Hierarchical Data Format Version 5 (HDF5)
Features, Tools, and Python Integration

Alvaro Aguilera
alvaro.aguilera@tu-dresden.de
`http://tu-dresden.de/zih`

Technische Universität Dresden

September 23, 2016

WHAT IS HDF?

"The Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of numerical data."

- stable, first versions in 1994
- open source (BSD-equivalent license)
- widespread in the scientific community
- written in C, but supported in many programming languages
- lots of tools and know-how available

WIKIPEDIA
The Free Encyclopedia

Website: `hdfgroup.org`

Why should I use HDF?

Many scientific projects progress like this:

- Early development stage
  - small amounts of test data produced (e. g. some KB of particle information, time-series, etc.)
  - data stored as text in ASCII files (e. g. CSV)
  - files organized in different directories

```
+ group
|     |-- file 1
|     |-- file 2
|     |-- ...
|     |-- file N
|     |-- summary
```

Why should I use HDF?

Many scientific projects progress like this:

- Early development stage
  - small amounts of test data produced (e. g. some KB of particle information, time-series, etc.)
  - data stored as text in ASCII files (e. g. CSV)
  - files organized in different directories

```
+ group
|     |-- file 1
|     |-- file 2
|     |-- ...
|     |-- file N
|     |-- summary
```

  - **1st Problems:**
    - slow I/O
    - big files (1 digit requires 1 byte instead of 3.3 bits)

WHY SHOULD I USE HDF?

Many scientific projects progress like this:

- Early development stage
  - small amounts of test data produced (e. g. some KB of particle information, time-series, etc.)
  - data stored as text in ASCII files (e. g. CSV)
  - files organized in different directories

```
+ group
|     |-- file 1
|     |-- file 2
|     |-- ...
|     |-- file N
|     |-- summary
```

  - **1st Problems:**
    - slow I/O
    - big files (1 digit requires 1 byte instead of 3.3 bits)
  - **Solution:** binary format
    - faster I/O & small files
    - requires more programming and documentation effort

Why should I use HDF?

- Early production stage
  - larger runs with thousands of files and directories

Why should I use HDF?

- Early production stage
  - larger runs with thousands of files and directories
  - **2^{nd} Problem:** file systems don't like too many files
    - *very* slow management operations (copy, listing, etc.)
    - slow search across files
    - inodes-outage

Why should I use HDF?

- Early production stage
  - larger runs with thousands of files and directories
  - **2nd Problem:** file systems don't like too many files
    - *very* slow management operations (copy, listing, etc.)
    - slow search across files
    - inodes-outage
  - **Solution:**
    - consolidation of small files into bigger ones

```
+ group 1
|     |-- objects
|     |-- summary
+ group 2
|     |-- objects
|     |-- summary
...
```

    - compressed collections of groups

```
group001-100.tar.gz
group101-200.tar.gz
...
```

WHY SHOULD I USE HDF?

- Production stage, it works but:
  - implementation was time consuming
  - searching the data is still slow and tedious
  - usually poorly documented
  - produced data is difficult to use by someone else

Why should I use HDF?

- Production stage, it works but:
  - implementation was time consuming
  - searching the data is still slow and tedious
  - usually poorly documented
  - produced data is difficult to use by someone else
- **Improvement:** use an I/O-library like HDF5!

File Organization

HDF5 files are containers for **objects**. An object can be either a **group** or a **dataset**.

A group is a collection of objects.

A dataset is a multidimensional array of data elements.

Attribute lists can be associated to any HDF5 object.

Groups define a hierarchical **path** similar to a directory structure:

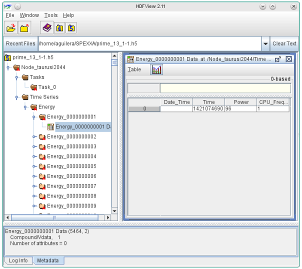
`/`	root group
`/foo`	object `foo` in group root
`/foo/bar`	object `bar` in group `foo` in group root

EXPLORING AN HDF5 FILE WITH *HDFView*

Graphical tool to browse HDF4 & HDF5 files.

Several useful features like:

- File creation
- Add, edit and modify file data
- Integrated plotting capabilities
- Available for most computer platforms (Java)



To start the application open a terminal and use the command: `hdfview`

Other Useful Tools

- **h5dump** – Dumps the contents of an HDF5 file to an ASCII file.
- **h5ls** – Lists specified features in the file.
- **h5diff** – Compares two files and report the differences.
- **h5repack** – Copies a file to a new file with different options.
- **h5repart** – Repartitions a file, creating a family of files.
- **h5copy** – Copies objects from a file to a new file
- **h5mkgrp** – Creates a new group in a file
- **h5stat** – Reports statistics regarding a file and the objects in it.

HDF5 for Python

- Available using the **h5py** library (http://h5py.org).
- Also distributed under BSD-like license.
- Used in this tutorial for simplicity.
- API follows to the traditional C interface.

