HDF5 I/O Performance

HDF and HDF-EOS Workshop VI

December 5, 2002

Goal of this talk

- Give an overview of the HDF5 Library tuning knobs for sequential and parallel performance

Challenging task

- HDF5 Library has to perform well on
  - Variety of UNIX Workstation (SGI, Intel, HP, Sun)
  - Windows
  - Cray
  - DOE supercomputers (IBM SP, Intel Tflops)
  - Linux clusters (Compaq, Intel)
- Variety of file systems (GPFS, PVFS, Unix FS)
- Variety of MPI-IO implementations
- Other tasks
  - Efficient memory and file space management
- *Applications are different (access patterns, many small objects vs. few large objects, parallel vs. sequential, etc.)*

Outline

- **Sequential performance**
  - ***Tuning knobs***
    - ***File level***
    - ***Data transfer level***
  - **Memory management**
  - **File space management: Fill values and storage allocation**
  - **Chunking**
    - **Compression**
    - **Caching**
  - **Compact storage**

Outline

- **Parallel performance**
  - ***Tuning knobs***
    - ***Data alignment***
    - ***MPI-IO hints***
    - ***HDF5 Split Driver***
  - **h5perf benchmark**

Sequential Performance

- **Tuning knobs**

Two Sets of Tuning Knobs

- **File level knobs**
  - Apply to the entire file
- **Data transfer level knobs**
  - Apply to individual dataset read or write

File Level Knobs

- H5Pset_meta_block_size
- H5Pset_cache

H5Pset_meta_block_size

- Sets the minimum metadata block size allocated for metadata aggregation.
- Aggregated block is usually written in a single write action
- Default is 2KB
- *Pro:*
  - Larger block size reduces I/O requests
- *Con:*
  - Could create “holes” in the file and make file bigger

H5Pset_meta_block_size

- **When to use:**
- **File is open for a long time and**
  - **A lot of objects created**
  - **A lot of operations on the objects performed**
  - **As a result metadata is interleaved with raw data**
  - **A lot of new metadata (attributes)**

H5Pset_cache

- **Sets:**
  - The number of elements (objects) in the meta data cache
  - The number of elements, the total number of bytes, and the preemption policy value (default is 0.75) in the raw data chunk cache

H5Pset_cache (cont.)

- **Preemption policy:**
  - Chunks are stored in the list with the most recently accessed chunk at the end
  - Least recently accessed chunks are at the beginning of the list
  - X*100% of the list is searched for the fully read/written chunk; X is called preemption value, where X is between 0 and 1
  - If chunk is found then it is deleted from cache, if not then first chunk in the list is deleted

H5Pset_cache (cont.)

- The right values of X
  - May improve I/O performance by controlling preemption policy
  - 0 value forces to delete the “oldest” chunk from cache
  - 1 value forces to search all list for the chunk that will be unlikely accessed
  - Depends on application access pattern

Data Transfer Level Knobs

- **H5Pset_buffer**
- **H5Pset_sieve_buf_size**

H5Pset_buffer

- Sets size of the internal buffers used during data transfer
- Default is 1 MB
- Pro:
  - Bigger size improves performance
- Con:
  - Library uses more memory

H5Pset_buffer

- **When should be used:**
  - Datatype conversion
  - Data gathering-scattering (e.g. checker board dataspace selection)

H5Pset_sieve_buf_size

- Sets the size of the data sieve buffer
- Default is 64KB
- Sieve buffer is a buffer in memory that holds part of the dataset raw data
- During I/0 operations data is replaced in the buffer first, then one big I/0 request occurs

H5Pset_sieve_buf_size

- Pro:
  - Bigger size reduces I/O requests issued for raw data access
- Con:
  - Library uses more memory
- When to use:
  - Data scattering-gathering (e.g. checker board)
  - Interleaved hyperslabs

HDF5 Application Memory Management

- H5garbage_collect()
  - Memory used by HDF5 application may grow with the growing number of the objects created and then released
    - Function walks through all the garbage collection routines of the library, freeing any unused memory
- When to use:
- Application creates-opens-releases substantial number of objects
- “Number of objects” is application and platform dependent

HDF5 File Space Management

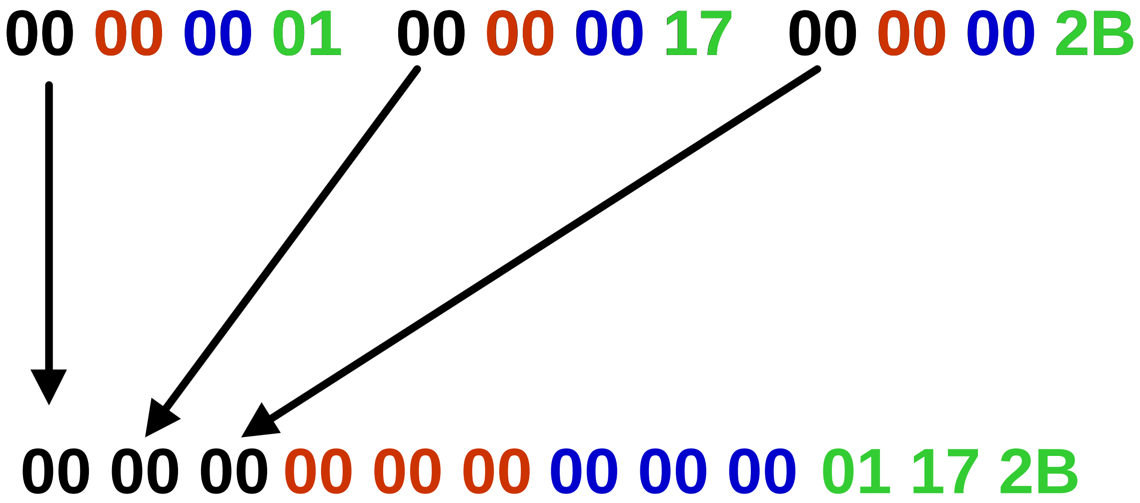
- H5Pset_alloc_time
  - Sets the time of data storage allocation for creating a dataset
    - `Early when dataset is created`
    - `Late when dataset is written`
- H5Pset_fill_time
  - Sets the time when fill values are written to a dataset
    - `When space allocated`
    - `Never`
  - Avoids unnecessary writes

