Intro to Parallel HDF5

Outline

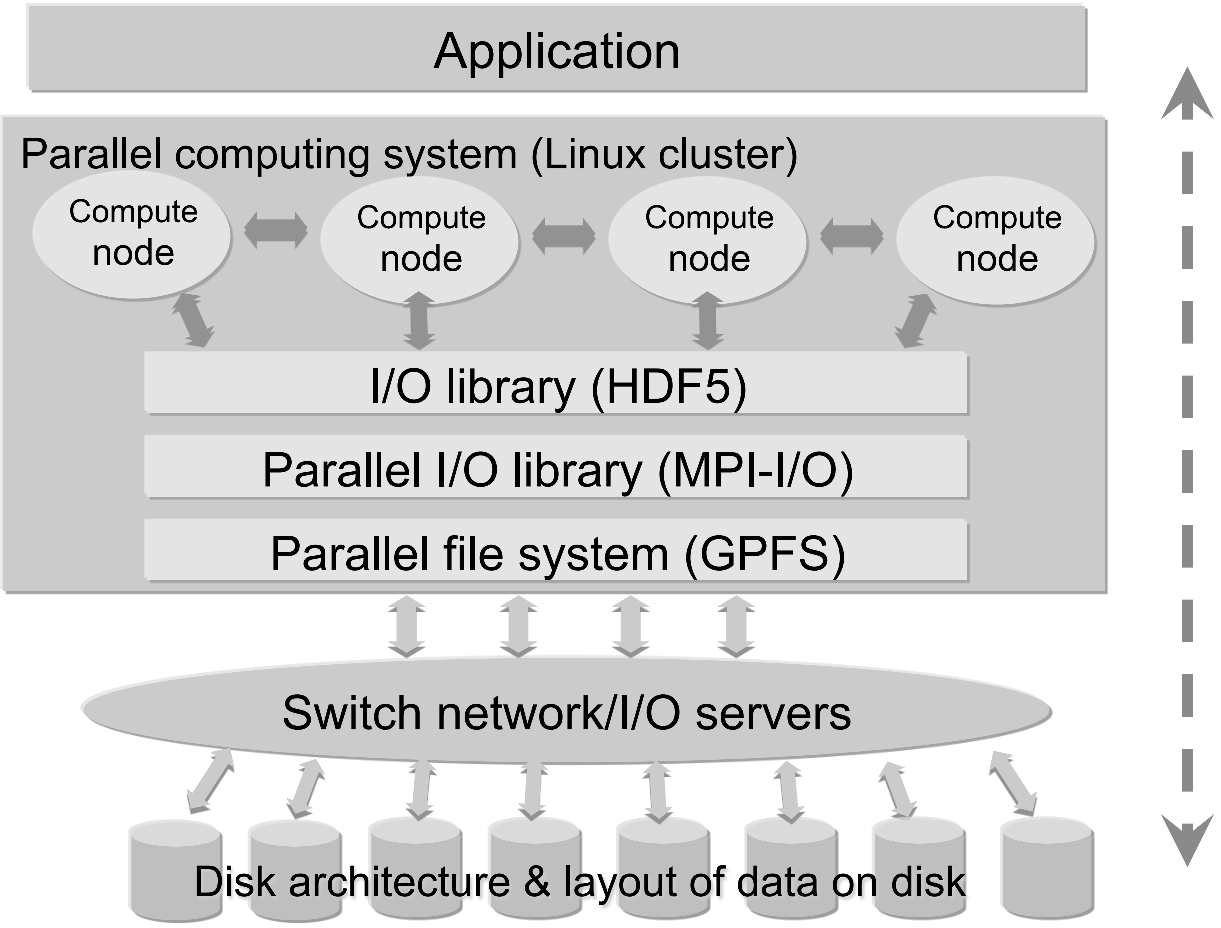
- Overview of Parallel HDF5 design
- Parallel Environment Requirements
- Performance Analysis
- Parallel tools
- PHDF5 Programming Model

