

# Parallel HDF5 Tutorial

Albert Cheng  
The HDF Group

# Parallel HDF5 Introductory Tutorial





## PHDF5 Requirements

- Support 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

September 9, 2008

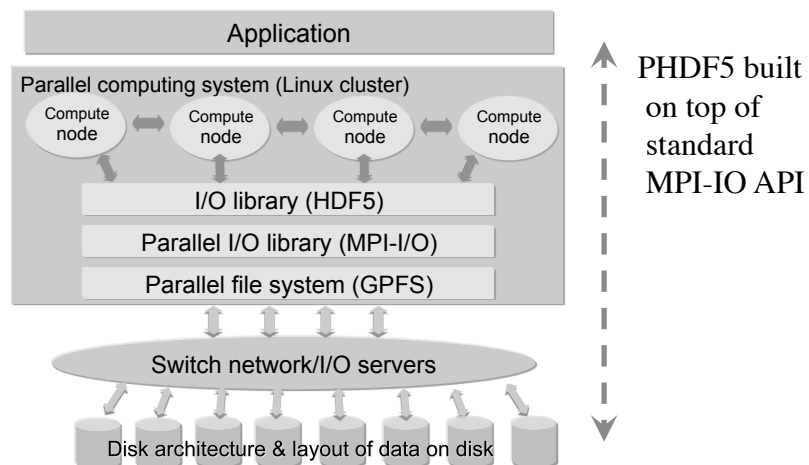
SPEEDUP Workshop - HDF5 Tutorial

5

www.hdfgroup.org



## PHDF5 Implementation Layers



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

6

www.hdfgroup.org



## Parallel Environment Requirements

- MPI with MPI-IO. E.g.,
  - MPICH2 ROMIO
  - Vendor's MPI-IO
- POSIX compliant parallel file system. E.g.,
  - GPFS
  - Lustre



## 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.



## 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
```



## Collective vs. Independent Calls

- MPI definition of collective call
  - 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



## 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
  - 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

- C and F90 language interfaces
- Platforms supported:
  - Most platforms with MPI-IO supported. E.g.,
    - IBM SP, Linux clusters, SGI Altrix, Cray XT3, ...



- 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
  - 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

```
23 comm = MPI_COMM_WORLD;
24 info = MPI_INFO_NULL;
26 /*
27     * Initialize MPI
28     */
29 MPI_Init(&argc, &argv);
33 /*
34     * Set up file access property list for MPI-IO access
35     */
->36 plist_id = H5Pcreate(H5P_FILE_ACCESS);
->37 H5Pset_fapl_mpio(plist_id, comm, info);
38
->42 file_id = H5Fcreate(H5FILE_NAME, H5F_ACC_TRUNC,
    H5P_DEFAULT, plist_id);
49 /*
50     * Close the file.
51     */
52 H5Fclose(file_id);
54 MPI_Finalize();
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

19

www.hdfgroup.org



## F90 Example Parallel File Create

```
23 comm = MPI_COMM_WORLD
24 info = MPI_INFO_NULL
26 CALL MPI_INIT(mpierror)
29 !
30 ! Initialize FORTRAN predefined datatypes
32 CALL h5open_f(error)
34 !
35 ! Setup file access property list for MPI-IO access
->37 CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id, error)
->38 CALL h5pset_fapl_mpio_f(plist_id, comm, info, error)
40 !
41 ! Create the file collectively.
->43 CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id,
    error, access_prp = plist_id)
45 !
46 ! Close the file.
49 CALL h5fclose_f(file_id, error)
51 !
52 ! Close FORTRAN interface
54 CALL h5close_f(error)
56 CALL MPI_FINALIZE(mpierror)
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

20

www.hdfgroup.org



## 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

```
56 file_id = H5Fcreate(...);
57 /*
58  * Create the dataspace for the dataset.
59  */
60 dims[0] = NX;
61 dims[1] = NY;
62 filespace = H5Screate_simple(RANK, dims, NULL);
63
64 /*
65  * Create the dataset with default properties collective.
66  */
->67 dset_id = H5Dcreate(file_id, "dataset1", H5T_NATIVE_INT,
68                    filespace, H5P_DEFAULT);

