Introduction to HDF5 and Xdmf

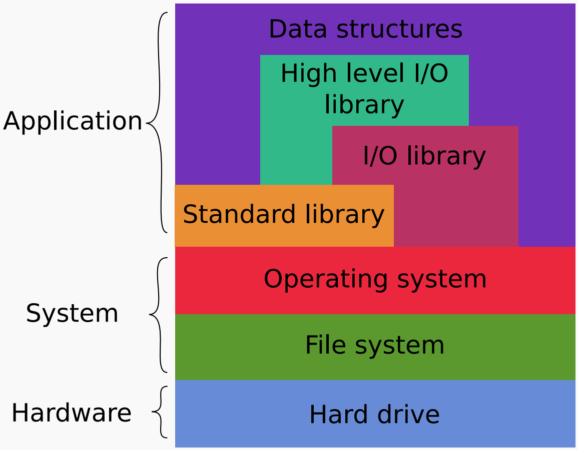
Matthieu Haefele

Saclay, November 2016,
ANF visualisation et données

Outline

- HDF5 in the context of Input/Output (IO)
- HDF5 Application Programming Interface (API)
- Playing with Dataspace
- Xdmf
- Hands on session

Hardware/Software stack



High level I/O libraries

The purpose of high level I/O libraries is to provide the developer a higher level of abstraction to manipulate computational modeling objects

- Meshes of various complexity (rectilinear, curvilinear, unstructured…)
- Discretized functions on such meshes
- Materials
- …

Until now, these libraries are mainly used in the context of visualization

Existing libraries

- Silo
  - Wide range of objects
  - Built on top of HDF5
  - “Native” format for VisIt
- Exodus
  - Focused on unstructured meshes and finite element representations
  - Built on top of NetCDF
- Famous/intensively used codes’ output format
- **eXtensible Data Model and Format (Xdmf)**

I/O libraries

Purpose of I/O libraries:

- Efficient I/O
- Portable binary files
- Higher level of abstraction for the developer

Two main existing libraries:

- Hierarchical Data Format: HDF5
- Network Common Data Form: NetCDF

HDF5 library

HDF5 file:

- HDF5 group: a grouping structure containing instances of zero or more groups or datasets
- HDF5 dataset: a multidimensional array of data elements

HDF5 dataset $\Leftrightarrow$ multidimensional array:

- Name
- Datatype (Atomic, NATIVE, Compound)
- Dataspace (rank, sizes, max sizes)
- Storage layout (contiguous, compact, chunked)

HDF5 High Level APIs

- **Dimension Scale** (H5DS): Enables to attach dataset dimension to scales
- **Lite** (H5LT): Enables to write simple dataset in one call
- **Image** (H5IM): Enables to write images in one call
- **Table** (H5TB): Hides the compound types needed for writing tables
- **Packet Table** (H5PT): Almost H5TB but without record insertion/deletion but supports variable length records
- …

HDF5 low level API

- **H5F**: File manipulation routines
- **H5G**: Group manipulation routines
- **H5S**: Dataspace manipulation routines
- **H5D**: Dataset manipulation routines
- ...

Just have a look at the outstanding on-line reference manual for HDF5 !

C order versus Fortran order

```c
/* C language */
#define NX 4
#define NY 3
int x,y;
int f[NY][NX];

for (y=0;y<NY;y++)
  for (x=0;x<NX;x++)
    f[y][x] = x+y;
```

```fortran
! Fortran language
integer, parameter :: NX=4
integer, parameter :: NY=3
integer                  :: x,y
integer, dimension(NX,NY) :: f

do y=1,NY
  do x=1,NX
    f(x,y) = (x-1) + (y-1)
  enddo
enddo
```

0	1	2	3	1	2	3	4	2	3	4	5

The memory mapping is identical, the language semantic is different !!

