Advanced HDF5 & XDMF

Matthieu Haefele

High Level Support Team
Max-Planck-Institut für Plasmaphysik, München, Germany

Autrans, 26-30 Septembre 2011,
École d'été Masse de données : structuration, visualisation

Outline

1. Advanced HDF5
   - start, stride, count, block
   - Playing with dataspaces
   - Playing with chunks

2. XDMF language
   - Concepts
   - Language syntax

Concept

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

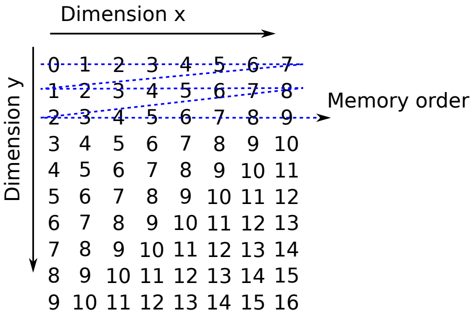
We consider:

- A 2D array $f(x,y)$ 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;
```

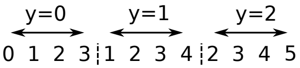
Illustration for count parameter

Selection of the box $((0,0),(3,2))$

Dimension x →

Dimension y ↓

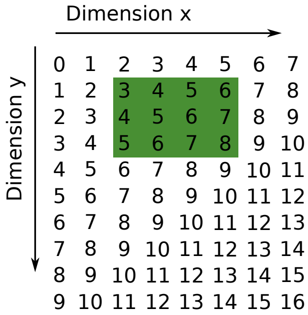
0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

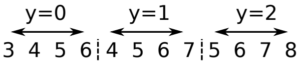



```
count[0] = 3;   count[1] = 4;
```

Illustration for start parameter

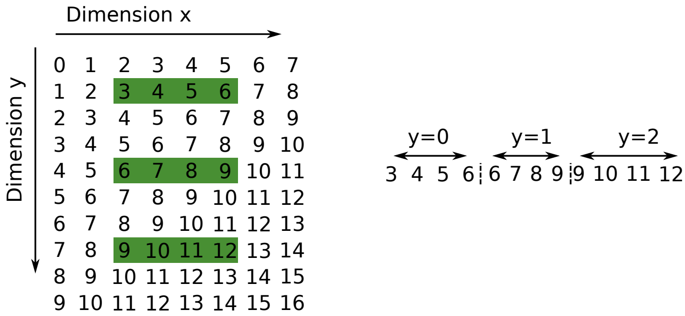
Selection of the box $((2,1),(5,3))$





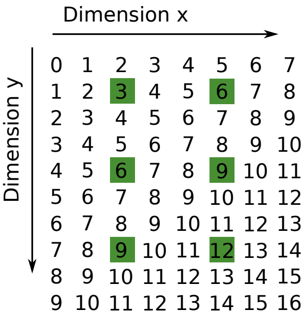

```
start[0] = 1;  start[1] = 2;
count[0] = 3;  count[1] = 4;
```

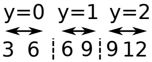
Illustration for stride parameter




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

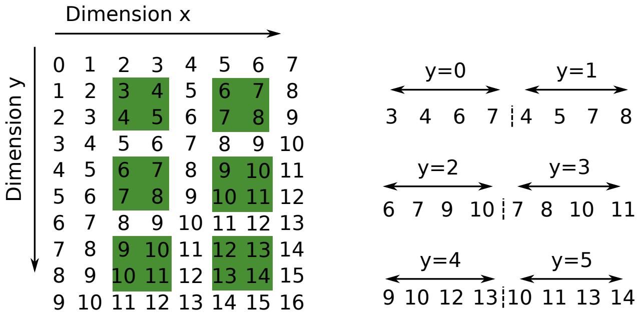
Illustration for stride parameter






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

Illustration for block parameter




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

Exercice 1

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

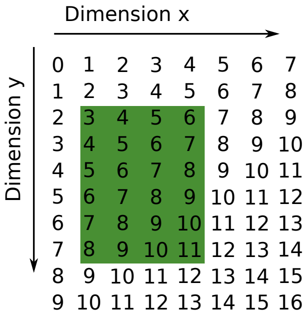
Dimension x →

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




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

Exercice 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;
```

Exercice 3

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

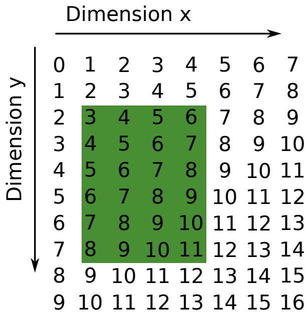
Dimension x →

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




```
 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 ?

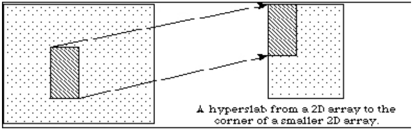


Figure: Access a sub-set of data with a hyperslab[1]

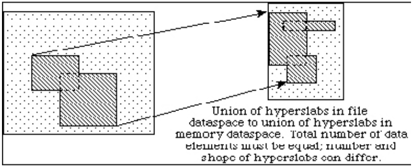


Figure: Build complex regions with hyperslab unions[1]

[1] Figures taken from HDF5 website

What is a dataspace for ?

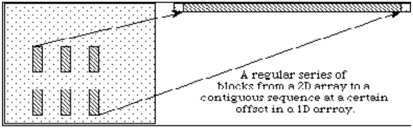


Figure: Use hyperslabs to gather or scatter data[2]

[2]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 dataspaces

- *space_id* is modified by *H5Sselect_hyperslab*, so it must exist
- *start, stride, count, block* arrays must be 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

```
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

What are chunks for ?