Chunking and Compression

- Chunking storage
  - Provides better partial access to dataset
  - Space is allocated when data is written
  - *Con:*
    - *Storage overhead*
    - *May degrade performance if cache is not set up properly*
- Compression (GZIP, SZIP in HDF5 1.5 release)
  - Saves space
  - User may easily turn on their own compression method
  - *Con:*
    - *May take a lot of time*
- Data shuffling (in HDF5 1.5 release)
  - Helps compression algorithms

Data shuffling

- See Kent Yang’s poster
- Not a compression; change of byte order in a stream of data
- Example
  - 1 23 43
- Hexadecimal form
  - 0x01 0x17 0x2B
- Big-endian machine
  - 0x00 0x00 0x00 0x01 0x00 0x00 0x00 0x17 0x00 0x00 0x00 0x2B
- Shuffling
  - 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 0x17 0x2B



00 00 00 01   00 00 00 17   00 00 00 2B

00 00 00 00 00 00 00 00 00 01 17 2B

Chunking and compression benchmark

- Write one 4-byte integer dataset 256x256x1024 (256MB)
- Using chunks of 256x16x1024 (16MB)
- Random integers between 0 and 255
- Tests with
  - Compression on/off
  - Chunk cache size 16MB to 256MB
  - Data shuffling

Chunking Benchmark
Time Definitions

- Total
  - Time to open file, write dataset, close dataset and close file
- Write time
  - Time to write the whole dataset
- Average chunk time
  - Total time/ number of chunks

Performance improvement

Release version	Total time (Open-write-close) in seconds	Average time to write a 16MB chunk In seconds
1.4.4 release	8.6950	0.4809
1.4.5-prerelease	4.5711	0.2447

Effect of Caching (H5Pset_cache)

Compression	Cache	Total time in seconds	Write time in seconds
No File size 268.4MB	16MB	5.607	5.43
No	256MB	5.79	3.63
Yes File size 102.9MB	16MB	672.58	630.89
Yes	256MB	674.66	3.48

Effect of data shuffling (H5Pset_shuffle + H5Pset_deflate)

File size	Total time	Write Time
102.9MB	671.049	629.45
67.34MB	83.353	78.268

Compression combined with shuffling provides

- Better compression ratio
- Better I/O performance

Effect of chunk caching and data shuffling

H5Pset_cache + H5Pset_shuffle + H5Pset_deflate

Cache	Total time	Write Time
16MB	83.353	78.268
128MB	82.942	43.257
256MB	82.972	3.476

- Caching improves chunk write time

Compact storage

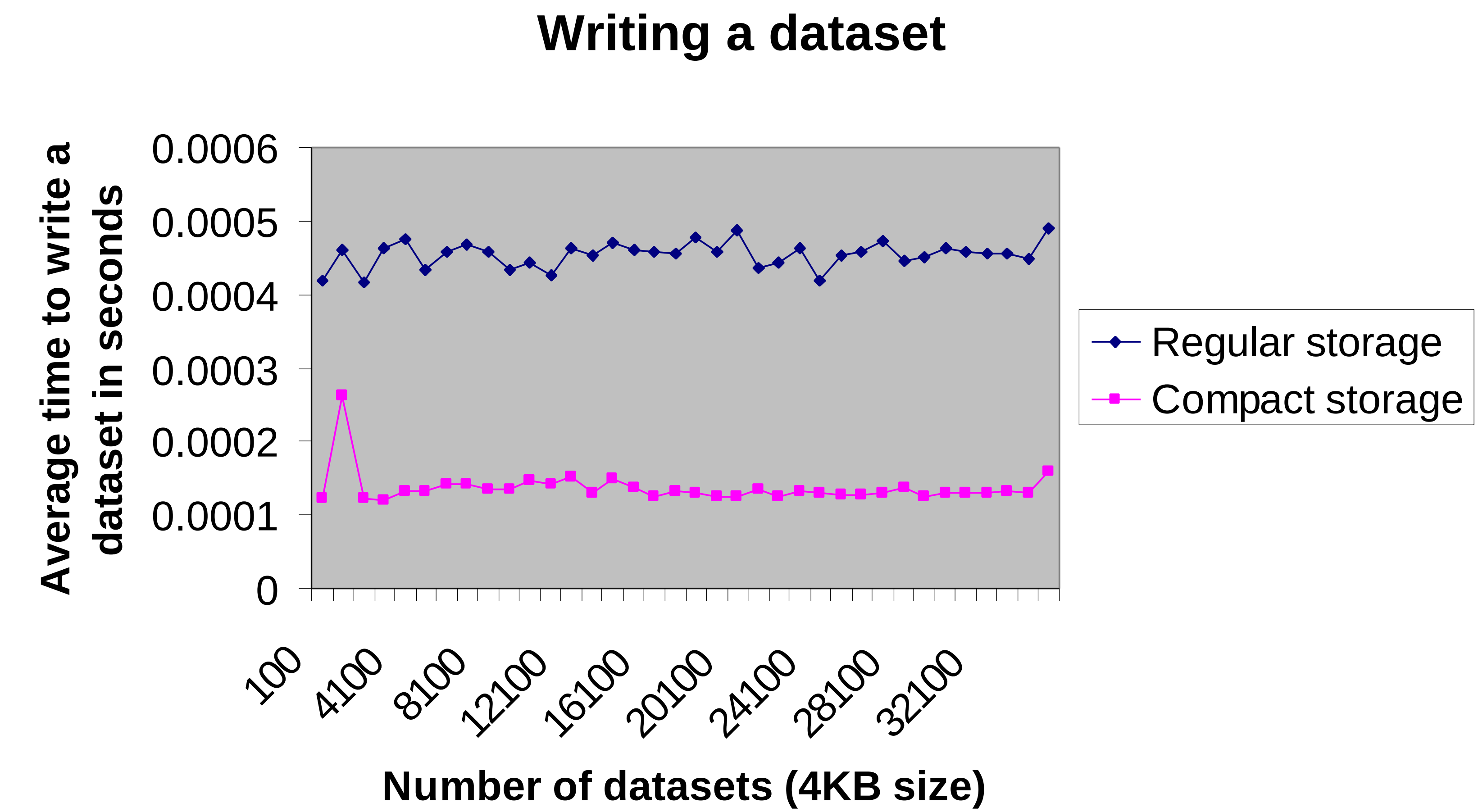
- Store small objects (e.g. 4KB dataset) in the file
  - C code example:

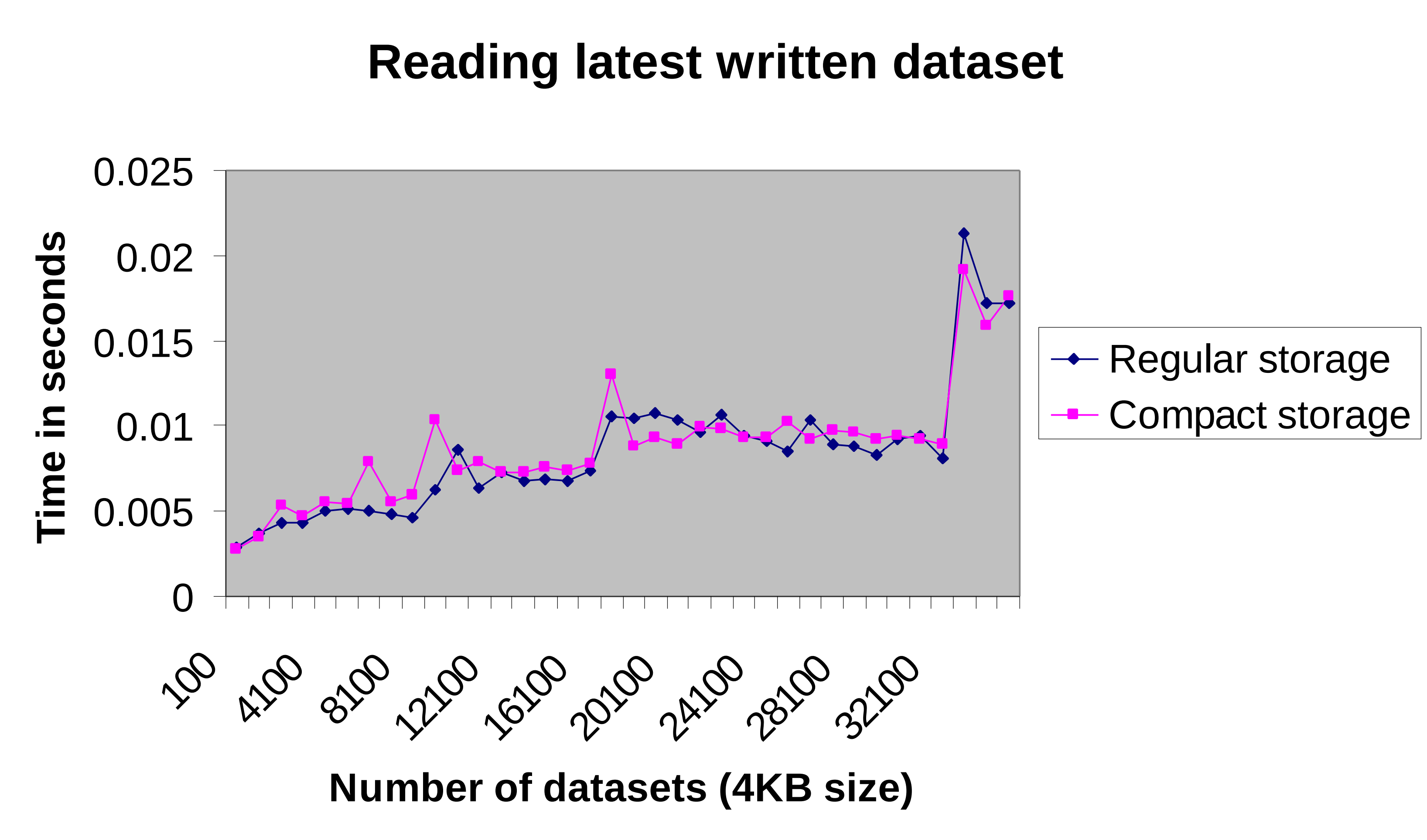
    ```
    plist = H5Pcreate(H5P_DATASET_CREATE);
    H5Pset_layout(plist, H5D_COMPACT);
    H5Pset_alloc_time(plist,H5D_ALLOC_TIME_EARLY);
    dataset = H5Dcreate(file,…, plist);
    ```

  - Raw data is stored in the dataset header
    - Metadata and raw data are written/read in one I/0 operation
    - Faster write and read

Compact storage benchmark

- Create a file with N 4KB datasets using regular and compact storage (100 < N < 35000)
- Measure average time needed to write/read a dataset in a file with N datasets
- Benchmark run on Linux 2.2.18 i686, 960MB memory
- `timeofday` function used to measure time


Writing a dataset
Average time to write a dataset in seconds
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
100
4100
8100
12100
16100
20100
24100
28100
32100
Number of datasets (4KB size)
Regular storage
Compact storage

