I/O Libraries and HDF5

CPS343

Parallel and High Performance Computing

Spring 2020

Outline

1. Introduction: I/O libraries
   - Why we need I/O libraries
   - The Hierarchical Data Format
   - Other file formats

2. HDF concepts and basic usage
   - HDF data model, file structure, and library API
   - HDF programming model
   - HDF tools
   - Use examples

Outline

I/O challenges

I/O presents two important challenges to scientific computing:

1. Performance — I/O is much more time-consuming than computation.
2. Portability — different kinds of computers can have different ways of representing real (floating point) numbers.

Storage formats

When you use a `PRINT` statement in Fortran or a `printf()` in C or output to `cout` in C++, you are asking the program to output data in human-readable form:

```
#include <math.h>
float x = M_PI;
printf("%f\n", x);
```

But what if the value that you want to output is a real number with lots of significant digits?

```
3.14159265358979323846264338327950288419716939937510...
```

Data output as text

When you output data as text, each character takes 1 byte. So if you output a number with lots of digits, then you're outputting lots of bytes.

For example, it takes 13 bytes to output

```
1.3456789E+23
```

as text.

This assumes we're using ASCII (American Standard Code for Information Interchange); the requirements can be higher for multilingual character sets.

Output data in binary

Inside the computer, a single precision real number (Fortran `REAL`, C/C++ `float`) typically requires 4 bytes, and a double precision number (`DOUBLE PRECISION` or `double`) typically requires 8.

That's less than 13.

Since I/O is expensive, it's better to output 4 or 8 bytes than 13 or more.

Happily programming languages allow one to output data as binary (internal representation) rather than as text.

Binary output problems

When you output data as **binary** rather than as text, you output substantially **fewer bytes**, so you save time (since I/O is expensive) and you save disk space.

But, you pay two prices:

Readability (Most) People can't read binary easily.

Portability Different kinds of computers have different ways of internally representing numbers.

Binary readability: no problem

Readability of binary data **isn't a problem** in scientific computing, because:

- You can always write a helper program to read in the binary data and display its text equivalent.
- If you have lots and lots of data then there is probably too much to read anyway, at least not without a program that read the data file and display small subsets of data.

Binary portability: big problem

Binary data portability is a **very big problem** in scientific computing, because data that's output on one kind of computer may not be readable on another, and so:

- You can't necessarily output the data on one kind of computer and then use them (for example, visualize, analyze) on another kind.
- Some day the kind of computer that output the data will be obsolete and there may be no computer in the world that can input it, and thus the data are lost. (Okay, that's a bit extreme, but at the very least it may not be convenient to read the data)

Portable binary data

To address this problem, the HPC community has developed a number of portable binary data formats.

Two of the most popular are:

- HDF (Hierarchical Data Format) from the National Center for Supercomputing Applications: http://www.hdfgroup.org
- NetCDF (Network Common Data Form) from Unidata: http://www.unidata.ucar.edu/software/netcdf

Advantages of portable I/O libraries

There are many obvious advantages to using a portable binary I/O package. For example, they:

- give you portable binary I/O;
- have simple, clear APIs;
- are available for free;
- run on most platforms;
- allow you to *annotate* your data (for example, put into the file the variable names, units, experiment name, grid description, etc).

Also, both HDF and netCDF support distributed parallel I/O.

Outline

What is HDF?

According to the HDF website (http://www.hdfgroup.org) HDF5 provides:

- A versatile data model that can represent very complex data objects and a wide variety of metadata.
- A completely portable file format with no limit on the number or size of data objects in the collection.
- A software library that runs on a range of computational platforms, from laptops to massively parallel systems, and implements a high-level APIs for C, C++, Fortran 90, Java, and Python, and many other languages
- A rich set of integrated performance features that allow for access time and storage space optimizations.
- Tools and applications for managing, manipulating, viewing, and analyzing the data in the collection.

Outline

netCDF

Another I/O library that is widely used by the HPC community is **netCDF** (Network Common Data Form).

netCDF provides many of the same capabilities as HDF5. It is widely used in atmospheric and meteorological applications.

