RFC: Actual I/O Mode

Jacob Gruber

Christian Chilan

Allow a user to determine which type of I/O was performed after the completion of a requested parallel I/O call. This is not necessarily the same as what was requested.

# Introduction

Collective I/O, which is requested by the user via a data transfer property list (DXPL), can perform I/O according to several optimization schemes. The HDF5 library either chooses one based on a set of user-adjustable parameters, or a user can request an optimization directly.

These optimization schemes may not perform pure collective I/O. Some schemes analyze each chunk in a dataset individually, and may access some collectively and others independently. Thus some independent I/O may still occur even when a collective operation is requested.

Additionally, until recently the HDF5 library was unable to perform certain optimizations under specific circumstances, such as the regularity of the dataset or the implementation of MPI. In these cases, the library would either choose an alternate optimization or switch to independent I/O[[1]](#footnote-1).

Currently, there is no way to check which optimization was chosen or whether collective or independent I/O was performed. This RFC proposes extensions to the HDF5 library allowing the user to determine the optimization and I/O mode(s) used by each process in an I/O operation, but not at the level of individual chunks. The extensions will also allow the user to determine what caused the HDF5 library to skip collective I/O in the local process and among all processes, if that was the case.

# Description

## Description of Optimizations

At the inception of this project, the parallel I/O code was poorly documented, both internally and externally. Careful study of the code was required to understand implementation details and their motivations often remained opaque. While the internal documentation has improved significantly, there still is no external documentation. In light of this, some brief descriptions of the various optimizations available to the library as it performs parallel I/O are provided here.

As this section of HDF5 is being reworked, some of this discussion may soon be obsolete. However, while details may change, the general thrust should remain intact.

### General Parallel I/O Concerns

Before we discuss specific optimizations, we should note that in certain circumstances, collective I/O will not be attempted at all, even if requested, and HDF5 will perform independent I/O instead. The following conditions[[2]](#footnote-2) bring about this switch:

* Datatype conversions need to be performed
* Data transforms need to be performed
* I/O is using the MPI POSIX driver
* One of the dataspaces is neither simple nor scalar
* There are point selections in one of the dataspaces[[3]](#footnote-3)
* The dataset is neither contiguous nor chunked
* Any filters need to be applied

If all of these checks pass, HDF5 chooses a collective I/O optimization scheme. If the dataset is contiguous, collective I/O proceeds without further consideration. If the dataset is chunked, one of three optimization schemes will be chosen. If the average number of processes addressing each chunk is above some threshold (the threshold defaults to 0, but can be set by the user), HDF5 performs “Link Chunk I/O”. If the threshold is not reached, HDF5 performs “Multi Chunk I/O”. In addition, the user can request either Link Chunk I/O or “Multi Chunk No Opt I/O”, a second version of Multi Chunk I/O that performs less optimization. Consult the flowcharts at the end of this document for the details of this decision process. Brief descriptions of the various optimizations follow:

### Contiguous I/O

Contiguous I/O performs a collective read or write operation on a contiguously stored dataset, combining the selections supplied by the individual processes. Unlike some of the more complicated optimizations, Contiguous I/O will never switch to independent.

### Link Chunk I/O

In Link Chunk I/O, one MPI derived datatype is created that contains the selection of all chunks and one collective I/O operation is performed. Link chunk I/O will not switch to independent I/O.

### Multi Chunk I/O

In Multi Chunk I/O, each chunk is evaluated separately. If the chunk’s elements are selected by at least a user-specified fraction of the processes, collective I/O is performed on the chunk. Otherwise independent I/O is performed. Using this scheme, a process can perform independent I/O, collective I/O or a mixture of the two, and since each process may have a different selection, they may perform different types of I/O.

Consider an application with two processes reading a dataset with two chunks. Process 0 selects both chunks and Process 1 selects only Chunk 0. Thus, Chunk 0 is selected by 100% of processes and Chunk 1 is selected by 50% of processes. If the fraction threshold is set to 60%, Chunk 0 will be read collectively by both processes and Chunk 1 will be read independently by Process 0 only. Here, Process 0 will perform both collective and independent I/O while process 1 will perform only collective I/O.

