“Extendable type-safe, thread-safe, asynchronous APIs for Neutron Science Data using modern C++ on top of HDF5”

William F Godoy, Addi Malviya Thakur, Steven E Hahn

Computer Science and Mathematics Division
Oak Ridge National Laboratory

Prepared for: HDF5 Users Group Meeting HUG2021 10/14/2021
Contents

• Background: Data at ORNL neutron science facilities, SNS/HFIR

• Challenges: bottlenecks, implementations, sustainability

• Proposed long-term solution:
  – Type-safe, threaded API on top of HDF5 using modern C++

• Future?
Background

ORNL neutron facilities, SNS and HFIR, fill us with interesting data

www.neutrons.ornl.gov
Event-based Raw Neutron Data

- Saved to HDF5 files using the standard NeXus schema [https://www.nexusformat.org/] capturing metadata annotations required for each instrument. (2,000 ~ 3,000 entries ....or more)

- < 5M events /s /instrument ~ 60 MB/s/instrument of raw data on the stream. Stored for 3 years at [https://analysis.sns.gov/] 1.2 TB/day, Grand Total of 1.6 PB as of 2020. Single Intel Xeon “nodes” for processing.

- Mantid [https://github.com/mantidproject/mantid] processes raw-event data into in-memory “workspaces” using generic loader used by several instrument data reduction workflows. Used across several neutron facilities.
NeXus

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Entry Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>group</td>
<td>/entry</td>
</tr>
<tr>
<td>attribute</td>
<td>/entry/NX_class</td>
</tr>
<tr>
<td>group</td>
<td>/entry/DASlogs</td>
</tr>
<tr>
<td>attribute</td>
<td>/entry/DASlogs/NX_class (NXlogs)</td>
</tr>
<tr>
<td>group</td>
<td>/entry/DASlogs/BL6:CS:DataType</td>
</tr>
<tr>
<td>attribute</td>
<td>/entry/DASlogs/BL6:CS:DataType/NX_class</td>
</tr>
<tr>
<td>dataset</td>
<td>/entry/DASlogs/BL6:CS:DataType/average_value</td>
</tr>
<tr>
<td>dataset</td>
<td>/entry/DASlogs/BL6:CS:DataType/average_value_error</td>
</tr>
<tr>
<td>group</td>
<td>/entry/bunk5_events</td>
</tr>
<tr>
<td>attribute</td>
<td>/entry/bunk5_events/NX_class (NXevent_data)</td>
</tr>
<tr>
<td>dataset</td>
<td>/entry/bunk5_events/event_id</td>
</tr>
<tr>
<td>dataset</td>
<td>/entry/bunk5_events/event_index</td>
</tr>
</tbody>
</table>

Challenges

Several metadata indexing, data, memory challenges were identified on Mantid:

• Currently several I/O “glue-layers” to HDF5 including the defunct NeXus API library: [https://github.com/nexusformat/code](https://github.com/nexusformat/code)

• Inefficient data access, current APIs on top of HDF5 not designed with performance in mind balancing computation, memory, I/O → appropriate “in-memory” index for processing, memory hogs for indexing

• Threading opens several HDF5 descriptors (1 per thread) and locks I/O operations

• Single files are becoming “too large”…multiple files API? Few MB to 100 GB

Short term improvements on Mantid Loader

- Introduced a new in-memory indexing methodology. Facility Time == $$$$
Proposed long-term solution

• No Cost I/O: NCIO (sorry for the pretentious name)
  Exploratory work: https://github.com/ORNL/ncio

• Domain specific API with the right level of abstraction on top of HDF5 (without doing a DSL approach):
  – NCIO: NeXus entry, bankID, histogram, log, instrument

• Different API levels:
  – Low-level “performance” API: pointers, deferred/lazy evaluation, key/value options, threaded? for backends
  – High-level: workflows on top of low-level APIs, bindings for end-users
NCIO Pluggable Architecture

- NCIO should leverage HDF5 API features: VOL, compression, chunking

### Consumers
- Data Reduction Workflows, Mantid

### IO Layers
- Reduced Quantities of Interests
- Data Descriptor $\rightarrow$ NeXus

### Domain-Specific Layers
- Virtual Transport Backend

### File System
- HDF5
- PHDF5
- HDF5 VOL?
NCIO Multithreaded API

- Type-safe: as close as possible to a domain of science semantics
- Take advantage of modern C++ (auto, threads)
- Thread-safe and truly-threaded (if/when backends allow)

