PERFORMANCE COMPARISON OF COLLECTIVE I/O AND INDEPENDENT I/O WITH DERIVED DATATYPES

Christian M. Chilan
Kent Yang
2006-6-23

I. Introduction

1. Independent IO and Collective IO

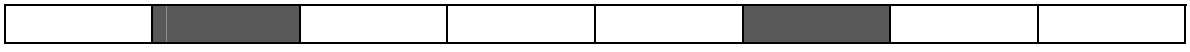
Parallel HDF5 uses the MPI-IO to allow multiple processors to write to one file on parallel systems. MPI-IO supports two ways of doing this: Independent IO and Collective IO. Independent IO means that each process does IO independently, while Collective IO requires all processes to participate when doing IO operations. The advantage of collective IO is that it allows MPI-IO to do optimization to improve IO performance. This is because system IO calls on most operating systems can only handle contiguous data in a file. For non-contiguous data, they must read or write in many small IO accesses, resulting in very poor IO performance.

Using the independent IO option means that each process does its own IO, so using independent IO without any optimization is just like doing general IO with many processes. If an application is only handling contiguous data, this will generally result in acceptable performance. However, for many applications each process needs to access noncontiguous data and performance will be poor. On the other hand, the MPI-IO library can optimize these accesses and improve performance by using the MPI-IO function call MPI_FILE_SET_VIEW and collective IO. Essentially, MPI-IO will assemble a big contiguous IO collectively by combining the noncontiguous data layout of each process. This is explained in greater detail in Chapter 3 of Using MPI-2[4].

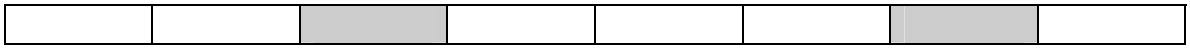
As a very simple example, suppose we have four processes with each process's view of the data as follows:

P0's view

P1's view



P2's view



P3's view

When doing independent IO, since the each process's view is noncontiguous, writing what is essentially four blocks on the above chart will require 8 individual IO access to the disk. However, when using collective IO, the IO access to the disk can be illustrated as follows:

P0 P1 P2 P3

This layout is contiguous. With an appropriate parallel file system, the previous 8 IOs can become a single IO operation. In real applications, of course, collective IO in MPI-IO can handle much more complicated cases.

In the current version of HDF5 I/O operations can be carried out in independent or collective mode, corresponding to the access types provided by MPI-IO.

2. Collective VS collective IO

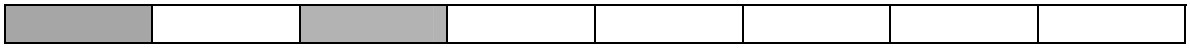
In the world of parallel HDF5, sometimes "collective" and "collective IO" are used interchangeably. However, "collective IO" is a small subset of "collective". In fact, "collective IO" really boils down to two MPI-IO functions: <MPI_File_write_all> and <MPI_File_read_all> within parallel HDF5. There are dozens of collective calls in the MPI world.

II. Independent I/O with Derived Datatype

1. Another I/O Option

MPI-IO also provides another technique called data-sieving technique [6] to improve performance with independent I/O. This technique provides a buffer to hold several non-contiguous, non-interleaved small I/O accesses and hopefully does one I/O in the MPI-IO layer. This can be illustrated as follows:

P0's view



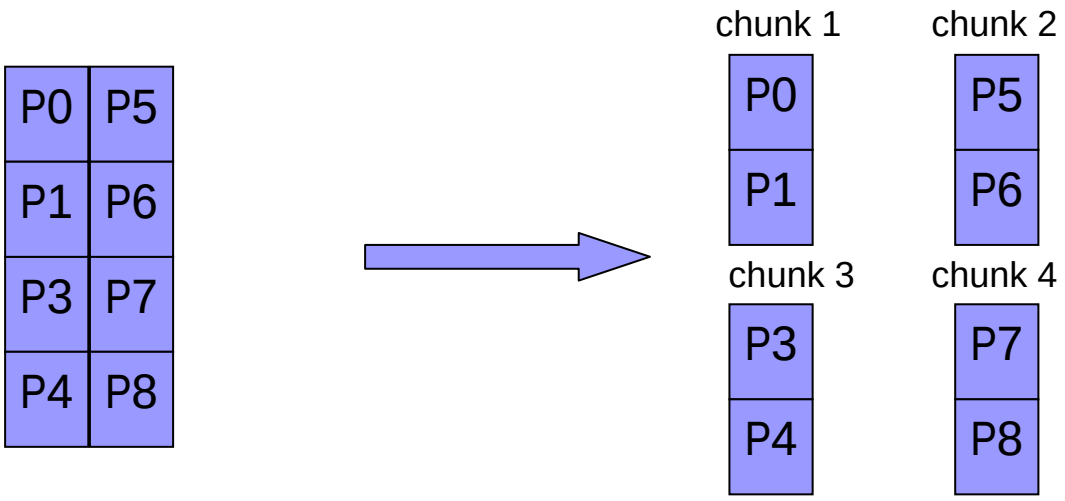
P1's view

Without using data sieving, process 0 may need two IO accesses to do IO independently. With data sieving, only one IO is needed. However, this performance gain doesn't come for free--it requires the application to describe the file layout pattern clearly and pass this information into the MPI-IO layer. Currently the only way to do this is by using an MPI derived datatype. Furthermore, there are two restrictions to this approach. The first is that an MPI collective call, MPI_file_set_view, has to be used to pass the file layout

information to the MPI-IO layer. This means although the application is doing independent IO it needs to do a collective call beforehand. This means that an application that wants to use independent IO with data sieving is actually in the "collective mode" category instead of the "independent mode" category. This is very important in order to understand why in HDF5 `H5FD_MPIO_COLLECTIVE` **`must`** be set in the MPIO dataset transfer property list before using `independent IO` with an MPI derived datatype. The second restriction to using a derived datatype is that it should NEVER be used for WRITING the data in the non-contagious interleaved case since this can cause data corruption!

2. Why not always use collective IO?

Collective IO is not free. Executing a collective IO call may require extra communication overhead in addition to the collective call MPI_File_set_view. Furthermore, it is very possible that only a small number of processors will participate in IOs in chunked storage when an application issues a collective IO request to the HDF5 library. This can be illustrated in the following chart:



Eight processors are used to do IOs for this dataset, which is divided into four chunks. Only two processors participate in IOs for each chunk, but the other six processors must still communicate with them when using collective IO. Depending on different MPI-IO implementations, it is possible that MPI-IO may also demand the other six processors to do IOs for this chunk. This is likely to cause poor performance compared with independent IO.

The purpose of the performance study is to verify whether independent IO with an MPI derived datatype(DDT) can provide better performance than collective IO inside HDF5.

To summarize the target of comparison, we use the following tables to describe the performance behaviors of independent IO, independent IO with DDT and Collective IO among the different file layouts. We will put question marks for the behaviors we don't know, and we will try to verify our hypothesis at the end of the table.

	Independent IO	Independent IO with DDT	Collective IO
contiguous	Good	Good	Good
Non-contiguous non-interleaving	Not Good	Good?	Good?
Non-contiguous interleaving	Not Good	Not Good?	Good

Table 1: Read Performance among different MPI-IO modes

	Independent IO	Independent IO with DDT	Collective IO
contiguous	Good	Good	Good
Non-contiguous non-interleaving	Not Good	Good?	Good?
Non-contiguous interleaving	Not Good	Should not use	Good

Table 2: Write Performance among different MPI-IO modes

In order to verify this hypothesis, we perform testing with non-interleaved selections in Bluesky, the NCAR IBM Power4 SP cluster, using two categories.

The first category models the common case in which all the processors of the communicator participate in the I/O operation. In the second category, we want to determine the effect on performance of using only a subset of processors out of 64 processors in the communicator. In all tests, we ran the test three times and reported the best result.

III. Testing with full processor participation

In these tests, we compare the performance of collective I/O and independent I/O access with derived datatypes for the common case in which all processors participate in the I/O operations. The configuration and geometry are shown in the following Table 3 and in Figure 1. The type of each element is a char. The dataset is 2-D, with height equal to Buffer_dim and length Dset_dim. We explicitly set the Dset_dim equal to the number of processor multiplied by Buffer_dim so that each processor selects a region of size Buffer_dim*Buffer_dim.

Ind-dd represents “independent with derived datatype”. Coll represents “collective.” The aggregate bandwidth(MB/second) is used for performance comparison. This is the y-axis for figure 2-5.