### Multi Chunk I/O No Opt

In Multi Chunk I/O No Opt, each chunk is evaluated independently, but inter-process communication overhead is reduced by performing a simpler optimization. Let *n* be the minimum number of chunks in any one process's selection. If the index of the chunk in the current process’s selection is less than *n*, then collective I/O is performed. Otherwise, independent I/O is performed.

Like the previous optimization, it is possible to have processes disagree. In a two process application, if Process 0 selects 1 chunk and Process 1 selects two chunks, Process 0 will perform one collective I/O operation and Process 1 will perform one collective and one independent I/O operation.

## Design of Properties

To track the type of I/O performed, two properties are proposed: actual\_chunk\_opt\_mode, to track the optimization scheme chosen in chunked datasets and actual\_io\_mode, to track whether independent I/O, collective I/O or some mix of the two took place.

Two properties are proposed instead of one composite property because, even though most optimization schemes are limited in what type of I/O they can perform, almost all optimizations have multiple values for the actual I/O mode and most of these modes are shared among several optimizations.

The two properties are described in more detail in the following Reference Manual entries.

# New API Functions RM Entries

## H5Pget\_mpio\_actual\_chunk\_opt\_mode

Signature:

herr\_t H5Pget\_mpio\_actual\_chunk\_opt\_mode(hid\_t dxpl\_id, H5D\_mpio\_actual\_chunk\_opt\_mode\_t \* actual\_chunk\_opt\_mode)

Purpose:

Retrieves the type of chunk optimization that HDF5 actually performed on the last parallel I/O call. This is not necessarily the type of optimization requested.

Motivation:

A user can request collective I/O via a data transfer property list (DXPL) that has been suitably modified with H5Pset\_dxpl\_mpio. The operation can be optimized in several different ways, some of which also can be requested by the user. However, HDF5 may not be able to satisfy requests for specific optimizations and may choose a different optimization scheme. This property allows the user to track which optimization was actually used. Used in conjunction with H5Pget\_mpio\_actual\_io\_mode, this property allows the user to determine exactly what HDF5 did when attempting collective chunked I/O.

Description:  
H5Pget\_mpio\_actual\_chunk\_opt\_mode retrieves the type of chunk optimization performed when collective I/O was requested. This property is set before I/O takes place, and will be set even if I/O fails.

Valid values returned in actual\_chunk\_opt\_mode:

H5D\_MPIO\_NO\_CHUNK\_OPTIMIZATION  
No chunk optimization was performed. Either no collective I/O was attempted or the dataset wasn't chunked. *(Default)*

H5D\_MPIO\_MULTI\_CHUNK  
Each chunk was individually assigned collective or independent I/O based on what fraction of processes access the chunk. If the fraction is greater than the multi chunk ratio threshold, collective I/O is performed on that chunk. The multi chunk ratio threshold can be set using H5Pset\_dxpl\_mpio\_chunk\_opt\_ratio. The default value is 60%.

H5D\_MPIO\_MULTI\_CHUNK\_NO\_OPT  
Each chunk is assigned collective or independent I/O based on how many chunks were accessed before it. Collective I/O is performed on the first chunk in each selection, then the second, and so on until one process finishes, then the remaining processes perform independent I/O on the rest of their selections.

H5D\_MPIO\_LINK\_CHUNK   
Collective I/O is performed on all chunks simultaneously.

Parameters:

hid\_t dxpl\_id   
IN: Dataset transfer property list identifier

H5D\_mpio\_actual\_chunk\_opt\_mode\_t \*actual\_chunk\_opt\_mode   
OUT: The type of chunk optimization performed by HDF5.

Returns:

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

## H5Pget\_mpio\_actual\_io\_mode

Signature:

herr\_t H5Pget\_mpio\_actual\_io\_mode(hid\_t dxpl\_id,   
H5D\_mpio\_actual\_io\_mode\_t \* actual\_io\_mode)

Purpose:   
Retrieves the type of I/O that HDF5 actually performed on the last parallel I/O call. This is not necessarily the type of I/O requested.

Motivation:   
A user can request collective I/O via a data transfer property list (DXPL) that has been suitably modified with H5Pset\_dxpl\_mpio. However, HDF5 will sometimes ignore this request and perform independent I/O instead. This property allows the user to see what kind of I/O HDF5 actually performed. Used in conjunction with H5Pget\_mpio\_actual\_chunk\_opt\_mode, this property allows the user to determine exactly HDF5 did when attempting collective I/O.