Comparing netCDF and HDF:

- the netCDF API is a bit simpler than that of HDF5
- the native netCDF-3 filespace is flat rather than hierarchical
- netCDF-4 (current version) uses HDF5 as the underlying file format but also works with netCDF-3 files
- parallel I/O is supported through the underlying HDF5 layer or with a spin-off library **Parallel netCDF** (or **PnetCDF**) from Argonne National labs which implements netCDF functionality on top of MPI-IO

FITS

Although not directly related to HPC, it's interesting (but unsuprising) to note that standardized binary file formats have been around for some time.

FITS stands for Flexible Image Transport System and is mostly used in astronomical and other image processing applications for storing and especially for archiving data.

First standardized in 1981, it has been updated since then but ensures that backwards compatibility will always be maintained.

Data management plans

Finally, it's worth pointing out that in in this age of "big data," government funding organizations like NSF and NIH require all grantees to develop and implement a *data management plan*.

For example, the National Science Foundation (NSF) requires that

> **Grantees from all fields will develop and submit specific plans to share materials collected with NSF support, except where this is inappropriate or impossible.** *These plans should cover how and where these materials will be stored at reasonable cost, and how access will be provided to other researchers, generally at their cost.*

Outline

HDF organization

The Hierarchical Data Format (HDF) implements a model for managing and storing data. It includes

1. an abstract data model,
2. an abstract storage model (the data format),
3. and libraries to implement the abstract model and to map the storage model to different storage mechanisms.

The HDF5 library

- provides a programming interface to a concrete implementation of the abstract models,
- implements a model of data transfer, i.e., efficient movement of data from one stored representation to another stored representation.

HDF abstract data model

File a contiguous string of bytes in a computer store (memory, disk, etc.), and the bytes represent zero or more objects of the model

Group a collection of objects (including groups)

Dataset a multidimensional array of data elements with attributes and other metadata

Dataspace a description of the dimensions of a multidimensional array

Datatype a description of a specific class of data element including its storage layout as a pattern of bits

Attribute a named data value associated with a group, dataset, or named datatype

Property List a collection of parameters (some permanent and some transient) controlling options in the library

Link the way objects are connected

HDF abstract storage model

HDF5 objects and data are mapped to a *linear address space*, assumed to be a contiguous array of bytes stored on some random access medium.

The HDF5 File Format Specification is organized in three parts:

- Level 0 File signature and super block
- Level 1 File infrastructure:
  B-link trees and B-tree nodes, Group, Group entry, Local heaps, Global heap, Free-space index
- Level 2 Data object:
  Data object headers, Shared data object headers, Data object data storage

Key abstraction goal: efficiently map logically multidimensional rectangular arrays to efficient storage mechanisms

HDF file structure

As its name implies, files in HDF have a hierarchical structure, much like the Unix file system structure.

- Every file has a *root* group named "/".
- A group typically maintains pointers to other groups or to datasets.
- The *path* to `dset2` in the lower right of the diagram is `/group2/group2/dset2`

