RFC: Expanding the HDF5 Hyperslab Selection Interface

Chao Mei, Quincey Koziol

Hyperslab selection is an important component of the HDF5 library, and has a wide usage in many applications. However, the current implementation provides only a limited number of interfaces for operating on hyperslabs. This document proposes adding new operations on hyperslab selection that allow the user to more flexibility operate on two hyperslab selections.

1 Introduction

In HDF5, when the user wants to read from or write to a portion or subset of a dataset, he or she needs to select a subset of the dataspace of the dataset, and then use that selection to read from or write to the dataset. The hyperslab selection method available in the public HDF5 API (i.e. *H5Sselect_hyperslab*) provides such a way to create a selection. It operates on a hyperslab with specified *offset*, *stride*, *block size* and *count*, a set of four parameters representing a regular pattern of elements in a dataspace.

While such a single function interface has been adequate to cover applications' use of selection operations so far, it would improve programming productivity to have a more flexible interface that addresses more use cases. For example, if application developers want to combine hyperslab selections from two different dataspaces and create a new dataspace containing the result, they are not easily able to perform such an operation with the existing selection routine because the hyperslab selection in a dataspace may not be represented simply by the four *offset*, *stride*, *block size* and *count* parameters.

Furthermore, providing new selection functions will improve the high-level abstractions of selection operations for user applications and potentially improve performance. For example, if application developers want to keep a current dataspace selection unchanged, but store the selection result in a new dataspace, they currently first need to copy the existing dataspace, and then use the copy to perform the specified selection operation. Therefore, extra code in the application to perform the copy is required in such situations. It is more desirable to leave such copying to the HDF library, which can perform both the hyperslab operation and dataspace copy in a single, more efficient step. Additionally, in some cases such as when the selection result is empty, the process is not efficient in terms of memory footprint and computation because of the extra copy of the existing dataspace.

To address the concerns above, three new interfaces for hyperslab selection operations are proposed in this RFC.

2 Approach

Among the three functions proposed in this RFC (in section 3, below), two of them (H5Smodify_select and H5Scombine_select) perform operations on two dataspaces (each containing a hyperslab selection). H5Smodify_select stores the result in the first dataspace (replacing the existing selection), while H5Scombine_select creates a new dataspace to store the resulting selection. Providing these two functions covers use cases where the existing hyperslab selection to operate on could not be represented simply by the set of four *offset*, *stride*, *block size* and *count* parameters, thus offering more flexibility in hyperslab operations than the existing hyperslab selection method (H5Sselect_hyperslab). The third proposed function (H5Scombine_hyperslab) has almost the same interface with the existing selection method, with the only difference that this new function will return a new dataspace to store the result. These three new functions only provide additional hyperslab selection methods, and therefore do not affect existing application code.

In summary, the three functions mentioned above, and the existing H5Sselect_hyperslab function, express the different equations as shown in the following table:

Function Name	**Equation Name**
H5Sselect_hyperslab	A = A <op> [St, Sd, Ct, Bk]
H5Scombine_hyperslab	C = A <op> [St, Sd, Ct, Bk]
H5Smodify_select	A = A <op> B
H5Scombine_select	C = A <op> B

Where

1. A, B represent the dataspace passed as function arguments, C represents the new dataspace created to contain the selection result.
2. <op> represents the operation that is allowed in the functions (i.e., NOT, AND, OR etc.).
3. [St, Sd, Ct, Bk] is a four-element tuple that describes the hyperslab passed as function arguments, representing the “start” (St), “stride” (Sd), “count” (Ct), “block” (Bk) of the hyperslab respectively.

