Introduction to Parallel HDF5

If you are new to HDF5 please read the *Learning the Basics* topic first.

Contents:

Overview of Parallel HDF5 (PHDF5) Design

There were several requirements that we had for Parallel HDF5 (PHDF5). These were:

- Parallel HDF5 files had to be compatible with serial HDF5 files and sharable between different serial and parallel platforms.
- Parallel HDF5 had to be designed to have a single file image to all processes, rather than having one file per process. Having one file per process can cause expensive post processing, and the files are not usable by different processes.
- A standard parallel I/O interface had to be portable to different platforms.

With these requirements of HDF5 our initial target was to support MPI programming, but not for shared memory programming. We had done some experimentation with thread-safe support for Pthreads and for OpenMP, and decided to use these.

Implementation requirements were to:

- Not use Threads, since they were not commonly supported in 1998 when we were looking at this.
- Not have a reserved process, as this might interfere with parallel algorithms.
- Not spawn any processes, as this is not even commonly supported now.

The following shows the Parallel HDF5 implementation layers:

Parallel Programming with HDF5

This tutorial assumes that you are somewhat familiar with parallel programming with MPI (Message Passing Interface).

If you are not familiar with parallel programming, here is a tutorial that may be of interest:

- Tutorial on HDF5 I/O tuning at NERSC

Some of the terms that you must understand in this tutorial are:

- **MPI Communicator:**

  Allows a group of processes to communicate with each other.

  Following are the MPI routines for initializing MPI and the communicator and finalizing a session with MPI:

```
C               Fortran          Description
    --              -------          -----------
    MPI_Init        MPI_INIT         Initialize MPI (MPI_COMM_WORLD usually)

    MPI_Comm_size   MPI_COMM_SIZE    Define how many processes are contained
                                     in the communicator

    MPI_Comm_rank   MPI_COMM_RANK    Define the process ID number within
                                     the communicator (from 0 to n-1)

    MPI_Finalize    MPI_FINALIZE     Exiting MPI
```

- **Collective:** MPI defines this to mean *all processes of the communicator must participate in the right order.*

Parallel HDF5 opens a parallel file with a communicator. It returns a file handle to be used for future access to the file.

All processes are required to participate in the collective Parallel HDF5 API. Different files can be opened using different communicators.

Examples of what you can do with the Parallel HDF5 collective API:

- File Operation: Create, open and close a file
- Object Creation: Create, open, and close a dataset
- Object Structure: Extend a dataset (increase dimension sizes)
- Dataset Operations: Write to or read from a dataset
  (Array data transfer can be collective or independent.)

Once a file is opened by the processes of a communicator:

- All parts of the file are accessible by all processes.
- All objects in the file are accessible by all processes.
- Multiple processes write to the same dataset.
- Each process writes to a individual dataset.

Please refer to the Supported Configuration Features Summary in the release notes for the current release of HDF5 for an up-to-date list of the platforms that we support Parallel HDF5 on.

Creating and Accessing a File with PHDF5

The programming model for creating and accessing a file is as follows:

1. Set up an access template object to control the file access mechanism.

2. Open the file.

3. Close the file.

Each process of the MPI communicator creates an access template and sets it up with MPI parallel access information. This is done with the H5P_CREATE call to obtain the file access property list and the H5P_SET_FAPL_MPIO call to set up parallel I/O access.

Following is example code for creating an access template in HDF5:

C:

```
23      MPI_Comm comm  = MPI_COMM_WORLD;
24      MPI_Info info  = MPI_INFO_NULL;
25
26      /*
27       * Initialize MPI
28       */
29      MPI_Init(&argc, &argv);
30      MPI_Comm_size(comm, &mpi_size);
31      MPI_Comm_rank(comm, &mpi_rank);
32
33      /*
34       * Set up file access property list with parallel I/O access
35       */
36 plist_id = H5Pcreate(H5P_FILE_ACCESS); 37 H5Pset_fapl_mpio(plist_id, comm, info);
```