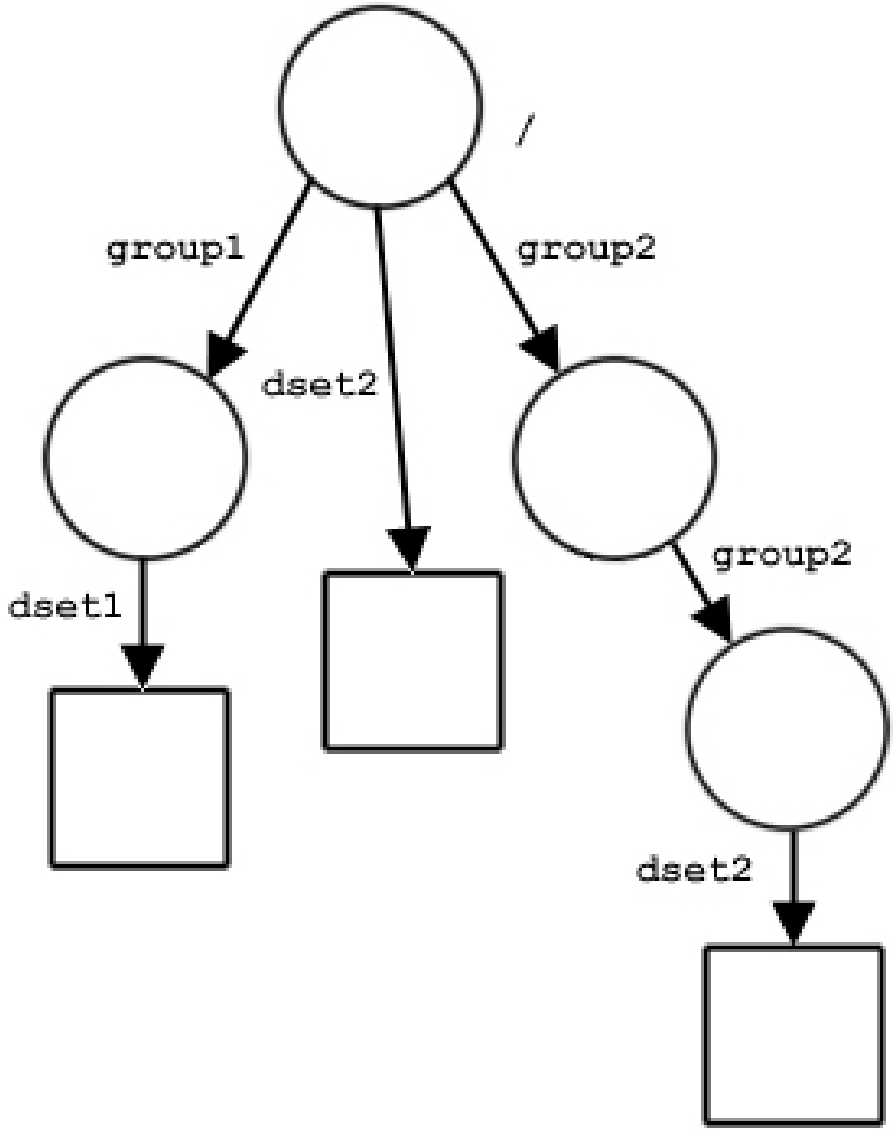


Image source: https://portal.hdfgroup.org/display/HDF5/HDF5+User%27s+Guide

HDF5 API naming scheme

Prefix	Operates on
H5A	Attributes
H5D	Datasets
H5E	Error reports
H5F	Files
H5G	Groups
H5I	Identifiers
H5L	Links
H5O	Objects
H5P	Property lists
H5R	References
H5S	Dataspaces
H5T	Datatypes
H5Z	Filters

Since the HDF language is implemented in C it does not provide object-oriented a namespace.

The naming convention, however, groups routines based on the type of data or structures they operate on.

For example, the function `H5Dcreate()` is used to create a new dataset, while `H5Gcreate()` can create a new group.

Outline

Creating an HDF file

The following code segment creates and then closes an HDF file. Here it is assumed that `fname` is a C character array containing a valid file name.

```
hid_t file;     /* declare file identifier */
/*
 * Create a new file (truncate and overwrite any file of
 * the same name) with default file creation properties
 * and default file access properties
 */
file = H5Fcreate(fname, H5F_ACC_TRUNC, H5P_DEFAULT,
                 H5P_DEFAULT);
/*
 * Close the file
 */
status = H5Fclose(file);
```

The parameter `H5F_ACC_EXCL` can be used instead of `H5F_ACC_TRUNC` to cause `H5Fcreate()` to fail if the file already exists.

Creating and initializing a dataset

The essential objects within a dataset are *datatype* and *dataspace*.