Overview of Parallel HDF5 Design

PHDF5 Requirements

- Support Message Passing Interface (MPI) programming
- 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 usable by different number of processes
- Standard parallel I/O interface
- Must be portable to different platforms

PHDF5 Implementation Layers


Application
Parallel computing system (Linux cluster)
Compute node
Compute node
Compute node
Compute node
I/O library (HDF5)
Parallel I/O library (MPI-I/O)
Parallel file system (GPFS)
Switch network/I/O servers
Disk architecture & layout of data on disk

PHDF5 built on top of standard MPI-IO API

MPI-IO vs. HDF5

- MPI-IO is an Input/Output API.
- It treats the data file as a “linear byte stream” and each MPI application needs to provide its own file view and data representations to interpret those bytes.
- All data stored are machine dependent except the “external32” representation.
- External32 is defined in Big Endianness
  - Little-endian machines have to do the data conversion in both read or write operations.
  - 64bit sized data types may lose information.

MPI-IO vs. HDF5 Cont.

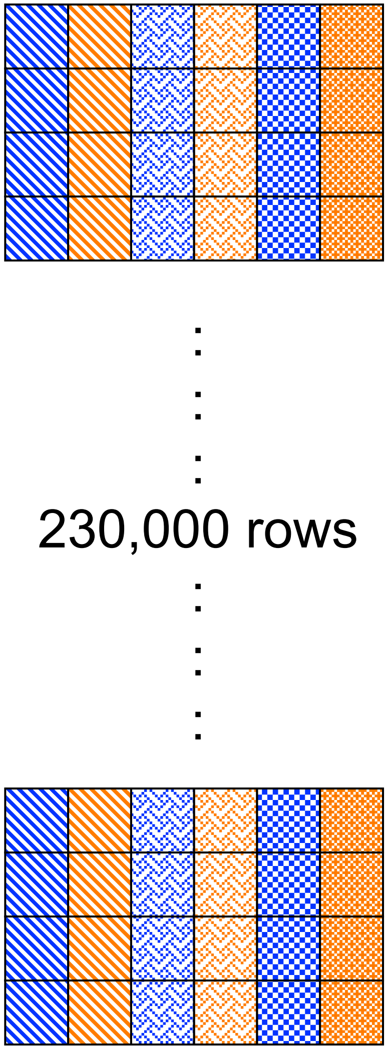
- HDF5 is a data management software.
- It stores the data and metadata according to the HDF5 data format definition.
  - HDF5 file is self-described.
  - Each machine can store the data in its own native representation for efficient I/O without loss of data precision.
  - Any necessary data representation conversion is done by the HDF5 library automatically.

Performance Analysis

- Some common causes of poor performance
- Possible solutions:
  - Use larger I/O sizes
  - Use specific I/O system hints
  - Independent vs. Collective access

Independent vs. Collective Access

- User reported Independent data transfer mode was much slower than the Collective data transfer mode
- Data array was tall and thin: 230,000 rows by 6 columns



Collective vs. Independent Calls

- MPI definition of collective calls
  - All processes of the communicator must participate in the right order. E.g.,
    - Process1 Process2
    - call A(); call B(); call A(); call B(); **right**
    - call A(); call B(); call B(); call A(); **wrong**
- Independent means not collective
- Collective is not necessarily synchronous

Debug Slow Parallel I/O Speed(1)

- Writing to one dataset
  - Using 4 processes == 4 columns
  - data type is 8 bytes doubles
  - 4 processes, 1000 rows == 4x1000x8 = 32,000 bytes
- % mpirun -np 4 ./a.out i t 1000
  - Execution time: 1.783798 s.
- % mpirun -np 4 ./a.out i t 2000
  - Execution time: 3.838858 s.
- # Difference of 2 seconds for 1000 more rows = 32,000 Bytes.
- # A speed of 16KB/Sec!!! Way too slow.