HDF5 first example

```
#define NX      5
#define NY      6
#define RANK    2

int main (void)
{
    hid_t       file, dataset, dataspace;
    hsize_t     dimsf[2];
    herr_t      status;
    int         data[NY][NX];

    init(data);
    file = H5Fcreate("example.h5", H5F_ACC_TRUNC, H5P_DEFAULT,\
                     H5P_DEFAULT);
    dimsf[0] = NY;
    dimsf[1] = NX;
```

HDF5 first example cont.

```
    dataspace = H5Screate_simple(RANK, dimsf, NULL);

    dataset = H5Dcreate(file, "IntArray", H5T_NATIVE_INT, \
        dataspace, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);

    status = H5Dwrite(dataset, H5T_NATIVE_INT, H5S_ALL, \
                      H5S_ALL,H5P_DEFAULT, data);

    H5Sclose(dataspace);
    H5Dclose(dataset);
    H5Fclose(file);

    return 0;
}
```

HDF5 high level example cont.

```
  status = H5LTmake_dataset_int(file, "IntArray", RANK, dimsf, data);

  H5Fclose(file);

  return 0;
}
```

Variable C type

```
hid_t    file, dataset, dataspace;
hsize_t  dimsf[2];
herr_t   status;
```

- hid_t: handler for any HDF5 objects (file, groups, dataset, dataspace, datatypes...)
- hsize_t: C type used for number of elements of a dataset (in each dimension)
- herr_t: C type used for getting error status of HDF5 functions

File creation

```
file = H5Fcreate("example.h5", H5F_ACC_TRUNC, H5P_DEFAULT,\
                 H5P_DEFAULT);
```

- "example.h5": file name
- H5F_ACC_TRUNC: File creation and suppress it if it exists already
- H5P_DEFAULT: file creation property list
- H5P_DEFAULT: file access property list (needed for MPI-IO)

Dataspace creation

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

- RANK: dataset dimensionality
- dimsf: size of the dataspace in each dimension
- NULL: specify max size of the dataset being fixed to the size

Dataset creation

```
dataset = H5Dcreate(file, "IntArray", H5T_NATIVE_INT, \
    dataspace, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
```

- file: HDF5 objects where to create the dataset. Should be a file or a group.
- "IntArray": dataset name
- H5T_NATIVE_INT: type of the data the dataset will contain
- dataspace: size of the dataset
- H5P_DEFAULT: default option for property list.

Datatype

- Predefined Datatypes: created by HDF5.
- Derived Datatypes: created or derived from the predefined data types.

There are two types of predefined datatypes:

- **STANDARD**: They defined standard ways of representing data. Ex: H5T_IEEE_F32BE means IEEE representation of 32 bit floating point number in big endian.
- **NATIVE**: Alias to standard data types according to the platform where the program is compiled. Ex: on an Intel based PC, H5T_NATIVE_INT is aliased to the standard predefined type, H5T_STD_32LE.

Datatype cont.

A data type can be:

- ATOMIC: cannot be decomposed into smaller data type units at the API level. Ex: integer
- COMPOSITE: An aggregation of one or more data types. Ex: compound data type, array, enumeration

Dataset writing

```
status = H5Dwrite(dataset, H5T_NATIVE_INT, H5S_ALL, \
                  H5S_ALL,H5P_DEFAULT, data);
```

- dataset: HDF5 objects representing the dataset to write
- H5T_NATIVE_INT: Type of the data in memory
- H5S_ALL: dataspace specifying the portion of memory that needs be read (in order to be written)
- H5S_ALL: dataspace specifying the portion of the file dataset that needs to be written
- H5P_DEFAULT: default option for property list (needed for MPI-IO).
- data: buffer containing the data to write

Closing HDF5 objects

```
H5Sclose(dataspace);
H5Dclose(dataset);
H5Fclose(file);
```

Opened/created HDF5 objects are closed.

Some comments

```
   status = H5LTmake_dataset_int(file, "IntArray", RANK, dimsf, data);

   H5Fclose(file);

   return 0;
}
```

This example is almost a **fwrite**, but:

- The generated file is portable
- The generated file can be accessed with HDF5 tools
- Attributes can be added on datasets or groups
- The type of the data can be fixed
- The storage layout can be modified
- Portion of the dataset can be written
- ...

