Portable scientific data formats are vital for scientific computing

A requirement for:

- Complex Workflows
- Reliable Data Storage
- Knowledge transfer
- Long-term maintainability & reproducibility

- The official HDF5 library is versatile and well supported, but it only provides a low-level C/C++ interface.
- Several C++ wrapper libraries exist, but are mostly domain-specific or incomplete.
I/O is an essential part of Neuroscience

HDF5 is critical to the Blue Brain Project. We require storing millions of neuron morphologies alongside their physiological properties, connections, and other data:

As our codebase is mostly written in >=C++11, we found the need for a suitable API for HDF5 in C++.
A Modern C++11 Wrapper

- Project started 7 years ago
- Active community
- 16 Releases (9 official). Latest: v2.4.1

Programmer Friendly

- Header-only library
- API enables concise code and provides sensible defaults

Wide Compatibility

- Cross platform: Windows, Linux, Mac
- Very few requirements: C++11, hdf5-1.8
- Supports Eigen, Boost and more

Stability & Performance

- Used in production at BBP
- Good test coverage, multiple scenarios
- Low overhead
HighFive: Looking under the hood
HighFive > RAII and resource management

HighFive utilizes RAII to handle object life-times and automatically manages reference counting on HDF5 objects from the C library.

The following example uses HighFive datatypes to create and open a dataset “/a/b” and fill it with four integers. The scope releases any associated resources:

```cpp
using namespace HighFive;
...
{
    File file("foo.h5", File::ReadWrite | File::Create);
    DataSet dataset = file.createDataSet("/a/b", std::vector<int>{1,2,3,4});
}
```
HighFive utilizes RAII to handle object life-times and automatically manages reference counting on HDF5 objects from the C library.

The following example uses HighFive datatypes to create and open a dataset “/a/b” and fill it with four integers. The scope releases any associated resources.

```cpp
using namespace HighFive;
...
{
    File file("foo.h5", File::ReadWrite | File::Create);
    DataSet dataset = file.createDataSet("/a/b", std::vector<int>{1, 2, 3, 4});
}
```

Internally, HighFive transparently manages the creation of the Group, DataSpace, Attributes, and more in HDF5.
HighFive > Type Conversion / Induction

The library uses C++ templating for automatic type mapping, even of non-contiguous types. This increases programmer productivity while reducing coding bugs:

Example with STL Container

```cpp
... 
std::vector<std::vector<double>> d2 = make_matrix();
file.createDataSet("/group/d2", d2);
...
```

Non-contiguous type conversion for read / write, and primitive types
In addition to the support for standard types (e.g., `std::vector`, `std::map`, ...), HighFive supports types from Boost, Eigen, XTensor, and others. Here is another example:

**Example with Boost**

```cpp
typename boost::numeric::ublas::matrix<double>;
boost::multi_array<double, 2> d2(boost::extents[5][3]);
file.createDataSet("/group/d2", d2);
```

**Example with Eigen**

```cpp
Eigen::MatrixXd d2 = Eigen::MatrixXd::Random(5, 3);
file.createDataSet("/group/d2", d2);
```

**Example with Boost uBLAS**

```cpp
using UBlasMatrix = typename boost::numeric::ublas::matrix<double>;
boost::numeric::ublas::matrix<double>
file.createDataSet("/group/d2", UBlasMatrix(5, 3));
```
HighFive > Example

With HighFive, we can easily create a source code example that illustrates the creation of an HDF5 file with:

1. A dataset with a vector of integers that has an attribute for the units.
2. A dataset with 2D matrix based on a non-contiguous datatype.

The example on the right also shows how to read back one of the datasets.

```cpp
using namespace HighFive;

...  
File file("tmp.h5", File::ReadWrite | File::Create);

// Create DataSet and write data (short form)
file.createDataSet("/group/d1",
   std::vector<int>{1,2,3,4,5});

// Attribute supported
file.createAttribute("/group/d1/units",
   std::string("cm/s"));

// Nested STL containers
std::vector<std::vector<double>> d2 = make_matrix();
file.createDataSet("/group/d2", d2);

// Reading
std::vector<int> d1_read;
file.getDataSet("/group/d1").read(d1_read);

...
HighFive > Advanced Features

HighFive is built with scientific applications in mind. The library supports advanced features that eases the development of complex C++ applications, while maintaining the source code readability. These are some of the most relevant:

- pHDF5 Support
- Chunking & Compression
- Native HDF5 Interaction
HighFive is built with scientific applications in mind. The library supports advanced features that ease the development of complex C++ applications while maintaining source code readability. These are some of the most relevant:

1. **Chunking & Compression**
2. **Native HDF5 Interaction**
3. **pHDF5 Support**

The only requirement is to use the `MPIOFileDriver` in the file opening. No other special API calls are required.

```cpp
File file("parallel_highfive.h5",
        File::ReadWrite | File::Create | File::Truncate,
        MPIOFileDriver(MPI_COMM_WORLD, MPI_INFO_NULL));
```

...
HighFive is built with scientific applications in mind. The library supports advanced features that eases the development of complex C++ applications, while maintaining the source code readability. **These are some of the most relevant:**

- pHDF5 Support
- Chunking & Compression
- Native HDF5 Interaction

Blue Brain Project 13
HighFive is built with scientific applications in mind. The library supports advanced features that ease the development of complex C++ applications, while maintaining source code readability. These are some of the most relevant:

- Group properties can be set for compression, chunking, and much more.
- pHDF5 Support
- Native HDF5 Interaction
- Chunking & Compression

```cpp
DataSetCreateProps props;
props.add(Chunking(std::vector<hsize_t>({2, 2})));
props.add(Deflate(9));
file.createDataSet("/group/d2", d2, props);
...```

Blue Brain Project
HighFive is built with scientific applications in mind. The library supports advanced features that eases the development of complex C++ applications, while maintaining the source code readability. **These are some of the most relevant:**

- pHDF5 Support
- Chunking & Compression
- Native HDF5 Interaction

---

Blue Brain Project
HighFive > Advanced Features (Continuation)

HighFive gives access to the native types of HDF5 and allows the user to call non-supported functionality.

These are some of the most relevant:

... File file("myfile.h5");
std::cout << H5Fget_freespace(file.getId()) << std::endl;
...

Native HDF5 Interaction
HighFive’s performance overhead compared to HDF5 code in C

- Naively writing row-by-row performs ~15 times slower.
- Fastest hand-written code took profiling and careful optimization and is substantially longer than the HighFive code (28 lines vs. 2 lines).
Challenges

Despite the longevity of the project, we are still working on several challenges:

- **Multi-threading within HDF5**
  - Multi-threaded I/O is funneled either by the library or MPI user. Fully parallel read-access would be a *really* useful feature to have.

- **# of datasets or groups scalings**
  - Inserting $O(1e6)$ of groups into a single HDF5 container on spinning disks gives notable latency of group retrieval, slow to construct such large files.

- **Support for mapping user defined, deeply nested, compound data types easily**
Thanks

Public repo: https://github.com/BlueBrain/HighFive
More information: https://go.epfl.ch/hi5

Thank you for listening

Questions?