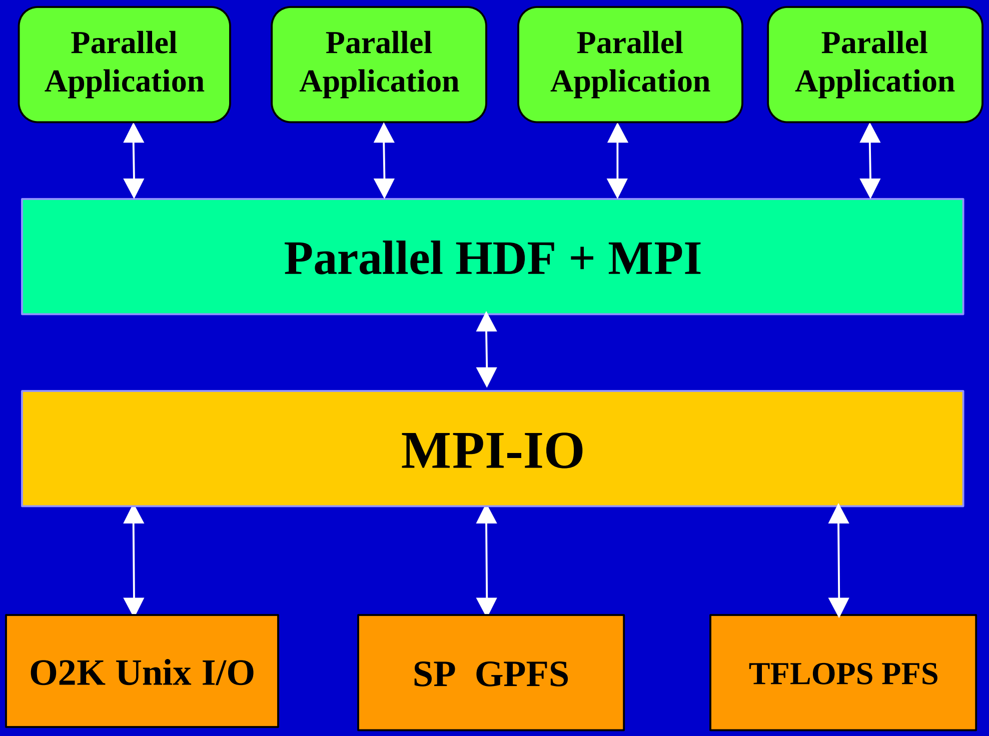


Reading latest written dataset
Time in seconds
0.025
0.02
0.015
0.01
0.005
0
100
4100
8100
12100
16100
20100
24100
28100
32100
Number of datasets (4KB size)
Regular storage
Compact storage

Parallel Performance

- ***Tuning knobs***
- **h5perf benchmark**

PHDF5 Implementation Layers



Parallel Application

Parallel Application

Parallel Application

Parallel Application

Parallel HDF + MPI

MPI-IO

O2K Unix I/O

SP GPFS

TFLOPS PFS

User Applications

HDF library

Parallel I/O layer

Parallel File systems

File Level Knobs (Parallel)

- H5Pset_alignment
- H5Pset_fapl_split
- H5Pset_fapl_mpio

H5Pset_alignment

- Sets two parameters
  - Threshold
    - Minimum size of object for alignment to take effect
    - Default 1 byte
  - Alignment
    - Allocate object at the next multiple of alignment
    - Default 1 byte
- Example: (threshold, alignment) = (1024, 4K)
  - All objects of 1024 or more bytes starts at the boundary of 4KB

H5Pset_alignment
Benefits

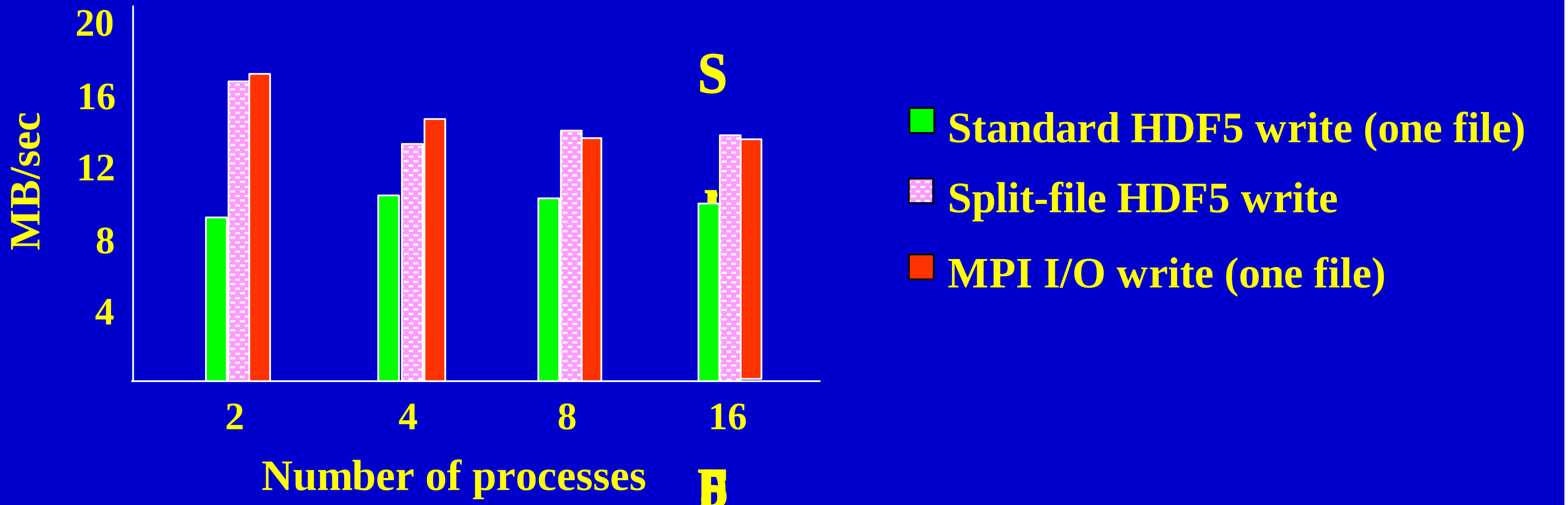
- In general, the default (no alignment) is good for single process serial access since the OS already manages buffering.
- For some parallel file systems such as GPFS, an alignment of the disk block size improves I/O speeds.

H5Pset_fapl_split

- **HDF5 splits to two files**
  - **Metadata file for metadata**
  - **Raw data file for raw data (array data)**
- **Significant I/O improvement if**
  - **metadata file is stored in Unix file systems (good for many small I/O)**
  - **raw data file is stored in Parallel file systems (good for large I/O).**