70 H5Dclose(dset_id);
71 /*
72  * Close the file.
73  */
74 H5Fclose(file_id);
```



## F90 Example: Create Dataset

```
43 CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id,  
    error, access_prp = plist_id)  
73 CALL h5screate_simple_f(rank, dimsf, filespace, error)  
76 !  
77 ! Create the dataset with default properties.  
78 !  
->79 CALL h5dcreate_f(file_id, "dataset1", H5T_NATIVE_INTEGER,  
    filespace, dset_id, error)  
90 !  
91 ! Close the dataset.  
92 CALL h5dclose_f(dset_id, error)  
93 !  
94 ! Close the file.  
95 CALL h5fclose_f(file_id, error)
```



## 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

```
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,
102                  memspace, filespace, plist_id, data);
```



## 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, &
                  file_space_id = filespace, &
                  mem_space_id = memspace, &
                  xfer_prp = plist_id)
```



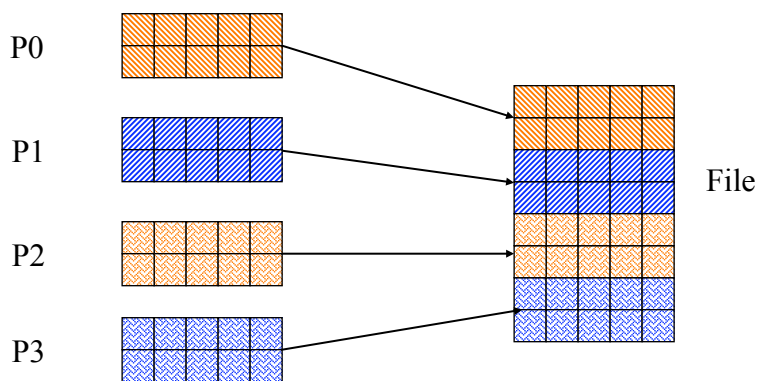
## Writing and Reading Hyperslabs

- 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

## Set up the Hyperslab for Read/Write

```
H5Sselect_hyperslab(  
    file_space, H5S_SELECT_SET,  
    offset, stride, count, block  
)
```

## Example 1: *Writing dataset by rows*





## 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
      }
    }
  }
}
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

31

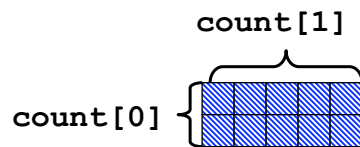
www.hdfgroup.org



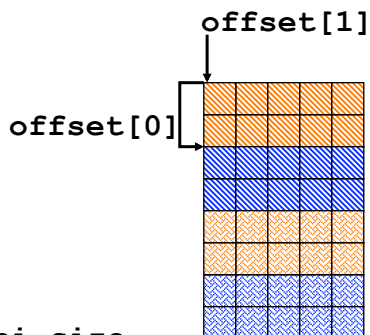
## Example 1: *Writing dataset by rows*

Process 1

Memory



File



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

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

32

www.hdfgroup.org

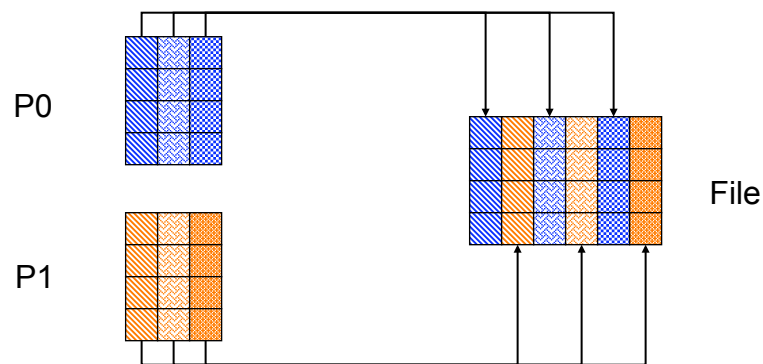


## Example 1: Writing dataset by rows

```
71 /*
72  * Each process defines dataset in memory and
73  * writes it to the hyperslab
74  * in the file.
75  */
76 count[0] = dimsf[0]/mpi_size;
77 count[1] = dimsf[1];
78 offset[0] = mpi_rank * count[0];
79 offset[1] = 0;
80 memspace = H5Screate_simple(RANK,count,NULL);
81 /*
82  * Select hyperslab in the file.
83  */
84 filespace = H5Dget_space(dset_id);
85 H5Sselect_hyperslab(filespace,
86 H5S_SELECT_SET,offset,NULL,count,NULL);
```



