

# Introduction to serial HDF5

Matthieu Haefele



Saclay, 6-7 March 2017,  
Parallel filesystems and parallel IO libraries PATC@Mds



# Training outline

## Day 1:

- AM: Serial HDF5 (M. Haefele)
- PM: Parallel IO and parallel HDF5 (M. Haefele)

## Day 2:

- AM 1: Luster file system @ TGCC (T. Leibovici)
- AM 2 + PM: XIOS (M-H. Nguyen)

Please do not forget to **fill the evaluation form** at

<https://events.prace-ri.eu/event/569/evaluation/evaluate>



# Outline Day 1

## **Morning:**

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

## **Afternoon:**

- Parallel IO issues & concepts
- Basic concepts of MPI-IO
- Parallel HDF5
- Hands on session



# IO in a nutshell

Doing Input / Output is about **TRANSPORTING**

Data stored in **memory**

to / from

Data stored on **disk**





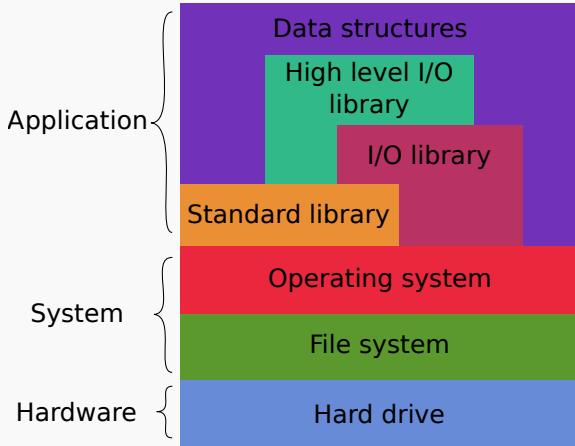
# IO in a nutshell

## Three criteria / metrics to balance

- Code development / maintenance time
- Performance
- Post-processing requirement



# 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)
- **XIOS (XML IO Server)**





# 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 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 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, dims, 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, dims, 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,\n                 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, dims, 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)



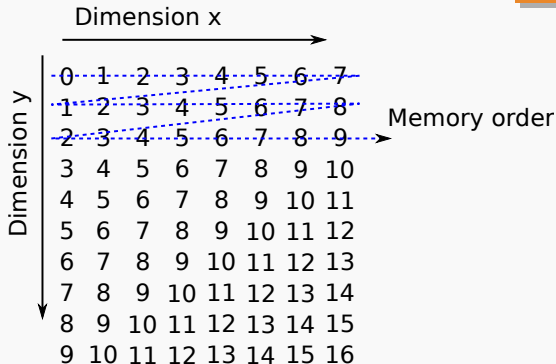
## Conventions for the examples

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
- ⇒ Dimension  $y$  is index=0
- ⇒ 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;
```

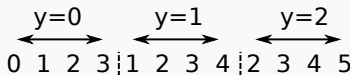


## Illustration for count parameter

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



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

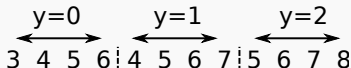


## Illustration for start parameter

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



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

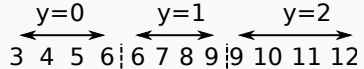


## Illustration for stride parameter

Dimension x →

Dimension y ↓

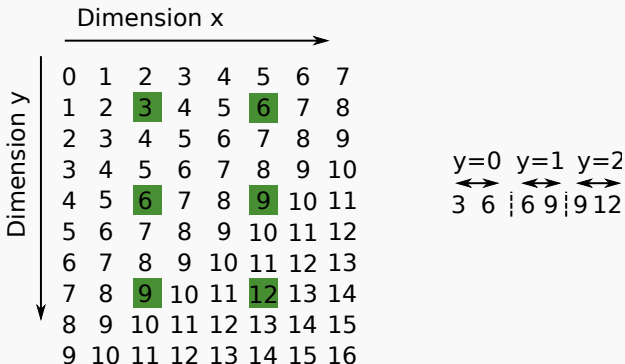
0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16



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



## Illustration for stride parameter



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





## Illustration for block 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

←-----→   ←-----→

3 4 6 7 | 4 5 7 8

y=2      y=3

←-----→   ←-----→

6 7 9 10 | 7 8 10 11

y=4      y=5

←-----→   ←-----→

9 10 12 13 | 10 11 13 14

```
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 ?

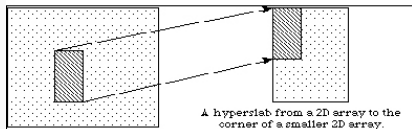


Figure : Access a sub-set of data with a hyperslab<sup>1</sup>

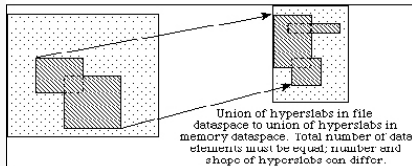


Figure : Build complex regions with hyperslab unions<sup>1</sup>

<sup>1</sup>Figures taken from HDF5 website



## What is a dataspace for ?

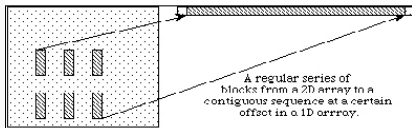


Figure : Use hyper-slabs to gather or scatter data<sup>2</sup>

<sup>2</sup>Figures taken from HDF5 website



## 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 dataspace

- *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 dataspace during a partial IO

```
status = H5Sselect_hyperslab(space_id_mem, H5S_SELECT_SET, \
start_mem, stride_mem, count_mem, block_mem);
```

```
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



## HDF5 command line tools

- HDF5 files are non ASCII files
- non human readable files
- ⇒ Tools provided to manipulate and get information contained in HDF5 files
- Three main ones: **h5ls**, **h5dump**, **h5diff**



# Hands on HDF5

**git clone [https://github.com/mathaefe/HDF5\\_hands-on.git](https://github.com/mathaefe/HDF5_hands-on.git)**

