Down to the bytes: can we simplify alternative access to HDF5?

Thomas Kluyver, European XFEL

European HDF5 User Group Meeting 2025, DESY

I work on

- h5py – Python wrapper for HDF5
- h5glance - Terminal (& Jupyter notebook) HDF5 viewer
- EXtra-data – Higher-level abstraction for European XFEL HDF5 data

```
f = h5py.File("my-data.h5")
arr = f["path/to/dataset"][5:10, :100]
```

```
$ h5glance example.h5
example.h5
├group1
│ ├subgroup1
│ │ ├dataset1   [uint64: 200]
│ │ └dataset2   [float32: 2 × 128 × 500]
│ └subgroup2
│   └dataset1   [int16: 12]
└softlink       -> /group1/subgroup1/dataset2
```

Motivation

- HDF5 is a very versatile file format
  - Chunking, filters, external data, custom data types…
- Many use cases using a small subset of features – but not the **same** subset
- Reimplementing HDF5 is a massive task
- Can we make HDF5 readers/writers for specific use cases?

The HDF5 file format

- 4 versions of superblock
- 24 header message types, many with multiple versions
- 2 B-tree formats
- 5 kinds of chunk index
- 4 ways to refer to another file
- **Most** lengths & offset may be 2/4/8 bytes

Layout: Array Property Description for Datatype Version 2

Byte	Byte	Byte	Byte
Dimensionality	Reserved (zero)		
Dimension #1 Size			
. . .			
Dimension #n Size			
Permutation Index #1			
. . .			
Permutation Index #n			
Base Type			

Fields: Array Property

Field Name	
Dimensionality	This value is the numb
Dimension #n Size	This value is the size o list of dimensions is th changing dimension.
	This value is the index

Reimplementations

- H5Coro (C++)
- H5Coro (Python)
- jhdf (Java)
- PureHDF (C#)
- pyfive (Python)
- jsfive (JavaScript)
- JLD2.jl (Julia)

Not an exhaustive list

Compatibility

Format Element	Supported	Contains	Missing
Field Sizes	Yes	1, 2, 4, 8, bytes	
Superblock	Partial	Version 0, 2	Version 1, 3
Base Address	Yes		
B-Tree	Partial	Version 1	Version 2
Group Symbol Table	Yes	Version 1	
Local Heap	Yes	Version 0	
Global Heap	No		Version 1
Fractal Heap	Yes	Version 0	
Shared Object Header Message Table	No		Version 0

From H5coro's README

Motivation

Specialised implementations could support:

- Parallel decompression
- Parsing other file sources (e.g. fsspec in Python)
- Avoiding performance ‘cliffs’
- Read strategies within chunks (e.g. Blosc)
- Parallel writing without MPI
- Streaming
- Error detection in virtual datasets