Single instruction multiple data (SIMD), C++ std::thread

Task-based parallelism C++
std::async: lazy evaluation

Thread 0
Thread 1
Thread 2

NCIO
NCIO
NCIO

Memory ops
Memory ops
Memory ops

File ops
File ops
File ops

Compute
NCIO

Memory ops
File ops
Type-safe using C++ templates, thread-safe API

- Concurrent I/O, compute API

```cpp
ncio::DataDescriptor fr = ncio.Open("data_async.h5", ncio::OpenMode::read);

// Get is type-safe lazy evaluation, ref and pointer based
fr.Get<ncio::schema::nexus::entry::bank1_events::total_counts>(totalCounts);
fr.Get<ncio::schema::nexus::entry::bank1_events::event_index>(eventIndex.data(), ncio::BoxAll);
// automatic reallocation when executing
std::vector<double> eventTimeZero;
fr.Get<ncio::schema::nexus::entry::bank2_events::event_time_zero>(eventTimeZero, ncio::BoxAll);

// HDF5 action happens in the background
std::future future = fr.ExecuteAsync(std::launch::async);
do_some_interesting_compute(); // overlap compute + I/O
future.get(); // data is available
fr.Close();
```

Task-based parallelism
C++ std::async

```
Thread 0
Thread 1
Compute
NCIO
```

Memory ops
File ops
Type-safe using C++ templates, thread-safe API

- SIMD thread API (always pre-allocate memory)

```cpp
// any callable
auto lf_ReadChunkThread = [](...){
  // start, count = f(threadID);
  fw.Get<ncio::schema::nexus::entry::bank1_events::event_time_offset>(
      &eventTimeOffset[start], {{start}, {count}}, threadID);
};

// thread-safe handler
ncio::DataDescriptor fr = ncio.Open("data_threads.h5",
    ncio::OpenMode::read);

// C++11 threads or OpenMP
std::vector<std::thread> threads;
threads.reserve(nThreads);

// launch thread task
for (auto threadID = 0; threadID < nThreads; ++threadID)
  threads.emplace_back(lf_ReadChunkThread, threadID,
    nThreads, std::ref(eventTimeOffset),
    std::ref(totalCounts),
    std::ref(fr));

for (auto &thread : threads) thread.join();
// data is available
fr.Close();
```
“Appropriate” type-safe in-memory index API

- Your favorite IDE would pick up these types (in case the user forgets)...

```cpp
ncio::DataDescriptor fr = ncio::Open(fileName, ncio::OpenMode::read);

const auto nxClassIndex =
    fr.GetMetadata<ncio::schema::nexus::index::model1,
    ncio::schema::nexus::model1_t>();
```

```
enum class index
{
    model0,
    model1,
    model2,
    model3,
};
```

```
using model0_t = std::set<std::string>;
using model1_t = std::map<std::string, std::set<std::string>>;
using model2_t = std::map<std::string, std::string>;
using model3_t = std::unordered_map<std::string, std::unordered_set<std::string>>;
```

“Appropriate” type-safe data entry API

- Your favorite IDE would pick up these types (in case the user forgets)…
- Catch errors way before runtime (or before compile time if using IDEs)
- Possible with C++17 auto template deduction (maps hierarchical entries nicely)

```cpp
fr.Get<ncio::schema::nexus::entry::bank1_events::event_index>(
    eventIndex, ncio::BoxAll);

fr.Get<ncio::schema::nexus::entry::bank2_events::event_time_zero>(
    eventTimeZero, ncio::BoxAll);

fr.Get<ncio::schema::nexus::entry::

fr.Execute();
fr.Close();

CHECK_EQ(totalCounts, 10);
CHECK_EQ(eventIndex, std::vector<std::string>);
CHECK_EQ(eventTimeZero,
    std::vector<double>{0.016666}[
```
Summary

• Tackling (array-based) data as well as “in-memory” metadata index access is essential for reduction workflows at ORNL neutron science facilities SNS/HFIR.

• Current data access implementations on top of HDF5 serve specific purposes and they map 1-to-1 to HDF5 API calls.

• We present a “extendable” thread-safe (concurrent and SIMD), type-safe, lazy API on top of HDF5 using modern C++ features (template auto, std::thread, std::async).

• [https://github.com/ORNL/ncio](https://github.com/ORNL/ncio) (still exploratory, but running nightly regression with actual NeXus HDF5 data)
Future?

• More data is being produced that won’t fit in memory: [https://neutrons.ornl.gov/sts](https://neutrons.ornl.gov/sts) Second Target Station

• Might need current high-performance computing (HPC), MPI, parallel file system, NVRAM, etc.

• Extension to high-level languages (Python, Julia, R) for the end-user have its own challenges:
  
  • “Just-in-time” type-safety
  • Python’s GIL, Global Interpreter Lock

• Some of these ideas need operational “patron” support… “quality software = large investment”
ACKNOWLEDGEMENT

Work at Oak Ridge National Laboratory was sponsored by the Division of Scientific User Facilities, Office of Basic Energy Sciences, US Department of Energy, under Contract no. DE-AC05-00OR22725 with UT-Battelle, LLC

Thanks to the HUG2021 organization, The HDF5 Group and HDF5 stakeholders

Thanks to the audience