

Improving Parallel I/O Performance Using Multithreaded Two-Phase I/O with Processor Affinity Management *

Yuichi Tsujita^{1,4}, Kazumi Yoshinaga^{1,4}, Atsushi Hori^{1,4},
Mikiko Sato^{2,4}, Mitaro Namiki^{2,4}, Yutaka Ishikawa³

¹ RIKEN Advanced Institute for Computational Science (AICS)

² Tokyo University of Agriculture and Technology

³ The University of Tokyo



⁴ JST CREST

* This research work was partially supported by JST CREST

Outline

1. Motivation
2. MPI-IO
3. HDF5
4. Two-Phase I/O
5. Multithreaded Two-Phase I/O
6. CPU Core Affinity Management in Multithreaded Two-Phase I/O
7. Performance Evaluation for Parallel HDF5
8. Related Work
9. Concluding Remarks

Motivation (1)

- Increase in data size due to recent huge scale of parallel computation
 **Parallel I/O**
- MPI-IO: Parallel I/O APIs in the MPI standard (MPI-2)
 - ROMIO: One of the widely used MPI-IO implementations
- Variety of I/O access patterns
 - Non-contiguous I/O: One of the performance bottlenecks
 **Performance improvement**
- Two-Phase I/O in ROMIO
 - Improvement in throughput for non-contiguous accesses in parallel I/O
 - ✗ Limitation in performance improvement

There is room to improve I/O throughput.

Motivation (2)

- Our proposal
 - **Multithreaded** Two-Phase I/O by using a Pthreads library
 - Overlapping file I/O with data communications
 - CPU core management for threads
- MPI-IO layer
 - Performance improvements
 - Minimization in memory footprints
- Parallel HDF5 layer
 - Further performance improvements

Evaluation of parallel HDF5
as an application example

MPI-IO (1)

- MPI-IO

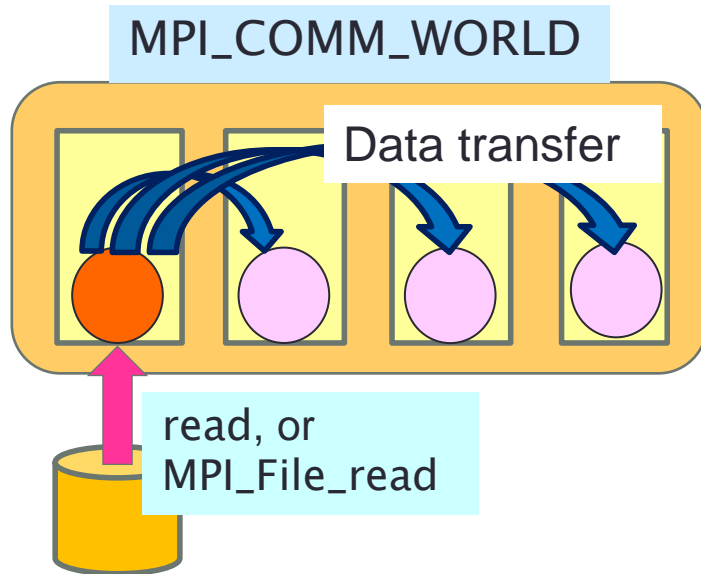
- I/O interface in the MPI standard (MPI-2)
- Provides various kinds of I/O interfaces including parallel I/O

- ROMIO

- An MPI-IO implementation in MPICH
 - Incorporated in other MPI implementations such as OpenMPI
 - Supports many parallel file systems such as Lustre or PVFS2 through an ADIO interface layer

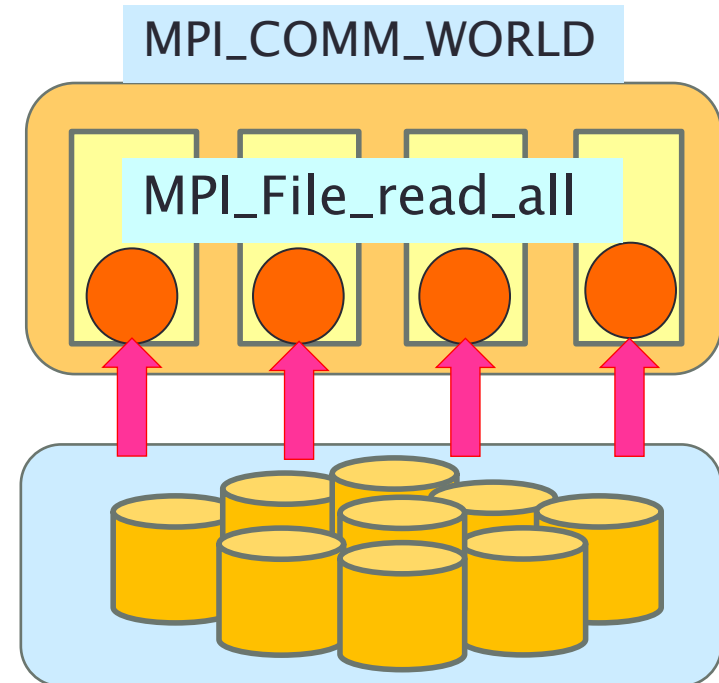
MPI-IO (2)

