# Parallel HDF5 Hints

MuQun Yang & Quincey Koziol
NCSA HDF group

## I. Some definitions

According to HDF5 user's guide [1], hyperslabs are portions of datasets. A hyperslab selection can be a logically contiguous collection of points in a dataspace, or it can be regular pattern of points or blocks in a dataspace. Four parameters are required to describe a completely general hyperslab. Each parameter is an array whose rank is the same as that of the dataspace. The parameters are shown in Table 4.

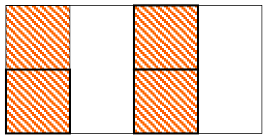
**Table 4**

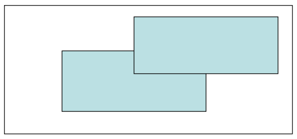
| **Parameter** | **Definition** |
|---|---|
| start | A starting location in the array for the hyperslab. |
| stride | The number of elements to separate each element or block to be selected. If the *stride* parameter is set to NULL, the stride size defaults to 1 in each dimension (i.e., no elements are skipped). |
| count | The number of elements or blocks to select along each dimension. |
| block | The size of the block selected from the dataspace. If the block parameter is set to NULL, the block size defaults to a single element in each dimension, as if the block array was set to all 1s. |

Based on the above definitions, we will define singular and regular hyperslabs.

1. regular hyperslab

A regular hyperslab is a hyperslab that generated inside an HDF5 program with only one H5Sselect_hyperslab routine call for the selected data space. Please refer to HDF5 reference manual [2] for the description of H5Sselect_hyperslab. The following two figures show illustrations of one regular hyperslab and one non-regular hyperslab.



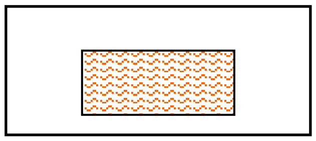


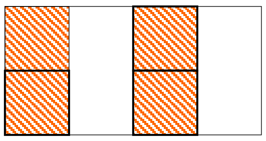
Regular hyperslab selection
(One H5Sselect_hyperslab call)

irregular hyperslab selection
(Two H5Sselect_hyperslab calls)

2. Singular hyperslab

A singular hyperslab is a special type of regular hyperslab of which the count of all dimensions is always 1. The following two figures show the illustration of one singular hyperslab and one non-singular hyperslab.





Singular hyperslab selection
(count =1 for both dimensions)

non-singular hyperslab selection
(count = 2 for X- dimension)

3. MPI-IO

It is the parallel I/O part of MPI-2. The specification was finalized in July 1997. According to [3], MPI-IO includes the following features:

- Noncontiguous access in both memory and file,
- Collective I/O operations,
- Use of explicit offsets to avoid separate seeks,
- Both individual and shared file pointers,
- Nonblocking I/O,
- Portable and customized data representations,
- Hints for the implementation of file system.

4. MPI Derived Data Type

The material describing MPI derived data type here is from the tutorial "Derived Data Types with MPI"[4].

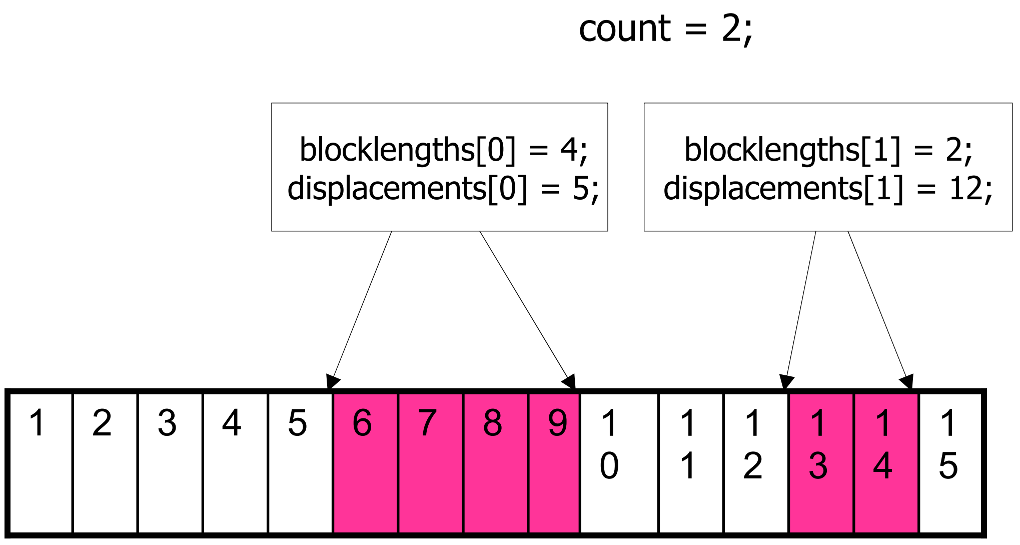
It is built from the basic MPI datatypes; it consists of sequence of basic datatypes and displacements. The reason to build MPI derived datatype is to provide a portable and efficient way to describe non-contiguous or mixed types in a message. MPI provides several routines to create MPI derived data type. They are MPI_Type_contiguous, MPI_Type_vector, MPI_Type_indexed and MPI_Type_struct.