## Example 2: Writing dataset by columns





## Writing by columns: *Output of h5dump*

```
HDF5 "SDS_col.h5" {
  GROUP "/" {
    DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE SIMPLE { ( 8, 6 ) / ( 8, 6 ) }
      DATA {
        1, 2, 10, 20, 100, 200,
        1, 2, 10, 20, 100, 200,
        1, 2, 10, 20, 100, 200,
        1, 2, 10, 20, 100, 200,
        1, 2, 10, 20, 100, 200,
        1, 2, 10, 20, 100, 200,
        1, 2, 10, 20, 100, 200,
        1, 2, 10, 20, 100, 200
      }
    }
  }
}
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

35

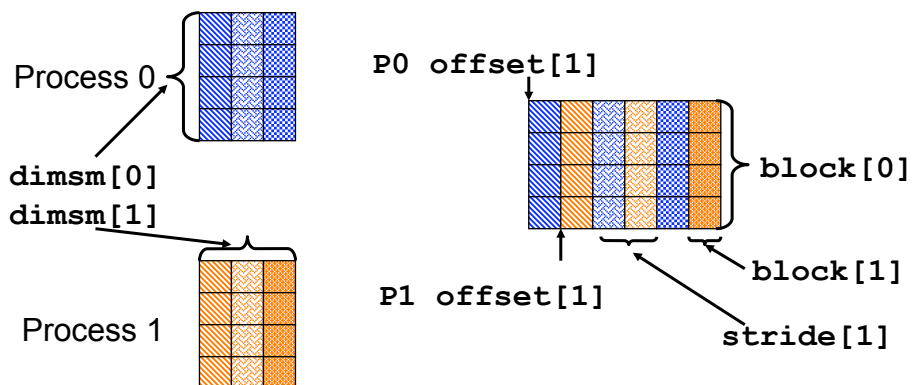
www.hdfgroup.org



## Example 2: *Writing dataset by column*

Memory

File



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

36

www.hdfgroup.org



## Example 2: Writing dataset by column

```
85      /*
86      * Each process defines hyperslab in
87      * the file
88      */
89      count[0] = 1;
90      count[1] = dimsm[1];
91      offset[0] = 0;
92      offset[1] = mpi_rank;
93      stride[0] = 1;
94      stride[1] = 2;
95      block[0] = dimsf[0];
96      block[1] = 1;
97
98      /*
99      * Each process selects hyperslab.
100     */
101     filesystem = H5Dget_space(dset_id);
102
103     H5Sselect_hyperslab(filespace,
104                        H5S_SELECT_SET, offset, stride,
105                        count, block);
```

September 9, 2008

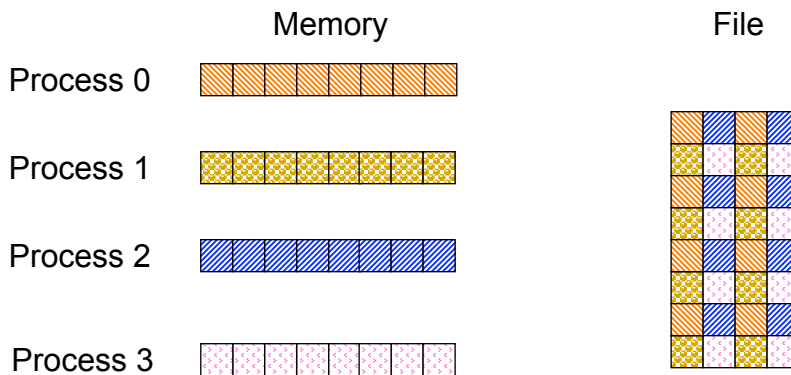
SPEEDUP Workshop - HDF5 Tutorial

37

www.hdfgroup.org



## Example 3: Writing dataset by pattern



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

38

www.hdfgroup.org



## Writing by Pattern: Output of h5dump