● Collective I/O



a. Independent I/O

- Large overhead in data communications
- Bottleneck in the representative process to perform local I/O



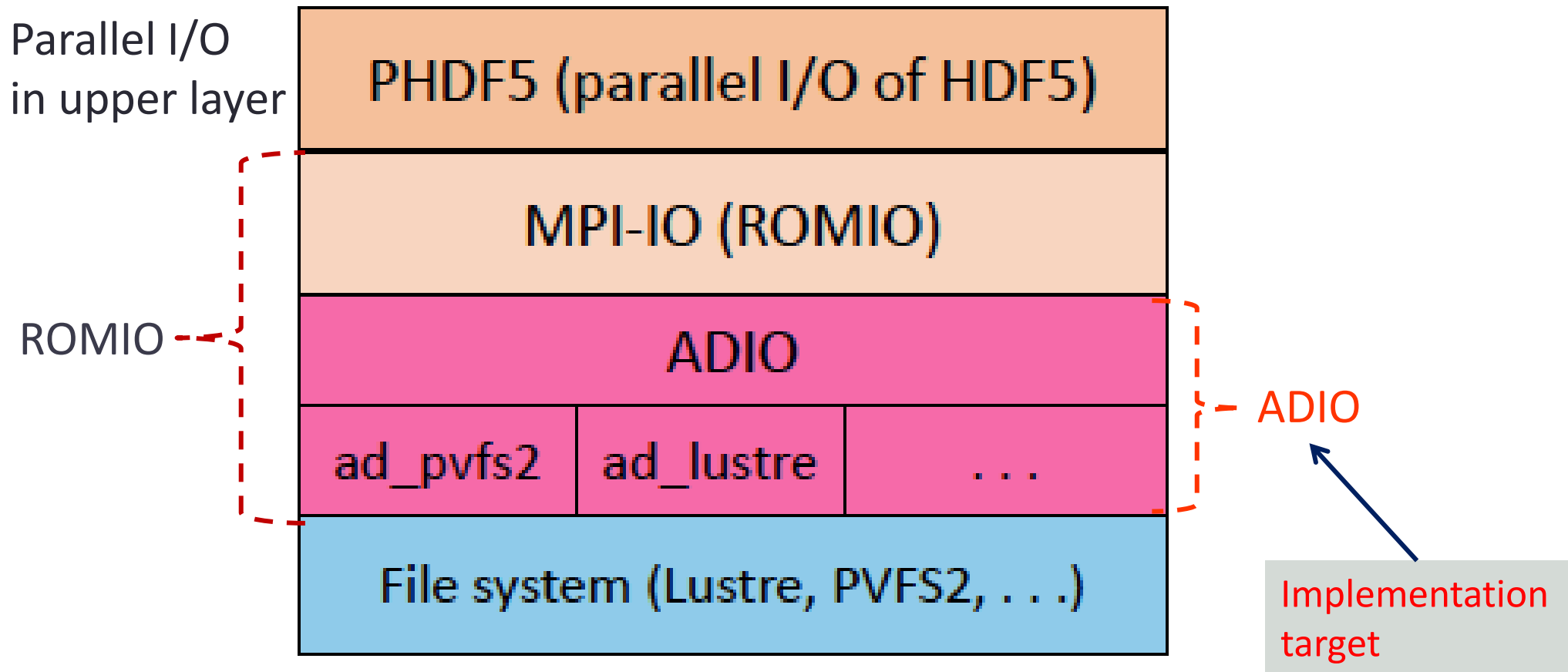
Parallel file system (e.g., Lustre, PVFS2, ...)

b. Collective I/O

- High throughput by using parallel I/O
- Suitable for parallel file systems such as Lustre

MPI-IO (3)