Here we just show the usage of MPI_Type_indexed. There are five parameters for this routine: count, blocklens[], offsets[], oldtype and newtype.
Count, blocklens, offsets and oldtype are inputting parameters;
Count is the number of blocks to be added.
Blocklens are number of elements in block – an array of length count.
Offsets are displacements for each block – an array of length count.
Oldtype is the datatype of each element.
The newtype will return the pointer for new derived type.

The C routine of MPI_Type_indexed is

```
MPI_Type_indexed(count,blocklengths,displacements,MPI_INT,&indextype);
```

The following figure shows an example of describing MPI_Type_Indexed.

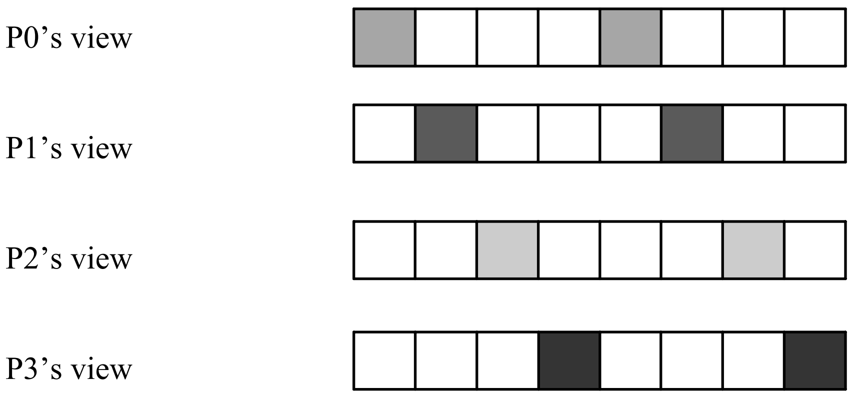


5. Independent IO and Collective IO

   Independent IO means that each process can do IO independently. It should not depend on or be affected by other processes.
   Collective IO is a way of doing IO defined as MPI-IO standard; contrary to independent IO, all processes must participate in doing IO. MPI-IO can do optimization to improve IO performance by using MPI_FILE_SET_VIEW routine with collective IO.

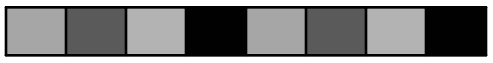
   The following figures show examples with 4 processes.

   1) With independent IO



When doing independent IO, for worst case it may require 8 individual IO access.

2) With collective IO



It may only need one IO access to the disk. Check [5] and the reference of that report for more information.

## II. Parallel HDF5 application software stacks

| |
|---|
| Applications |
| **Parallel HDF5** |
| **MPI-IO(ROM-IO, etc.)** |
| Parallel File System(GPFS, PVFS, Lustre) |
| Hardware(Myrinet, infinite band etc.) |

There are several layers that can affect performances for the parallel HDF5 application according to the table. In this document, we will only focus on the performance hint on how to wisely use parallel HDF5 to gain better performance through MPI-IO.

## III. Functionality of Parallel HDF5

There are three different storage layouts for raw data in HDF5. They are contiguous storage, chunking storage and compact storage.

Regardless of performance issue, users can choose to use either collective IO or independent IO. From parallel HDF5's perspective, there are no restrictions to enforce applications not to use either option.

## IV. Internal implementations of parallel HDF5

Since internal implementations of parallel HDF5 can help understanding performance issue, here we will give an overview of ideas of current implementation inside parallel HDF5. These descriptions only reflect the current implementation (HDF5 1.6.3 release) and should not be treated as the final implementation since HDF5 is still in the process of being developed.