Concept of start, stride, count block

Considering a *n*-dimensional array, start, stride, count and block are arrays of size *n* that describe a subset of the original array

- **start**: Starting location for the hyperslab (default 0)
- **stride**: The number of elements to separate each element or block to be selected (default 1)
- **count**: The number of elements or blocks to select along each dimension
- **block**: The size of the block (default 1)

Conventions for the examples

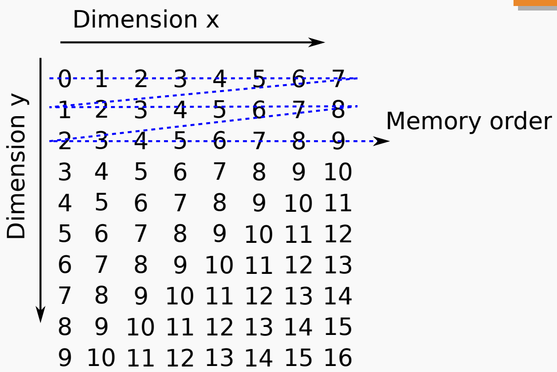
We consider:

- A 2D array $f[N_y][N_x]$ with $N_x = 8, N_y = 10$
- Dimension x is the dimension contiguous in memory
- Graphically, the x dimension is represented horizontal
- Language C convention is used for indexing the dimensions

$\Rightarrow$ Dimension y is index=0

$\Rightarrow$ Dimension x is index=1

Graphical representation




```
int start[2], stride[2], count[2], block[2];
 start[0] = 0;   start[1] = 0;
stride[0] = 1;  stride[1] = 1;
 block[0] = 1;   block[1] = 1;
```

Illustration for count parameter

Dimension x →

Dimension y ↓

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

y=0 y=1 y=2

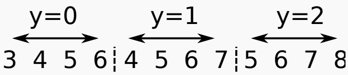
0 1 2 3 ¦ 1 2 3 4 ¦ 2 3 4 5

```
count[0] = 3;  count[1] = 4;
```

Illustration for start parameter

Dimension x (across) / Dimension y (down)

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16




```
start[0] = 1;  start[1] = 2;
count[0] = 3;  count[1] = 4;
```

Illustration for stride parameter

Dimension x

Dimension y

0	1	2	3	4	5	6	7
1	2	**3**	**4**	**5**	**6**	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	**6**	**7**	**8**	**9**	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	**9**	**10**	**11**	**12**	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

y=0 y=1 y=2

3 4 5 6 ¦ 6 7 8 9 ¦ 9 10 11 12

```
start[0] = 1;   start[1] = 2;
count[0] = 3;   count[1] = 4;
stride[0] = 3;  stride[1] = 1;
```

Illustration for stride parameter

Dimension x →

Dimension y ↓

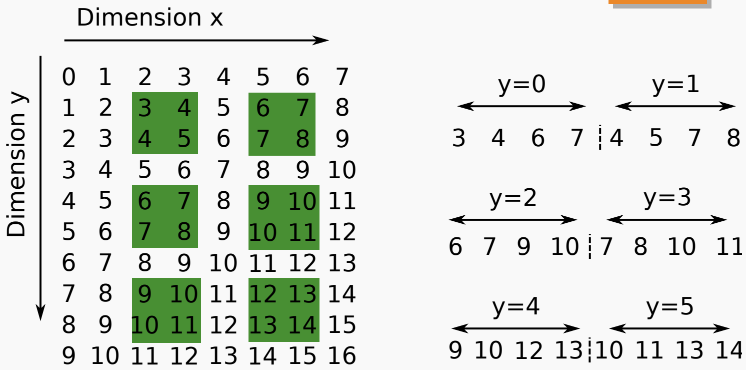
0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

y=0 y=1 y=2

3 6 ¦ 6 9 ¦ 9 12

```
 start[0] = 1;  start[1] = 2;
 count[0] = 3;  count[1] = 2;
stride[0] = 3; stride[1] = 3;
```