Debug Slow Parallel I/O Speed(2)

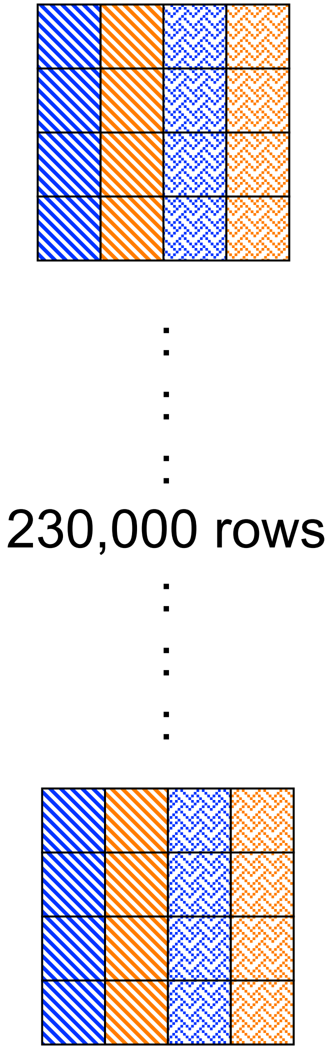
- Build a version of PHDF5 with
  - ./configure --enable-debug --enable-parallel …
  - This allows the tracing of MPIO I/O calls in the HDF5 library.
- E.g., to trace
  - MPI_File_read_xx and MPI_File_write_xx calls
  - % setenv H5FD_mpio_Debug “rw”

Debug Slow Parallel I/O Speed(3)

- % setenv H5FD_mpio_Debug ’rw’
- % mpirun -np 4 ./a.out i t 1000 # Indep.; contiguous.
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=2056 size_i=8
- in H5FD_mpio_write mpi_off=2048 size_i=8
- in H5FD_mpio_write mpi_off=2072 size_i=8
- in H5FD_mpio_write mpi_off=2064 size_i=8
- in H5FD_mpio_write mpi_off=2088 size_i=8
- in H5FD_mpio_write mpi_off=2080 size_i=8
- …
- # total of 4000 of this little 8 bytes writes == 32,000 bytes.

Independent calls are many and small

- Each process writes one element of one row, skips to next row, write one element, so on.
- Each process issues 230,000 writes of 8 bytes each.
- Not good==just like many independent cars driving to work, waste gas, time, total traffic jam.

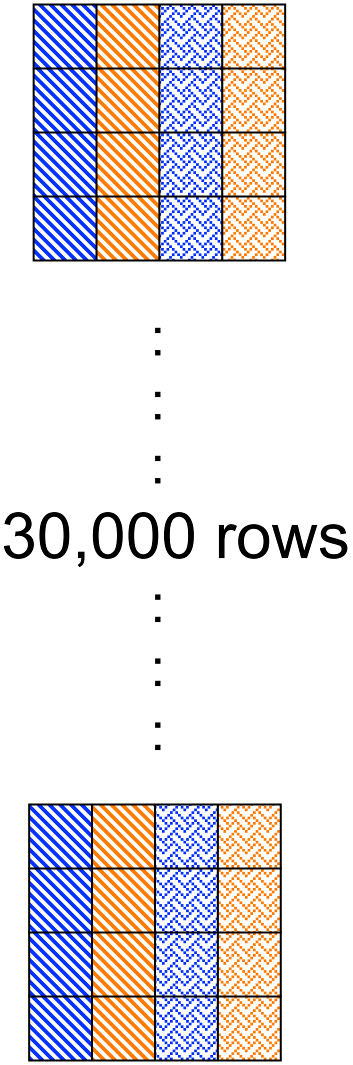


Debug Slow Parallel I/O Speed (4)

