Overview of Parallel HDF5

Outline

- Overview of Parallel HDF5 design
- Setting up parallel environment
- Programming model for
  - Creating and accessing a File
  - Creating and accessing a Dataset
  - Writing and reading Hyperslabs

PHDF5 Initial Target

- Support for MPI programming
- Not for shared memory programming
  - Threads
  - OpenMP
- Has some experiments with
  - Thread-safe support for Pthreads
  - OpenMP if called "correctly"

PHDF5 Requirements

- PHDF5 files compatible with serial HDF5 files
  - Shareable between different serial or parallel platforms
- Single file image to all processes
  - One file per process design is undesirable
    - Expensive post processing
    - Not useable by different number of processes
- Standard parallel I/O interface
  - Must be portable to different platforms

Collective vs. Independent Calls

- MPI definition of collective call
  - All processes of the communicator must participate in the right order
- Independent means not collective
- Collective is not necessarily synchronous

Programming Restrictions

- Most PHDF5 APIs are collective
- PHDF5 opens a parallel file with a communicator
  - Returns a file-handle
  - Future access to the file via the file-handle
  - All processes must participate in collective PHDF5 APIs
  - Different files can be opened via different communicators

Examples of PHDF5 API

- Examples of PHDF5 collective API
  - File operations: H5Fcreate, H5Fopen, H5Fclose
  - Objects creation: H5Dcreate, H5Dopen, H5Dclose
  - Objects structure: H5Dextend (increase dimension sizes)
- Array data transfer can be collective or independent
  - Dataset operations: H5Dwrite, H5Dread

What Does PHDF5 Support ?

- After a file is opened by the processes of a communicator
  - All parts of file are accessible by all processes
  - All objects in the file are accessible by all processes
  - Multiple processes write to the same data array
  - Each process writes to individual data array

Creating and Accessing a File Programming model

- HDF5 uses access template object (property list) to control the file access mechanism
- General model to access HDF5 file in parallel:
  - Setup MPI-IO access template (access property list)
  - Open File
  - Close File

- 11 -

Creating and Opening Dataset

- All processes of the MPI communicator open/close a dataset by a collective call
  - C: H5Dcreate or H5Dopen; H5Dclose
  - F90: h5dcreate_f or h5dopen_f; h5dclose_f
- All processes of the MPI communicator extend dataset with unlimited dimensions before writing to it
  - C: H5Dextend
  - F90: h5dextend_f

- 15 -

Accessing a Dataset

- All processes that have opened dataset may do collective I/O
- Each process may do independent and arbitrary number of data I/O access calls
  - C: H5Dwrite and H5Dread
  - F90: h5dwrite_f and h5dread_f

Accessing a Dataset Programming model

- Create and set dataset transfer property
  - C: H5Pset_dxpl_mpio
    - H5FD_MPIO_COLLECTIVE
    - H5FD_MPIO_INDEPENDENT (default)
  - F90: h5pset_dxpl_mpio_f
    - H5FD_MPIO_COLLECTIVE_F
    - H5FD_MPIO_INDEPENDENT_F (default)
- Access dataset with the defined transfer property

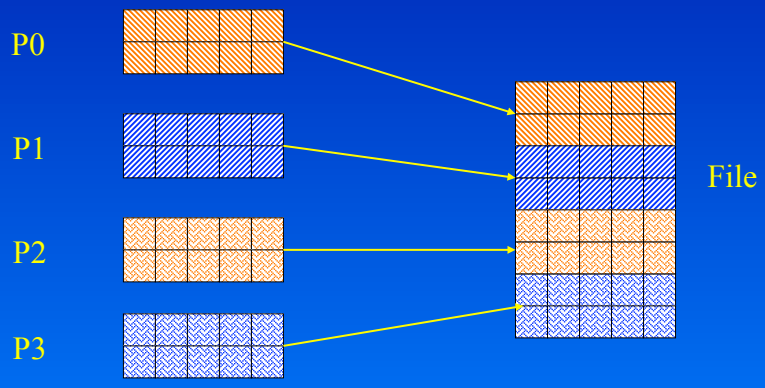
F90 Example: Collective write

```
88  ! Create property list for collective dataset write
89  !
90  CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, error)
91  CALL h5pset_dxpl_mpio_f(plist_id, &
                            H5FD_MPIO_COLLECTIVE_F, error)
92
93  !
94  ! Write the dataset collectively.
95  !
96  CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, data, &
                    error, filespace, memspace, &
                    xfer_prp = plist_id)
```


Writing and Reading Hyperslabs Programming model

- Distributed memory model: data is split among processes
- PHDF5 uses hyperslab model
- Each process defines memory and file hyperslabs
- Each process executes partial write/read call
  - Collective calls
  - Independent calls

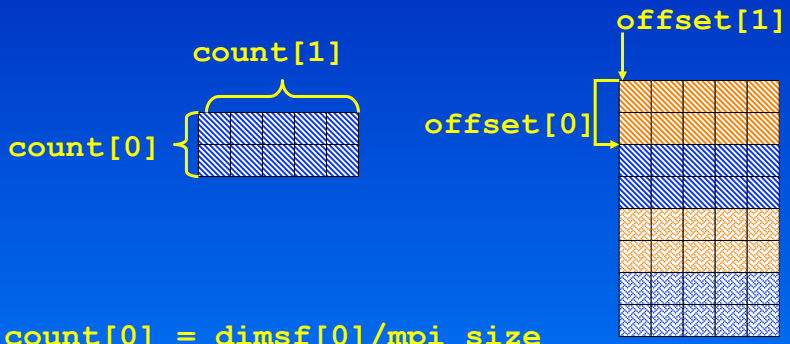
Hyperslab Example 1
Writing dataset by rows



Example 1
Writing dataset by rows

P1 (memory space)

File




```
count[0] = dimsf[0]/mpi_size
count[1] = dimsf[1];
offset[0] = mpi_rank * count[0];  /* = 2 */
offset[1] = 0;
```

Hyperslab Example 3
Writing dataset by chunks