# Accelerating Data Compression

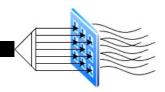
Through Parallel Filter Processing

**Frederick Neu** 



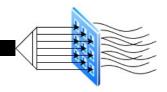
2025 HDF5 User Group Meeting





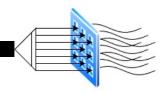
### Motivation (Background)

- "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



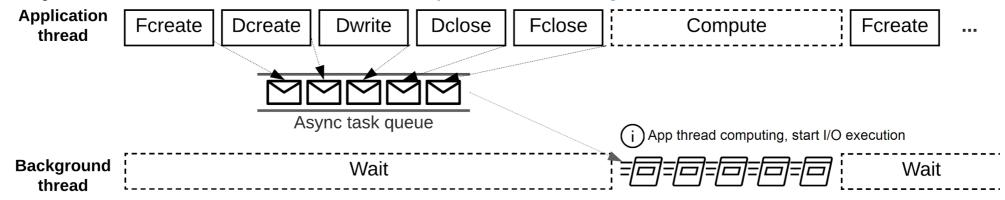
#### Now: Writing a chunked dataset

- 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**

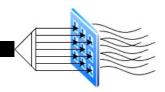
HDF5 Asynchronous I/O VOL connector (<u>https://hdf5-vol-async.readthedocs.io/en/latest/</u>)



- Split chunk (older prototype)
  - Chunks split into "sub chunks"
  - <sup>o</sup> Sub chunks asynchronously filtered

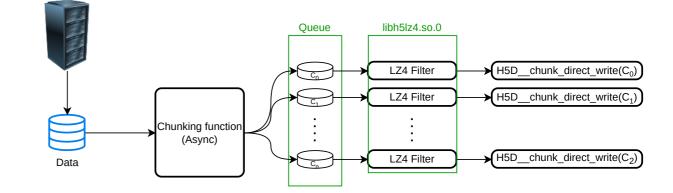
### Ways to write

- 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

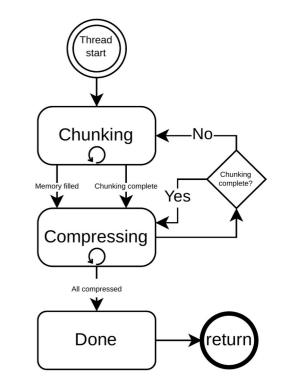
- Setting up the Properties of a Dataset Object remains unchanged
- Introducing a new API (*H5Dwrite\_filter\_parallel()*)
  - Taking the whole dataset as parameter
  - Internally using thread pool
    - Chunking
    - Applying filters as promised(!) to PList
    - Writing (through VFD)

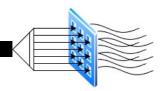


#### Into the Pool

#### • Chunking

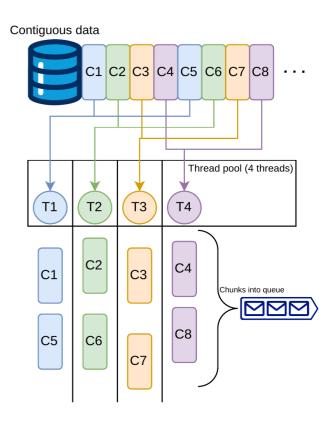
- Asynchronous chunk creation (subset for each thread)
- Memory limiting possibility (save state, jump to **Compressing**)
- Compressing
  - Dynamically loading filter into memory
    - Applying to chunk
  - Writing to storage
    - potentially returning to **Chunking**, if previously Memory Limited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





## **Asynchronous Chunking**

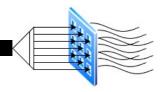
- Each thread has own subset
  - Similar speeds (using *memcpy*) per chunk
- Completely thread independent chunking function
  - Protecting only adding addresses into queue
- Flexible changing of status
  - Chunking or Compressing
  - Currently interrupted chunk is remembered
    - Continues on return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



### **Parallel Filtering**

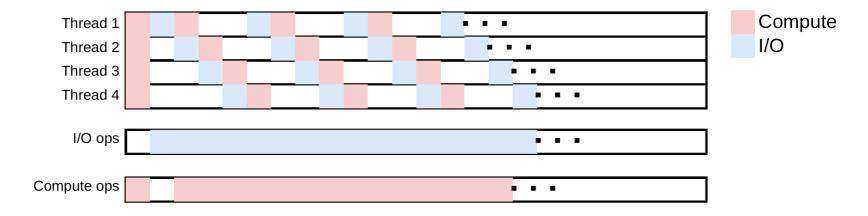
- Dynamically loaded from shared library
  - Based on Property List information (Trust)
    - Fault, if filter not found
  - Potentially all HDF5 registered Filters usable
    - H5Z\_class2\_t struct registered
- Filter instructions fetched from Property List
  - cd\_nelmts, cd\_values