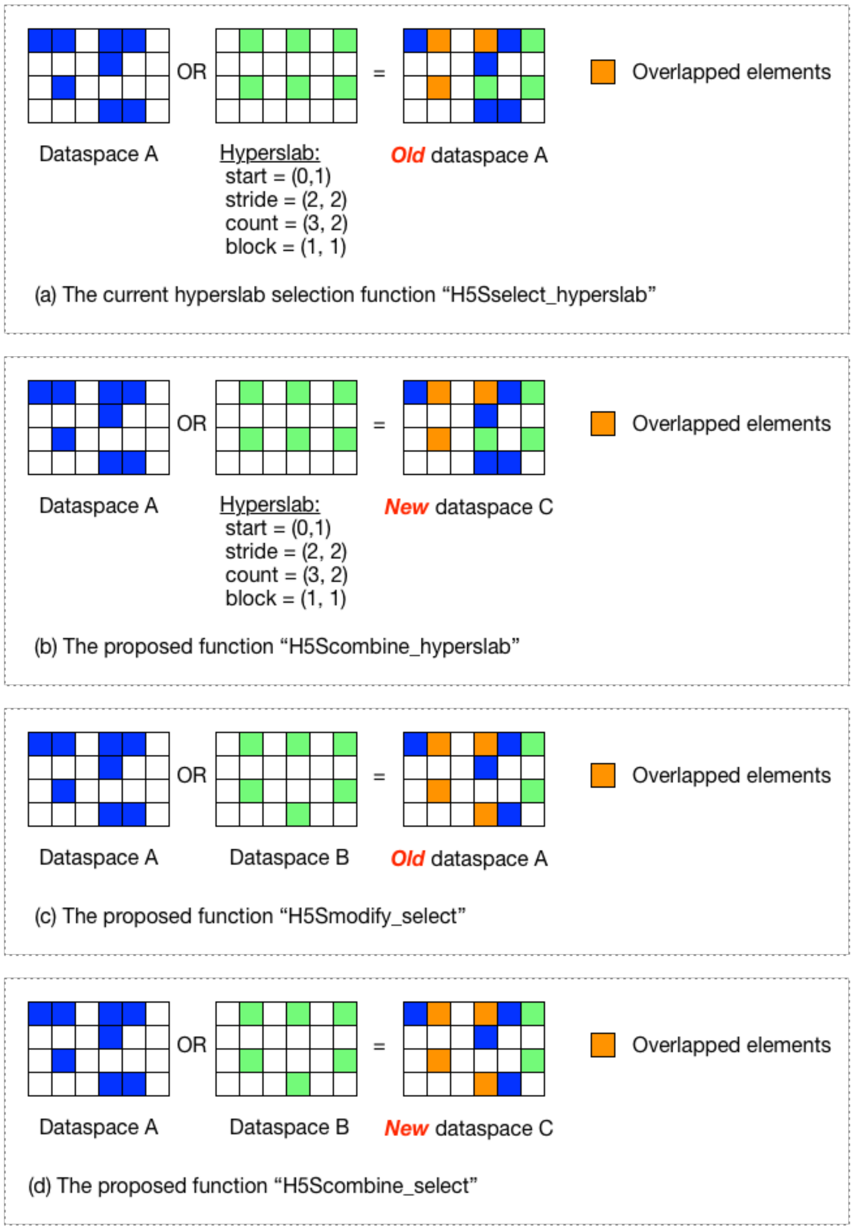
OR
=
Overlapped elements
Dataspace A
Hyperslab:
start = (0,1)
stride = (2, 2)
count = (3, 2)
block = (1, 1)
Old dataspace A
(a) The current hyperslab selection function "H5Sselect_hyperslab"
New dataspace C
(b) The proposed function "H5Scombine_hyperslab"
Dataspace B
(c) The proposed function "H5Smodify_select"
(d) The proposed function "H5Scombine_select"

Figure 1: Hyperslab Selection Operations

Figure 1 shows four examples where the H5S_SELECT_OR operation is performed on 2D dataspaces, demonstrating the differences between the existing hyperslab selection method (figure 1(a)) and the three proposed ones (figures 1(b), 1(c), 1(d)). As for the current method illustrated by figure 1(a), the selection result is stored in the first dataspace "A", and second selection could be represented by four parameters. Figure 1(b) differs from 1(a) in that the result in stored in a new dataspace "C". In figure 1(c) and 1(d), the second selection in dataspace "B" is not a regular pattern, and could not be simply represented by four hyperslab parameters. This is the biggest difference from the other two figures 1(a) and 1(b). If we compare figure 1(c) and 1(d), the difference is analogous with that of figure 1(a) and (b) as to which dataspace stores the selection result.