```
HDF5 "SDS_pat.h5" {
  GROUP "/" {
    DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE SIMPLE { ( 8, 4 ) / ( 8, 4 ) }
      DATA {
        1, 3, 1, 3,
        2, 4, 2, 4,
        1, 3, 1, 3,
        2, 4, 2, 4,
        1, 3, 1, 3,
        2, 4, 2, 4,
        1, 3, 1, 3,
        2, 4, 2, 4
      }
    }
  }
}
```

September 9, 2008

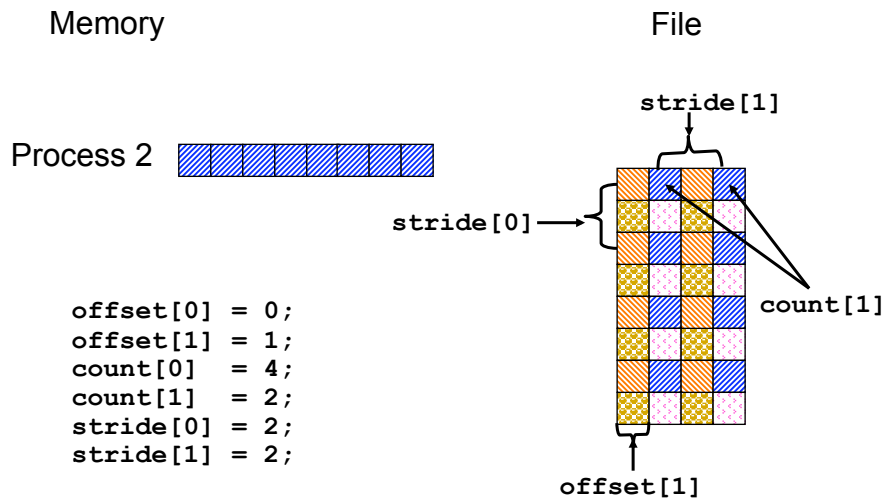
SPEEDUP Workshop - HDF5 Tutorial

39

www.hdfgroup.org



## Example 3: Writing dataset by pattern



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

40

www.hdfgroup.org



## Example 3: Writing by pattern

```
90      /* Each process defines dataset in memory and
91      * writes it to the hyperslab in the file.
92      */
93      count[0] = 4;
94      count[1] = 2;
95      stride[0] = 2;
96      stride[1] = 2;
97      if(mpi_rank == 0) {
98          offset[0] = 0;
99          offset[1] = 0;
100     }
101     if(mpi_rank == 1) {
102         offset[0] = 1;
103         offset[1] = 0;
104     }
105     if(mpi_rank == 2) {
106         offset[0] = 0;
107         offset[1] = 1;
108     }
109     if(mpi_rank == 3) {
110         offset[0] = 1;
111         offset[1] = 1;
112     }
```

September 9, 2008

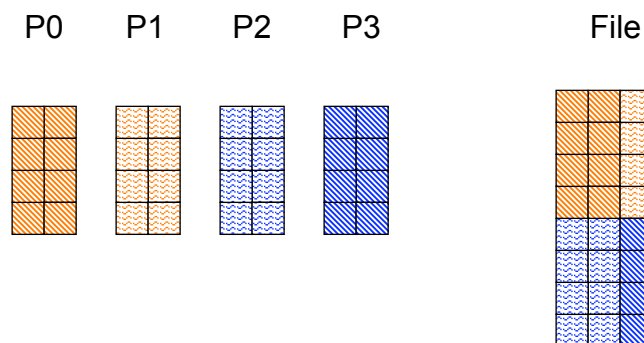
SPEEDUP Workshop - HDF5 Tutorial

41

[www.hdfgroup.org](http://www.hdfgroup.org)



## Example 4: Writing dataset by chunks



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

42

[www.hdfgroup.org](http://www.hdfgroup.org)



## Writing by Chunks: Output of h5dump