Description:   
H5Pget\_mpio\_actual\_io\_mode retrieves the type of I/O performed on the selection of the current process. This property is set after all I/O is completed; if I/O fails, it will not be set.

Valid values returned in actual\_io\_mode:

H5D\_MPIO\_NO\_COLLECTIVE\_IO  
No collective I/O was performed. Collective I/O was not requested or collective I/O isn't possible on this dataset. (Default)

H5D\_MPIO\_CHUNK\_INDEPENDENT  
HDF5 performed one the chunk collective optimization schemes and each chunk was accessed independently.

H5D\_MPIO\_CHUNK\_COLLECTIVE  
HDF5 performed one the chunk collective optimization schemes and each chunk was accessed collectively.

H5D\_MPIO\_CHUNK\_MIXED  
HDF5 performed one the chunk collective optimization schemes and some chunks were accessed independently, some collectively.

H5D\_MPIO\_CONTIGUOUS\_COLLECTIVE   
Collective I/O was performed on a contiguous dataset.

**Note:**

All processes need not return the same value. For example, if I/O is being performed using the multi chunk optimization scheme, one process's selection may include only chunks accessed collectively, while another may include only chunks accessed independently and a third may involve both types. In this case, the first process will report H5D\_MPIO\_CHUNK\_COLLECTIVE while the second will report H5D\_MPIO\_CHUNK\_INDEPENDENT and the third H5D\_MPIO\_CHUNK\_MIXED.

Parameters:

hid\_t dxpl\_id  
IN: Dataset transfer property list identifier

H5D\_mpio\_actual\_io\_mode\_t \* actual\_io\_mode  
OUT: The type of I/O performed by this process.

Returns:   
Returns a non-negative value if successful; otherwise returns a negative value.

## H5Pget\_mpio\_no\_collective\_cause

Signature:

herr\_t H5Pget\_mpio\_no\_collective\_cause(hid\_t dxpl\_id,   
uint32\_t \* local\_no\_collective\_cause,   
uint32\_t \* global\_no\_collective\_cause)

Purpose:   
Retrieves local and global causes that broke collective I/O on the last parallel I/O call.

Motivation:   
A user can request collective I/O via a data transfer property list (DXPL) that has been suitably modified with H5Pset\_dxpl\_mpio. However, there are conditions that can cause HDF5 to forgo collective I/O and perform independent I/O. Such causes can be different across the processes of a parallel application. This function allows the user to determine what caused the HDF5 library to skip collective I/O locally, that is in the local process, and globally, across all processes.

Description:   
H5Pget\_mpio\_no\_collective\_cause serves two purposes. It can be used to determine whether collective I/O was used for the last preceding parallel I/O call. If not, it retrieves the local and global causes that broke collective I/O on that parallel I/O call. The properties retrieved by this function are set before I/O takes place and are retained even when I/O fails.

Valid values returned on the property are as follows; the numbers on the right are bitmask values:

H5D\_MPIO\_COLLECTIVE = 00000000  
Collective I/O was performed successfully. (Default)

H5D\_MPIO\_SET\_INDEPENDENT = 00000001  
Collective I/O was not performed because independent I/O was requested.

H5D\_MPIO\_DATATYPE\_CONVERSION = 00000010  
Collective I/O was not performed because datatype conversions were required.

H5D\_MPIO\_DATA\_TRANSFORMS = 00000100  
Collective I/O was not performed because data transforms needed to be applied.

H5D\_MPIO\_SET\_MPIPOSIX = 00001000  
Collective I/O was not performed because the selected file driver was MPI-POSIX.

H5D\_MPIO\_NOT\_SIMPLE\_OR\_SCALAR\_DATASPACES = 00010000  
Collective I/O was not performed because one of the dataspaces was neither simple nor scalar.

H5D\_MPIO\_POINT\_SELECTIONS = 00100000  
Collective I/O was not performed because there were point selections in one of the dataspaces.

H5D\_MPIO\_NOT\_CONTIGUOUS\_OR\_CHUNKED\_DATASET = 01000000  
Collective I/O was not performed because the dataset was neither contiguous nor chunked.