- % setenv H5FD_mpio_Debug ’rw’
- % mpirun -np 4 ./a.out i h 1000   # Indep., Chunked by column.
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=0 size_i=96
- in H5FD_mpio_write mpi_off=3688 size_i=8000
- in H5FD_mpio_write mpi_off=11688 size_i=8000
- in H5FD_mpio_write mpi_off=27688 size_i=8000
- in H5FD_mpio_write mpi_off=19688 size_i=8000
- in H5FD_mpio_write mpi_off=96 size_i=40
- in H5FD_mpio_write mpi_off=136 size_i=544
- in H5FD_mpio_write mpi_off=680 size_i=120
- in H5FD_mpio_write mpi_off=800 size_i=272
- …
- Execution time: 0.011599 s.

Use Collective Mode or Chunked Storage

- Collective mode will combine many small independent calls into few but bigger calls
- Chunks of columns speeds up too



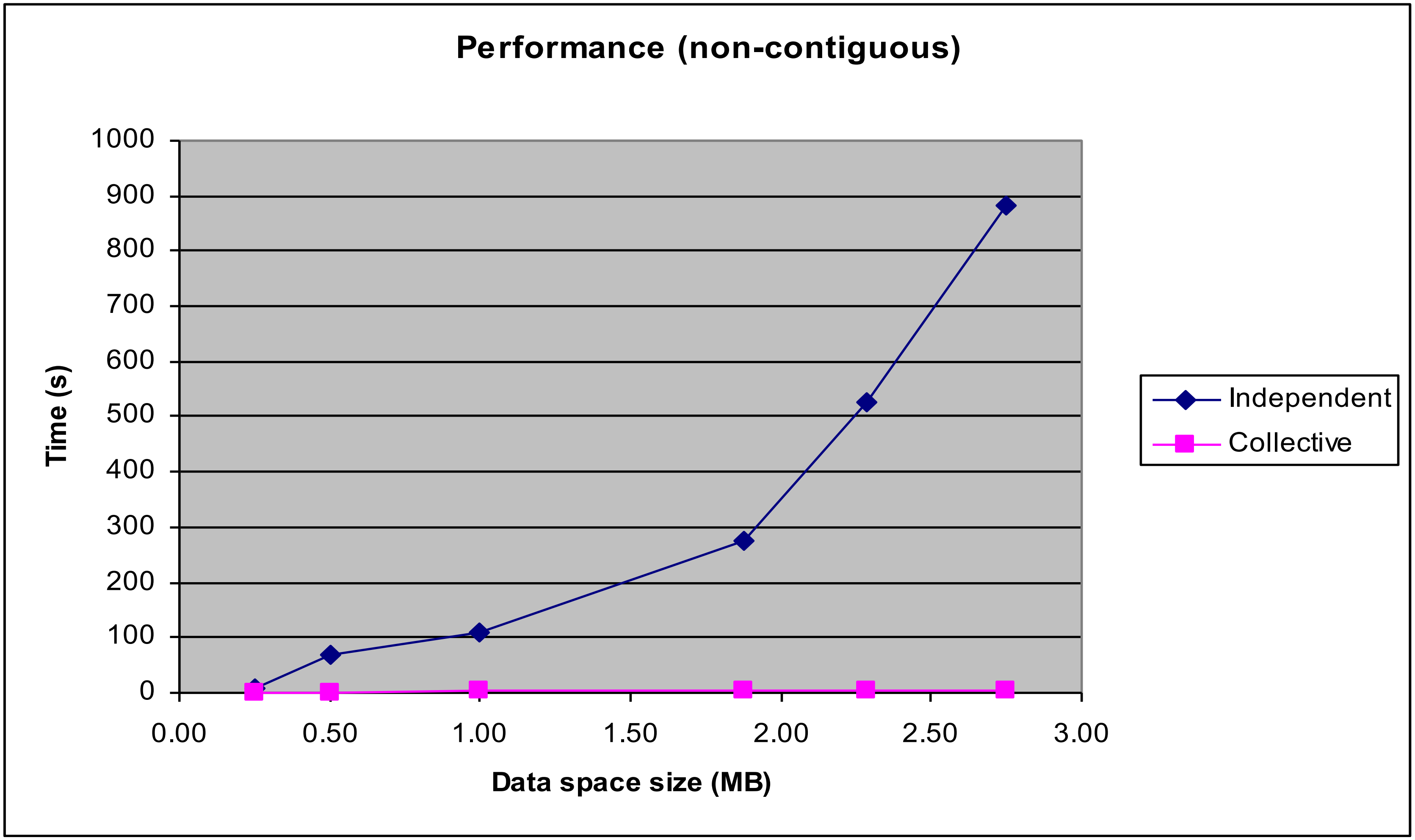
230,000 rows

Independent vs. Collective write

6 processes, IBM p-690, AIX, GPFS

# of Rows	Data Size (MB)	Independent (Sec.)	Collective (Sec.)
16384	0.25	8.26	1.72
32768	0.50	65.12	1.80
65536	1.00	108.20	2.68
122918	1.88	276.57	3.11
150000	2.29	528.15	3.63
180300	2.75	881.39	4.12

Independent vs. Collective write (cont.)



Parallel Tools

- h5perf
  - Performance measuring tools showing I/O performance for different I/O API

h5perf

- An I/O performance measurement tool
- Test 3 File I/O API
  - POSIX I/O (open/write/read/close…)
  - MPIO (MPI_File_{open,write,read,close})
  - PHDF5
    - H5Pset_fapl_mpio (using MPI-IO)
- An indication of I/O speed upper limits

h5perf: Some features

- Check (-c) verify data correctness
- Added 2-D chunk patterns in v1.8
- -h shows the help page.

Useful Parallel HDF Links

- Parallel HDF information site

  http://www.hdfgroup.org/HDF5/PHDF5/
- Parallel HDF5 tutorial available at