Test	Processors	Buffer size per processor(bytes) (Buffer_dim)	The whole data size per processor(bytes) (Dset_dim)
16, 1K*1K	16	1K x 1K	1K x 16K
16, 2K*2K	16	2K x 2K	2K x 32K
32,1K*1K	32	1K x 1K	1K x 32K

Table3 Test parameters

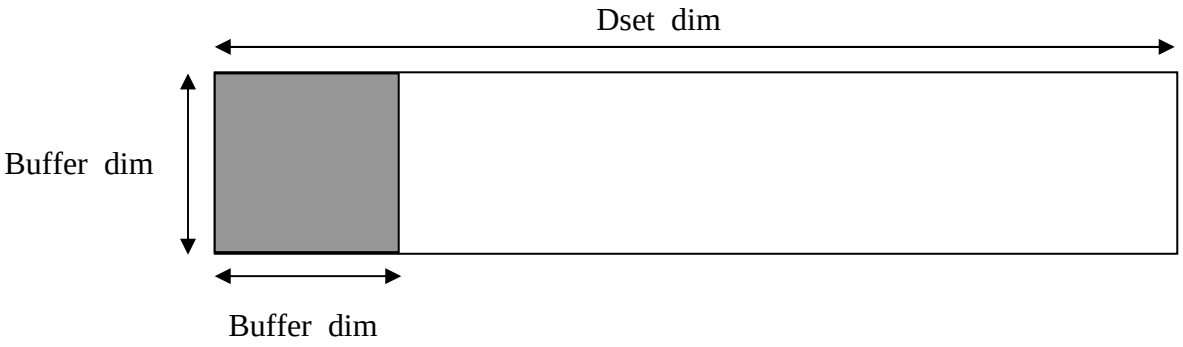


Figure1 the selection pattern per processor per I/O operation. The shaded area here is the selection.

The shaded area in Figure 1 represents the processor selection during the first I/O operation. During the subsequent operations, the shaded area shifts to right so that it covers the entire area of the dataset.

The results of our testing are shown in Figure 2. The aggregate bandwidth (y-axis in MB/second) is used for performance comparison. Note that the benefit of using independent I/O access with MPI DDT is an improvement in performance in READ operations. However, we see that this advantage decreases as the region not selected becomes much larger than the processor selection per I/O operation.

