Outline

• Overview of parallel HDF5
• General best practices which effect parallel performance
• Best methods for HDF5 parallel I/O
• Using Parallel I/O instrumentation for tuning
Resources

• HDF5 home page: http://hdfgroup.org/HDF5/
• HDF5 Jira: https://jira.hdfgroup.org
• Documentation: https://portal.hdfgroup.org/display/HDF5/HDF5

• HDF5 repo: https://bitbucket.hdfgroup.org/projects/HDFFV/repos/hdf5/
  • We are moving to Github! Stay tuned for announcement

• Latest releases: https://portal.hdfgroup.org/display/support/Downloads
  • HDF5 1.8.21
  • HDF5 1.10.6
  • HDF5 1.12.0
HDF5 Version for parallel HDF5

• CGNS scaling for different versions of HDF5 (Summit, ORNL).
Parallel HDF5 Overview
Parallel HDF5 Overview

• In this section we will remind you about basics of parallel HDF5
• If you are new to parallel HDF5, see:
  • Online tutorials https://portal.hdfgroup.org/display/HDF5/Introduction+to+Parallel+HDF5
  • In-person tutorials
    • Super Computing Conference (MPI IO)
    • ECP annual meetings
    • National Laboratories (Argonne Training Program on Extreme-Scale Computing (ATPESC) )
Why Parallel HDF5?

• Take advantage of high-performance parallel I/O while reducing complexity
  • Use a well-defined high-level I/O layer instead of POSIX or MPI-IO
  • Use only a single or a few shared files
    • “Friends don’t let friends use file-per-process!” 😕

• Maintained code base, performance and data portability
  • Rely on HDF5 to optimize for underlying storage system
Benefit of Parallel I/O – Strong Scaling Example

CGNS – SUMMIT, ORNL
PHDF5 implementation layers

APPLICATION

MPI I/O LIBRARY

HDF5 LIBRARY

HDF5 file on Parallel File System

Interconnect Network + I/O Servers

Disk Architecture and Layout of Data on Disk
Parallel HDF5 (PHDF5) 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”
Parallel HDF5 Etiquette

- PHDF5 opens a shared file with an MPI communicator
  - Returns a file ID (as usual)
  - All future access to the file via that file ID
- Different files can be opened via different communicators
- 🌟 All processes must participate in collective PHDF5 APIs
- 🌟 All HDF5 APIs that modify the HDF5 namespace and structural metadata are collective!
  - File ops., group structure, dataset dimensions, object life-cycle, etc.
  - Raw data operations can either be collective or independent
    - For collective, all processes must participate, but they don’t need to read/write data.

Example of a PHDF5 C Program

Starting with a simple serial HDF5 program:

```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:

```c
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();
```
General HDF5 Programming Parallel Model for raw data I/O

- Each process defines selections in memory and in file (aka HDF5 hyperslabs) using H5Sselect_hyperslab
- The hyperslab parameters define the portion of the dataset to write to
  - Contiguous hyperslab
  - Regularly spaced data (column or row)
  - Pattern
  - Blocks
- Each process executes a write/read call using selections, which can be either collective or independent
Collective vs. Independent Operations

- MPI Collective Operations:
  - All processes of the communicator must participate, in the right order.
  
  E.g.,

  Process 1: call A(); call B();
  Process 2: call A(); call B();          ...CORRECT

  Process 1: call A(); call B();
  Process 2: call B(); call A();          ...WRONG

- Collective operations are not necessarily synchronous, nor must they require communication
  - It could be that only internal state for the communicator changes

- Collective I/O attempts to combine multiple smaller independent I/O ops into fewer larger ops; neither mode is preferable \textit{a priori}
Object Creation (Collective vs. Single Process)

The diagram illustrates the time (in seconds) required for object creation as the number of processes increases. The blue line represents Collective Object Creation, while the red line represents One Processes Object Creation. The graph shows a general trend of increasing time with the number of processes, with Collective Object Creation taking significantly longer than One Processes Object Creation.
CAUTION: Object Creation (Collective vs. Single Process)

- In sequential mode, HDF5 allocates chunks incrementally, i.e., when data is written to a chunk for the first time.
  - Chunk is also initialized with the default or user-provided fill value.

- In the parallel case, chunks are always allocated when the dataset is created (not incrementally).
  - The more ranks there are, the more chunks need to be allocated and initialized/written, which manifests itself as a slowdown.
Set HDF5 to never fill chunks (H5Pset_fill_time with H5D_FILL_TIME_NEVER)
Parallel Compression (HDF5 1.10.2 and later)
General HDF5 Best Practices Effecting Parallel Performance
Memory considerations

• **Open Objects**
  - Open objects use up memory. The amount of memory used may be substantial when many objects are left open. Application should:
    - Delay opening of files and datasets as close to their actual use as is feasible.
    - Close files and datasets as soon as their use is completed.
    - If opening a dataspace in a loop, be sure to close the dataspace with each iteration, as this can cause a large temporary "memory leak".

• There are APIs to determine if objects are left open.
  - `H5Fget_obj_count` will get the number of open objects in the file,
  - `H5Fget_obj_ids` will return a list of the open object identifiers.
HDF5 Dataset I/O

- Issue large I/O requests
  - At least as large as file system block size
- Avoid **datatype conversion**
  - Use the same data type in the file as in memory
- Avoid **dataspace conversion**
  - One dimensional buffer in memory to two-dimensional array in the file

ℹ️ Can break collective operations; check what mode was used

H5Pget_mpio_actual_io_mode, and why

H5Pget_mpio_no_collective_cause
HDF5 Dataset – Storage Type

• Use **contiguous storage** if no data will be added and compression is not used
  • Data will not be cached by HDF5

• Use **compact** storage when working with small data (<64K)
  • Data becomes part of HDF5 internal metadata and is cached (metadata cache)

• Avoid data duplication to reduce file sizes
  • Use links to point to datasets stored in the same or external HDF5 file
  • Use VDS to point to data stored in other HDF5 datasets
HDF5 Dataset – Chunked Storage

- Chunking is required when using extendibility and/or compression and other filters
- I/O is always performed **on a whole chunk**
- Understand how **chunking cache** works
  [https://portal.hdfgroup.org/display/HDF5/Chunking+in+HDF5](https://portal.hdfgroup.org/display/HDF5/Chunking+in+HDF5) and consider
  - Do you access the same chunk often?
  - What is the best chunk size (especially when using compression)?
HDF5 Parallel Performance
Performance Tuning is a Multi-layer Problem

- **Application**
  (Semantic organization, standards compliance ...)

- **HDF5**
  (cache chunk size, independent/collective ...)

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

- **Parallel File System**
  (Lustre – stripe factor and stripe size)

- **Storage Hardware**

Our focus today is on HDF5 and PFS.
Parallel File Systems – Lustre, GPFS, etc.

• Scalable, POSIX-compliant file systems designed for large, distributed-memory systems
• Uses a client-server model with separate servers for file metadata and file content
Effects of Software/Hardware Changes

- Poor/Improved performance can be a result of FS changes
- Single shared file using MPI-IO performance degradation [Byna, NERSC].
Effects of influencing object’s in the file layout

- H5Pset_alignment – controls alignment of file objects on addresses.
How to pass hints to MPI from HDF5

• To set hints for MPI using HDF5, see: H5Pset_fapl_mpio
• Use the 'info' parameter to pass these kinds of low-level MPI-IO tuning tweaks.
• C Example – Controls the number of aggregators on GPFS:

```c
MPI_Info info;
MPI_Info_create(&info); /* MPI hints: the key and value are strings */
MPI_Info_set(info, "bg_nodes_pset", "1");
H5Pset_fapl_mpio(plist_id, MPI_COMM_WORLD, info);
/* Pass plist_id to H5Fopen or H5Fcreate */
file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
```
Use Case CGNS

Performance tuning
• 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
Performance issue: Slow opening of an HDF5 File …

• Opening an existing file
  • CGNS reads the entire HDF5 file structure, loading a lot of (HDF5) metadata
  • Reads occur independently on ALL ranks competing for the same metadata

"Read Storm"
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");
```
HDF5 Metadata Read Storm Problem (II)

• HDF5 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

- Hint that metadata access is done collectively
  - H5Pset_coll_metadata_write, H5Pset_all_coll_metadata_ops
  - A property on an access property list
  - 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
  - If set, MPI rank 0 will issue the read for a metadata entry to the file system and broadcast to all other ranks
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
Diagnostics and Instrumentation Tools
I/O monitoring and profiling 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
• Profilers have to compromise between
  • A lot of detail ➔ large trace files and overhead
  • Aggregation ➔ loss of detail, but low overhead

• Examples of I/O benchmarks:
  • h5perf (in the HDF5 source code distro and binaries)
  • IOR  https://github.com/hpc/ior

• Examples of profilers
  • Darshan  https://www.mcs.anl.gov/research/projects/darshan/
  • Recorder  https://github.com/uiuc-hpc/Recorder
  • TAU built with HDF5
“Poor Man’s Debugging”

• Build a version of PHDF5 with
  ```
  ./configure --enable-build-mode=debug --enable-parallel ...
  ```

``` setenv H5FD_mpio_Debug "rw" ```

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

• You’ll get something like this…
“Poor Man’s Debugging” (cont’d)
Example - 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

…
“Poor Man’s Debugging” (cont’d)
Debugging Collective Operations

setenv H5_COLL_API_SANITY_CHECK 1

- HDF5 library will perform an MPI_Barrier() call inside each metadata operation that modifies the HDF5 namespace.
- Helps to find which rank is hanging in the MPI barrier
Use Case

Tuning PSDNS with Darshan
Darshan (ECP DataLib team)

• 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, sequentially, access size
  • Cumulative I/O time, intervals of I/O activity

• An excellent starting point
Darshan Use-Case (Blue Waters, NCSA)

- PSDNS code solves the incompressible Navier-Stokes equations in a periodic domain using pseudo-spectral methods.
- Uses custom sub-filing by collapsing the 3D in-memory layout into a 2D arrangement of HDF5 files.
- Uses virtual dataset which combines the datasets distributed over several HDF5 files into a single logical dataset.

Slow read times.

Ran experiments on 32,768 processes with **Darshan** 3.1.3 to create an I/O profile.
Darshan Use-Case (Blue Waters, NCSA)

... total_POSIX_SIZE_READ_0_100: 196608 total_POSIX_SIZE_READ_100_1K: 393216 total_POSIX_SIZE_READ_1K_10K: 617472 total_POSIX_SIZE_READ_10K_100K: 32768 total_POSIX_SIZE_READ_100K_1M: 2097152 total_POSIX_SIZE_READ_1M_4M: 0 total_POSIX_SIZE_READ_4M_10M: 0 total_POSIX_SIZE_READ_10M_100M: 0 total_POSIX_SIZE_READ_100M_1G: 0 total_POSIX_SIZE_READ_1G_PLUS: 0 ...

⚠️ Large numbers of reads of only small amounts of data.

🛠️ Multiple MPI ranks independently read data from a small restart file which contains a virtual dataset.
“Broadcast” the restart file:

1. Rank 0: read the restart file as a byte stream into a memory buffer.
2. Rank 0: broadcasts the buffer.
3. All MPI ranks open the buffer as an HDF5 file image, and proceed as if they were performing reads against an HDF5 file stored in a file system.

Eliminates the “read storm”,

....