H5D\_MPIO\_FILTERS = 10000000  
Collective I/O was not performed because filters needed to be applied.

The above name/value pairs are members of the H5D\_mpio\_no\_collective\_cause\_t enumeration.

**~~Note:~~**

~~Section 8 illustrates the current decision process that determines whether collective I/O is possible.~~ Each process determines whether it can perform collective I/O and broadcasts the result. Those results are combined to make a collective decision; collective I/O will be performed only if all processes can perform collective I/O.

If collective I/O was not used, the causes that prevented it are reported by individual process by means of an enumerated set. The causes may differ among processes, so H5Pget\_mpio\_no\_collective\_cause returns two property values. The first value is the one produced by the local process to report local causes. This local information is encoded in an enumeration, the H5D\_mpio\_no\_collective\_cause\_t described above, with all individual causes combined into a single enumeration value by means of a bitwise OR operation. The second value reports global causes; this global value is the result of a bitwise-OR operation across the values returned by all the processes. [[4]](#footnote-4)

Parameters:

hid\_t dxpl\_id  
IN: Dataset transfer property list identifier

uint32\_t \* local\_no\_collective\_cause  
OUT: A enumerated set value indicating the causes that prevented collective I/O in the local process.

uint32\_t \* global\_no\_collective\_cause  
OUT: An enumerated set value indicating the causes across all processes that prevented collective I/O.

Returns:   
Returns a non-negative value if successful; otherwise returns a negative value.

# Notes

The actual\_chunk\_opt\_mode and actual\_io\_mode properties are not strictly independent and not all combinations of the properties are possible. In a collective operation, the values available to actual\_io\_mode are dependent on the value of actual\_chunk\_opt\_mode.

The possible combinations are:

|  |  |  |
| --- | --- | --- |
|  | H5D\_MPIO\_NO\_CHUNK\_OPTIMIZATION | H5D\_MPIO\_NO\_COLLECTIVE\_IO H5D\_MPIO\_CONTIGUOUS\_COLLECTIVE |
|  | H5D\_MPIO\_MULTI\_CHUNK | H5D\_MPIO\_NO\_COLLECTIVE\_IO H5D\_MPIO\_CHUNK\_INDEPENDENT H5D\_MPIO\_CHUNK\_COLLECTIVE H5D\_MPIO\_CHUNK\_MIXED |
|  | H5D\_MPIO\_MULTI\_CHUNK\_NO\_OPT | H5D\_MPIO\_NO\_COLLECTIVE\_IO H5D\_MPIO\_CHUNK\_INDEPENDENT H5D\_MPIO\_CHUNK\_COLLECTIVE H5D\_MPIO\_CHUNK\_MIXED |
|  | H5D\_MPIO\_LINK\_CHUNK | H5D\_MPIO\_CHUNK\_COLLECTIVE |

Also, at the present time, there is no way of telling whether a specific chunk was read collectively or independently.

# Usage

If a user is experiencing difficulties with parallel I/O, support personnel could use these properties to get extra diagnostic information. Additionally, a user could use these functions to ensure that a specific optimization is chosen to prevent unexpected slowdown of parallel applications.

# Example

The following pseudo code illustrates the use of the actual I/O mode properties in determining whether a process performed collective I/O, independent I/O or both in an application with three processes. In this example Process 0 will report collective I/O, Process 1 will report both collective and independent I/O and Process 2 will report independent I/O. This example is contrived, but it isn’t too hard to imagine that if the processes’ selections were determined by a computation or user input, a similar scenario might arise.

H5D\_mpio\_actual\_chunk\_opt\_mode\_t actual\_chunk\_opt\_mode;

H5D\_mpio\_actual\_io\_mode\_t actual\_io\_mode;

<set up mpi\_rank and mpi\_size>

<open file collectively>

<create space>

<create dataset with three chunks>

<create file and memory spaces>

if (mpi\_rank == 0) {

<select hyperslab in Chunk 0>

} else if (mpi\_rank == 1) {

<select hyperlab in Chunk 0 and Chunk 1>

} else if (mpi\_rank == 2) {

<select hyperslab in Chunk 2>

}

dxpl = H5Pcreate(H5P\_DATASET\_XFER);