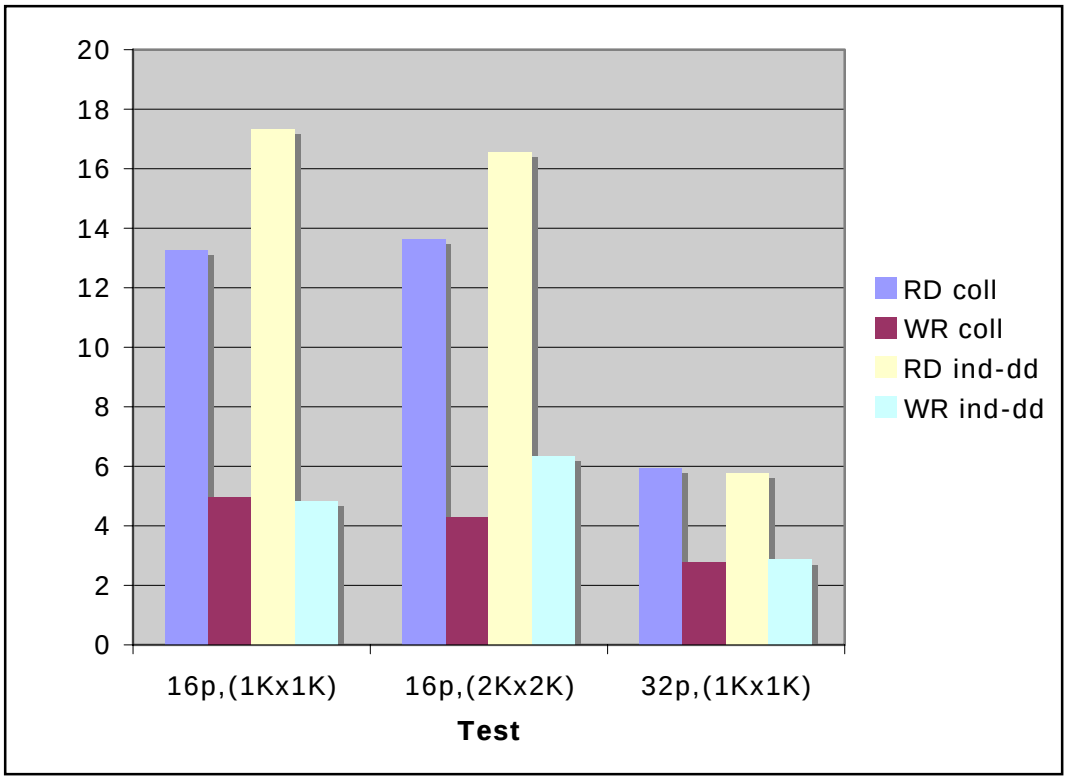


Figure 2 Performance of tests with full processor participation

IV. Testing with subset of processors

In these tests, we wanted to determine the performance impact of using only a small subset of processors to execute I/O operations. The total number of processors is 64 but the actual number of processors that perform I/O for a given test varies as shown in the x-axis of Figure 4 and 5.

The selection pattern per processor is shown in Figure 3. The type of each element is an integer.

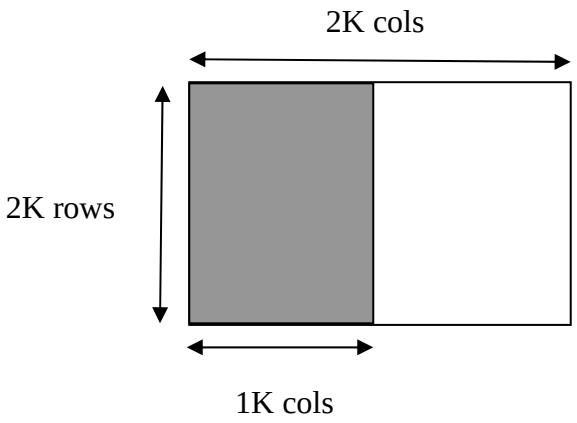


Figure 3 the selection pattern per processor

The results of our testing are shown in Figures 4 and 5. As we see, independent access with derived datatypes always provides better performance than collective access. This is more evident when the subset of processors is much smaller than the total number of processors in the communicator.