Results for ASCI Red machine at Sandia National Laboratory

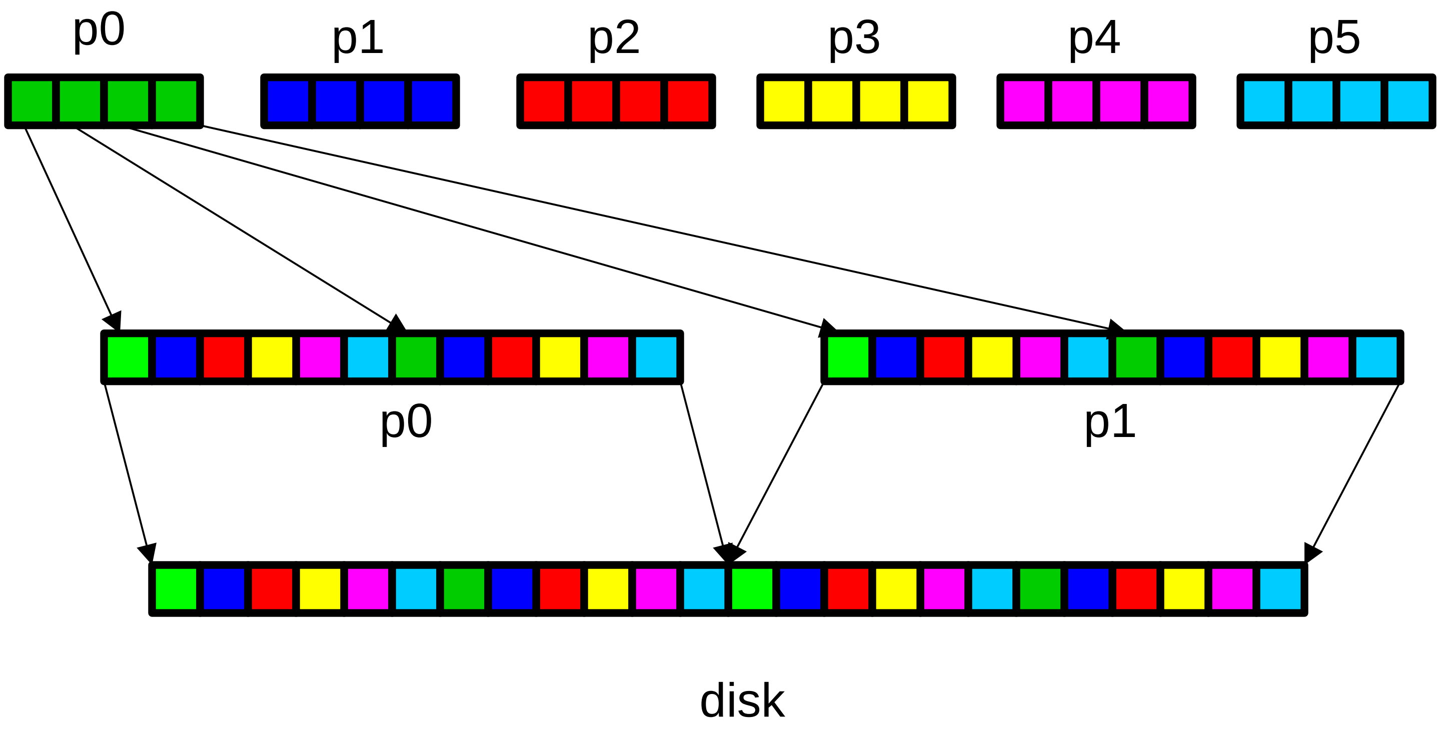
- Each process writes 10MB of array data



I/O Hints via H5Pset_fapl_mpio

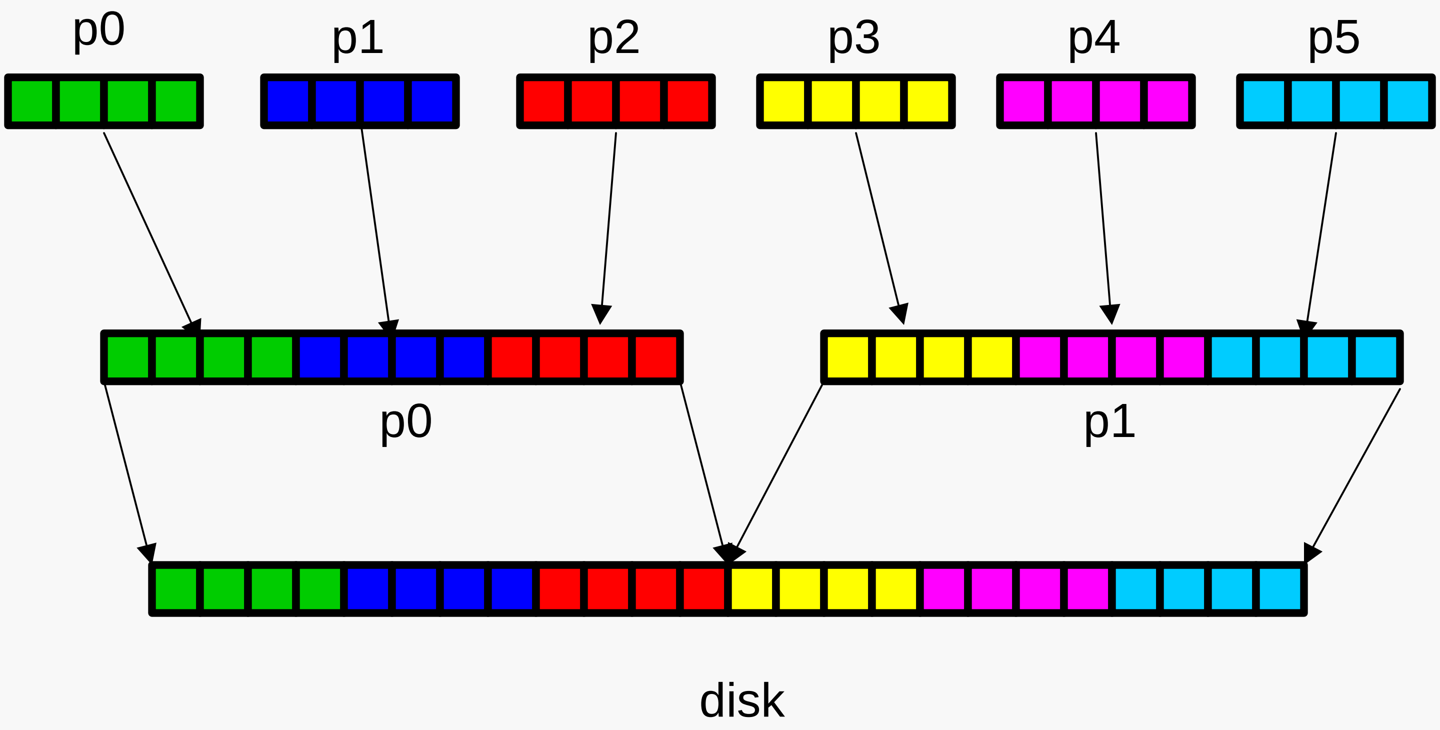
- MPI-IO hints can be passed to the MPI-IO layer via the Info parameter of H5Pset_fapl_mpio
- Examples
  - Telling Romio to use 2-phase I/O speeds up collective I/O in the ASCI Red machine at Livermore National Laboratory
  - Setting IBM_largeblock_io=true speeds up GPFS write speeds