H5Pset\_dxpl\_mpio(dxpl, H5FD\_MPIO\_COLLECTIVE);

/\* Set the average number of processes per chunk required for Link   
 \* Chunk I/O (or, conversely, the upper limit for Multi Chunk I/O).   
 \* Here, Link Chunk I/O will only occur if the average number of   
 \* processes per chunk is twice mpi\_size. This is, of course,   
 \* impossible, and effectively forces Multi Chunk I/O.   
 \*/

H5Pset\_dxpl\_mpio\_chunk\_opt\_num(dxpl, mpi\_size\*2);

/\* Set the threshold fraction of processes per chunk for  
 \* collective I/O. Here, collective I/O will only occur   
 \* if a process is selected by at least 40% of processes.  
 \*/

H5Pset\_dxpl\_mpio\_chunk\_opt\_ratio(dxpl, 40);

H5Dwrite(dataset, data\_type, mem\_space, file\_space, dxpl, buffer);

H5Pget\_mpio\_actual\_io\_mode(dxpl, &actual\_io\_mode);

H5Pget\_mpio\_actual\_chunk\_opt\_mode(dxpl, &actual\_chunk\_opt\_mode);

/\* Check properties against expected values \*/

assert(actual\_chunk\_opt\_mode == H5D\_MPIO\_MULTI\_CHUNK);

if (mpi\_rank == 0) {

assert(actual\_io\_mode == H5D\_MPIO\_CHUNK\_COLLECTIVE);

} else if (mpi\_rank == 1) {

assert(actual\_io\_mode == H5D\_MPIO\_CHUNK\_MIXED);

} else if (mpi\_rank == 2) {

assert(actual\_io\_mode == H5D\_MPIO\_CHUNK\_INDEPENDENT);

}

The next example illustrates the use of the no collective cause property in determining why collective I/O was skipped. In this case, a file is open using the MPI-POSIX driver and a collective write operation is requested. The returned property value indicates that collective I/O could not be performed because of the MPI-POSIX driver is in use.

H5D\_mpi\_no\_collective\_cause\_t local\_no\_collective\_cause;

H5D\_mpi\_no\_collective\_cause\_t global\_no\_collective\_cause;

<set up mpi\_rank and mpi\_size>

fapl = H5Pcreate(H5P\_FILE\_ACCESS);

H5Pset\_fapl\_mpiposix(fapl, MPI\_COMM\_WORLD, 0);

<open file collectively>

<create space>

<create contiguous dataset>

<create file and memory spaces>

<hyperslab selection divides dataset equally among processes>

dxpl = H5Pcreate(H5P\_DATASET\_XFER);

H5Pset\_dxpl\_mpio(dxpl, H5FD\_MPIO\_COLLECTIVE);

H5Dwrite(dataset, data\_type, mem\_space, file\_space, dxpl, buffer);

H5Pget\_mpi\_no\_collective\_cause(dxpl, &local\_no\_collective\_cause, &global\_no\_collective\_cause);