- Software stack of ROMIO

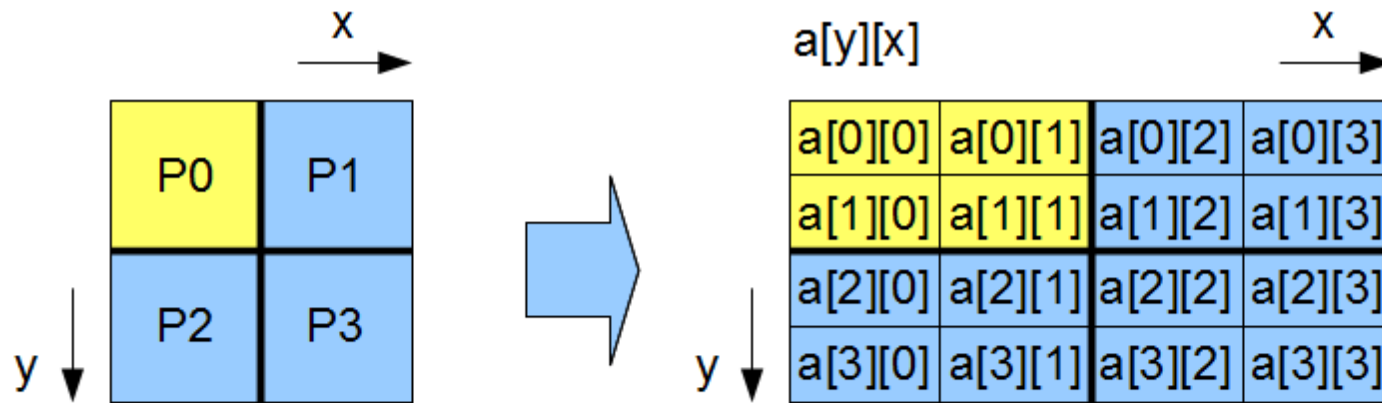


HDF5

- Hierarchical Data Format 5 (HDF5)
 - Developed by the HDF Group (<http://www.hdfgroup.org/>)
 - Hierarchical and self-describing data format
 - HDF5 file organization
 - HDF5 group : a grouping structure containing HDF5 objects
 - HDF5 dataset : A multidimensional array of data elements
 - Parallel I/O part (parallel HDF5)
 - realized by using MPI-IO interface APIs
 - MPI-IO functions (independent, **collective**)
 - MPI functions to generate **derived data types**, and so forth

Collective I/O for derived data type access patterns plays a big role to manage data-intensive scientific applications.

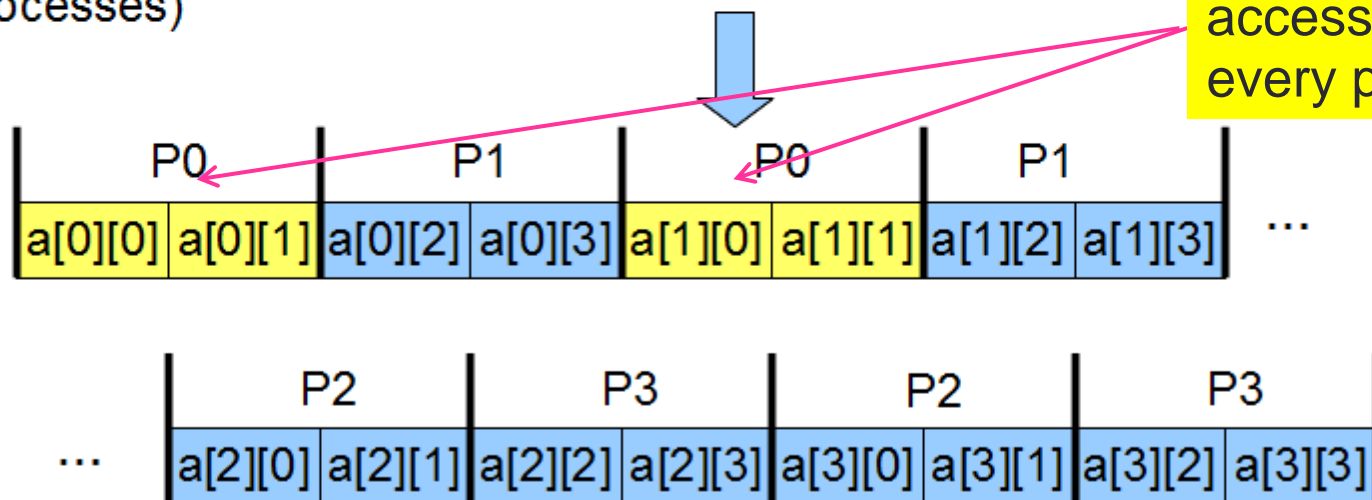
Two-Phase I/O (1)



Decomposition for 2-D
data file image
(4 processes)

e.g., decomposition into four
regions

Non-contiguous
accesses in
every process



Real data access pattern on a target file
(row-order)

Two-Phase I/O improves collective I/O throughput in non-contiguous accesses.