2-Phase I/O



- Interleaving

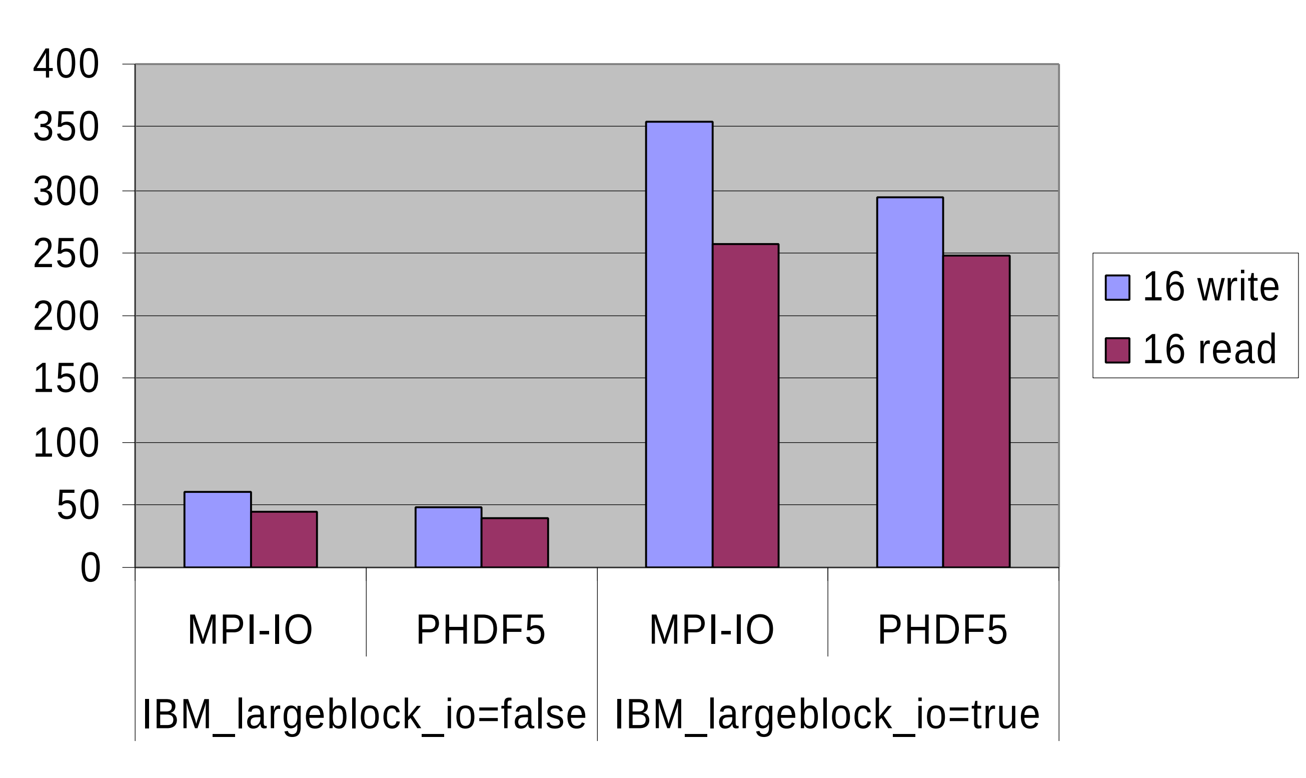
2-Phase I/O



- Aggregation (available in ROMIO 1.2.4); useful for
  - filling I/O buffers
  - moving data to processors that have better connectivity

Effects of I/O Hints
IBM_largeblock_io

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



Parallel I/O
Benchmark Tool

- h5perf
  - Benchmark test I/O performance
  - Comes with HDF5 binaries
  - Writes datasets into the file by hyperslabs
  - Variables:
    - Number of datasets
    - Number of processes
    - Number of bytes per process per dataset to write/read
    - Threshold for data alignment
    - Size of transfer buffer (memory buffer) and block per process
    - Collective vs. Independent
    - Interleaved blocks vs. contiguous blocks

Parallel I/O
Benchmark Tool

- Four kinds of API
  - Parallel HDF5
  - MPI-IO
  - Native parallel (e.g., gpfs, pvfs)
  - POSIX (open, close, lseek, read, write)
- Provides standard approach to measure and compare performance results