- Chunks can improve performance during partial IO.
- Each opened dataset has a chunk cache enabling some kind of out-of-core programming.
- Z compression can be activated on chunked datasets

How to play with chunks

```
hid_t dataset_property, group, dataspace;
hsize_t dims[2], chunk_dims[2];

dims[0] = ny; dims[1] = nx;
chunk_dims[0] = dims[0]/2; chunk_dims[1] = dims[1]/2;
dataset_property = H5Pcreate(H5P_DATASET_CREATE);
status = H5Pset_chunk(dataset_property, RANK, chunk_dims);
status = H5Pset_deflate(dataset_property, 1);

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

HDF5 chunks performance pitfalls

When performing partial IO, chunked dataset can really improve performance but:

- Chunks should not be neither too small nor too large
- Cache size should be large enough to store as many chunks needed by an IO
- Number of chunks hash values

Functions H5Pset_cache, H5Pset_chunk_cache are your friend !

HDF5 main drivers

- MPI-IO: enables parallel IO from a MPI program
- core: enables to handle in-memory file
- Distributed Shared Memory (DSM): enables a MPI program to write into a DSM

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 binary or HDF5 files

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>
```

Tag XDMF and Domain

- A single XDMF tag must contain all the document: this is an XML constraint
- Several domain can exist in an XDMF file

Tag DataItem

The DataItem tag is the most nested tag in XDMF language.
It can contain:

- data
- references to data contained in external files
- other DataItem

Tag DataItem cont.

Possible attributes:

- Name: name of the DataItem
- ItemType: type of the DataItem
- Format: format of the underlying data
  - XML: in ASCII directly in the XML file
  - HDF: in a separate HDF5 file
  - Binary: in a separate binary file
- NumberType: type of the data
  - Float
  - Int
  - UInt
  - Char
  - UChar
- Dimensions: number of elements in each dimensions
- Precision: 1, 4 or 8 Bytes per elements
- Endian: Big, Little or Native (only for Binary Format)

Tag DataItem cont.

Possible value for attribute ItemType:

- Uniform: A single array of values (default)
- Collection: A one dimension array of DataItems
- Tree: a hierarchical structure of DataItems
- HyperSlab: contains two data items. The first selects the start, stride and count indexes of the second DataItem
- Coordinates: contains two DataItems. The first selects the parametric coordinates of the second DataItem
- Function: calculates an expression.

Tag Grid

A Grid describes how space and time are discretized.
Possible attributes

- Name: name of the grid
- GridType: type of the grid
  - **Uniform**: a homogeneous single grid
  - Collection: an array of Uniform grids all with the same Attributes
  - Tree: a hierarchical group
  - Subset: a portion of another Grid
- CollectionType (meaningfull only for GridType=Collection)
  - Spatial: domain decomposition
  - Temporal: time evolution of a grid
- Section (meaningfull only for GridType=Subset)
  - DataItem: subset decribed in the following DataItem
  - All: select the whole grid

Tag Topology

The Topology tag describes the kind of topology of the current grid

- For structured grid, connectivity is implicit
- For unstructured grid, connectivity must be explicitely given in a DataItem

Possible attributes:

- TopologyType: type of the topology
- NumberOfElement or Dimensions: number of cells
- NodesPerElement (Meaningful only for Polyvertex, Polygon and Polyline)
- BaseOffset (eventually needed for binary files)

Tag Topology cont.

Possible values for TopologyType:

- Polyvertex
- Polyline
- Triangle
- . . .
- Edge_3 - Quadratic line with 3 nodes
- . . .
- Mixed
- 2DCoRectMesh - 2D rectilinear mesh, Axis are perpendicular and spacing is constant
- 2DRectMesh - 2D rectilinear mesh
- 2DSMesh - 2D curvilinear mesh
- . . .

Tag Geometry

The Geometry tag describes the position of the nodes of the mesh

One single mandatory attribute: GeometryType

- XYZ: Interlaced locations
- XY: like XYZ, but Z is set to 0.0
- X_Y_Z: X,Y, and Z are separate arrays
- VXVYVZ: Three arrays, one for each axis
- ORIGIN_DXDYDZ: Six Values : Ox,Oy,Oz + Dx,Dy,Dz (only used for CoRectMesh)

Tag Attribute

The Attribute tag describes how values are mapped on a grid

Possible attributes:

- Name: name of the Attribute
- Center: where values are located on the mesh
  - Node
  - Cell
  - Grid
  - Face
  - Edge
- AttributeType
  - Scalar
  - Vector
  - Tensor
  - Tensor6
  - Matrix

Attribute Reference

- Each tag seen previously can have a Reference attribute.
- Instead of describing something new, it can just refer to an existing tag in the XDMF file.

XDMF first example

```
<?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>
```

XDMF issues

- Lack of description (ex: No 1D curve)
- Inconsistencies between specification and implementation
- XDMF library very difficult to compile and to use
- Is the project about to die ?

Conclusion

Four levels of interfaces to perform I/O:

- High level I/O libraries
- I/O libraries
- Standard library
- Kernel call

I/O and high level I/O libraries

- need to be mastered
- introduce a software dependency, so portability and durability issues
- provide higher level API, so less code and more maintainable code