RFC: New HDF5 API routines for HPC Applications

Peter Cao

Quincey Koziol

Jonathan Kim

The HDF5 library allows a data access operation to access one dataset at a time, whether access is collective or independent I/O. Accessing multiple datasets will require the user issuing an I/O call for each dataset. For example, if you access five datasets in a file, you will need at least five IO calls -- at least one for each dataset. This RFC proposes new routines to allow users to access data in multiple datasets with one I/O call.

This RFC describes the new API routines, *H5Dread\_multi()* and *H5Dwrite\_multi()*, which take information about multiple datasets and perform a single dataset access to the file. The new functions can improve I/O performance when collective I/O access is used. Although the new functions can be used for independent I/O (serial or parallel), users are not likely to see any benefits from the new functions when used that way.

# Introduction

Accessing collectivelybycan

However the current HDF5 library does not support a single I/O call for multiple datasets. For example, if you access five datasets in a file, you will need at least five I/O calls.

We propose to add three new functions to the HDF5 library: *H5Dread\_multi()*, *H5Dwrite\_multi() and H5Dclose\_multi()*. Using the proposed new read/write functions, users will be able to make a single function call to read or write data to multiple datasets. The new functions can be used for both independent and collective I/O access. Our discussion below will focus on the collective I/O case since only the collective I/O will take the advantage of the new functions.

# Use Cases

## Improving FLASH I/O

FLASH code was designed to simulate thermonuclear flashes on a Cartesian, structured mesh. The mesh consists of cells that contain physical quantities such as density, pressure and temperature (also known as mesh variables). Each cell is assigned to a self-contained block. In the FLASH file layout, a block is stored in an HDF5 file and mesh variables are stored as 4D datasets in the file.

In a FLASH simulation, the time spent on file I/O is a common bottleneck. Using collective I/O[1] improves I/O performance for HPC applications like FLASH. Current parallel HDF5 performs collective I/O on a single dataset and requires many I/O calls in FLASH simulations since there are frequently many variables accessed during each time step. Using the proposed collective I/O on multiple datasets will reduce the number of I/O calls. In an experimental study, Rob Latham, Chris Daley, etc.[2] have showed that the average time for writing a file is reduced by half when collective I/O on multiple variables is used:

“*The standard file layout approach (storing application data in multiple library objects), however, offers a slight performance trade-off. Each function call represents a relatively expensive I/O operation. All other factors aside, if the goal is to achieve the highest I/O performance a better approach would describe the entire application I/O pattern and then execute a single call. If the application places all mesh variables into a single I/O library object, as in the experimental file layout approach, then a single I/O library call could be issued to service all application variables instead of N separate calls. Experiments confirm that this approach does improve performance*.”[2]

# Approach

The basic approach for multi-dataset collective IO is not much different from the current implementation of chunk collective IO on a single dataset. The major change for the work is that we add data information from multiple datasets to the I/O mapping list and construct MPI information for collective IO operations.

The following example chart shows the conceptual implementation approach for the new API functions



## New API Functions

Three new functions, *H5Dread\_multi()*, *H5Dwrite\_multi()* and H5Dclose\_multi() are proposed here, and a common type structure is used for the APIs as defined below .

typedef struct H5D\_rw\_multi\_t

{

hid\_t dataset\_id;

hid\_t mem\_type\_id;

hid\_t mem\_space\_id;

hid\_t file\_space\_id;

unsigned int close\_flags; /\* handle close/free resources \*/

void \* buf; /\* data buffer \*/

} H5D\_rw\_multi\_t;

### H5Dread\_multi()

This routine performs collective or independent I/O reads from multiple datasets. In collective mode, all members of the file communicator associated with the HDF5 file must participate in the call. Each process loads the information required to perform each read into the H5D\_rw\_multi\_t structure, and passes an array of the structures through to *H5Dread\_multi()*.

The API function description is as shown below.

herr\_t H5Dread\_multi(hid\_t file\_id,

size\_t count,

struct H5D\_rw\_multi\_t info[],

hid\_t dxpl\_id);

Parameters:

* file\_id: file or group id for the location of datasets
* count: the number of datasets.
* Info: the array of dataset information and read buffers.
* dxpl\_id: dataset transfer property.

