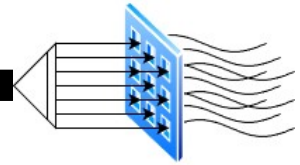


Accelerating Data Compression

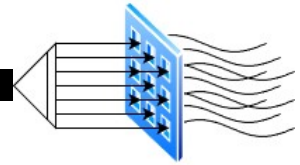
Through Parallel Filter Processing

Frederick Neu

Scientific Computing
Universität Hamburg
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

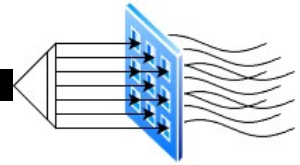
05/28/25

2025 HDF5 User Group Meeting



Motivation (Background)

- “Create some dataset and let’s see, how HDF5 handles it using LZ4 Filtering”
  - Compression to reduce I/O load
  - Counting ascending, odd numbers being “1”
  - LZ4 compression ratio roughly 1.993 : 1 (Block size 8192 Byte)
- Tracing the dataset
  - From API until final write
  - Caching (LRU)
- Simply writing (a lot) to storage (no caching needed)
  - Potential to introduce parallelism?
  - --enable-threadsafe (dead)locking



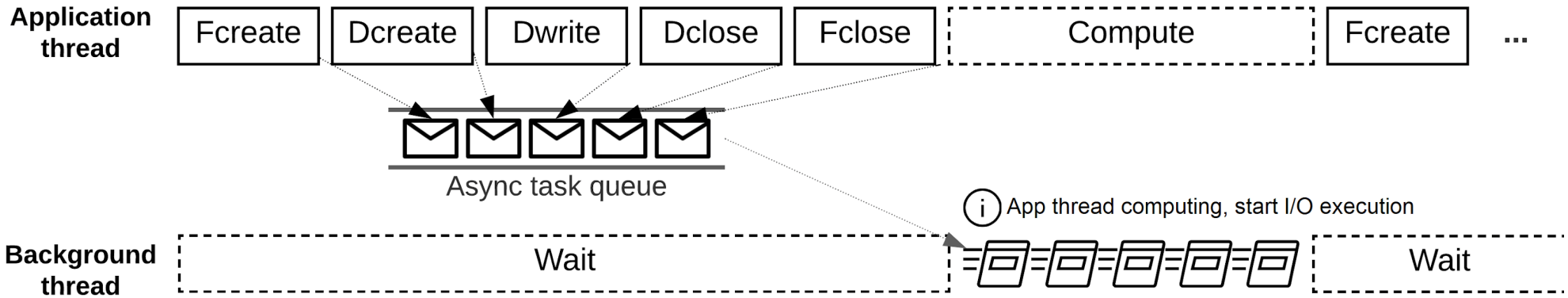
Now: Writing a chunked dataset

- H5Dwrite(…)
  - Native VOL connector
  - POSIX VFD
- H5D__chunk_write() (./src/H5Dchunk.c)

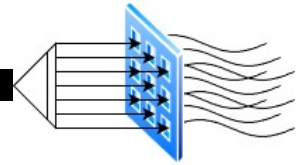
```
3245.   /* Iterate through nodes in chunk skip list */
3246.   chunk_node = H5D_CHUNK_GET_FIRST_NODE(dset_info);
3247.   while (chunk_node) {
                        ⋮ (… creating chunk, place into cache, filter, write …)
3359.       /* Advance to next chunk in list */
3360.       chunk_node = H5D_CHUNK_GET_NEXT_NODE(dset_info, chunk_node);
3361.   }
```

Currently using multithreading

- HDF5 Asynchronous I/O VOL connector (https://hdf5-vol-async.readthedocs.io/en/latest/)



- Split chunk (older prototype)
  - Chunks split into "sub chunks"
  - Sub chunks asynchronously filtered