These are independent objects and are created separately from any dataset to which they may be attached. Hence, creating a dataset requires, at a minimum, the following steps:

1. Create and initialize a dataspace for the dataset
2. Define a datatype for the dataset if not using a predefined datatype
3. Create and initialize the dataset

Creating and initializing a dataset: 2D Array

```
hid_t dataset, datatype, dataspace;  /* identifiers */

/* Create dataspace for a fixed-size 2-D dataset. */

dimsf[0] = NX;
dimsf[1] = NY;
dataspace = H5Screate_simple(2, dimsf, NULL);

/* Define a datatype for the data in the dataset */

datatype = H5Tcopy(H5T_NATIVE_INT);
status = H5Tset_order(datatype, H5T_ORDER_LE);

/* Create a new dataset with default dataset
 * creation properties */

dataset = H5Dcreate(file, DATASETNAME, datatype, dataspace,
                    H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
```

Examples of predefined HDF datatypes

HDF provides several predefined datatypes. For example

Datatype	Description
`H5T_STD_I32LE`	Four-byte, little-endian, signed, two's complement int
`H5T_STD_U16BE`	Two-byte, big-endian, unsigned integer
`H5T_IEEE_F32BE`	Four-byte, big-endian, IEEE floating point
`H5T_IEEE_F64LE`	Eight-byte, little-endian, IEEE floating point
`H5T_C_S1`	One-byte, null-terminated string of eight-bit characters

Examples of predefined native HDF datatypes

HDF also supports native datatypes that correspond to datatypes available on the local platform.

Native Datatype	Corresponding C Type
`H5T_NATIVE_INT`	int
`H5T_NATIVE_FLOAT`	float
`H5T_NATIVE_CHAR`	char
`H5T_NATIVE_DOUBLE`	double
`H5T_NATIVE_LDOUBLE`	long double

Closing objects

When closing an HDF file, one should also close the associated dataset, datatype (if one was created) and dataspace.

This releases resources allocated for these objects.

```
H5Tclose(datatype);
H5Dclose(dataset);
H5Sclose(dataspace);
```