a)

b)

3 Interface

The prototypes for the proposed functions are:

a) `hid_t` H5Scombine_hyperslab(*space_a, op, start, stride, count, block*)

`hid_t` *space_a*;	IN: Dataspace ID which contains selection A
`H5S_seloper_t` *op*;	IN: Operation to perform to combine selection in dataspaces
`hssize_t *start`;	IN: Array containing the offset of the starting coordinate for the hyperslab
`hssize_t *stride`;	IN: Array containing the elements between each block's starting location
`hssize_t *count`;	IN: Array containing the number of blocks in each dimension
`hssize_t *block`;	IN: Array containing the size of each block in each dimension

- **Purpose**: Performs an operation on a hyperslab and an existing selection and returns the resulting selection in a new dataspace, which will have the same extent as that of the *space_a* dataspace.
- **Returns**: The ID of dataspace with selection defined by operation is returned on success; negative value is returned on failure.
- **Description**: Combines hyperslab specified with an existing hyperslab selection (which must have the same number of dimensions) using the operation specified to form the selection specified in a new data space, which is returned. The extent of the dataspace for selection A will be used for the extent of the dataspace returned. The following operations are defined:
    - H5S_SELECT_SET: replacing existing selection with the parameters provided
    - H5S_SELECT_OR: logical OR of elements in selection A and selection B
    - H5S_SELECT_AND: logical AND of elements in selection A and selection B
    - H5S_SELECT_XOR: logical XOR of elements in selection A and selection B
    - H5S_SELECT_NOTB: subtract selection B from selection A
    - H5S_SELECT_NOTA: subtract selection A from selection B
- **Comments**: Same as existing H5Sselect_hyperslab API call, except a new dataspace is created and returned.

b) `herr_t` H5Smodify_select(*space_a, op, space_b*)

`hid_t` *space_a*;	IN/OUT: Dataspace ID which contains selection A
`H5S_seloper_t` *op*;	IN: Operation to perform to combine selection in dataspaces
`hid_t` *space_b*;	IN: Dataspace ID which contains selection B

- **Purpose**: Performs an operation on the hyperslab selections of two dataspaces and returns the result in the first dataspace (replacing the current selection).
- **Returns**: non-negative value is returned on success; negative value is returned on failure.
- **Description**: Combines two hyperslab selections (which must have the same number of dimensions) using the operation specified to form a new selection that replaces selection A in *space_a*. The following operations are defined:
  - H5S_SELECT_OR: logical OR of elements in selection A and selection B
  - H5S_SELECT_AND: logical AND of elements in selection A and selection B
  - H5S_SELECT_XOR: logical XOR of elements in selection A and selection B
  - H5S_SELECT_NOTB: subtract selection B from selection A
  - H5S_SELECT_NOTA: subtract selection A from selection B
- **Comments**: Similar to existing H5Sselect_hyperslab API call, except a selection is used to combine with an existing selection, instead of specifying a hyperslab offset, stride, block count and size to combine with a dataspace's selection.

c) `hid_t` H5Scombine_select(*space_a, op, space_b*)

`hid_t` *space_a*;	IN: Dataspace ID which contains selection A
`H5S_seloper_t` *op*;	IN: Operation to perform to combine selection in dataspaces
`hid_t` *space_b*;	IN: Dataspace ID which contains selection B

- **Purpose**: Performs an operation on the hyperslab selections of two dataspaces and returns the resulting selection in a new dataspace, which will have the same extent as that of the *space_a* dataspace.
- **Returns**: The ID of dataspace with selection defined by operation is returned on success; negative value is returned on failure.
- **Description**: Combines two hyperslab selections (which must have the same number of dimensions) using the operation specified to form the selection specified in a new data space, which is returned. The extent of the dataspace for selection A will be used for the extent of the dataspace returned. The following operations are defined:
  - H5S_SELECT_OR: logical OR of elements in selection A and selection B
  - H5S_SELECT_AND: logical AND of elements in selection A and selection B
  - H5S_SELECT_XOR: logical XOR of elements in selection A and selection B
  - H5S_SELECT_NOTB: subtract selection B from selection A
  - H5S_SELECT_NOTA: subtract selection A from selection B
- **Comments**: Same as H5Smodify_select, except a new dataspace is created and returned.