  http://www.hdfgroup.org/HDF5/Tutor/
- Parallel FAQ

  https://www.hdfgroup.org/hdf5-quest.html#PARALLEL
- HDF Help email address

  help@hdfgroup.org

Questions?

How to Compile PHDF5 Applications

- h5pcc – HDF5 C compiler command
  - Similar to mpicc
- h5pfc – HDF5 F90 compiler command
  - Similar to mpif90
- To compile:
  - % h5pcc h5prog.c
  - % h5pfc h5prog.f90

h5pcc/h5pfc -show option

- -show displays the compiler commands and options without executing them, i.e., dry run

```
% h5pcc -show Sample_mpio.c
mpicc -I/home/packages/phdf5/include \
-D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE \
-D_FILE_OFFSET_BITS=64 -D_POSIX_SOURCE \
-D_BSD_SOURCE -std=c99 -c Sample_mpio.c

mpicc -std=c99 Sample_mpio.o \
-L/home/packages/phdf5/lib \
home/packages/phdf5/lib/libhdf5_hl.a \ /home/packages/
phdf5/lib/libhdf5.a -lz -lm -Wl,-rpath \
-Wl,/home/packages/phdf5/lib
```

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

Collective vs. Independent Calls

- MPI definition of collective calls
  - All processes of the communicator must participate in the right order. E.g.,
    - Process1 Process2
    - call A(); call B(); call A(); call B(); **right**
    - call A(); call B(); call B(); call A(); **wrong**
- Independent means not collective
- Collective is not necessarily synchronous

Examples of PHDF5 API

- Examples of PHDF5 collective API
  - File operations: H5Fcreate, H5Fopen, H5Fclose
  - Objects creation: H5Dcreate, H5Dclose
  - Objects structure: H5Dextend (increase dimension sizes)
- Array data transfer can be collective or independent
  - Dataset operations: H5Dwrite, H5Dread
  - Collectiveness is indicated by function parameters, not by function names as in MPI API