```
HDF5 "SDS_chnk.h5" {
  GROUP "/" {
    DATASET "IntArray" {
      DATATYPE  H5T_STD_I32BE
      DATASPACE SIMPLE { ( 8, 4 ) / ( 8, 4 ) }
      DATA {
        1, 1, 2, 2,
        1, 1, 2, 2,
        1, 1, 2, 2,
        1, 1, 2, 2,
        3, 3, 4, 4,
        3, 3, 4, 4,
        3, 3, 4, 4,
        3, 3, 4, 4
      }
    }
  }
}
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

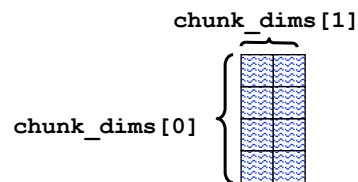
43

www.hdfgroup.org



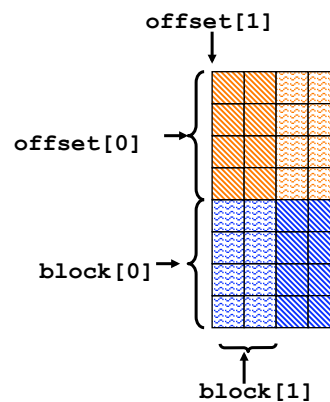
## Example 4: Writing dataset by chunks

Process 2: Memory



```
block[0] = chunk_dims[0];
block[1] = chunk_dims[1];
offset[0] = chunk_dims[0];
offset[1] = 0;
```

File



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

44

www.hdfgroup.org



## Example 4: Writing by chunks

```
97     count[0] = 1;
98     count[1] = 1;
99     stride[0] = 1;
100    stride[1] = 1;
101    block[0] = chunk_dims[0];
102    block[1] = chunk_dims[1];
103    if(mpi_rank == 0) {
104        offset[0] = 0;
105        offset[1] = 0;
106    }
107    if(mpi_rank == 1) {
108        offset[0] = 0;
109        offset[1] = chunk_dims[1];
110    }
111    if(mpi_rank == 2) {
112        offset[0] = chunk_dims[0];
113        offset[1] = 0;
114    }
115    if(mpi_rank == 3) {
116        offset[0] = chunk_dims[0];
117        offset[1] = chunk_dims[1];
118    }
```



# Parallel HDF5 Intermediate Tutorial



## Outline

- Performance
- Parallel tools

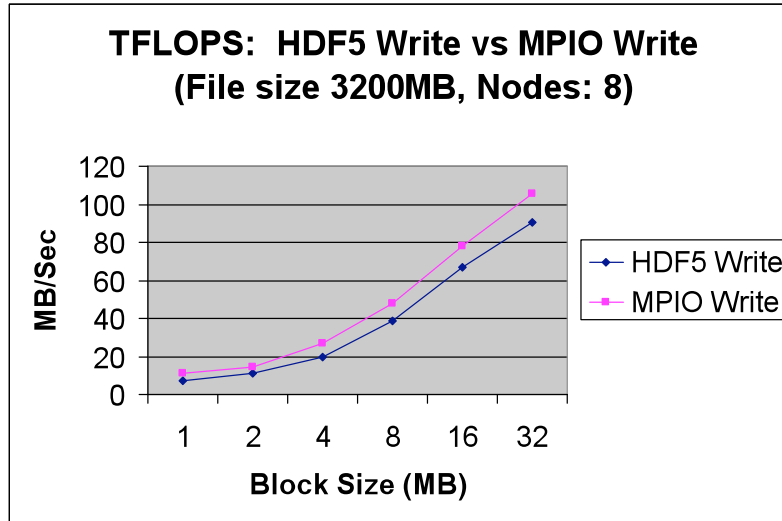


## My PHDF5 Application I/O is slow

- If my application I/O performance is slow, what can I do?
  - Use larger I/O data sizes
  - Independent vs. Collective I/O
  - Specific I/O system hints
  - Increase Parallel File System capacity

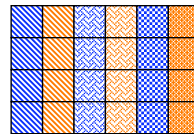


## Write Speed vs. Block Size



## Independent Vs Collective Access

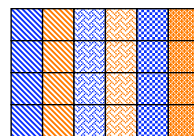
- User reported Independent data transfer mode was much slower than the Collective data transfer mode
- Data array was tall and thin: 230,000 rows by 6 columns



⋮

230,000 rows

⋮





## 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 ==  $4 \times 1000 \times 8 = 32,000$  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.

