RFC: A New Library to generate arbitrary hyperslabs.

Rishi Sinha and Robert E. McGrath
September 30, 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 correction and efficiency.

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. Num_OR – These are the number of distinct hyperslabs that would be generated to be ORed together. This is necessary because one complex hyperslab generated by the algorithm would be a contiguous hyperslab, where as we also need to test

discontiguous hyperslabs. Though this part also doesn't guarantee presence of discontiguous hyperslabs, but it increases the chance of such hyperslabs.

3. Max_Planes – For each dimension what is the maximum number of planes to consider in that dimensions.

3.2 Generating the hyperslab.

Any hyperslab can be represented by an n dimensional array. This array would represent the intersection of various hyperplanes involved in the hyperslab. Each point in the array would have a 0, or a ${}^{n}C_2$ dimensional tuple of pairs, representing the form of intersection (The reason for the tuples is explained in the next paragraph). A 0 means that at the planes don't intersect. For every hyperplane in that intersection we also need to provide which direction that the hyperplane intersects from, this can be a positive direction, represented by 1 or a negative direction represented by -1. For example a point (2, 3) in a 2 dimensional matrix would represent how hyperplanes x = 2, y = 3 would intersect. For a non-0 value of the point, it would mean that the hyperplanes intersect at that point. This intersection is shown in the following diagram.

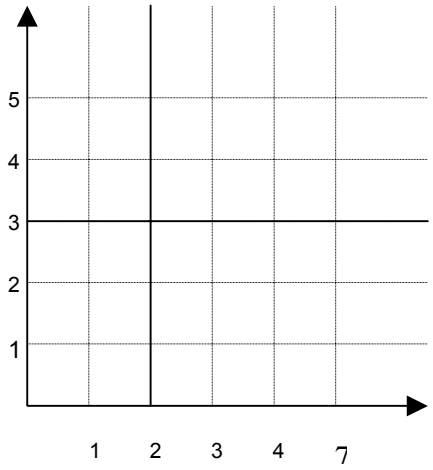

Figure 1

This intersection doesn't denote any structure of the hyperslab; hence we need direction of intersection too. For example both if x = 2 and y = 3 intersect from the positive direction then the red lines would indicate one corner of the hyperslab being generated. We need 2 such directions for each pair of intersecting hyperplanes. Since for an n dimensional hyperslab we have ${}^{n}C_2$ different intersections of hyperplanes, we need ${}^{n}C_2$ different pairs of numbers representing the intersection. None of these pairs can be 0,0 because for a closed hyperslab we need hyperplanes from all dimensions to meet. Thus value in the tuple containing ${}^{n}C_2$ different pairs shows the direction of the intersection for hyperplane of the corresponding dimension. Here it would be ((1, 1)).

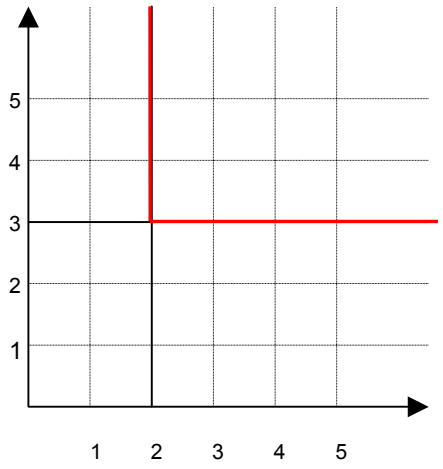

Figure 2

Hence the 2 dimensional array:

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

Table 1

for the hyperplanes x =2, x = 3, y = 3, y = 4 would represent the hyperslab.

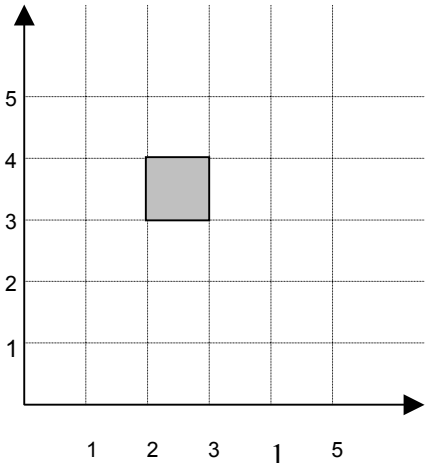

Figure 3

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.

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 if we generate n dimensional arrays that conform to these constraints we are in essence generating hyperslabs. Thus if we can randomly generate these arrays, we have a method of generating any arbitrary complex hyperslab.

For example Figure 4 shows the hyperslab for the array in Table 2

	x = 1	x = 2	x = 3	x = 4	x = 5	x = 6
y = 7	-1, 1	0	0	0	0	-1, -1
y = 6	0	-1, 1	0	0	0	1, -1
y = 5	0	1, 1	0	0	-1, -1	0
y = 4	0	0	-1, 1	-1, -1	0	0
y = 3	0	-1, 1	1, -1	0	0	0
y = 2	0	1, 1	0	1, -1	0	0
y = 1	1, 1	0	0	0	1, -1	0

Table 2

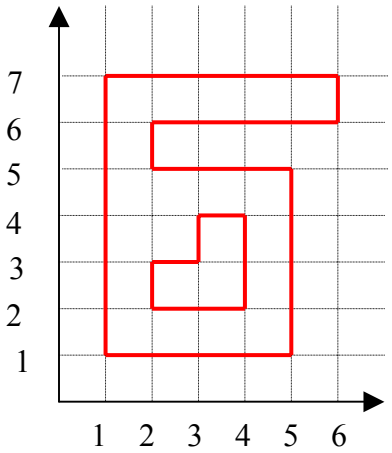

Figure 4

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.

This algorithm runs in polynomial time with respect to the number of dimensions. Thus going in very high dimensions may not be desirable. This is not that big a limitation because for very high dimensions dataset, the memory also becomes a limitation.

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>