/\* check property against expected value \*/

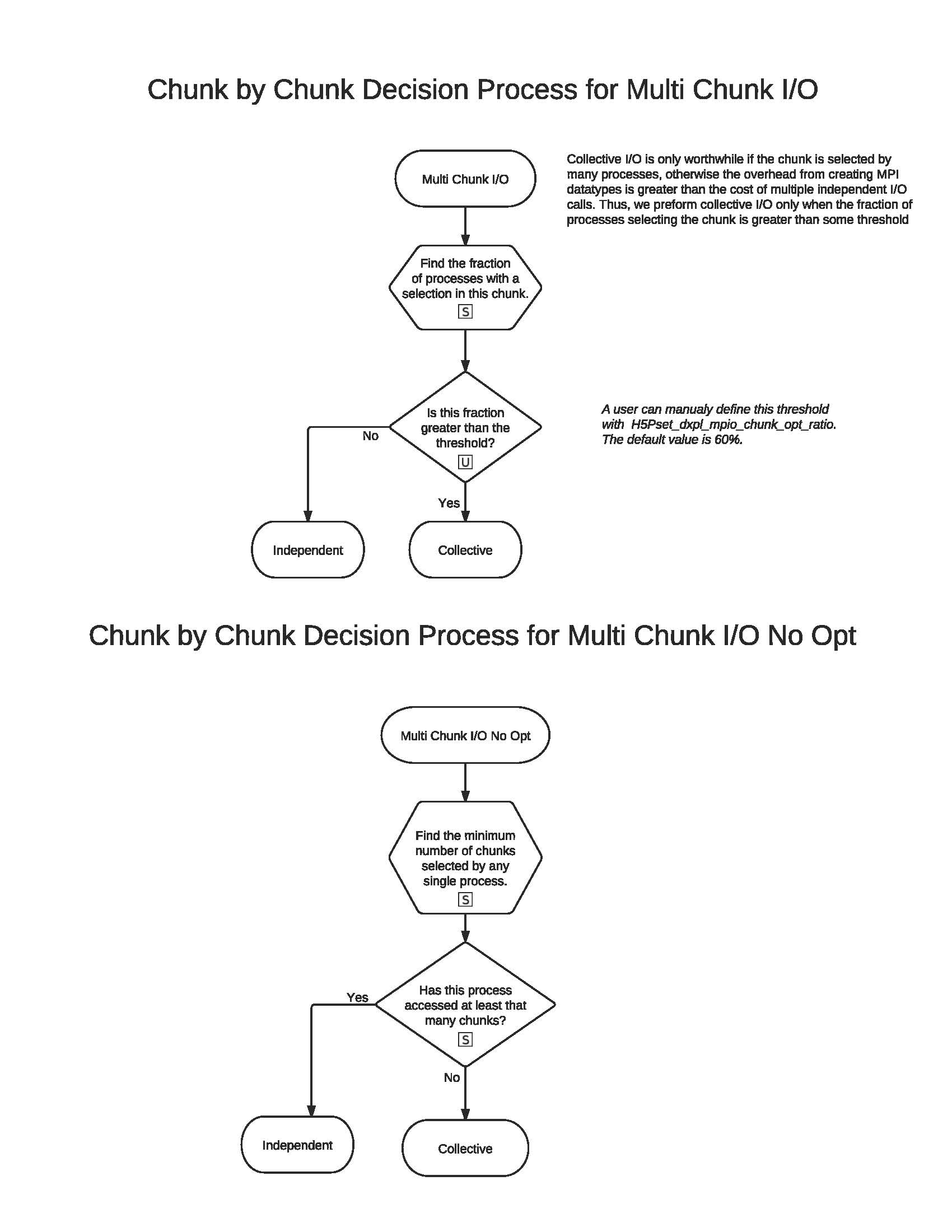
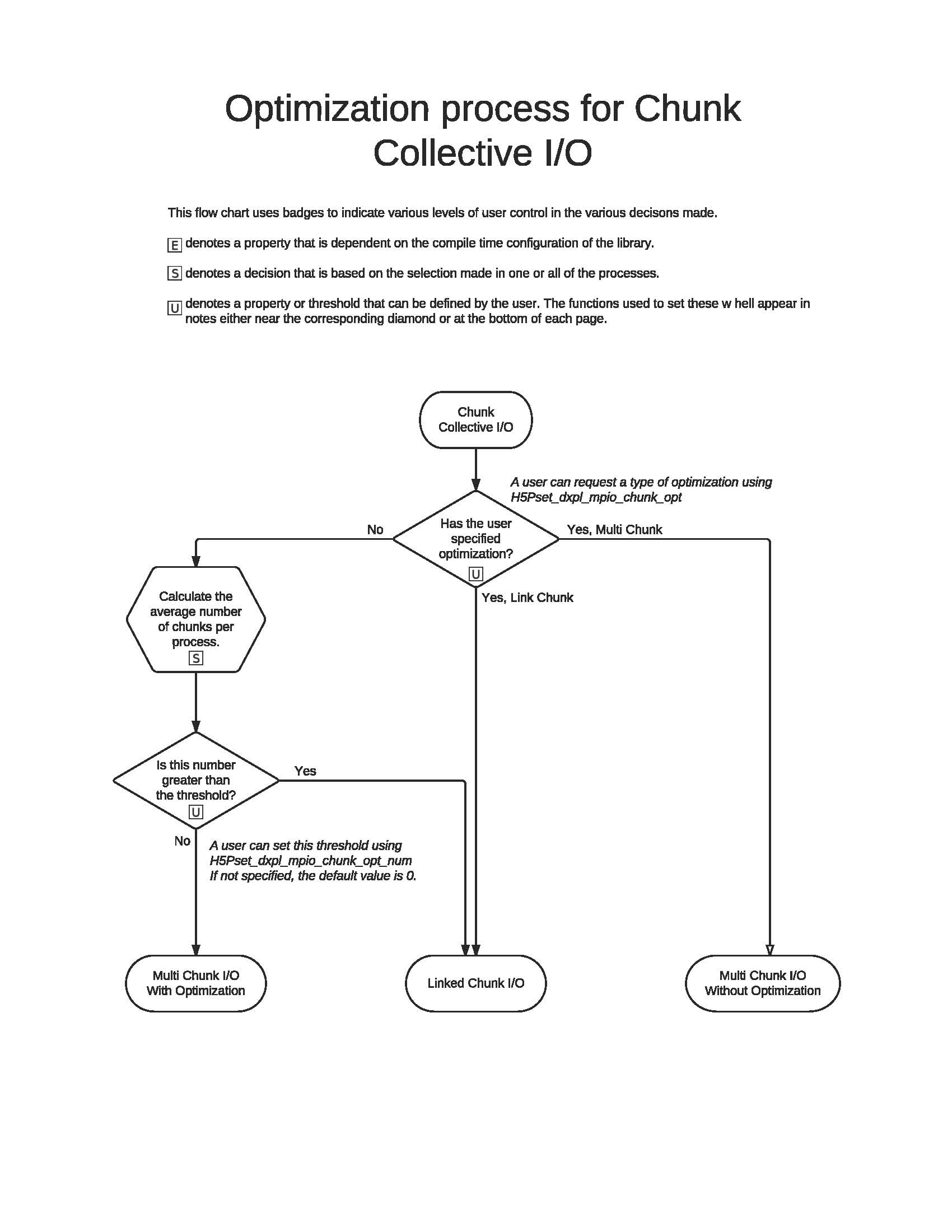
assert(local\_no\_collective\_cause == H5D\_MPIO\_SET\_MPIPOSIX);