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

51

[www.hdfgroup.org](http://www.hdfgroup.org)



## 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"`

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

52

[www.hdfgroup.org](http://www.hdfgroup.org)





## Debug Slow Parallel I/O Speed (4)

- % setenv H5FD\_mpio\_Debug 'rw'
- % mpirun -np 4 ./a.out i h 1000 # Indep., Chunked.
- 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.

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

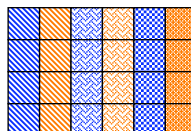
55

www.hdfgroup.org



## Use Collective Mode or Chunked Storage

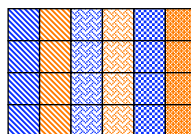
- Collective mode will combine many small independent calls into few but bigger calls==like people going to work by trains collectively.
- Chunks of columns speeds up too==like people live and work in suburbs to reduce overlapping traffics.



⋮

230,000 rows

⋮



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

56

www.hdfgroup.org



## 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              |

September 9, 2008

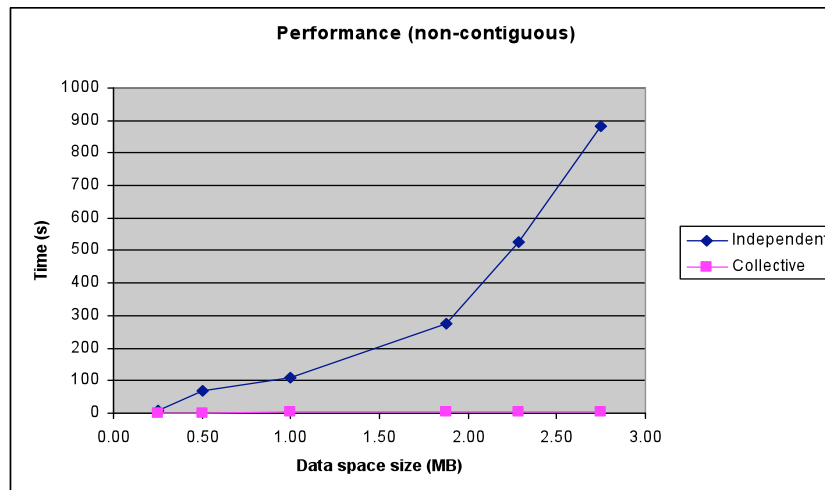
SPEEDUP Workshop - HDF5 Tutorial

57

www.hdfgroup.org



## Independent vs. Collective write (cont.)



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

58

www.hdfgroup.org



## Effects of I/O Hints: IBM\_largeblock\_io

- GPFS at LLNL Blue
  - 4 nodes, 16 tasks
  - Total data size 1024MB
  - I/O buffer size 1MB

| Tasks |              | IBM_largeblock_io=false |       | IBM_largeblock_io=true |       |
|-------|--------------|-------------------------|-------|------------------------|-------|
|       |              | MPI-IO                  | PHDF5 | MPI-IO                 | PHDF5 |
| 16    | write (MB/S) | 60                      | 48    | 354                    | 294   |
| 16    | read (MB/S)  | 44                      | 39    | 256                    | 248   |

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

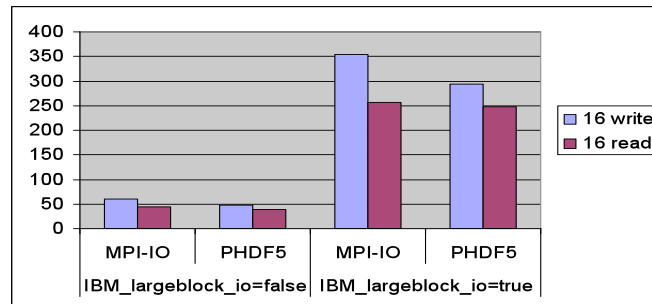
59

www.hdfgroup.org



## Effects of I/O Hints: IBM\_largeblock\_io

- GPFS at LLNL ASCI Blue machine
  - 4 nodes, 16 tasks
  - Total data size 1024MB
  - I/O buffer size 1MB