4 Example

a) Using *H5Scombine_hyperslab*:
```
/* assign values to start, stride, count, block */
start[0] = …; start[1] = …; …… ; start[M] = …;
stride[0] = …; stride[1] = …; …… ; stride[M] = …;
count[0] = …; count[1] = …; …… ; count[M] = …;
block[0] = …; block[1] = …; …… ; block[M] = …;
space_a = … /* create first dataspace, with hyperslab selection */

/* Perform “And” operation, and store the result in a new dataspace space2 */
 if((space_c = H5Scombine_hyperslab(space_a, H5S_SELECT_AND,
                                   start, stride, count, block)) < 0)
       fprintf(stderr, “Error in H5Scombine_hyperslab”);
```

b) Using *H5Smodify_select*:
```
space_a = … /* create first dataspace, with hyperslab selection */
space_b = … /* create second dataspace, with hyperslab selection */

/* “And” two spaces, and the result is stored in space_a */
if(H5Smodify_select(space_a, H5S_SELECT_AND, space_b) < 0)
      fprintf(stderr, “Error in H5Smodify_select”);
```

c) Using *H5Scombine_select*:
```
space_a = … /* create first dataspace, with hyperslab selection */
space_b = … /* create second dataspace, with hyperslab selection */

/* “And” two spaces, and the result is stored in a new dataspace space3 */
if((space_c = H5Scombine_select(space_a, H5S_SELECT_AND, space_b)) < 0)
      fprintf(stderr, “Error in H5Scombine_select”);
```
d)

5 Implementation

The basic work flow of these three new functions will be similar to the existing H5Sselect_hyperslab function. First, three parts of two dataspaces’ interaction are calculated, i.e., two for the selection regions that belong to only one of the dataspaces and one for the region that belongs to both dataspaces. Finally, depending on the specified selection operation (AND, OR, XOR, etc), those three parts are combined to form the newly created dataspace to store the result or to replace the selection in the existing dataspace.

For a newly created dataspace that combines two existing dataspaces, depending on the selection operation, especially the bounding box of the two existing selections, it is possible to avoid copying the selection inside each dataspace. For example, if the two dataspaces' selections are detected to be non-overlapping, and the selection operation is H5S_SELECT_AND, the newly created dataspace will obviously have an empty selection. Therefore, it is desired to implement the new API routines with as few copies of selection regions as possible in order to achieve the best performance. For example:

c) If two dataspace selections don't overlap and the operator is H5S_SELECT_AND, then it is not necessary to copy both selection regions.

d) If two dataspace selections don't overlap and the operator is H5S_SELECT_NOTB, then it is not necessary to copy the selected region in the second space.

Although the basic work flow of these new functions is similar to each other, the handling of the memory footprint is quite different. *H5Smodify_select* requires the resulted hyperslab selection to be integrated into the first dataspace, while the other two (*H5Scombine_select*, *H5Scombine_hyperslab*) require a new dataspace to be created to contain the result.

Revision History

June 27, 2011:	Version 1 circulated for comment within The HDF Group.
July 18, 2011:	Version 1.1 circulated for comment within The HDF Group after first-round reviews
July 25, 2011:	Version 1.2 circulated for comment within The HDF Group after second-round reviews
August 11, 2011:	Version 1.3 circulated for comment with The HDF Group after third-round reviews
June 13, 2014:	Changed from H5Sselect_select() ➔ H5Smodify_select. Redrew figure to correct typos. Refactored document contents for clarity.