RFC: A New Library to Generate Arbitrary Hyperslabs.

Rishi Sinha and Robert E. McGrath
October 12, 2004

1. Introduction

There is a need for improved testing of the complex HDF5 hyperslabs. One of the main hurdles in the testing process is the definition of an arbitrary hyperslab. A simple cuboidal hyperslab is relatively easy to define but if we want to define a complex hyperslab it is not very apparent how we can define one. The purpose of this tool is that given a set of input parameters the tool would generate a set of arbitrary hyperslabs and then test these for correctness and efficiency.

The library described in this document is one component of an overall testing tool. The output of the library is a description of an HDF5 selection (see section 4). Another module reads these descriptions and generates selections. Yet another component reads data from the selections and checks the result.

The module described in this document implements one algorithm to generate HDF5 selections. Other module could be implemented to generate selections using different algorithms, writing the description in the standard language. Also, manually defined test cases can be written. Figure 1 suggests the data flow.

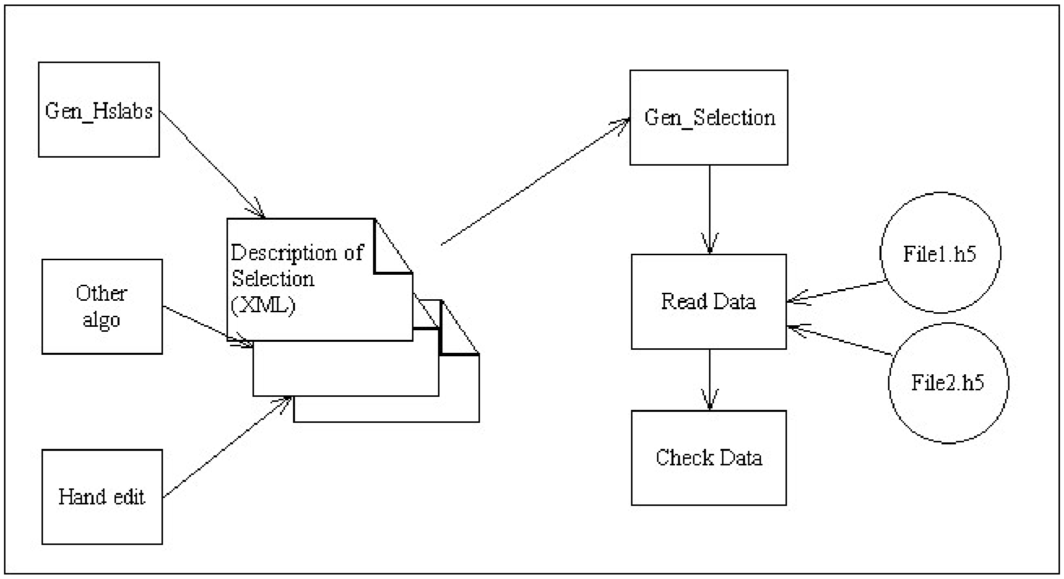

Figure 1

2. Proposed Library

Name: Generate_HSlabs

Methods:

Gen_Hslabs
Parameters: char *ifname, (The name of the file that has the input parameters).
Char *ofname (The name of the file where the output has to be logged.)
Return value: +ve if all the tests have passed.
-ve if any of the tests have failed. The value of the negative number represents the number of tests that have failed.
Other output: The output file, where all the information regarding the hyperslab selection is logged.

Description:
This library is intended to create arbitrary shaped hyperslabs and test the HDF5 libraries for these hyperslabs. Since this module could be used to test both correctness and efficiency of the underlying libraries, it should be stand-alone.

3. Details.

3.1 Input Parameters.

The input of the algorithm could be provided through a simple XML file defining the input parameters. The parameters that need to be defined are

1. Rank – The number of dimensions.
2. Max_Planes – For each dimension what is the maximum number of planes to consider in that dimensions.

3.2 Generating the hyperslab.

A hyperslab consists of a series of bounding hyperplanes. Thus a hyperslab can be represented as a set of intersections of hyperplanes. Thus if we can generate a series of hyperplanes and their intersections, we can generate any arbitrary hyperslab. For example if we start with the hyperplanes $x = 2$, $y = 3$, shown in Figure 2, we can find an intersection at the point (2, 3).