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 may write to the same data array
  - Each process may write to individual data array

PHDF5 API Languages and Platforms

- C and F90 language interfaces
- Platforms supported:
  - Most platforms with MPI-IO supported. E.g.,
    - IBM AIX
    - Linux clusters
    - Crays
  - For performance parallel file system is needed, e.g., Lustre or GPFS

Programming model for creating and accessing a file

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

Setup MPI-IO access template

Each process of the MPI communicator creates an access template and sets it up with MPI parallel access information

C:

```
herr_t H5Pset_fapl_mpio(hid_t plist_id,
       MPI_Comm comm,  MPI_Info info);
```

F90:

```
h5pset_fapl_mpio_f(plist_id, comm, info)
integer(hid_t) :: plist_id
integer        :: comm, info
```

`plist_id` is a file access property list identifier

C Example Parallel File Create

```
comm = MPI_COMM_WORLD;
info = MPI_INFO_NULL;
/*
 * Initialize MPI
 */
MPI_Init(&argc, &argv);
/*
 * Set up file access property list for MPI-IO access
 */
plist_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(plist_id, comm, info);

file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC,
   H5P_DEFAULT, plist_id);
/*
 * Close the file.
 */
H5Fclose(file_id);
MPI_Finalize();
```

Creating and Opening Dataset

- All processes of the 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 communicator must extend an unlimited dimension dataset before writing to it
  - ✓C: H5Dextend
  - ✓F90: h5dextend_f

C Example: Create Dataset

```
  file_id = H5Fcreate(…);
  /*
   * Create the dataspace for the dataset.
   */
  dimsf[0] = NX;
  dimsf[1] = NY;
  filespace = H5Screate_simple(RANK, dimsf, NULL);

  /*
   * Create the dataset with default properties collective.
   */
->dset_id = H5Dcreate(file_id, “dataset1”, H5T_NATIVE_INT,
                      filespace, H5P_DEFAULT);

  H5Dclose(dset_id);
  /*
   * Close the file.
   */
  H5Fclose(file_id);
```