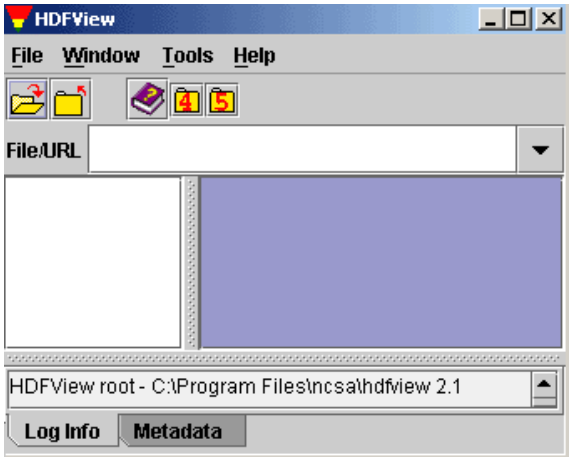
Outline

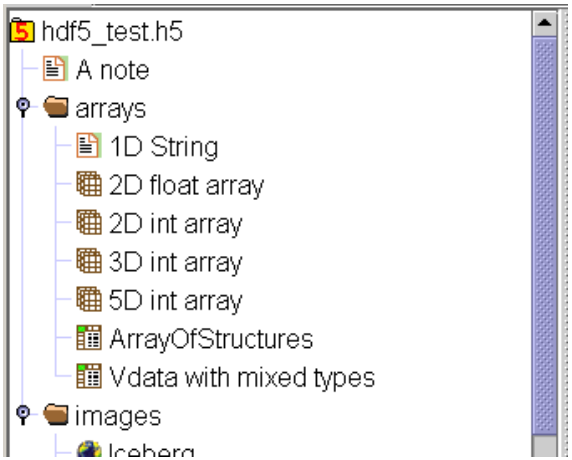
Some HDF command-line tools

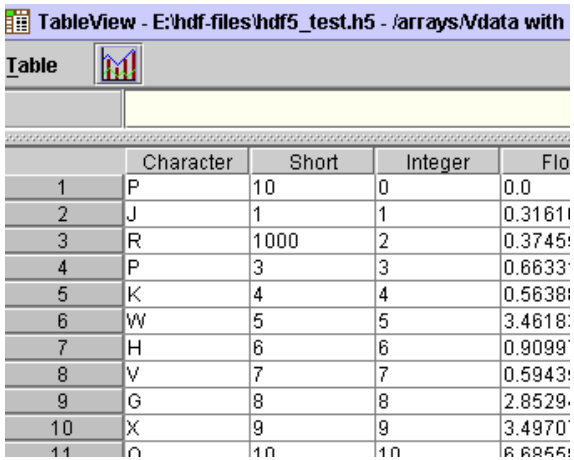
h5cc, h5c++	Compiler front-ends.
h5debug	Debugs an existing HDF5 file at a low level.
h5diff	Compares two HDF5 files and reports the differences.
h5dump	Enables the user to examine the contents of an HDF5 file and dump those contents to an ASCII file.
h5import	Imports ASCII or binary data into HDF5.
h5jam/h5unjam	Add/Remove text to/from User Block at the beginning of an HDF5 file.
h5ls	Lists selected information about file objects in the specified format.
h5perf	Measures Parallel HDF5 performance.
h5repack	Copies an HDF5 file to a new file with or without compression/chunking.
h5repart	Repartitions a file or family of files.

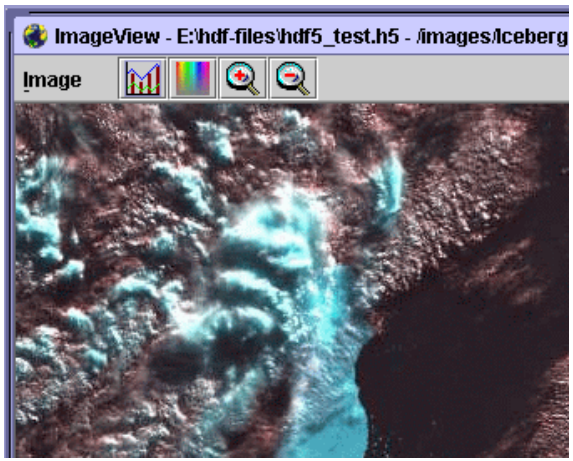
HDFView

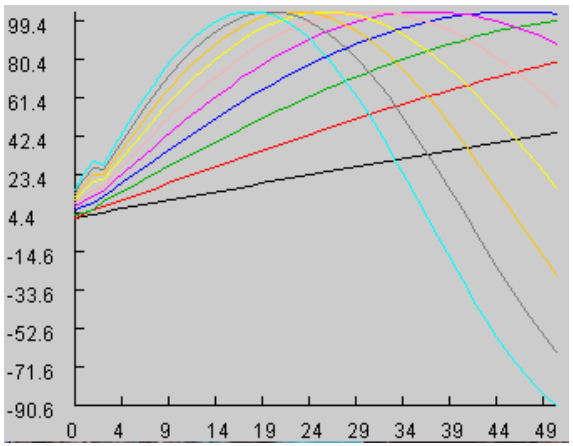
HDFView is an interactive program written in Java for viewing HDF files.