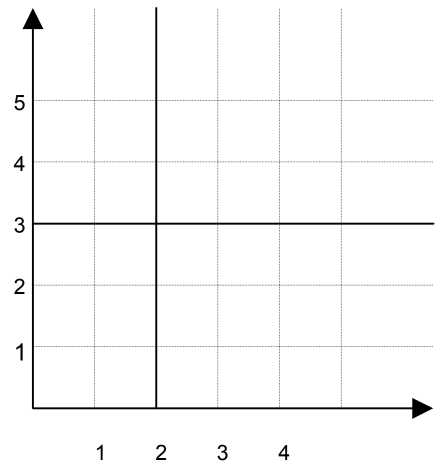

Figure 2

This intersection does not really tell us anything about the shape of the hyperslab until we say which direction this intersection is from. Knowing that the direction of intersection is +ve from x direction, and +ve from y direction (where +ve is the direction opposite to the direction of axis arrow), we know it corresponds to the corner shown in red in **Figure 3**.

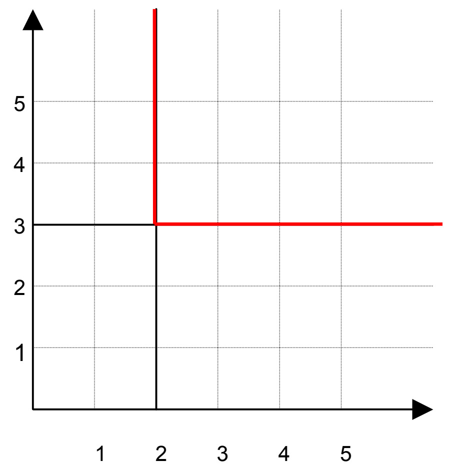

Figure 3

Hence if we add x = 3, and y = 4 hyperplanes to the set of hyperplanes, then we would get the hyperslab represented by Table 1. This would be the hyperslab shown in **Figure 4**.

	x = 2	x = 3
y = 3	1, 1	1, -1
y = 4	-1, 1	-1, -1

Table 1

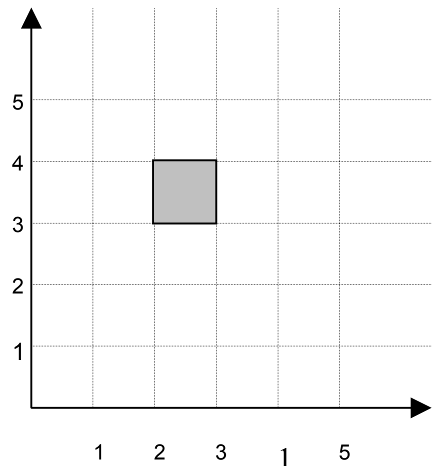

Figure 4

In order to generate this hyperplane we need to generate 2 hyperplanes in x-axis and 2 in y-axis. These are x = 2, x = 3 and y=1, y = 2. We separate the hyperplanes in to sets based on the axis they are constrained on. Thus the sets would be X = {x = 2, x = 3}, and Y = {y = 3, y = 4}. We order these in a canonical order of the coordinates. Thus we start with hyperplanes in set X. We sort the hyperplanes in X in order of the intersecting coordinate. In this order we find the intersection of each hyperplane with all the hyperplanes in higher axes. Thus we generate intersections for x = 2, since we have only two planes it can intersect with we make it intersection with both. The same is done for x = 3. This would lead to intersections at points (2, 3), (2, 4), (3, 3), (3, 4). These hyperplanes represent the hyperslab shown in Figure 4.

To generate the hyperslab shown in Figure 5, we would start with generating the hyperslabs intersecting with the x-axis. Thus we generate 6 different hyperplanes X = {x = 1, x = 2, x = 3, x = 4, x = 5, x = 6} and the 7 different hyperplanes intersecting the y-axis Y = {y = 1, y = 2, y = 3, y = 4, y = 5, y = 6, y =7}. Now we need to start with generating possible intersecting hyperplanes for x = 1. In the figure we have chosen y = 1, and y = 7. So for x = 1 we store a list containing {y = 1, y = 7}, also we insert x = 1 into the set for y = 1, and y = 7. Similarly we generate intersections for all hyperplanes in x-coordinate. Since the intersections for all x coordinates also generate all intersections for y coordinates. Have already been generated we don't need to generate intersections for y coordinates.

