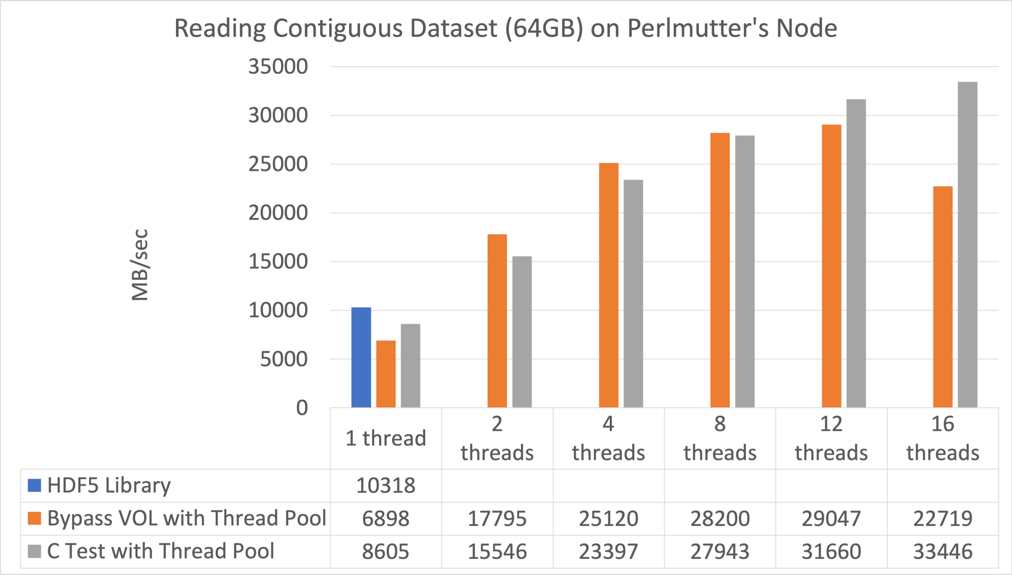


Dataset Size = 64GB on Linux						
	1 thread	2 threads	3 threads	4 threads	5 threads	6 threads
HDF5 library	2264					
Bypass VOL w	2240	3526	4458	4763	4686	4532
C Test with Th	2227	3564	4494	4732	4618	4561



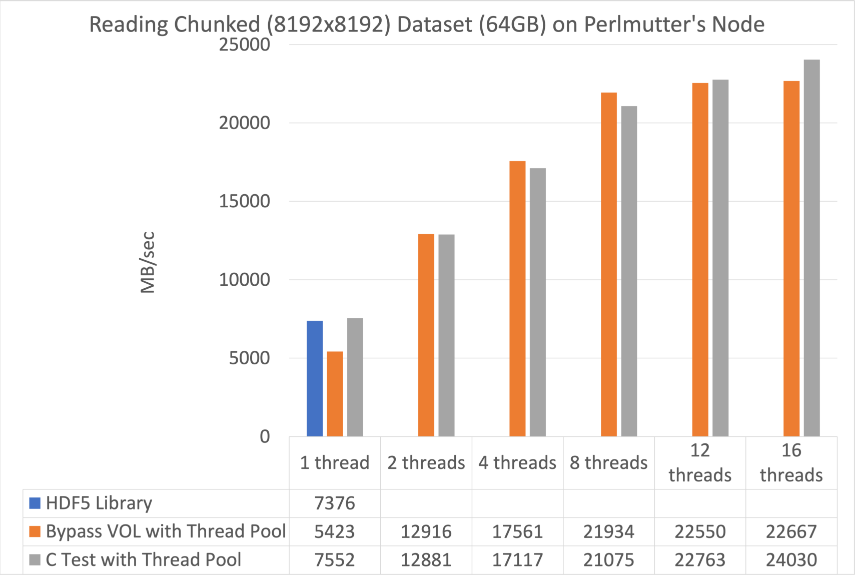
Dataset Size = 64GB on Linux								
	1 thread	2 threads	3 threads	4 threads	5 threads	6 threads	7 threads	8 threads
HDF5 library	1227							
Bypass VOL w	1230	1787	2564	3182	3691	4110	4382	4323
C Test with Th	1223	1942	2575	3140	3695	4117	4510	4288

Perlmutter results (64GB dataset)



