RFC: New HDF5 API routines for HPC Applications

Peter Cao

Quincey Koziol

The HDF5 library allows a data access operation to access one dataset at a time, whether access is collective or independent. 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. The proposed work is to bring the IO to a new level: a single IO call for multiple datasets.

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

The current HDF5 library does not support a single I/O call for multiple datasets. Using the proposed new functions, *H5Dread\_multi()* and *H5Dwrite\_multi()*, 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.

Parallel HDF5 (PHDF5) supports both independent and collective dataset access. When collective I/O is used, all processes that have opened the dataset may do collective data access by calling *H5Dread()* or *H5Dwrite()* on the dataset with the transfer property set for collective access. In many cases, collective dataset access using the MPIO VFD improves I/O performance by many multiples,[1] since data can be aggregated by MPI into large contiguous accesses to disk instead of small non-contiguous ones.

Currently, collective dataset access works on a single dataset. For example, if you access five datasets in a file, you will need at least five I/O calls. The proposed work is to use collective I/O to perform a single I/O operation for multiple datasets.

We propose to add two new functions to the HDF5 library: H5Dread\_multi() and H5Dwrite\_multi(). H5Dread\_multi() will perform a single collective read on one or more datasets and H5Dwrite\_multi() does the same for collective writes. The details of the functions are discussed in the following sections.

# 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 timestep. 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 collective IO on a single dataset. The major change for the new work is that we add data information from different datasets to the I/O mapping list and construct information for collective IO operations. The following example chart explains the implementation approach for the new API functions



## New API Functions

Two new functions, *H5Dread\_multi()* and *H5Dwrite\_multi()*, are proposed here.

### H5Dread\_multi()

This omnibus routine performs collective reads from multiple datasets. 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 a structure, and passes an array of such structures through to *H5Dread\_multi()*.

The structure used for this purpose is *H5D\_read\_multi\_t*, and is defined below:

struct H5D\_read\_multi\_t

{

*hid\_t* dataset\_id; /\* as per H5Dread() \*/

*hid\_t* mem\_type\_id; /\* as per H5Dread() \*/

*hid\_t* mem\_space\_id; /\* as per H5Dread() \*/

*hid\_t* file\_space\_id; /\* as per H5Dread() \*/

*void \** buf; /\* as per H5Dread() \*/

};

With the *H5D\_read\_multi\_t* in hand, we may declare *H5Dread\_multi()* as follows:

herr\_t H5Dread\_multi(size\_t count, struct H5D\_read\_multi\_t reads[],

hid\_t xfer\_plist\_id);

Very briefly, processing inside *H5Dread\_multi()* will be as follows. Note that all error checking has been omitted for brevity.

* Each process in the collective read scans the list of data set reads indicated by the reads[] array, and constructs a derived MPI type describing the sections of the HDF5 file to be read.
* Each process then calls *MPI\_File\_read\_all()* to perform the desired reads.
* On return from *MPI\_File\_read\_all()*, 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 reads[] array.

### H5Dwrite\_multi()

This omnibus routine performs collective writes to multiple datasets. 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 a structure, and passes an array of such structures through to *H5Dwrite\_multi()*.

The structure used for this purpose is *H5D\_write\_multi\_t*, and is defined below:

struct H5D\_write\_multi\_t

{

*hid\_t* dataset\_id; /\* as per H5Dwrite() \*/

*hid\_t* mem\_type\_id; /\* as per H5Dwrite() \*/

*hid\_t* mem\_space\_id; /\* as per H5Dwrite() \*/

*hid\_t* file\_space\_id; /\* as per H5Dwrite() \*/

*const void \** buf; /\* as per H5Dwrite() \*/

};

With the *H5D\_write\_multi\_t* in hand, we may declare *H5Dwrite\_multi()* as follows:

herr\_t H5Dwrite\_multi(size\_t count, struct H5D\_write\_multi\_t writes[],

hid\_t xfer\_plist\_id);

Very briefly, processing inside *H5Dwrite\_multi()* will be as follows. Note that all error checking has been omitted for brevity.

* Each process in the collective write scans the list of data set writes indicated by the writes[] array, and constructs a derived MPI type describing the sections of the HDF5 file to be written.
* Each process then calls *MPI\_File\_write\_all()* to perform the desired writes.
* On return from *MPI\_File\_write\_all()*, each process tidies up, and returns.

[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. |
|  |  |
|  |  |
|  |  |