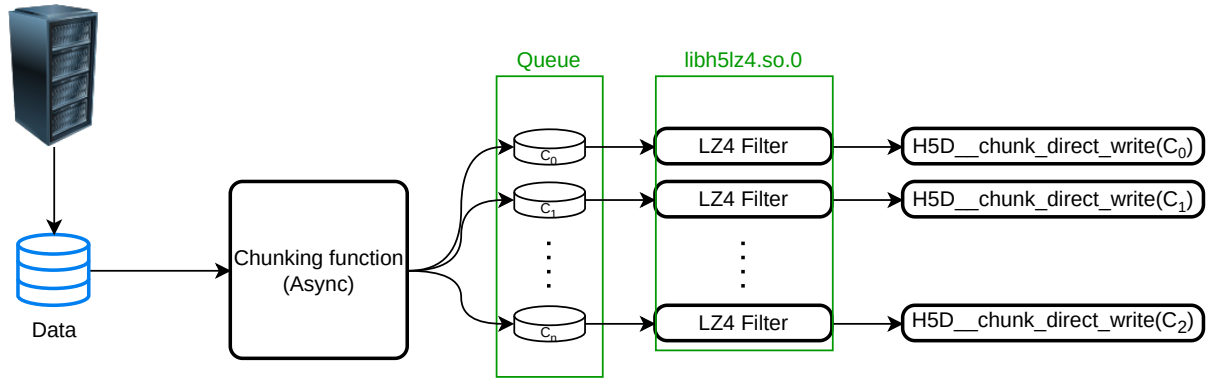


Ways to write

- Native VOL connector offers
  - Standard (through *H5Dwrite*)
  - Optional (through *H5Dwrite_chunk*)
    - Unfortunately API function, deadlock likely if **--enable-threadsafe** used
- Bypass
  - RFC: Direct Chunk Write
    - Raymond Lu (https://support.hdfgroup.org/releases/hdf5/documentation/rfc/DECTRIS%20Integration%20RFC%202012-11-29.pdf)
    - "[…] bypass the library's data conversion and filter pipeline and write data chunks directly to a dataset in the file."
    - Used by native VOL connector's optional write

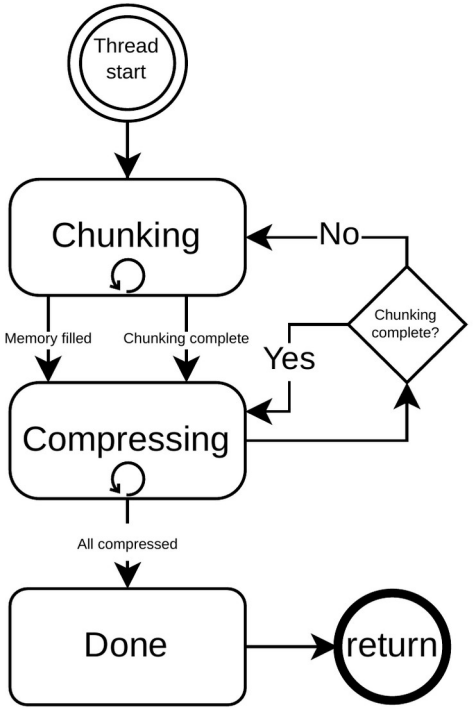
Trust me, Property List

- Setting up the Properties of a Dataset Object remains unchanged
- Introducing a new API (*H5Dwrite_filter_parallel()*)
  - Taking the whole dataset as parameter
  - Internally using thread pool
    - Chunking
    - Applying filters as promised(!) to PList
    - Writing (through VFD)



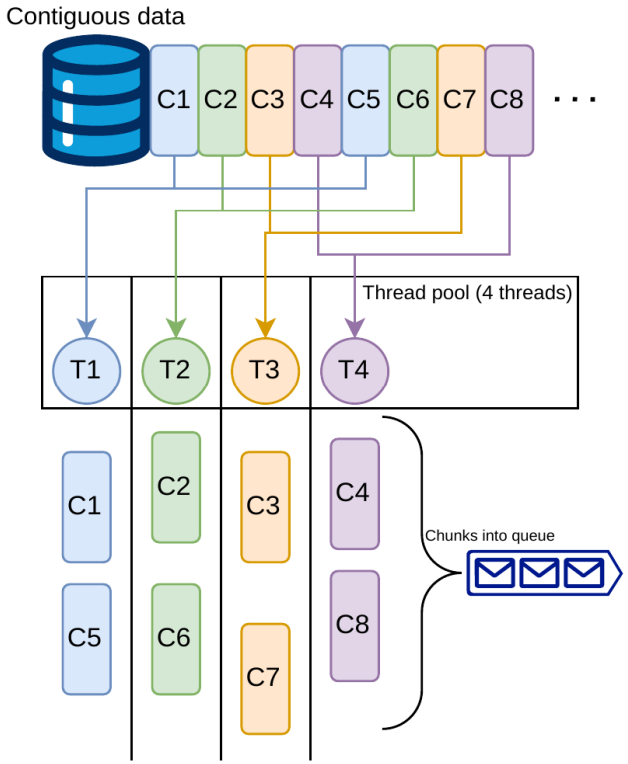
Into the Pool

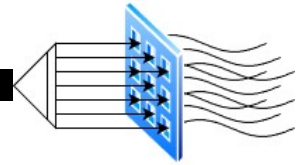
- Chunking
  - Asynchronous chunk creation (subset for each thread)
  - Memory limiting possibility (save state, jump to **Compressing**)
- Compressing
  - Dynamically loading filter into memory
    - Applying to chunk
  - Writing to storage
    - potentially returning to **Chunking**, if previously Memory Limited
- Done
  - Thread can be collected by parent process