1. Raw data with contiguous storage
   Parallel HDF5 supports both independent IO and collective IO.

   1) Collective IO

   For regular hyperslab selection, parallel HDF5 uses MPI derived data type routine MPI_TYPE_VECTOR to build a MPI derived data type. Through using MPI_FILE_SET_VIEW and the MPI derived data type, the potential improvement of IO performance through using MPI-IO will be accomplished. For non-regular hyperslab selection, parallel HDF5 uses independent IO internally for this option.

   2) Independent IO

   Parallel HDF5 supports independent IO for any hyperslab selections.

2. Raw data with chunking storage

   1) Collective IO

   The current parallel HDF5 implementation can do collective IO in chunking storage with more restrictions. If the application setting matches either case of the two special cases below, the collective IO is accomplished.

   The first case is: hyperslab selection in each process must be regular and all hyperslab selections must be within one chunk.

   The second case is: hyperslab selection in each process must be singular and the number of chunks that covers the hyperslab selection for each process must be equal.

Internally, the implementation for these two cases will call routines that generate MPI derived data type in contiguous storage to accomplish collective IO. For all other cases, parallel HDF5 uses independent IO for chunking storage.

2) Independent IO

Parallel HDF5 supports independent IO for any hyperslab selections.

3. Raw data with compact storage

The current parallel HDF5 implementation does not use parallel IO for this storage. Applications should assure the consistency of data among processes. In other words, IO access to data in a dataset should be exactly the same for all processes.

4. Metadata IO

The current parallel HDF5 implementation does not use parallel IO for metadata. Applications should assure the consistency of metadata among processes. In other words, IO access to metadata should be exactly the same for all processes.

## V. Performance Hints for Parallel HDF5 applications

1. General hint

Use collective IO option of parallel HDF5 in your application if possible!

2. Collective IO hints for contiguous storage

Since parallel HDF5 currently only supports collective IO for regular hyperslab, so if possible, always use ***regular hyperslab*** selection in your application to take advantage of the collective IO feature inside HDF5.

3. Collective IO hints for chunking storage

According to section IV part 2, the current parallel HDF5 implementation only supports two special cases for collective IO for chunking storage.

Case 1 requires that

- hyperslab selection in each process must be regular
- all hyperslab selections must be within one chunk

Case 2 requires that

- hyperslab selection in each process must be singular
- the number of chunks that covers the hyperslab selection for each process must be equal

Since the current implementation will do collective IO access per chunk, so to avoid multiple IO accesses with multiple chunks, we recommend at first to try setting hyperslab in your application ***as case 1 required(that is regular hyperslab selection with all selections inside one chunk)*** to gain better performance.

However, if your application cannot fulfill the requirement of case 1, setting the hyperslab as case 2 requires should also gain better performance than using independent IO. For this case, try to make the chunk size large.

4. MPI POSIX driver

Besides MPI-IO driver, there is another parallel IO driver called MPI POSIX driver implemented in HDF5. It is a "combination" MPI-2 and posix I/O driver. It uses MPI for coordinating the actions of several processes and posix I/O calls to do the actual I/O to the disk. There is no collective I/O mode with this driver.This will almost certainly not work correctly for files accessed on distributed parallel systems with the file located on a non-parallel filesystem.

On some systems, Using MPI POSIX driver may perform better than using MPI-IO driver with independent IO mode.

## VI. Undergoing Work

HDF5 group is currently in the process of enhancing the support of using for general hyperslab selection. Some information can be found from [6].

## VII. Reference:

(1) HDF5 user's guide: http://hdf.ncsa.uiuc.edu/HDF5/doc/UG/
(2) HDF5 reference manual: http://hdf.ncsa.uiuc.edu/HDF5/doc/RM_H5Front.html
(3) William Gropp, Ewing Lusk, Rajeev Thakur, 1999: Using MPI-2. The MIT Press
(4) Derived Data types with MPI: http://www.msi.umn.edu/tutorial/scicomp/general/MPI/content6.html at supercomputing institute of University of Minnesota
(5) Investigation of Parallel NetCDF with ROMS: http://hdf.ncsa.uiuc.edu/apps/WRF-ROMS/parallel-netcdf.pdf
(6) HDF5 collective chunk IO power point slides: HDF innards seminar