Illustration for block parameter




```
start[0] = 1;   start[1] = 2;
count[0] = 3;   count[1] = 2;
stride[0] = 3;  stride[1] = 3;
block[0] = 2;   block[1] = 2;
```

Exercise 1

Please draw the elements selected by the start, stride, count, block set below

Dimension x →

Dimension y ↓

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

```
start[0] = 2;   start[1] = 1;
count[0] = 6;   count[1] = 4;
```

Solution 1

Dimension x

Dimension y

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

```
start[0] = 2;  start[1] = 1;
count[0] = 6;  count[1] = 4;
```

Exercise 2

Please draw the elements selected by the start, stride, count, block set below

Dimension x

Dimension y

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

```
start[0] = 2;  start[1] = 1;
count[0] = 1;  count[1] = 1;
block[0] = 6;  block[1] = 4;
```

Solution 2

Dimension x

Dimension y

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

```
start[0] = 2;   start[1] = 1;
count[0] = 1;   count[1] = 1;
block[0] = 6;   block[1] = 4;
```

Exercise 3

Please draw the elements selected by the start, stride, count, block set below

Dimension x →

Dimension y ↓

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

```
 start[0] = 2;  start[1] = 1;
 count[0] = 3;  count[1] = 2;
stride[0] = 2; stride[1] = 2;
 block[0] = 2;  block[1] = 2;
```

Solution 3

Dimension x →

Dimension y ↓

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

```
 start[0] = 2;   start[1] = 1;
 count[0] = 3;   count[1] = 2;
stride[0] = 2;  stride[1] = 2;
 block[0] = 2;   block[1] = 2;
```

What is a dataspace ?

Dataspace Objects

- Null dataspaces
- Scalar dataspaces
- Simple dataspaces
  - rank or number of dimensions
  - current size
  - maximum size (can be unlimited)

Dataspaces come into play:

- for performing partial IO
- to describe the shape of HDF5 dataset

What is a dataspace for ?

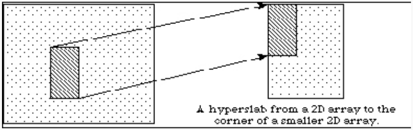


Figure : Access a sub-set of data with a hyperslab[1]

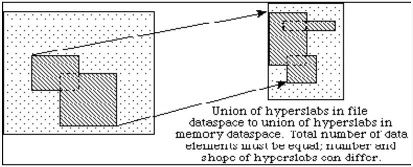


Figure : Build complex regions with hyperslab unions[1]

[1] Figures taken from HDF5 website

What is a dataspace for ?

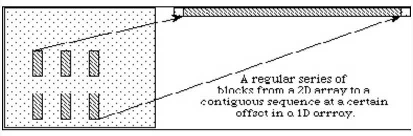


Figure : Use hyper-slabs to gather or scatter data[2]

[2] Figures taken from HDF5 website

Typical usage

Simple dataspace

- Created with *H5Screate_simple*
- Used to define the shape of a dataset during its creation
- Modified by *H5Sselect_hyperslab* to define a selection

Selection

- Created by modifying a simple dataspace thanks to *H5Sselect_hyperslab* to define a selection / portion of memory / dataset
- Used in *H5Dwrite* to perform partial IOs
- Do not use them for dataset creation

How to play with dataspaces

```
hid_t space_id;
hsize_t dims[2], start[2], count[2];
hsize_t *stride=NULL, *block=NULL;

dims[0] = ny; dims[1] = nx;
start[0] = 2;  start[1] = 1;
count[0] = 6;  count[1] = 4;

space_id = H5Screate_simple(2, dims, NULL);

status = H5Sselect_hyperslab(space_id, H5S_SELECT_SET, start,\
stride, count, block);
```