Asynchronous Chunking

- Each thread has own subset
  - Similar speeds (using *memcpy*) per chunk
- Completely thread independent chunking function
  - Protecting only adding addresses into queue
- Flexible changing of status
  - Chunking or Compressing
  - Currently interrupted chunk is remembered
    - Continues on return with no data loss





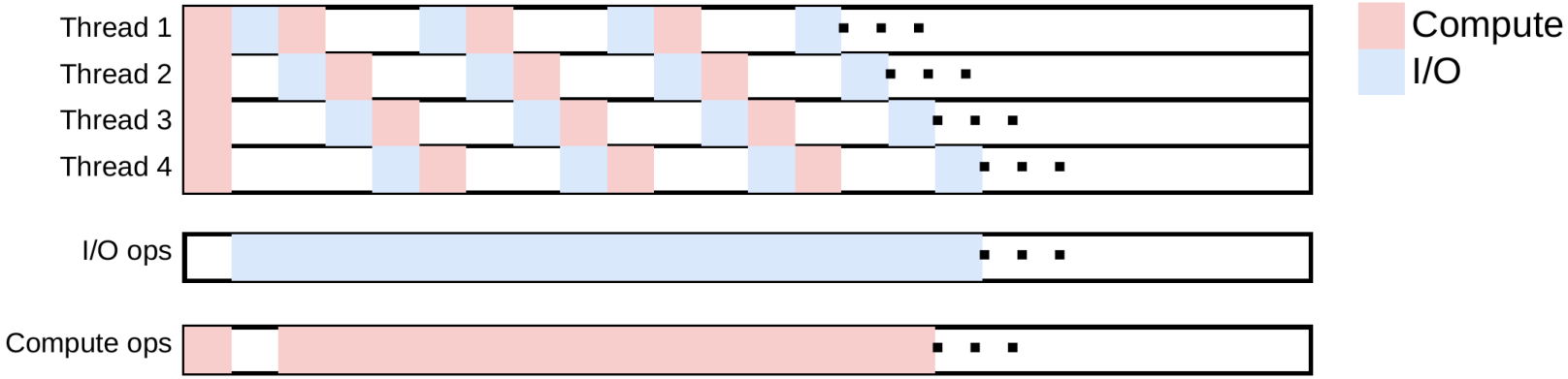
Parallel Filtering

- Dynamically loaded from shared library
  - Based on Property List information (Trust)
    - Fault, if filter not found
  - Potentially all HDF5 registered Filters usable
    - **H5Z_class2_t** struct registered
- Filter instructions fetched from Property List
  - cd_nelmts, cd_values

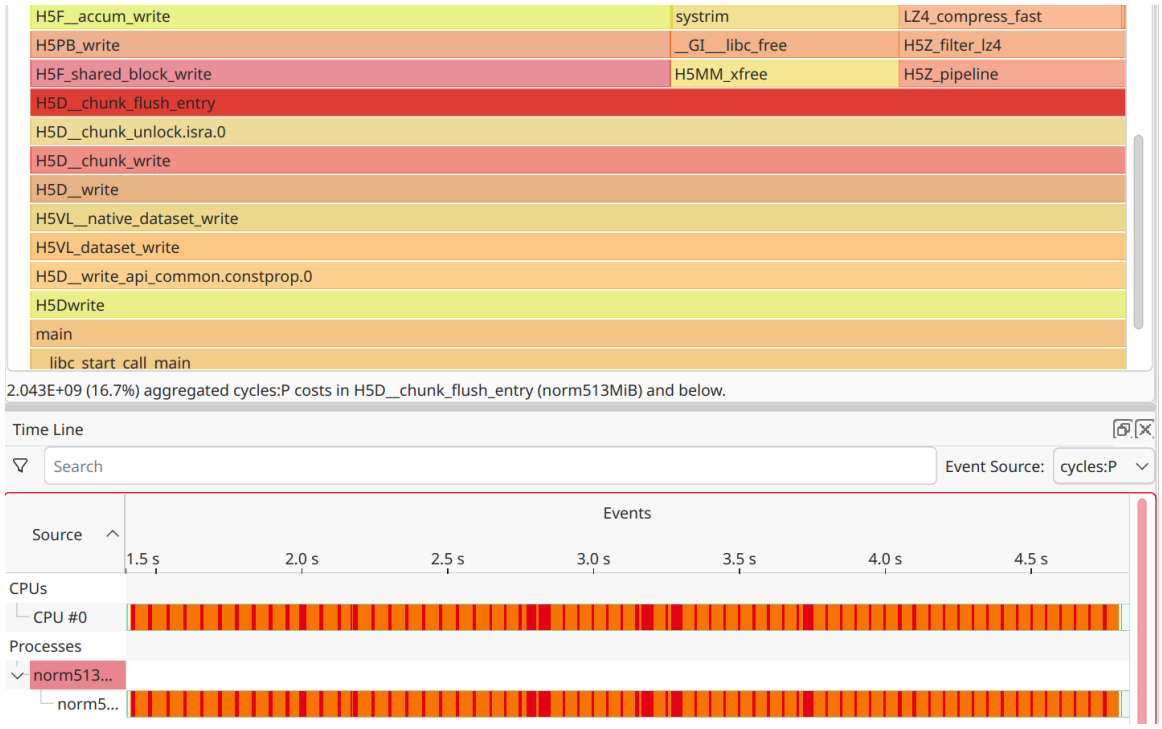
- Filter(s) applied to each chunk independently
  - After fetched from queue (protected)
  - Itself asynchronous
- *H5D__chunk_direct_write* as soon as filtered
  - Avails memory again
  - Function mutex locked
    - Causes dependency on next chunk write
    - Threads with filtered chunks ready wait
      - No further filtering
      - Reduces to Parallel Filtering