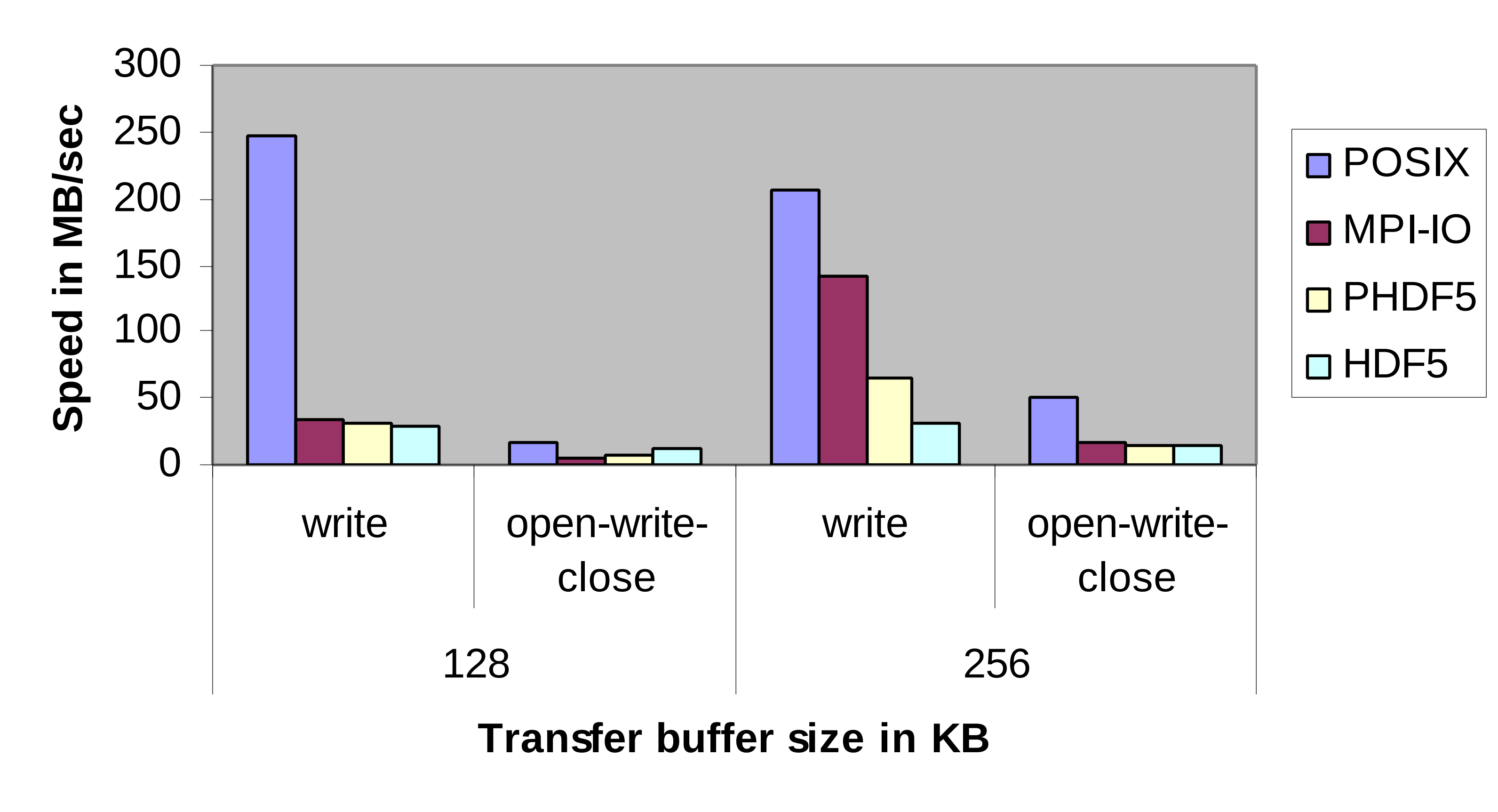
Parallel I/O Tuning

- Challenging task
  - Performance vary from platform to platform
  - Complex access patterns
  - Many layers can be involved
    - SAF-HDF5-MPIO-GPFS

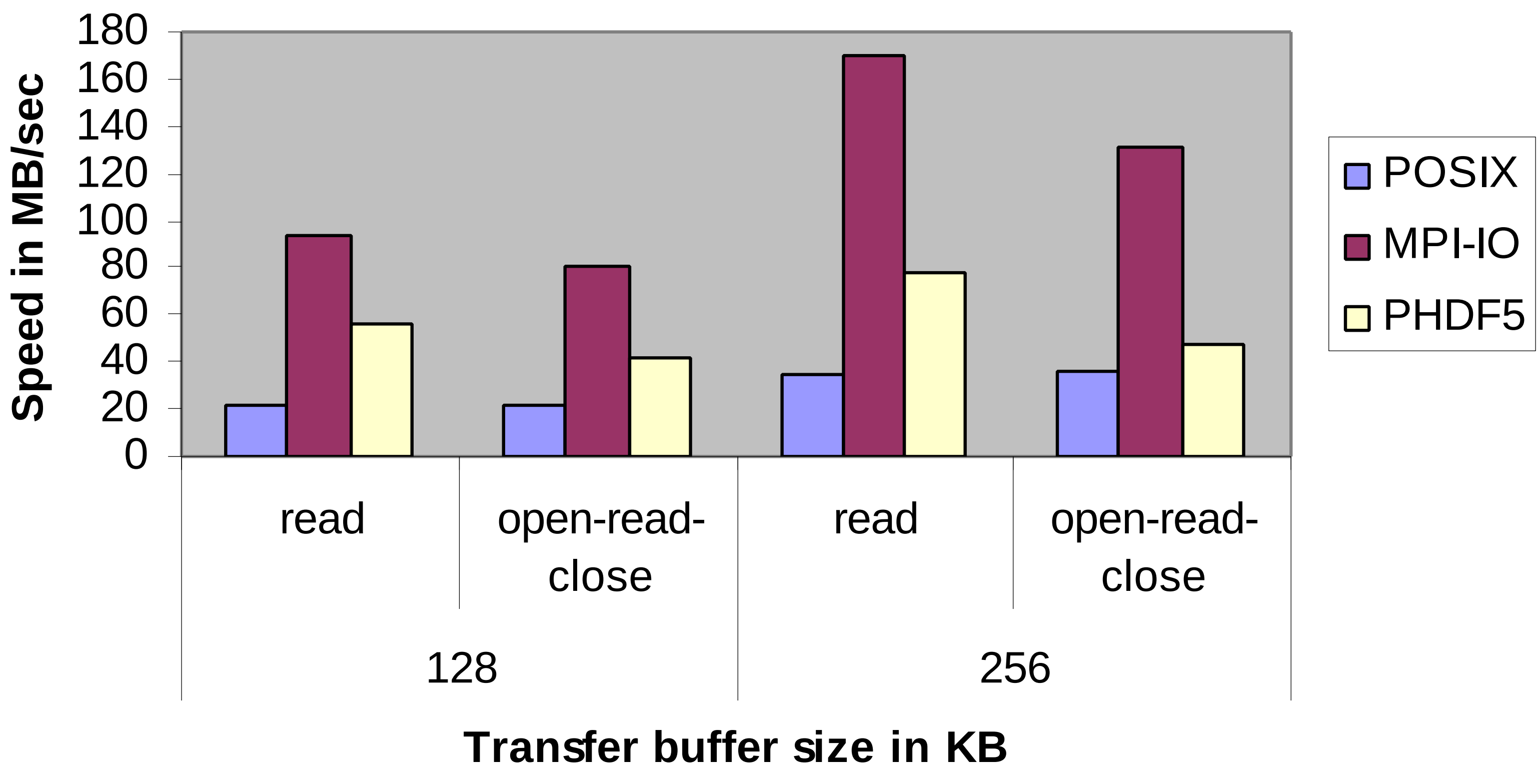
Parallel I/O Tuning
Example of transfer buffer effect

