Using HDF5 as a serialization format

First impressions

Vijay Kartik
Scientific Computing @ DESY

Context: why serialize?

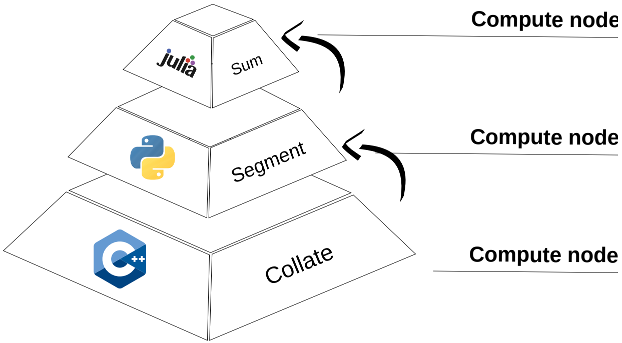
Data analysis today (@DESY)

	Then: monolithic analysis	Now: streaming analysis
Structure	Single program	Multiple blocks
Output	Final result	Intermediate results
Intermediate results	Program-internal	Passed between blocks
Location	Single machine (or MPI)	Spread over machines/languages

The problem: sharing n-dimensional data

"Streaming" data analysers -

- share pipeline
- pass intermediate results
- use C, C++, Python, Julia, etc.



How to ship multi-dimensional data around? Serialize + send

Serialize: why HDF5?

`H5Dwrite` = “serialize”

HDF5 file = serial buffer

HDF5 as a wire format

- Self-describing
- Multi-language
- Easy set-up

“Free” goodies

- Compression
- Metadata/attributes
- “Parser”

Usability = ☺

HDF5 as a serializer? Yes (but…)

- File Images - very nice to use
- On-disk (for kicks) - could be a backup?
- “Universal” access

Usability = ☺

> Heavy lifting, bookkeeping done by HDF5

> N-dim. tensors/arrays, scalars, everything

> Painless IPC

  - No homemade header, magic number
  - No protocol spec
  - Every receiver ‘speaks’ HDF5

> Quick implementation with `h5cpp` (ESS/DESY) [GitHub link]

But…

Speed = 😑

- OK-ish speeds (`H5Dwrite`)
  - buffer sizes/increments - tunable
  - copies by default
- `DONT_COPY | DONT_RELEASE`
  - 15% speedup
- Compression `OFF`: ~600 MB/S
- Compression `ON` : ~200-300 MB/s
- Remember: Deserialization on top

Impressions

- HDF5 very easy to use as a serialization format
- Lots of extras come “for free”
- Perfectly sufficient for several use cases
- Write speed is strictly okay
- Other (less user*-friendly) candidates? (Apache Arrow?)

More questions? Get in touch!

vijay.kartik@desy.de