Two-Phase I/O (2)

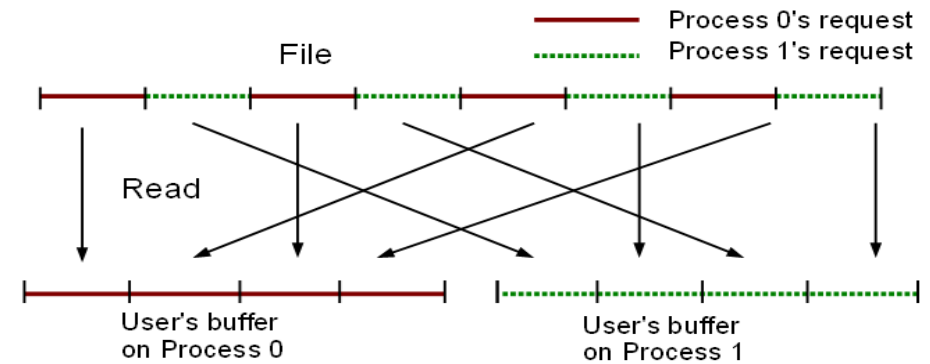
- Two-Phase I/O

- File accesses
- Data exchanges

- Collective read accesses

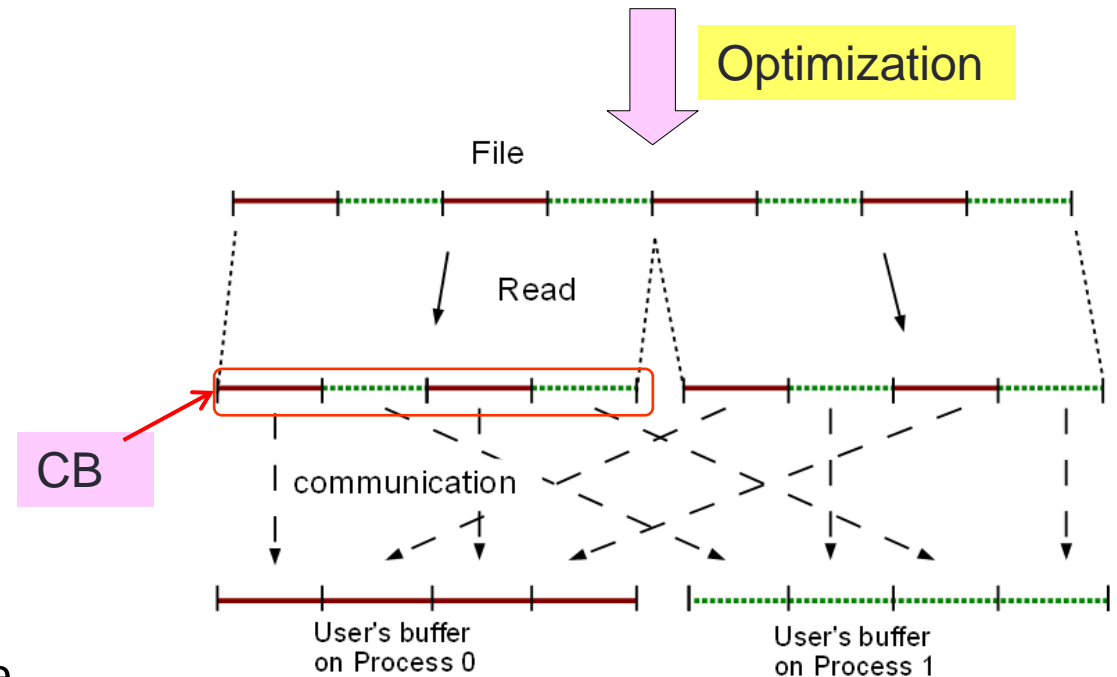
1. Total file region is evenly divided between MPI processes which play I/O (I/O aggregator).
2. Read from an assigned partial file region to a temporary buffer named collective buffer (CB)
3. Data communications to collect target data

* Collective write accesses are inversely operated.