FORTRAN90:

```
23       comm = MPI_COMM_WORLD
24       info = MPI_INFO_NULL
25
26       CALL MPI_INIT(mpierror)
27       CALL MPI_COMM_SIZE(comm, mpi_size, mpierror)
28       CALL MPI_COMM_RANK(comm, mpi_rank, mpierror)
29       !
30       ! Initialize FORTRAN interface
31       !
32       CALL h5open_f(error)
33
34       !
35       ! Setup file access property list with parallel I/O access.
36       !
37 CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id, error) 38 CALL h5pset_fapl_mpio_f(plist_id, comm,
info, error)
```

The following example programs create an HDF5 file using Parallel HDF5: C F90

Creating and Accessing a Dataset with PHDF5

The programming model for accessing a dataset with Parallel HDF5 is:

- Create or open a Parallel HDF5 file with a collective call to:

  H5D_CREATE
  H5D_OPEN

- Obtain a copy of the file transfer property list and set it to use *collective* or *independent I/O*.
  Do this by first passing a data transfer property list class type to:
  H5P_CREATE

  Then set the data transfer mode to either use *independent I/O* access or to use *collective I/O*, with a call to:
  H5P_SET_DXPL_MPIO

  Following are the parameters required by this call:

  C:
  ```
  herr_t H5Pset_dxpl_mpio (hid_t dxpl_id, H5FD_mpio_xfer_t  xfer_mode )
       dxpl_id    IN: Data transfer property list identifier
       xfer_mode  IN: Transfer mode:
                      H5FD_MPIO_INDEPENDENT - use independent I/O access
                                              (default)
                      H5FD_MPIO_COLLECTIVE  - use collective I/O access
  ```

  F90:
  ```
  h5pset_dxpl_mpi_f (prp_id, data_xfer_mode, hdferr)
       prp_id         IN: Property List Identifer (INTEGER (HID_T))
       data_xfer_mode IN: Data transfer mode  (INTEGER)
                            H5FD_MPIO_INDEPENDENT_F (0)
                            H5FD_MPIO_COLLECTIVE_F (1)
       hdferr         IN: Error code  (INTEGER)
  ```

- Access the dataset with the defined transfer property list.
  All processes that have opened a dataset may do collective I/O. Each process may do an independent and arbitrary number of data I/O access calls, using:
  H5D_WRITE
  H5D_READ

  If a dataset is unlimited, you can extend it with a *collective call* to:

  H5D_EXTEND

The following code demonstrates a collective write using Parallel HDF5:

C:
```
    95      /*
    96       * Create property list for collective dataset write.
    97       */
98 plist_id = H5Pcreate (H5P_DATASET_XFER); 99 H5Pset_dxpl_mpio (plist_id, H5FD_MPIO_COLLECTIVE);
   100
   101      status = H5Dwrite (dset_id, H5T_NATIVE_INT, memspace, filespace,
   102                plist_id, data);
```

F90:
```
   108       ! Create property list for collective dataset write
   109       !
110 CALL h5pcreate_f (H5P_DATASET_XFER_F, plist_id, error) 111 CALL h5pset_dxpl_mpio_f (plist_id,
H5FD_MPIO_COLLECTIVE_F, error)
   112
   113       !
   114       ! Write the dataset collectively.
   115       !
   116       CALL h5dwrite_f (dset_id, H5T_NATIVE_INTEGER, data, dimsfi, error, &
   117          file_space_id = filespace, mem_space_id = memspace, xfer_prp = plist_id)
```

The following example programs create a dataset in an HDF5 file using Parallel HDF5: C F90

Writing and Reading Hyperslabs

The programming model for writing and reading hyperslabs is:

- Each process defines the memory and file hyperslabs.
- Each process executes a partial write/read call which is either collective or independent.

The memory and file hyperslabs in the first step are defined with the H5S_SELECT_HYPERSLAB.

The *start* (or *offset*), *count*, *stride*, and *block* parameters define the portion of the dataset to write to. By changing the values of these parameters you can write hyperslabs with Parallel HDF5 by contiguous hyperslab, by regularly spaced data in a column/row, by patterns, and by chunks:

by Contiguous Hyperslab

by Regularly Spaced Data

by Pattern

by Chunk