Reading Contiguous Dataset (64GB) on Perlmutter's Node

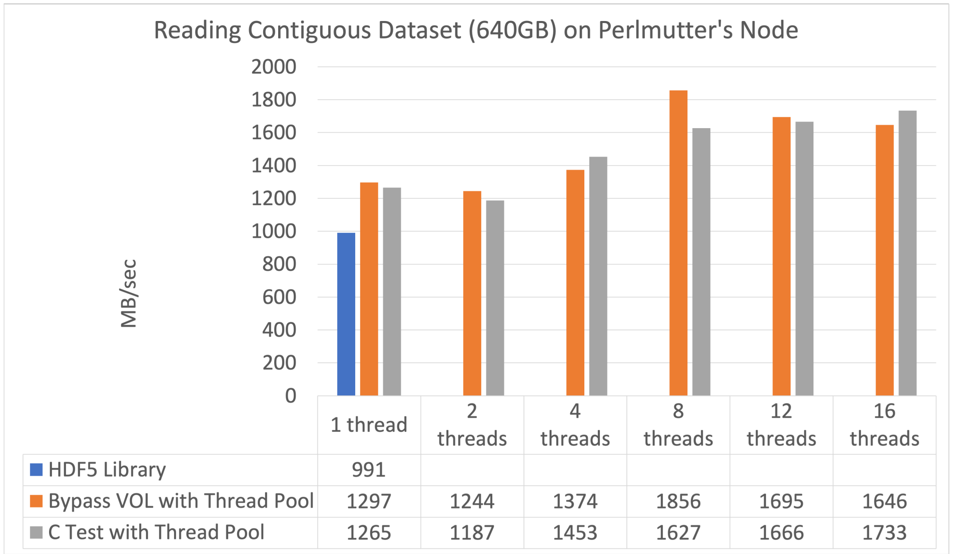
	1 thread	2 threads	4 threads	8 threads	12 threads	16 threads
🟦 HDF5 Library	10318					
🟧 Bypass VOL with Thread Pool	6898	17795	25120	28200	29047	22719
◼ C Test with Thread Pool	8605	15546	23397	27943	31660	33446



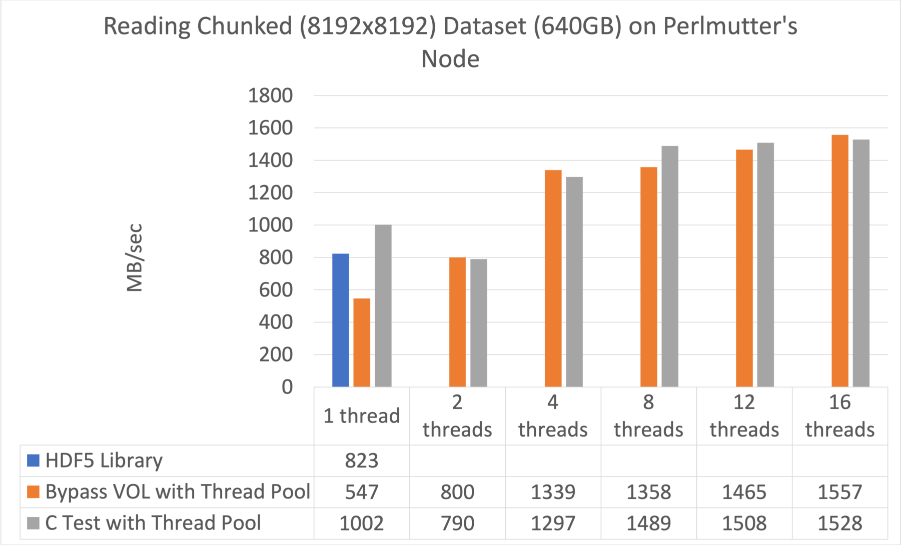
Reading Chunked (8192x8192) Dataset (64GB) on Perlmutter's Node

	1 thread	2 threads	4 threads	8 threads	12 threads	16 threads
🟦 HDF5 Library	7376					
🟧 Bypass VOL with Thread Pool	5423	12916	17561	21934	22550	22667
◼ C Test with Thread Pool	7552	12881	17117	21075	22763	24030

Perlmutter results (640GB dataset)



	1 thread	2 threads	4 threads	8 threads	12 threads	16 threads
🟦 HDF5 Library	991					
🟧 Bypass VOL with Thread Pool	1297	1244	1374	1856	1695	1646
⬜ C Test with Thread Pool	1265	1187	1453	1627	1666	1733



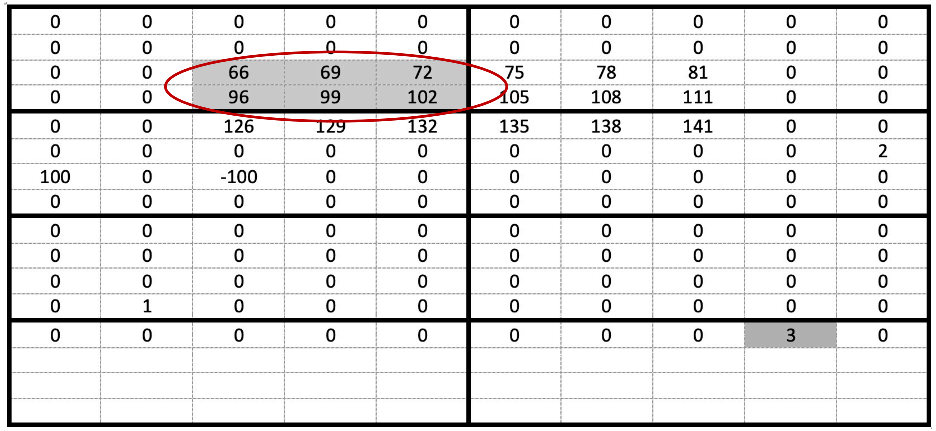
	1 thread	2 threads	4 threads	8 threads	12 threads	16 threads
🟦 HDF5 Library	823					
🟧 Bypass VOL with Thread Pool	547	800	1339	1358	1465	1557
⬜ C Test with Thread Pool	1002	790	1297	1489	1508	1528