Return:

* a non-negative value if successful; otherwise returns a negative value.

Brief description is as follows. Note that details have been omitted for brevity.

* Each process obtains the list of dataset info from the info[] array structure, and constructs a derived MPI type describing the sections of the HDF5 file to be read.
* For the collective I/O operation, each process end up calling *MPI\_File\_read\_at\_all()* to perform the desired reads.
* For the independent I/O operation, each process end up calling *MPI\_File\_read\_at()* to perform the desired reads. This operation doesn’t require constructing derived MPI type.
* Each process tidies up, and then returns with the desired data in the buffers pointed to by the buf fields of the elements of the info[] array structure.

### H5Dwrite\_multi()

This routine performs collective or independent I/O writes to multiple datasets. In collective mode, all members of the file communicator associated with the HDF5 file must participate in the call. Each process loads the information required to perform each write into the H5D\_rw\_multi\_t structure, and passes an array of the structures through to *H5Dwrite\_multi()*.

The API function description is as shown below.

herr\_t H5Dwrite\_multi(hid\_t file\_id,

size\_t count,

struct H5D\_rw\_multi\_t info[],

hid\_t dxpl\_id);

Parameters:

* file\_id: file or group id for the location of datasets
* count: the number of datasets.
* Info: the array of dataset information and write buffers.
* dxpl\_id: dataset transfer property.

Returns:

* a non-negative value if successful; otherwise returns a negative value.

Brief description is as follows. Note that details have been omitted for brevity.

* Each process obtains the list of dataset info from the info[] array structure, and constructs a derived MPI type describing the sections of the HDF5 file to be written.
* For the collective I/O operation, each process end up calling *MPI\_File\_write\_at\_all()* to perform the desired writes.
* For the independent I/O operation, each process end up calling *MPI\_File\_write\_at()* to perform the desired writes. This operation doesn’t require constructing derived MPI type.

### H5Dclose\_multi()

This routine performs close and free resources used by *H5Dread\_multi()* or *H5Dwrite\_multi()* APIs.

A user can also close or free resources that are used by *H5Dread\_multi()* or *H5Dwrite\_multi()* manually. Thus, this routine is an optional to be used. This routine is added mainly to provide convenience for user.

The API function description is as shown below.

herr\_t H5Dclose\_multi(size\_t count,

H5D\_rw\_multi\_t info[]);

Parameters:

* count: the number of datasets.
* Info: the array of dataset information.

Returns:

* a non-negative value if successful; otherwise returns a negative value.

A user can set below values as bitwise to ‘*close\_flags*’ which is a member variable of *H5D\_rw\_multi\_t* struct. This close function will perform close ids and free buffer accordingly.

H5D\_MULTI\_CLOSE\_ALL (0x0000u) /\* close and free all, Default \*/

H5D\_MULTI\_CLOSE\_DATASET (0x0001u) /\* close dataset only \*/

H5D\_MULTI\_CLOSE\_MSPACE (0x0002u) /\* close mem space only \*/

H5D\_MULTI\_CLOSE\_FSPACE (0x0004u) /\* close file space only \*/

H5D\_MULTI\_CLOSE\_DATATYPE (0x0008u) /\* close datatype only \*/

H5D\_MULTI\_FREE\_BUF (0x0010u) /\* free data buffer only \*/

[1] Yang M and Koziol Q, 2006. Using collective IO inside a high performance IO software package—HDF5 Technical Report National Center of Supercomputing Applications

[2] Rob Latham, Chris Daley, etc., March 2012. A case study for scientific I/O: improving the FLASH astrophysics code, <http://iopscience.iop.org/1749-4699/5/1/015001/article>

# Revision History

|  |  |
| --- | --- |
| *August 28, 2012:* | Version 1 by Peter Cao. Circulated internally. |
| *Sep 27, 2012:* | Version 2: updated based on internal reviews. |
| Feb 15, 2013: | Version 3: Updated based on internal reviews. Revised APIs and related contents.  (HDFFV-8313) |
|  |  |
|  |  |