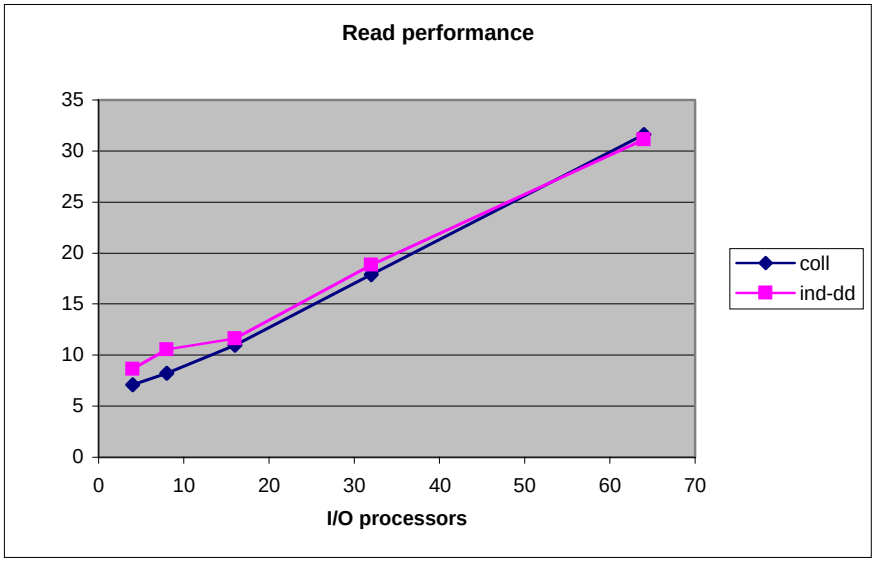


Figure 4 Read performance using a subset of processors for I/O

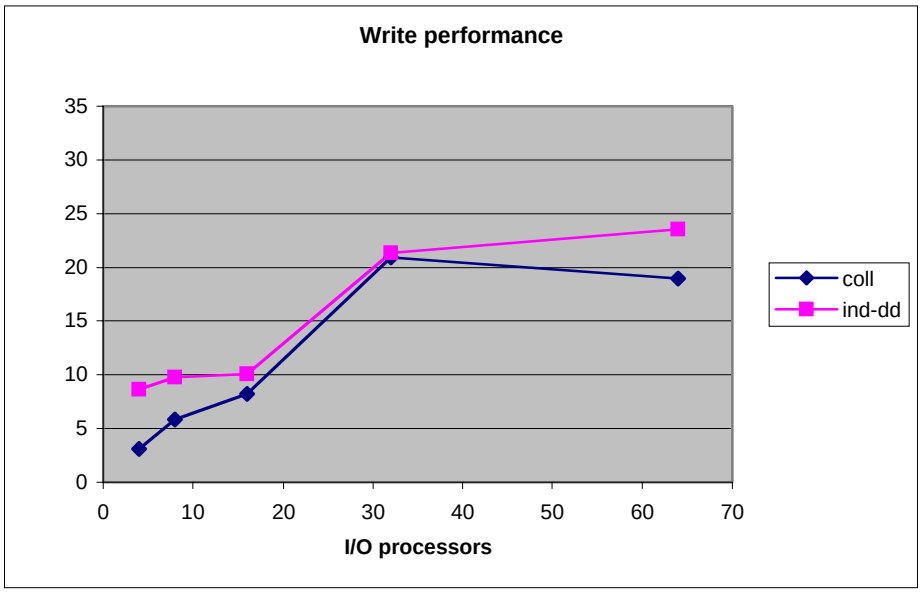


Figure 5 Write performance using a subset of processors for I/O

V. Conclusions

Since we did not find a case in which independent I/O access with derived datatypes reduces the performance significantly with respect to collective I/O operations, we believe that it is a valid option to include in HDF5. To use collective I/O inside HDF5 has some restrictions. For example, the collective I/O cannot be done if the compression filter or data conversion are enabled inside HDF5. The condition to use independent I/O access with derived datatypes inside HDF5 is the same as the condition to use collective I/O inside HDF5.

The magnitude of the performance improvement of independent IO access with derived datatypes depends on the size of the selection for a given IO operation relative to the dataset size. Read performance is good when all processors are participating in IO and can sometimes be 20% better than collective IO. When only small number of processors participate in IO, the performance of independent I/O access with derived datatype also improves significantly compared with collective I/O access.

	Independent IO	Independent IO with DDT	Collective IO
Contiguous	Good	Good	Good
Non-contiguous non-interleaving	Not Good	Good(possibly better than collective)	Good
Non-contiguous interleaving	Not Good	Not Good?	Good

Table 4: Read Performance among different MPI-IO modes

	Independent IO	Independent IO with DDT	Collective IO
Contiguous	Good	Good	Good
Non-contiguous non-interleaving	Not Good	Good(possibly better than collective)	Good
Non-contiguous interleaving	Not Good	Should not use	Good

Table 5: Write Performance among different MPI-IO modes

Appendix:

The new API for doing independent IO with DDT

Name: H5Pset_dxpl_mpio_collective_opt
Signature:

```
herr_t H5Pset_dxpl_mpio_collective_opt
       (hid_t dxpl_id, H5FD_mpio_collective_opt_t opt_mode)
```

Purpose:

Applications that set the data transfer property list to `H5FD_MPIO_COLLECTIVE` can set a flag in this API to use MPI-IO independent I/O functions inside HDF5. This API allows control of the low-level type of I/O while maintaining the same collective interface at the application level.

Description:

This API is an optional API. It **should only be used** when **`H5FD_MPIO_COLLECTIVE`** is set through data transfer API `H5Pset_dxpl_mpio`.

When the application sets the flag to `H5FD_MPIO_INDIVIDUAL_IO, the library will use low-level MPI independent I/O functions. Otherwise, collective I/O functions are used. The library will do collective I/O if this API is not called.`

Valid flags are as follows:

`H5FD_MPIO_COLLECTIVE_IO`
: Use collective I/O access(default)

`H5FD_MPIO_INDIVIDUAL_IO`
: Use independent I/O access

Parameters:

hid_t dxpl_id in: Data transfer property list identifier

H5FD_mpio_collective_opt_t opt_mode
in: The flag to determine the usage of collective I/O or independent I/O.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.