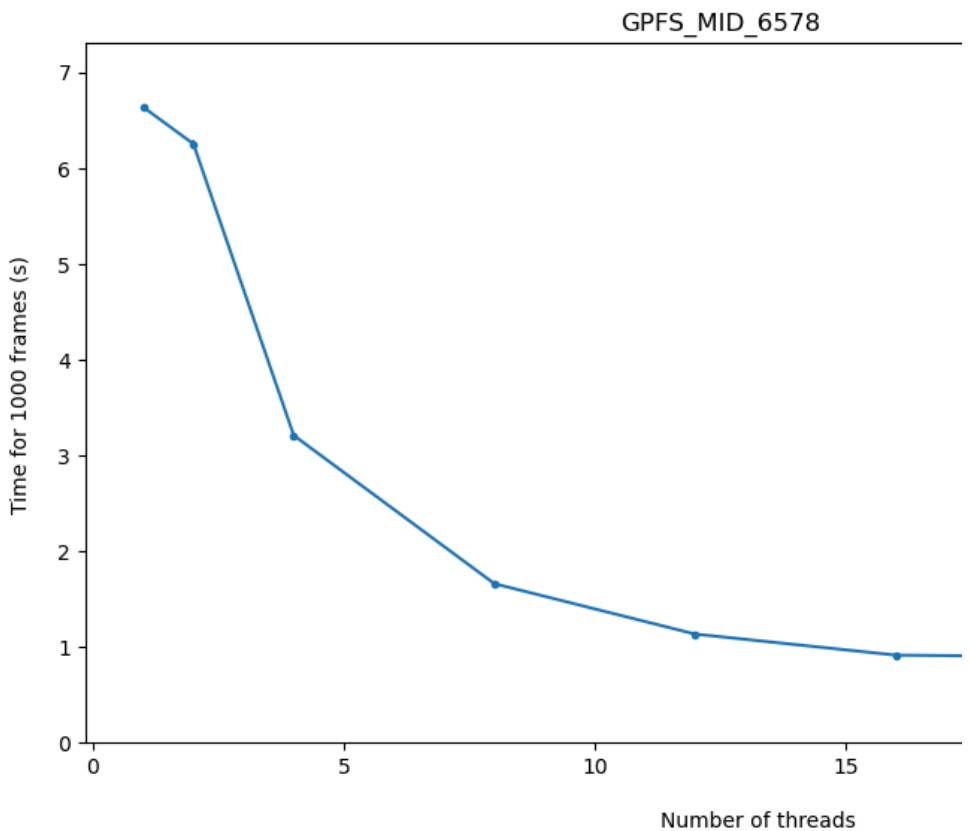
EXtra-data

- Python library for data access at European XFEL
  - Built on h5py
  - Combine many files from one ‘run’
  - Align data based on indexes
  - Split data to limit memory use or for parallel processing
  - Bring data into numpy, xarray, pandas, dask

```
run = open_run(proposal=4321, run=56)
arr = run.alias['photon_energy'].ndarray()
```

EXtra-data

- Parallel decompression
  - Around 6× speedup (so far)
  - Python thread pool
  - Byte shuffle using numpy
  - Very specific filters & chunking patterns



A pattern

- Check our expectations:
  - Data: datatype, chunking, filters
  - Access: selection, conversion
  - Consistency: all datasets similar?
- Fast path if expectations met
- Fallback to regular HDF5 access?
  - Performance cliff

In EXtra-data :

```
class ShuffleDeflateDecompressor:
  @classmethod
  def for_dataset(cls, dset):
      ...

  def apply_filters(self, chunk, fmask, out):
      ...
```

In b2h5py :

```
try:
    sel = opt_slice_check(self, args)
except NoOptSlicingError:
    return self.__dset.__getitem__(args)
with h5phil:
    return opt_selection_read(self.__dset, sel)
```

Improving h5py?

- More low-level details may need to be exposed as public attributes
- New APIs to register read/write accelerators?
  - b2h5py monkeypatches h5py instead for now
- Seeing through virtual datasets
- Higher level abstractions across datasets/files?

Low level APIs

- `H5Dread_chunk` & `H5Dwrite_chunk` (since 2018)
  - Bypass the filter pipeline
- `H5Dget_chunk_info[_by_coord]` (since 2019) & `H5Dchunk_iter` (since 2022)
  - Bypass file reading / writing (and filters)
  - Get offset & size within file
  - Can build external indexes, as in kerchunk
- All exposed in h5py, e.g. `dset.id.get_chunk_info()`

Going further

- Faster reads/writes are low-hanging fruit
- Other ideas involve getting between HDF5 and I/O
  - File drivers do this in theory, but limited in practice
- Separate (de)serialisation from I/O
  - Difficult for HDF5 overall
  - But object header messages are a large part of the format
  - Abstracting different versions of a given struct would be valuable

Machine-readable specifications

- Created from specification document
- Kaitai Struct language (YAML)
- Generate parsers in various languages, and (experimentally) serialisers
- Written for most of HDF5 specification: superblocks, object headers, symbol tables
  - ~1500 lines
- Not extensively tested yet
- Goal: publish this along with a Python library using it for low-level HDF5 access

```yaml
data_layout_chunked_v3:
  params:
    - id: offset_size
      type: u1
  seq:
    - id: ndims
      type: u1
    - id: address
      type:
        switch-on: offset_size
        cases:
          2: u2
          4: u4
          8: u8
    - id: shape
      type: u4
      repeat: expr
      repeat-expr: ndims
      doc: "Shape of one chunk"
    - id: element_size
      type: u4
      doc: "Size of one element in bytes"
```