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

Programming model for dataset access

- 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

C Example: Collective write

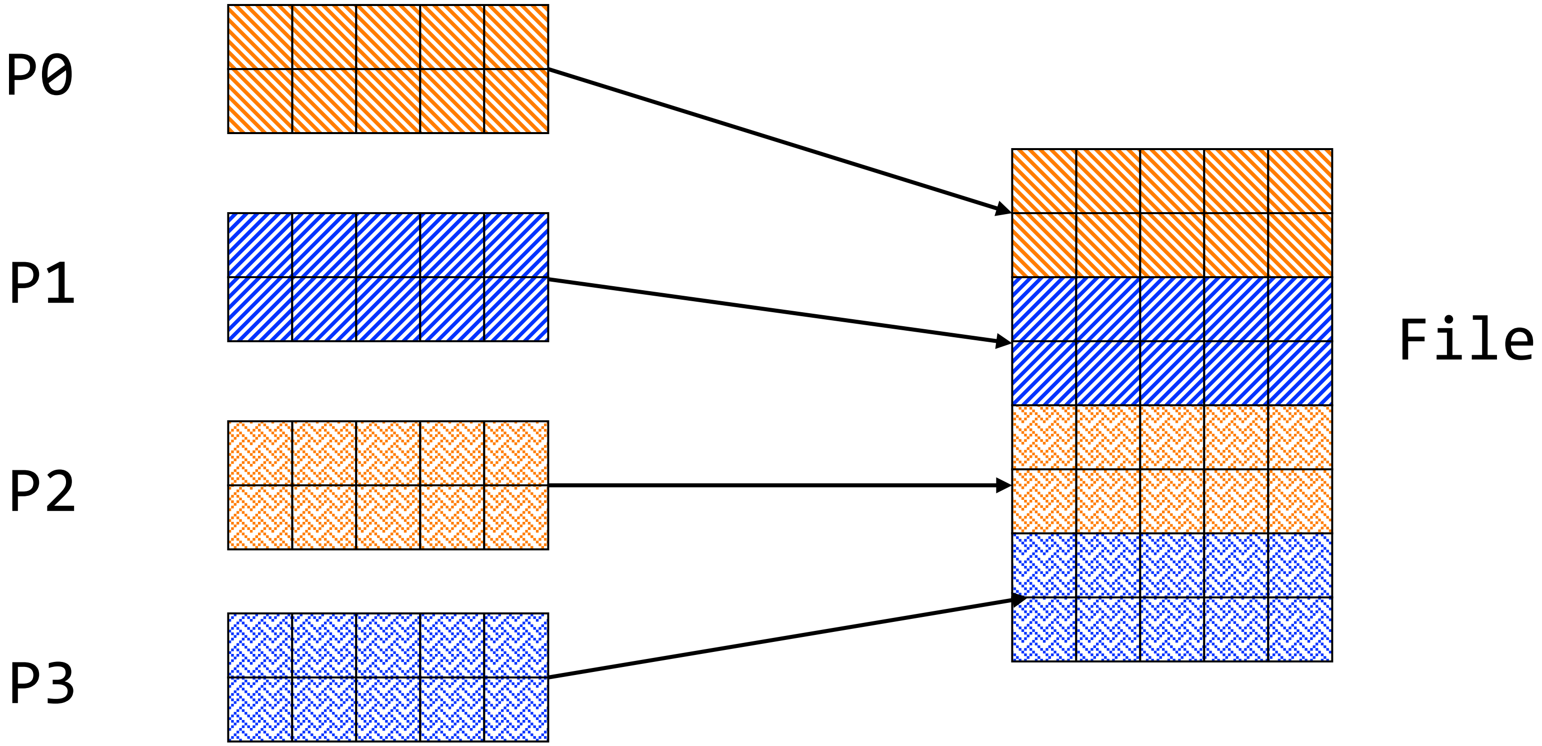
```
  /*
   * Create property list for collective dataset write.
   */
  plist_id = H5Pcreate(H5P_DATASET_XFER);
->H5Pset_dxpl_mpio(plist_id, H5FD_MPIO_COLLECTIVE);

  status = H5Dwrite(dset_id, H5T_NATIVE_INT,
                   memspace, filespace, plist_id, data);
```

Writing and Reading Hyperslabs

- Distributed memory model: data is split among processes
- PHDF5 uses HDF5 hyperslab model
- Each process defines memory and file hyperslabs
- Each process executes partial write/read call
    - Collective calls
    - Independent calls