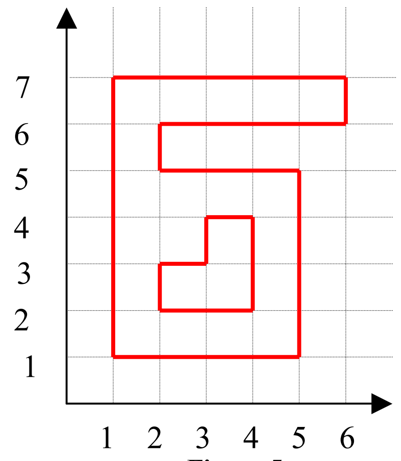

Figure 5

Thus we can represent any arbitrary hyperslab as a list of intersecting hyperplanes for every hyperplane. The direction of intersection of the hyperplane is determined by its position in the list. For example for the list for x = 2 in Figure 4, {y = 2, y = 3, y = 5, y = 7}. This means x = 2 intersects y = 2 from the +ve direction, y = 3 from the –ve direction, y = 5 from the +ve direction and y = 6 from the –ve direction. Thus the direction of each intersection is clearly defined just by this list. Hence the exact shape of the hyperslab is clearly defined.

Hence every hyperslab can be represented by an *n* dimensional array, but not every *n* dimensional array represents a hyperslab. The constraints that a matrix has to follow are the following:

i. If we order all the hyperplanes in one dimension according to their axis intersection value
   a. Then all the intersecting hyperplanes would intersect the lowest hyperplane from the positive direction.
   b. And all the intersecting hyperplanes would intersect the highest hyperplane from the negative direction.
   c. If a hyperplane 1 in dimension x, intersects with a hyperplane 2 in dimension y, then the direction of intersection of 2 with 1 is determined by whether or not 2 has intersected with a lower hyperplane in dimension x. If it has then the direction would be opposite of the intersection of the closest lower hyperplane, if not it would be positive. The same applies for the direction of 1.
   d. The last hyperplane in all directions must close all of the open hyperplanes. An open hyperplane is a hyperplane that intersects an odd number of hyperplanes in a particular axis.

ii. The number of intersections of a particular hyperplane should be even. This makes sure that we have a closed hyperslab.

iii. The hyperplane shouldn't cross over the boundaries of the closest smaller hyperplane if it is in negative direction. For example if x = 2 intersects with y = 3, y = 4, then x = 3 shouldn't intersect with any pair of y hyperplanes where y >4 and y < 4, or a pair where y < 3 and y > 3.

Thus as long as we follow the constraints on the hyperplanes we can generate any arbitrary hyperslab, and be randomly generating the intersections we can generate the hyperslabs randomly. This hyperslab would then need to be converted to a series of calls to the HDF5 library so that we can create this hyperslab. This algorithm is being worked on.

3.3. Coverage and Limitations of this Algorithm

This algorithm generates hyperslabs that can be described by a set of bounding hyperplanes. This algorithm works for any number of dimensions. The number of planes to be intersected determines the "complexity" of the hyperslab.

The algorithm runs in a $O(n^2m^2)$ where n is the number of dimensions and m the maximum number of hyperplanes in any dimension. Thus even if we have a large number of dimensions, the algorithm shouldn't have any problem generating the hyperslab.

Any hyperslab generated by this algorithm can be used as an HDF5 selection.

This algorithm does not test the features of the H5Sset_selection function. For any given selection, there may be many different sequences of calls to H5Sset_selection to generate the full selection. This algorithm will use a single canonical sequence of calls.

4. Output logging.

Since every run of the generation algorithm would create a different hyperslab, (random generation) hence we would need to log the structure of the hyperslab every time we create a hyperslab. Also all the calls that are being made would also be logged. After the hyperslab has been read from the dataset we, would then check if every item on the dataset corresponds to the correct value.

This logging would again be done in an XML file. For every run the XML file would contain the following.

```
        <HSlabFile>Name</HSlabFile> (The name of the file where the array is being
stored.)
        <HSlabCalls>
                <1>
                        <Count>
                                <1>#</1>
                                <2>#</2>
                                <3>#</3>
                        </Count>
                        <Start>
                                <1>#</1>
                                <2>#</2>
                                <3>#</3>
                        </Start>
                </1>
                <2> …
                </2>
        </HSlabCalls>
        <NoOfDataErrors>#</NoOfDataErrors>
        <DataValueFile>Name</DataValueFile> (The Name of the file where the details
of the check on the individual data items are stored.)
```

The format of the file, which stores the check on every incorrect data item, is

```
<Coordinate>
        <1>Coordinated value</1>
        <2>Coordinated value</2>
        <3>Coordinated value</3>
        <ActualValue></ActualValue>
        <CalculatedValue></CalculatedValue>
</Coordinate>
```