How to play with dataspaces

- *space_id* is modified by *H5Sselect_hyperslab*, so it must exist
- *start, stride, count, block* arrays must be at least the same size as the rank of *space_id* dataspace
- H5S_SELECT_SET replaces the existing selection with the parameters from this call.
- Other operations : H5S_SELECT_OR, AND, XOR, NOTB and NOTA
- *stride, block* arrays are considered as 1 if NULL is passed

Using dataspaces during a partial IO

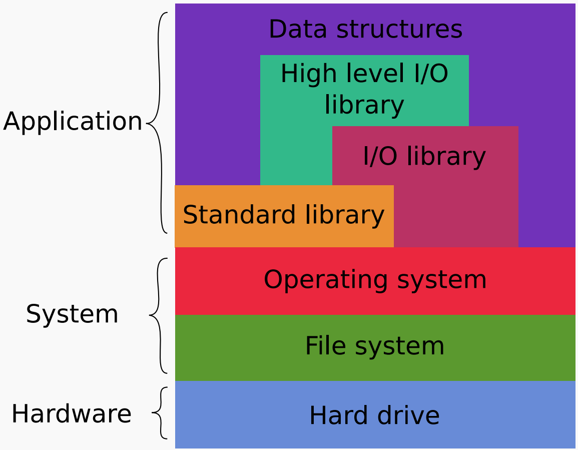
```
space_id_mem = H5Screate_simple(2, dims_mem, NULL);
status = H5Sselect_hyperslab(space_id_mem, H5S_SELECT_SET, \
start_mem, stride_mem, count_mem, block_mem);

space_id_disk = H5Screate_simple(2, dims_disk, NULL);
status = H5Sselect_hyperslab(space_id_disk, H5S_SELECT_SET, \
start_disk, stride_disk, count_disk, block_disk);

status = H5Dwrite(dataset, H5T_NATIVE_INT, space_id_mem, \
                  space_id_disk,H5P_DEFAULT, data);
```

- The two dataspace can describe non contiguous data and can be of different dimension
- But the number of elements must match

Hardware/Software stack



Xdmf

Xdmf is an XML language that allows one to describe complex computational modeling objects from a set of datasets

An Xdmf representation consists of:

- **Light data**: An XML file containing Xdmf language statements and references to datasets contained in the heavy data
- **Heavy data**: A set of ASCII, binary or HDF5 files, even SQL databases

A flexible design

1. Existing data can be easily brought into the framework
   $\Rightarrow$ **XML file written by hand**
2. Existing I/O procedures can be kept untouched
   $\Rightarrow$ **XML file written in addition within the procedure**
3. I/O procedures are modified to write data through Xdmf API
   $\Rightarrow$ **Both heavy and light data written by the Xdmf library**

Xdmf first example

```xml
<?xml version="1.0" ?>
<!DOCTYPE Xdmf SYSTEM "Xdmf.dtd">
<Xdmf Version="2.0">
 <Domain>
  <Grid Name="Structured mesh" GridType="Uniform">
   <Topology TopologyType="2DRectMesh" Dimensions="3 4"/>
   <Geometry GeometryType="VXVYVZ">
    <DataItem Format="XML" Dimensions="3" NumberType="Float" Precision="4">
     0.0 0.5 1.0
    </DataItem>
    <DataItem Format="XML" Dimensions="4" NumberType="Float" Precision="4">
     0.0 1.0 2.0 3.0
    </DataItem>
   </Geometry>
   <Attribute Name="Node Centered Values" Center="Node">
    <DataItem Format="HDF" Dimensions="12" NumberType="Int">
     basic_topology2d.h5:/values
    </DataItem>
   </Attribute>
  </Grid>
 </Domain>
</Xdmf>
```

Hands on HDF5

Fetch the hands-on:

- HDF5 hands-on: git clone https://github.com/mathaefele/HDF5_hands-on.git
- Xdmf hands-on: git clone https://github.com/mathaefele/Xdmf_hands-on.git

Do the hands-on

- Hands-on hdf5_1
- The HDF5 way:
  - Hands-on hdf5_2
  - Hands-on hdf5_3
- The Xdmf way:
  - example
  - vector_single
  - vector_time