What’s to expect?

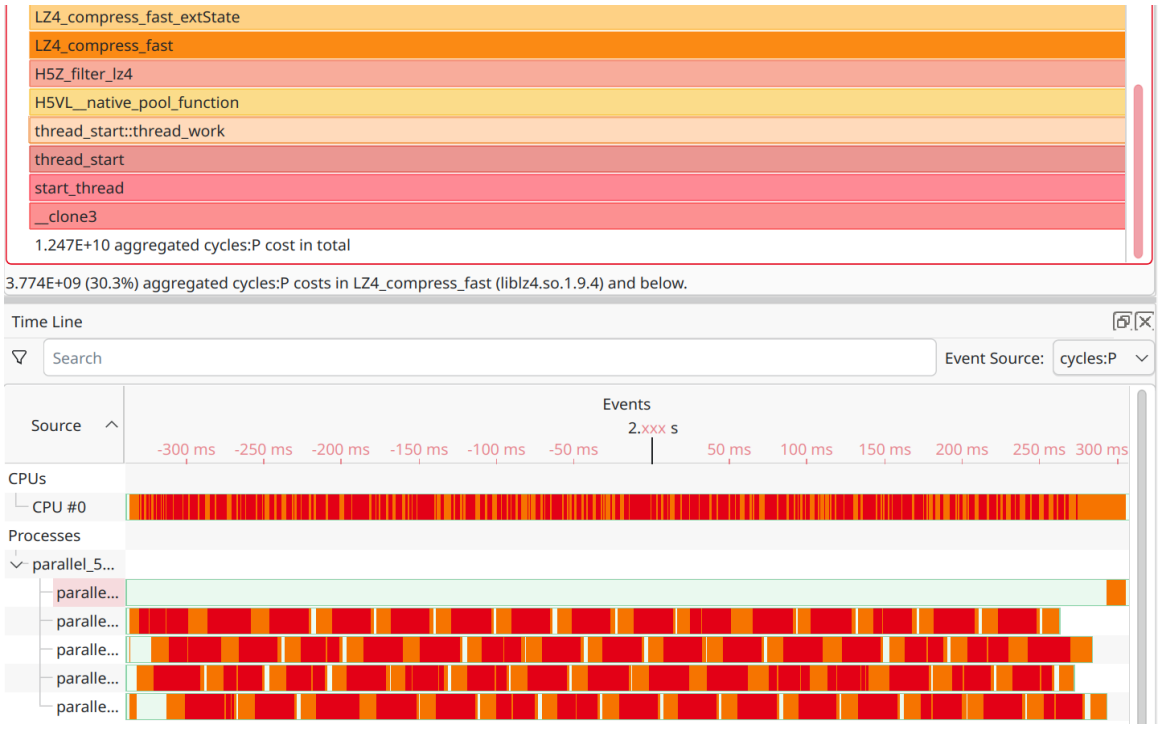
- Thread alternating writes
- Continuous I/O
  - With sufficient threads
- More efficient CPU usage
  - Filtering while writing



Native H5Dwrite



Pthread Parallel Filtering



Benchmarks

- System: DKRZ Levante HPC (compute node, 2x AMD 7763 CPU; 128 cores in total)
- Problem: 3GiB raw data, 4MiB Chunks; (100 samples)

DKRZ
DEUTSCHES KLIMARECHENZENTRUM