Example: *Writing dataset by rows*



This pattern is similar to JPSS aggregation

Writing by rows: *Output of h5dump*

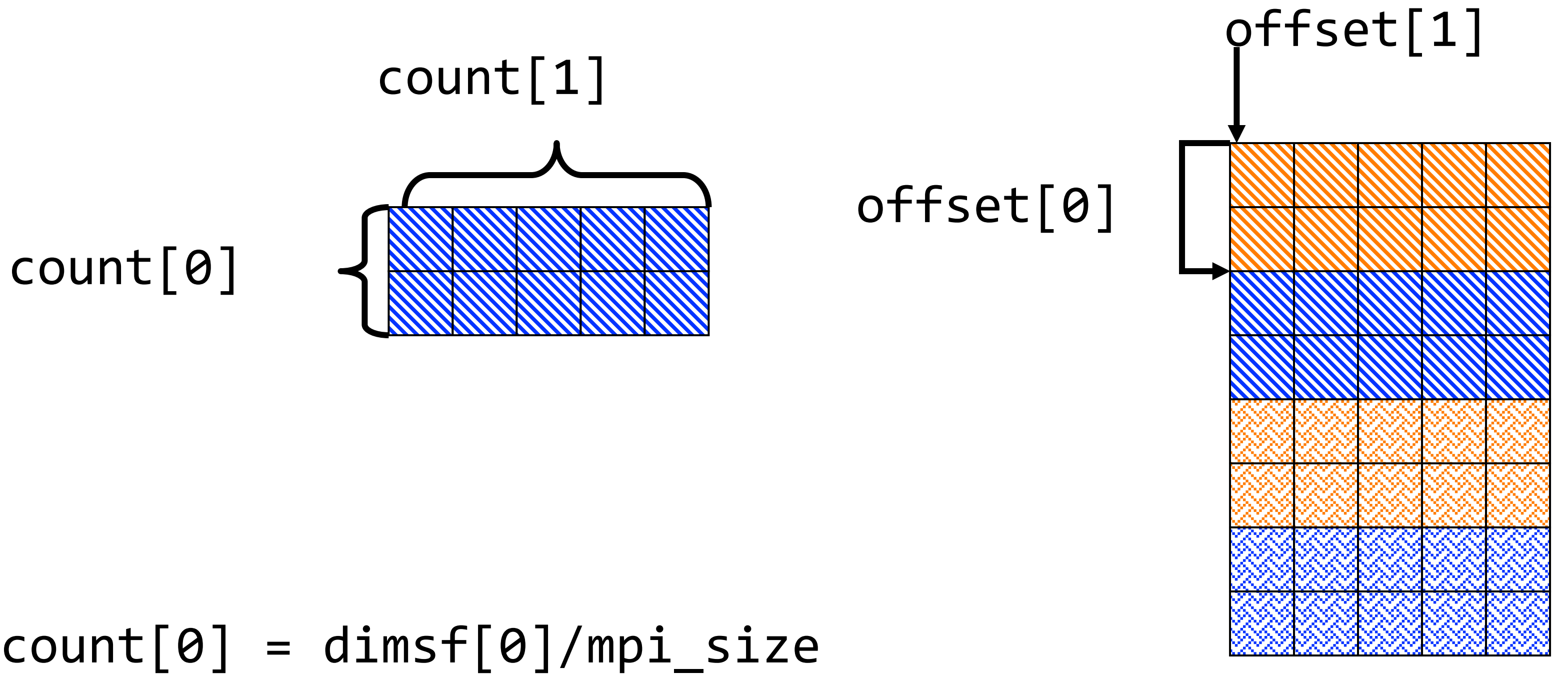
```
HDF5 "SDS_row.h5" {
GROUP "/" {
   DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE  SIMPLE { ( 8, 5 ) / ( 8, 5 ) }
      DATA {
         10, 10, 10, 10, 10,
         10, 10, 10, 10, 10,
         11, 11, 11, 11, 11,
         11, 11, 11, 11, 11,
         12, 12, 12, 12, 12,
         12, 12, 12, 12, 12,
         13, 13, 13, 13, 13,
         13, 13, 13, 13, 13
      }
   }
}
}
```

Example: *Writing dataset by rows*

Process P1

Memory

File



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

Example: *Writing dataset by rows*

```
/*
 * Each process defines dataset in memory and
 * writes it to the hyperslab
 * in the file.
 */
 count[0] = dimsf[0]/mpi_size;
 count[1] = dimsf[1];
 offset[0] = mpi_rank * count[0];
 offset[1] = 0;
 memspace = H5Screate_simple(RANK,count,NULL);

/*
 * Select hyperslab in the file.
 */
 filespace = H5Dget_space(dset_id);
 H5Sselect_hyperslab(filespace,
     H5S_SELECT_SET,offset,NULL,count,NULL);
 H5Dwrite (dset_id, …, memspace,filespace,buf);
```

Questions?

Acknowledgement

This work was supported by SGT under Prime Contract No. NNG12CR31C,

funded by NASA.

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of SGT or NASA.

Thank You!

Questions?