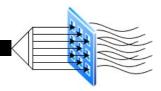
- Filter(s) applied to each chunk independently
  - After fetched from queue (protected)
  - Itself asynchronous
- *H5D\_chunk\_direct\_write* as soon as filtered
  - Avails memory again
  - Function mutex locked
    - Causes dependency on next chunk write
    - Threads with filtered chunks ready wait
      - 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

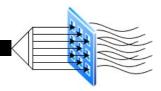




#### **Native H5Dwrite**

		ess_fast_ext5	State									
	LZ4_compress_fast H5Z_filter_lz4											
	_GI_sbrk											
			cycles:P cost in total									
4.07	7F+09 (33.39	%) aggregate	ed cycles:P costs in LZ4_	compress fast (liblz	4 so 1 9 4) and below	N				ت		
		, of aggine gate			inseries in and seree				6			
Time Line									(ð)	[×]		
$\nabla$	Search							Event Source:	cycles:P	$\sim$		
					Events							
9	Source ^											
		1.5 s	2.0 s	2.5 s	3.0 s	3.5 s	4.0 s	4.5 s				
CPL	Js											
	CPU #0											
Processes												
v norm513												
	norm5											

H5Faccum_write				systrim	LZ4	_compress_fast		
H5PB_write	GIlibc_free	H52	_filter_lz4					
H5F_shared_block_write				H5MM_xfree	H52	_pipeline		
H5Dchunk_flush_entry	(							
H5D_chunk_unlock.isra.0 H5D_chunk_write								
H5VLnative_dataset_w	rite							
H5VL_dataset_write								
H5Dwrite_api_commo	n.constprop.0							
H5Dwrite								
main								
libc start call main								
43E+09 (16.7%) aggregat	ed cycles:P costs in H5	Dchunk_flush_ent	ry (norm513MiB) and	d below.				
43E+09 (16.7%) aggregat ne Line	ed cycles:P costs in H5	Dchunk_flush_ent	ry (norm513MiB) and	d below.			Ð	
	ed cycles:P costs in H5	Dchunk_flush_ent	ry (norm513MiB) and	d below.		Event Source:		
ne Line	ed cycles:P costs in H5	Dchunk_flush_ent	ry (norm513MiB) and	d below.		Event Source:		
ne Line	ed cycles:P costs in H5			3.5 s	4.0 s	Event Source:	cycles:P	
Source		D_chunk_flush_ent	Events		4.0 s	J	cycles:P	
Source			Events		4.0 s	J	cycles:P	
Source ^ 1.5 s			Events		4.0 s	J	cycles:P	
Source ^ 1.5 s Us CPU #0			Events		4.0 s	J	cycles:P	



#### **Pthread Parallel Filtering**

LZ4_compres		s_fast_extState								
L	_Z4_compres	j_fast								
H	H5Z_filter_lz4 H5VLnative_pool_function									
ł										
t	hread_start::	thread_work								
t	hread_start									
s	start_thread									
_	_clone3									
1	1.247 <mark>E</mark> +10 ag	gregated cycles:P cost in total								
774	4E+09 (30.3%	) aggregated cycles:P costs in LZ4_compress_fast (liblz4.so.1.9.4) and below.								
		- 55 - 5 5		ලි						
	e Line	ie المعالم الم								
7	Search		Event Source:	cycles:P 🚿						
S	ource ^	Events           2.xxx s           -300 ms         -200 ms         -150 ms         -100 ms         -50 ms         50 ms         100 ms         150 ms         2	200 ms 250 n	ns 300 ms						
۶U	s									
C	CPU #0									
ос	esses									
- p	oarallel_5									
	paralle									
	– paralle									
	– paralle									
	– paralle									
	paralle									

	H5PB_write									
	H5F_shared_block_write									
	H5D_chunk_direct_write									
	H5VL_native_pool_function									
	thread_start::thread_work									
	thread_star	thread_start								
	start_threa	start_thread								
	_clone3	_clone3								
	1.247E+10	aggregated cycles:P cost in total	J							
1.4	467E+09 (11.8	8%) aggregated cycles:P costs in H5D_chunk_direct_write (parallel_513MiB) and below.								
Ti	me Line	(ଟି.)	ζ							
$\nabla$	Search	Event Source: cycles:P 🗸								
	Source ^	Events         2.xxx s           -300 ms         -250 ms         -150 ms         -100 ms         -50 ms         50 ms         100 ms         150 ms         200 ms         250 ms         300 ms								
СР	PUs									
	CPU #0									
Processes										
$\checkmark$	parallel_5									
	paralle									
	— paralle									
	paralle									
	paralle.									
	paralle.									

#### Benchmarks