assert(global\_no\_collective\_cause == H5D\_MPIO\_SET\_MPIPOSIX);

# Recommendation

The HDF5 API extensions proposed in this RFC have been implemented, but the parallel I/O code is changing. Thus the details of this RFC and the associated code will probably need to be revisited.

# Optimization Flowcharts



**CollectiveIO6.pdf**

# **RFC Revision History**

|  |  |
| --- | --- |
| August 04, 2011 | Version 1 posted for public comment. Comments should be sent to gruber1@hdfgroup.org |
| August 22, 2011 | Minor tweaks after comments from Quincey. |
| September 6, 2012 | Minor update for H5Pget\_mpio\_no\_collective\_cause section. (Property name changes, local cause change.) |

1. Though these cases have been removed, I think they still deserve mention, in order to better understand the initial motivation for this feature. This paragraph my also be the only documentation of this behavior. [↑](#footnote-ref-1)
2. Some of these conditions are pretty opaque to me, and my descriptions are little more than educated guesses. The decision process is illustrated in a flowchart in Section 8. If you need more detail, look at H5Dmpio.c, specifically in H5D\_mpio\_opt\_possible. [↑](#footnote-ref-2)
3. Allowing collective I/O on point selections is actively being worked on and should be supported soon. [↑](#footnote-ref-3)
4. Note to developers: Section 8 of the RFC illustrates the current process for determining whether collective I/O is possible. Each process determines a bitmask value indicating whether the process can perform collective I/O and, if not, all the causes that prevent it. Each processes then broadcasts its binary value so that a consensus may be determined; collective I/O will be performed only if all the processes can perform collective operations.

   The value determined by each process is encoded in an enumerated set. Since the causes may be different among processes, H5Pget\_mpio\_no\_collective\_cause returns two property values. The first value is the local value indicating the causes that prevent collective I/O in the local process. The second value is the result of an all-reduce bitwise-OR operation across the values returned by all of the processes. In this way, the second value consolidates the causes that prevented collective I/O globally across all processes. [↑](#footnote-ref-4)