- h5perf run on NERSC IBM SP
- 4 processes, 4 nodes, 1MB file, 1 dataset, 256KB data per process to write
- Maximum achieved write speed in MB/sec

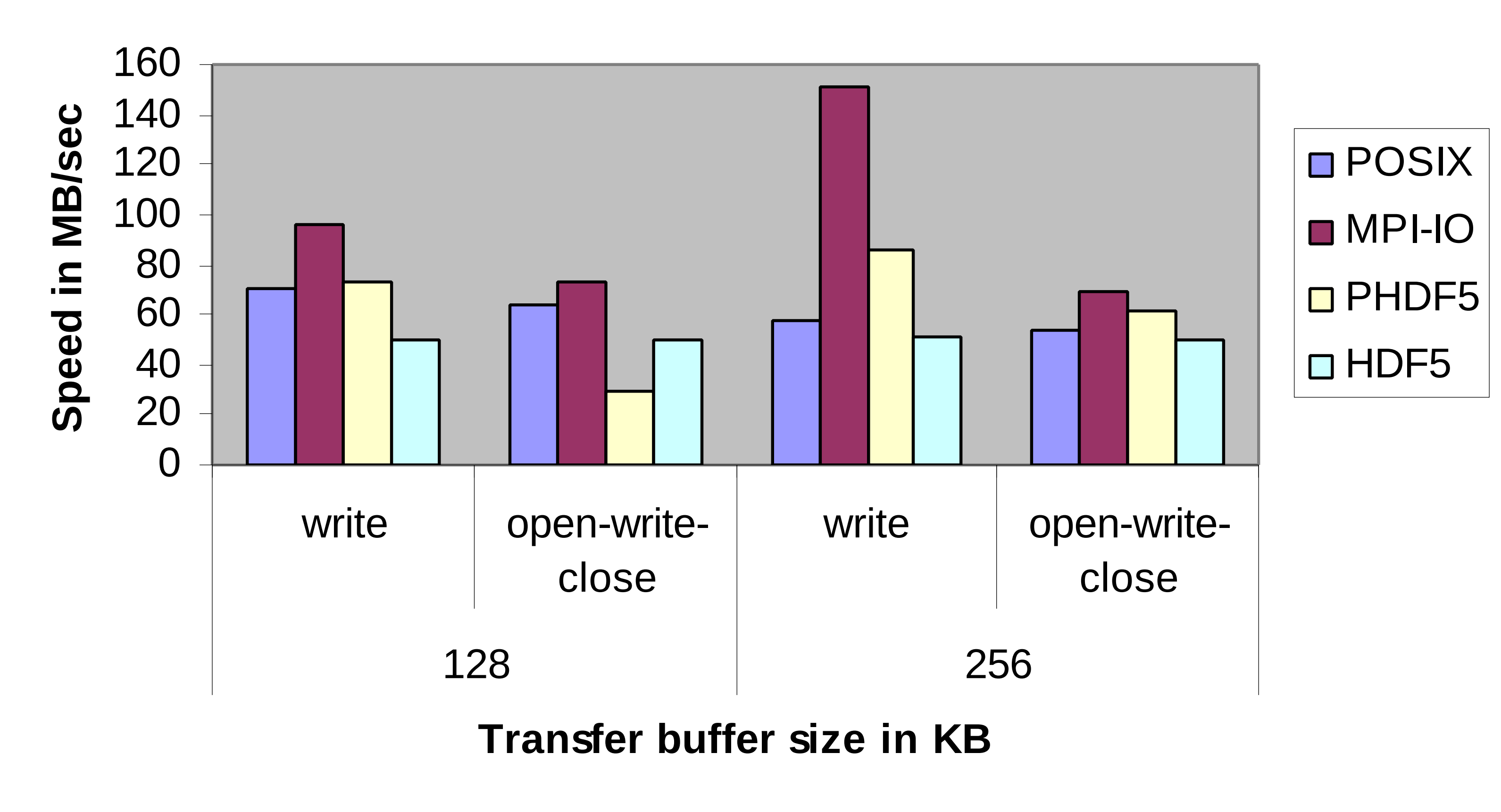
Parallel I/O Tuning
Example of transfer buffer effect on SP2



Parallel I/O Tuning
Example of transfer buffer effect on SP2



Parallel I/O Tuning
Example of transfer buffer effect on SGI



Summery results for Blue

- h5perf run on ASCI IBM SP Blue
- 1 to 4 processes per node, 16 nodes, 256KB data per process to write/read, 256 KB transfer size, 256KB block size
- Varied:
  - Number of tasks per node (1 – 4)
  - Number of datasets 50, 100, 200
  - Independent or collective calls

HDF5 collective write results

Number of datasets	Tasks per node (TPN) 1 – 2 Speed in MB.sec	Tasks per node (TPN) 3 – 4 Speed in MB/sec	MPI-IO best result Speed in MB/sec
50	20 – 30	50 – 60	68.47 3 TPN
100	40 – 60	50 – 60	66.24 3 TPN
200	40 – 60	25 – 45	52.66 2 TPN

HDF5 independent write results

Number of datasets	Tasks per node 1 – 2 Speed in MB.sec	Tasks per node 3 – 4 Speed in MB/sec	MPI-IO best result Speed in MB/sec
50	20 – 35	20 – 35	32.14 2 TPN
100	20 – 35	20 – 35	31.09 4 TPN
200	20 – 35	35 – 60	61.06 3 TPN

Read performance

Mode	PHDF5 Speed in MB/sec	MPI-IO Speed in MB/sec
Collective	75 - 200	335 - 1800
Independent	100 - 650	330 - 2935

Future Parallel HDF5 Features

- Flexible PHDF5
  - Reduces the needs of collective calls
  - Set aside a process for independent calls coordination
  - Estimated release date: end of 2002

Useful Parallel HDF Links

- Parallel HDF information site
  - http://hdf.ncsa.uiuc.edu/Parallel_HDF/
- Parallel HDF mailing list
  - hdfparallel@ncsa.uiuc.edu
- Parallel HDF5 tutorial available at
  - http://hdf.ncsa.uiuc.edu/HDF5/doc/Tutor