CREATING A HDF5 FILE

```
file = h5py.File(name, mode='a', driver=None, libver=None,
userblock_size, **keywords)
```

Parameters:

- **name:** file name or h5f.FileID
- **mode:**

`r`	Read-only, file must exist
`r+`	Read/write, file must exist
`w`	Create file, truncate if exists
`w-` or `x`	Create file, fail if exists
`a`	Read/write if exists, create otherwise (default)

- **driver:** desired driver
- **libver:** compatibility level
- **userblock_size:** Size in bytes of the user block (0 or power of 2 $\geqslant$ 512)
- **keywords:** options for the driver

HDF5 File Drivers

```
file = h5py.File(name, mode='a', driver=None, libver=None,
userblock_size, **keywords)
```

Common drivers:

- **None:** Default and recommended. Usually unbuffered POSIX-I/O.
- **core:** In-memory file
  - *backing_store:* `False`, contents are discarded when closing file. `True`, contents are saved to disk when closing file.
  - *block_size:* increment (in bytes) by which memory is extended (64KiB default)

File Object

```python
import h5py

f = h5py.File("test.h5", "w")

f.close()
```

Interesting object members:

- **close()** file
- **flush()** buffers to disk
- **create_group(**`name`**)**
- **create_dataset(**`name, options`**)**

HDF5 Groups

group = File.create_group(*name*)

group = Group.create_group(*name*)

File object acts as the root group.

```python
import h5py

f = h5py.File("example0.h5", "w")

group1 = f.create_group("group1")
subgroup1 = group1.create_group("subgroup1")
subgroup2 = f.create_group("/group2/subgroup2")
group2 = f["/group2"]

f.close()
```

HDF5 Datasets

Homogeneous collections of data elements of immutable datatype.

```
dataset = Group.create_dataset(name, shape, dtype, data,
**kwds)
```

- **name:** Name of the dataset
- **shape:** Dimensions of the array with data (tuple)
- **dtype:** Data type (optional, defaults to float)
- **data:** NumPy array with initial data (optional)
- **kwds:** Additional parameters (optional)
  - **chunks:** chunking shape
  - **compression:** enabling dataset compression
  - **maxshape:** maximun dimension for resizing
  - **fillvalue:** default value when reading uninitialized data
  - **etc.**

DATA TYPES

Prefixed with `h5py.h5t.`

- Floating point: IEEE_F32LE, IEEE_F32BE, IEEE_F64LE, IEEE_F64BE
- Integer: STD_I8LE, STD_I16LE, STD_I32LE, STD_I64LE. STD_U8LE, ... (also with BE)
- Strings: C_S1 (Null-terminated fixed-length string), FORTRAN_S1 (Zero-padded fixed-length string), VARIABLE (Variable-length string)
- etc.

Creating a dataset

```
import h5py
#
# Create a new file using defaut properties.
#
file = h5py.File('example1.h5', 'w')
#
# Create a dataset under the Root group.
#
dataset = file.create_dataset("dset", (4, 6), h5py.h5t.STD_I32BE)
print("Dataset dataspace is", dataset.shape)
print("Dataset datatype is", dataset.dtype)
print("Dataset name is", dataset.name)
print("Dataset is a member of the group", dataset.parent)
print("Dataset was created in the file", dataset.file)
#
# Close the file before exiting
#
file.close()
```

WRITTING TO A DATASET

```python
import h5py
import numpy as np

# Open an existing file using default properties.
file = h5py.File('example1.h5', 'r+')

# Open "dset" dataset under the root group.
dataset = file['/dset']
# Initialize data object with 0.
data = np.zeros((4,6))

# Assign new values
for i in range(4):
    for j in range(6):
        data[i][j]= i*6+j+1

# Write data
print("Writing data...")
dataset[...] = data

# Read data back and print it.
print("Reading data back...")
data_read = dataset[...]
print("Printing data...")
print(data_read)

# Close the file before exiting
file.close()
```

Chunking

By default HDF5 creates **contiguous** datasets.

Chunking divides the contiguous data into **chunks** that are irregularly stored on disk and indexed using a B-tree.

Example in which the data will be read and written in blocks with shape (100,100):

```
dset = f.create_dataset("chunked", (1000, 1000),
chunks=(100, 100))
```

Automatic chunking can be enabled using `chunks=True`.
The recommended size for the chunk are between 10 KiB and 1 MiB.

CREATING COMPRESSED FILE

```python
import h5py
import numpy as np

# Create hdf file.
file = h5py.File('example2.h5','w')

# Create /DS1 dataset; in order to use compression, dataset has to be chunked.
dataset = file.create_dataset('DS1', (32,64), 'i', chunks=(4,8), compression='gzip',
       compression_opts=9)

# Initialize data.
data = np.zeros((32,64))
for i in range(32):
    for j in range(64):
        data[i][j]= i*j-j

# Write data.
print("Writing data...")
dataset[...] = data

file.close()

# Read data back; display compression properties and dataset max value.
file = h5py.File('example2.h5','r')
dataset = file['DS1']
print("Compression method is", dataset.compression)
print("Compression parameter is", dataset.compression_opts)
data = dataset[...]
file.close()
```

CREATE ATTRIBUTES

HDF5 is a self-describing format.

This is achieved through attributes (metadata) attached to the groups and datasets.

```
import h5py
import numpy as np
#
# Open an existing file using defaut properties.
#
file = h5py.File('example1.h5', 'r+')
#
# Open "dset" dataset.
#
dataset = file['/dset']
#
# Create string attribute.
#
attr_string = "Meter per second"
dataset.attrs["Units"] = attr_string
#
# Close the file before exiting
#
file.close()
```

Practical Session

- Create an HDF5 file with a 100x100 integer dataset inside the group "experiment1"
- Initialize each element to the sum of its coordinates.
- Measure the execution time for the default and core driver (with and without storing on close)
- Turn on compression and compare time and resulting size using different chunking shapes.

Sources and References

[1] The HDF Group
HDF5 Tutorial
https://www.hdfgroup.org/HDF5/Tutor

[2] National Energy Research Scientific Computing Center (NERSC)
Introduction to Scientific I/O
http://www.nersc.gov/users/training/online-tutorials/introduction-to-scientific-i-o

[3] Andrew Collette et al.
HDF5 for Python
http://h5py.org

THANK YOU FOR YOUR ATTENTION

More information: `http://www.sppexa.de`

This work is supported by the German Research Foundation (DFG)