```
total_POSIX_SIZE_READ_0_100: 6
total_POSIX_SIZE_READ_100_1K: 0
total_POSIX_SIZE_READ_1K_10K: 0
total_POSIX_SIZE_READ_10K_100K: 2
total_POSIX_SIZE_READ_100K_1M: 0
total_POSIX_SIZE_READ_1M_4M: 0
```

```
total_POSIX_SIZE_READ_4M_10M: 0
total_POSIX_SIZE_READ_10M_100M: 0
total_POSIX_SIZE_READ_100M_1G: 32768
```

```
total_POSIX_SIZE_READ_1G_PLUS: 0
```
Use Case

Tuning HACC (Hardware/Hybrid Accelerated Cosmology Code) with Recorder
Recorder

• Multi-level I/O tracing library that captures function calls from HDF5, MPI and POSIX.
• It keeps every function and its parameters. Useful to exam access patterns.
• Built-in visualizations for access patterns, function counters, I/O sizes, etc.
• Also reports I/O conflicts such as write-after-write, write-after-read, etc. Useful for consistency semantics check (File systems with weaker consistency semantics).

https://github.com/uiuc-hpc/Recorder
Write Pattern Effects – Data location in the file

Pattern 1 – HDF5 pattern

Variable 1 (v1)  Variable 2 (v2)  Variable N (vN)

\[ \begin{array}{ccc}
P_0 & P_1 & P_2 \\
\end{array} \quad \begin{array}{ccc}
P_0 & P_1 & P_2 \\
\end{array} \quad \begin{array}{ccc}
P_0 & P_1 & P_2 \\
\end{array} \]

Variables are **contiguously** stored in the file

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

\[ \begin{array}{ccc}
v1 & v2 & vN \\
\end{array} \quad \begin{array}{ccc}
v1 & v2 & vN \\
\end{array} \quad \begin{array}{ccc}
v1 & v2 & vN \\
\end{array} \]

\[ \begin{array}{ccc}
P_0 & P_0 & \ldots \\
\end{array} \quad \begin{array}{ccc}
P_0 & P_1 & P_1 \\
\end{array} \quad \begin{array}{ccc}
P_1 & P_2 & P_2 \\
\end{array} \quad \begin{array}{ccc}
P_2 \\
\end{array} \]

Variables are **interleaved** in the file
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: HDF5 access patterns

HDF5 with individual dataset

HDF5 with compound datatype

HDF5 with allocate multi
Will Collective I/O make the access pattern (on the left) of individual dataset better?
- Problem size: 8GB per variable, 72GB in total
- Lustre config: Stripe count 32, Stripe Size 512M
- Each rank writes 9 variables
- The size of each write is 8GB/1024 Processes = 8MB

ROMIO:
- romio_cb_read/write = automatic
- "When set to automatic, ROMIO will use heuristics to determine when to enable the optimization."

HACC-IO: access patterns of HDF5 with collective I/O

Writes are aggregated by 32 nodes whereas reads remain the same
Interleaved is not always better, and neither is collective IO

- Write bandwidth with different stripe size.
- Individual dataset is better when using large stripe sizes.
Interleaved is not always better, and neither is collective IO

- When the request size is big, the collective communication overhead increases and the benefits from collective I/O becomes limited.
- Request size is 8MB in our case.
- Collective writes are indeed much faster: 83 seconds vs 1539 seconds in independent mode.
- However, the cost for communication is too high
Interleaved is not always better, and neither is collective IO

- Write bandwidth with different stripe size.
- Individual dataset is better when using large stripe sizes.
HACC-IO: MPI vs HDF5

• Same access pattern, but why MPI is faster?

- HDF5 writes 2048 bytes metadata at the beginning of the file.
- This causes the alignment issue for the data writes.
• HDF Knowledge base
  https://portal.hdfgroup.org/display/knowledge/Parallel+HDF5
• HDF-FORUM https://forum.hdfgroup.org/
• HDF Helpdesk help@hdfgroup.org
Acknowledgement

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences under Award Number DE-AC05-00OR22725.

Disclaimer
This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
THANK YOU!

Questions & Comments?