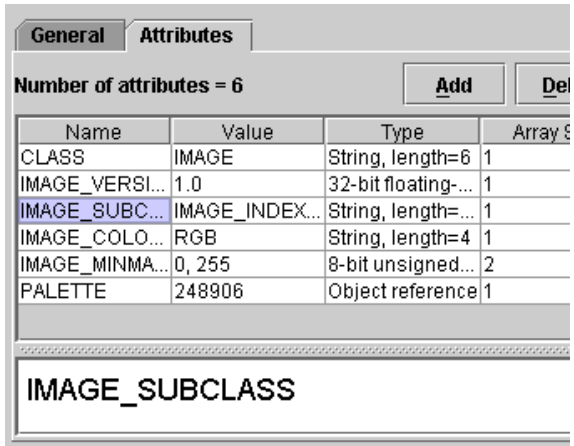


Image source: http://www.hdfgroup.org/hdf-java-html/hdfview/

Outline

Example: Storing a matrix (1)

```
/* Declare necessary variables */
const char* fname = "matrix.h5";
hid_t file_id, dataset_id, dataspace_id;
hsize_t dims[2];
int rows, cols, stride;
double* A;

/*
 * ...
 * Assign rows and cols and
 * create and fill matrix A here
 * ...
 */

/* Create HDF5 file.  Truncate if file already exists */
file_id = H5Fcreate(fname, H5F_ACC_TRUNC,
                    H5P_DEFAULT, H5P_DEFAULT);
```

Example: Storing a matrix (2)

```
/* Create the data space for dataset */
dims[0] = rows;
dims[1] = cols;
dataspace_id = H5Screate_simple(2, dims, NULL);

/* Create the dataset */
dataset_id = H5Dcreate(file_id, "/Matrix", H5T_IEEE_F64LE,
                       dataspace_id, H5P_DEFAULT,
                       H5P_DEFAULT, H5P_DEFAULT);

/* Write matrix data to file */
status = H5Dwrite(dataset_id, H5T_NATIVE_DOUBLE,
                  H5S_ALL, H5S_ALL, H5P_DEFAULT, A);

/* release resources and close file */
status = H5Dclose(dataset_id);
status = H5Sclose(dataspace_id);
status = H5Fclose(file_id);
```

Example: Reading a matrix (1)

```
/* Declare necessary variables */
const char* fname = "matrix.h5";
hid_t file_id, dataset_id, dataspace_id, file_dataspace_id;
hsize_t* dims;
hssize_t num_elem;
int rank;
int ndims;
int rows, cols, stride;
double* A;

/* Open existing HDF5 file */
file_id = H5Fopen(fname, H5F_ACC_RDONLY, H5P_DEFAULT);

/* Open existing dataset */
dataset_id = H5Dopen(file_id, "/Matrix", H5P_DEFAULT);
```

Example: Reading a matrix (2)

```
/* Determine dataset parameters */
file_dataspace_id = H5Dget_space(dataset_id);
rank = H5Sget_simple_extent_ndims(file_dataspace_id);
dims = (hsize_t*) malloc(rank * sizeof(hsize_t));
ndims = H5Sget_simple_extent_dims(file_dataspace_id, dims,
                                  NULL);
if (ndims != rank)
{
  fprintf(stderr, "Expected dataspace to be dimension ");
  fprintf(stderr, "%d but appears to be %d\n", rank, ndims);
}

/* Allocate matrix */
num_elem = H5Sget_simple_extent_npoints(file_dataspace_id);
A = (double*) malloc(num_elem * sizeof(double));
rows = dims[0];
cols = dims[1];
stride = cols;
```

Example: Reading a matrix (3)

```
/* Create dataspace */
dataspace_id = H5Screate_simple(rank, dims, NULL);
free(dims);

/* Read matrix data from file */
status = H5Dread(dataset_id, H5T_NATIVE_DOUBLE, dataspace_id
                 file_dataspace_id, H5P_DEFAULT, A);

/* Release resources and close file */
status = H5Dclose(dataset_id);
status = H5Sclose(dataspace_id);
status = H5Sclose(file_dataspace_id);
status = H5Fclose(file_id);

/*
 * Do something with matrix
 */

free(A);
```

Acknowledgements

Material used in creating these slides comes from

- The HDF5 User's Guide https://portal.hdfgroup.org/display/HDF5/HDF5+User%27s+Guide
- Various slide presentations:
  - http://www.oscer.ou.edu/ncsiworkshop2012intropar_sipe_scilibs_20120803.pdf
  - http://www.spscicomp.org/ScicomP12/Presentations/User/Yang.pdf
  - http://www.unidata.ucar.edu/software/netcdf/workshops/2007/hdf5/ncw07-hdf5.pdf