Sparse Data Storage in HDF5

New storage paradigm for sparse and variable-length data

New Storage Paradigm: Structured Chunk

Chunked dataset



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	66	69	72	75	78	81	0	0
0	0	96	99	102	105	108	111	0	0
0	0	126	129	132	135	138	141	0	0
0	0	0	0	0	0	0	0	0	2
100	0	-100	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3	0

0 may represent a value that is not-defined

If we write a shown sub-array using hyperslab selection how the chunk will be stored in the file?

Chunked storage: all chunk elements are stored

```
0 0 0 0 0 0 0 0 0 0 0 0 66 69 72 0 0 96 99 96 102
```

Structured Chunk storage for sparse data:

Locations and values of defined elements
(specified by the "hyperslab" selection)
are stored in different sections of the chunk

Section 0	Encoded selection
Section 1	66 69 72 96 99 96 102

- Structured chunk may have more than 2 sections
- New way of storing variable-length data in HDF5
  - 3 sections when storing sparse variable-length data
- Each section can be compressed with its own compression (or filter pipeline)
- No changes to the programming model
- A few new APIs including H5Pset_filter to solve argument passing issue for the compression filters

Sparse Storage Implementation Status

- Design documents can be found in Lifeboat GitHub repo(see References slide)
  - Programming model and APIs
  - File Format extensions
  - Shared chunk cache
    - Better performing chunk cache including multi-threaded implementation
  - Improved I/O pipeline in HDF5 library
- Current status
  - File Format is finalized (subject to change until implementation is completed)
  - Started implementation of APIs and command-line tools to support sparse storage
    - White paper with benchmark results that motivate structured chunk compression https://github.com/LifeboatLLC/SparseHDF5/blob/main/benchmarks/Sparse-VL-Benchmarks-2024-01-16.pdf
  - Designs for shared chunk cache and I/O pipeline are under development
- Prototype release in Spring 2025

Integrity of Data in HDF5

HDF5 Encryption

Native Encryption in HDF5

Why?

- *Long standing feature request that will enhance HDF5 data security and integrity.*

How?

- *Use VFD layer to convert all HDF5 file I/O to paged I/O and then encrypt / decrypt as required.*
- *This allows random access to an encrypted file – transparent to the HDF5 library proper.*
- *Concept can be applied to parallel using the sub-filing infrastructure.*

Status – currently in Phase I:

- *Serial only proof of concept version near completion – currently in test and debug.*
- *Linux only for now. Supports AES and Two Fish using gcrypt library.*
- *Will be delivered as a pair of built in VFDs in HDF5 1.14.3*
- *Looking for reviewers – in particular, comments / suggestions on the API and key distribution.*
- *If this is something you need, please consider writing a letter of support for the Phase II proposal.*

Native Encryption in HDF5

Plans for Phase II

- *Finalize API / key distribution design as required.*
- *Write production versions of VFDs developed in Phase I:*
  - *Implement production API, key management, etc.*
  - *Make VFDs as encryption library and algorithm agnostic as practical*
  - *Performance enhancements – vector I/O support, thread pools.*
    - *Multi-thread and possibly selection I/O needed for parallel.*
  - *Make VFDs plug-able*
- *Finish and/or extend the HDF5 plug-able VFD support as needed.*
- *Finish out sub-filing as required for encryption in parallel – must*
  - *Retro-fit multi-thread support on VFD layer*
  - *Update sub-filing VFD to support at least vector I/O*
  - *Update I/O concentrators to use the VFD layer*

References

- Documentation and code for multithreaded project are available from
  https://github.com/LifeboatLLC/MT-HDF5/tree/main/design_docs
  https://github.com/LifeboatLLC/Experimental/ “1_14_2_multithread” branch

- Documentation for sparse projects is available from
  https://github.com/LifeboatLLC/SparseHDF5/tree/main/design_docs
  https://gamma.hdfgroup.org/ftp/pub/outgoing/vchoi/SPARSE/H5.format.html#ChangesForStructChunk (in-progress)

Acknowledgement

This work is supported by the U.S. Department of Energy, Office of Science under award number:

- DE-SC0023583 for Phase II SBIR project "Toward multi-threaded concurrency in HDF5”
- DE-SC0023583 for Phase II SBIR project “Supporting Sparse Data in HDF5”
- DE-SC0024823 for Phase I SBIR project “Protecting the confidentiality and integrity of data stored in HDF5"

Thank you!

Questions?