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

60

www.hdfgroup.org



## Parallel Tools

- ph5diff
  - Parallel version of the h5diff tool
- h5perf
  - Performance measuring tools showing I/O performance for different I/O API



## ph5diff

- An parallel version of the h5diff tool
  - Supports all features of h5diff
  - An MPI parallel tool
  - Manager process (proc 0)
    - coordinates each the remaining processes (workers) to “diff” one dataset at a time;
    - collects any output from each worker and prints them out.
  - Works best if there are many datasets in the two files with few differences.
  - Available in v1.8.



## 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)
    - H5Pset\_fapl\_mpio\_posix (using POSIX I/O)
- 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.



## h5perf: example output 1/3

```
%mpirun -np 4 h5perf # Ran in a Linux system
Number of processors = 4
  Transfer Buffer Size: 131072 bytes, File size: 1.00 MBs
  # of files: 1, # of datasets: 1, dataset size: 1.00 MBs
  IO API = POSIX
  Write (1 iteration(s)):
    Maximum Throughput: 18.75 MB/s
    Average Throughput: 18.75 MB/s
    Minimum Throughput: 18.75 MB/s
  Write Open-Close (1 iteration(s)):
    Maximum Throughput: 10.79 MB/s
    Average Throughput: 10.79 MB/s
    Minimum Throughput: 10.79 MB/s
  Read (1 iteration(s)):
    Maximum Throughput: 2241.74 MB/s
    Average Throughput: 2241.74 MB/s
    Minimum Throughput: 2241.74 MB/s
  Read Open-Close (1 iteration(s)):
    Maximum Throughput: 756.41 MB/s
    Average Throughput: 756.41 MB/s
    Minimum Throughput: 756.41 MB/s
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

65

www.hdfgroup.org



## h5perf: example output 2/3

```
%mpirun -np 4 h5perf
...
  IO API = MPIO
  Write (1 iteration(s)):
    Maximum Throughput: 611.95 MB/s
    Average Throughput: 611.95 MB/s
    Minimum Throughput: 611.95 MB/s
  Write Open-Close (1 iteration(s)):
    Maximum Throughput: 16.89 MB/s
    Average Throughput: 16.89 MB/s
    Minimum Throughput: 16.89 MB/s
  Read (1 iteration(s)):
    Maximum Throughput: 421.75 MB/s
    Average Throughput: 421.75 MB/s
    Minimum Throughput: 421.75 MB/s
  Read Open-Close (1 iteration(s)):
    Maximum Throughput: 109.22 MB/s
    Average Throughput: 109.22 MB/s
    Minimum Throughput: 109.22 MB/s
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

66

www.hdfgroup.org



## h5perf: example output 3/3

```
%mpirun -np 4 h5perf
...
IO API = PHDF5 (w/MPI-I/O driver)
Write (1 iteration(s)):
  Maximum Throughput: 304.40 MB/s
  Average Throughput: 304.40 MB/s
  Minimum Throughput: 304.40 MB/s
Write Open-Close (1 iteration(s)):
  Maximum Throughput: 15.14 MB/s
  Average Throughput: 15.14 MB/s
  Minimum Throughput: 15.14 MB/s
Read (1 iteration(s)):
  Maximum Throughput: 1718.27 MB/s
  Average Throughput: 1718.27 MB/s
  Minimum Throughput: 1718.27 MB/s
Read Open-Close (1 iteration(s)):
  Maximum Throughput: 78.06 MB/s
  Average Throughput: 78.06 MB/s
  Minimum Throughput: 78.06 MB/s
Transfer Buffer Size: 262144 bytes, File size: 1.00 MBs
# of files: 1, # of datasets: 1, dataset size: 1.00 MBs
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

67

www.hdfgroup.org



## 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/>
- HDF Help email address  
help@hdfgroup.org

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

68

www.hdfgroup.org



# Questions?