Performance degradation by large number of Independent small I/O accesses

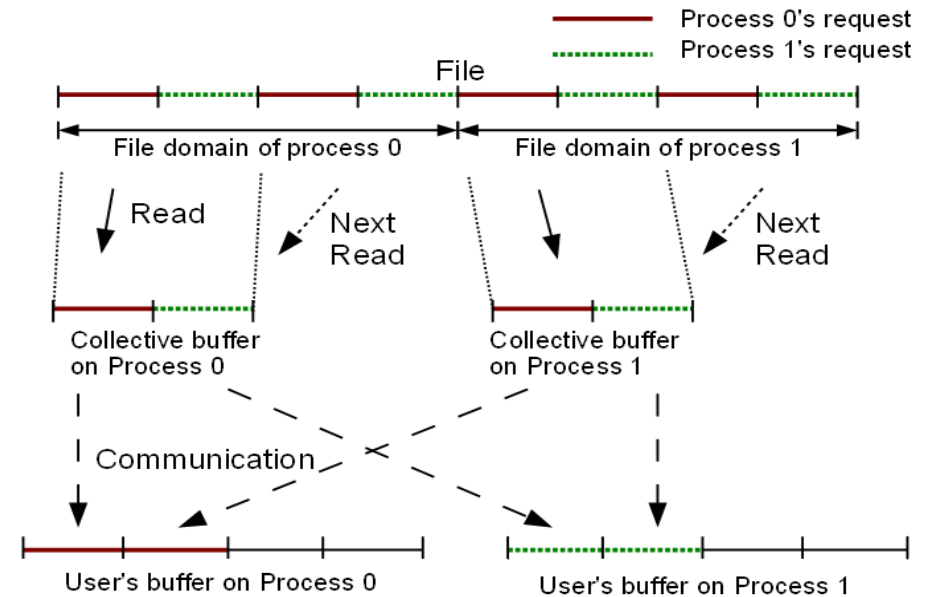
Optimization



Two-Phase I/O optimization

Two-Phase I/O (3)

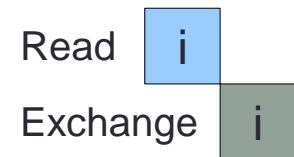
- TP-I/O cycles
 - One I/O access is aligned to CB size.
 - Multiple accesses (TP-I/O cycles)
 - 1 cycle consisting file I/O and data communications is repeated
 - TP-IO cycles are repeated until whole data region is accessed.



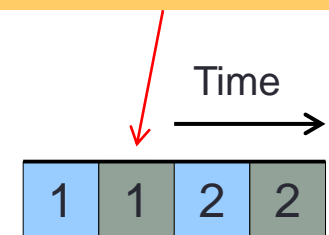
< Task flow in the case of two-cycles >

- We still have room to overlap file I/O phases with data communication phases.

i-th cycle request



Serialized operations !



< Two-Phase I/O flow in each MPI process >

Multithreaded Two-Phase I/O (1)

● Pipelined operations

- Overlap of data exchange with file I/O
- Increase in the number of cycles leads to higher overlap ratio between data exchange and file I/O.
 - Minimization of total utilized memory size

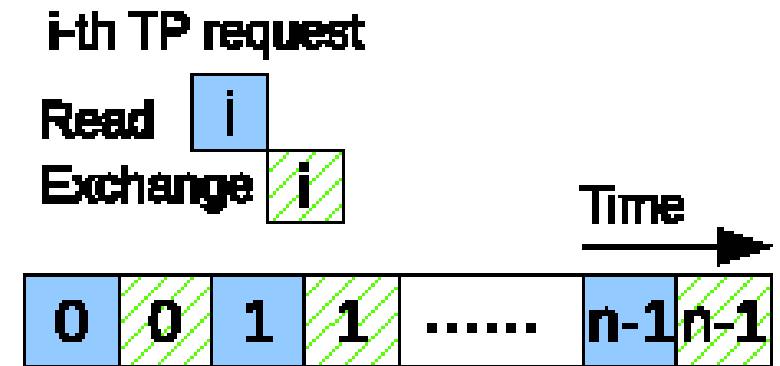


● Two proposed Implementations

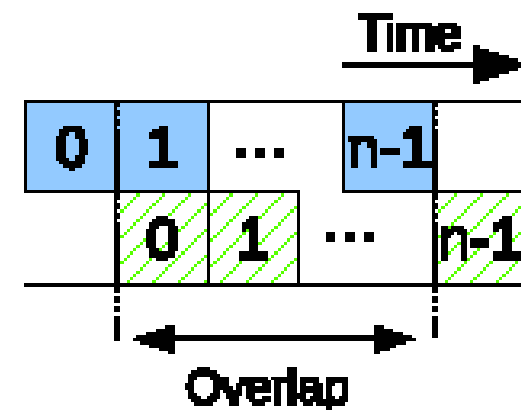
1. Multithreaded implementation using a Pthreads library

- Available on many platforms
- Portable API

2. Asynchronous I/O API

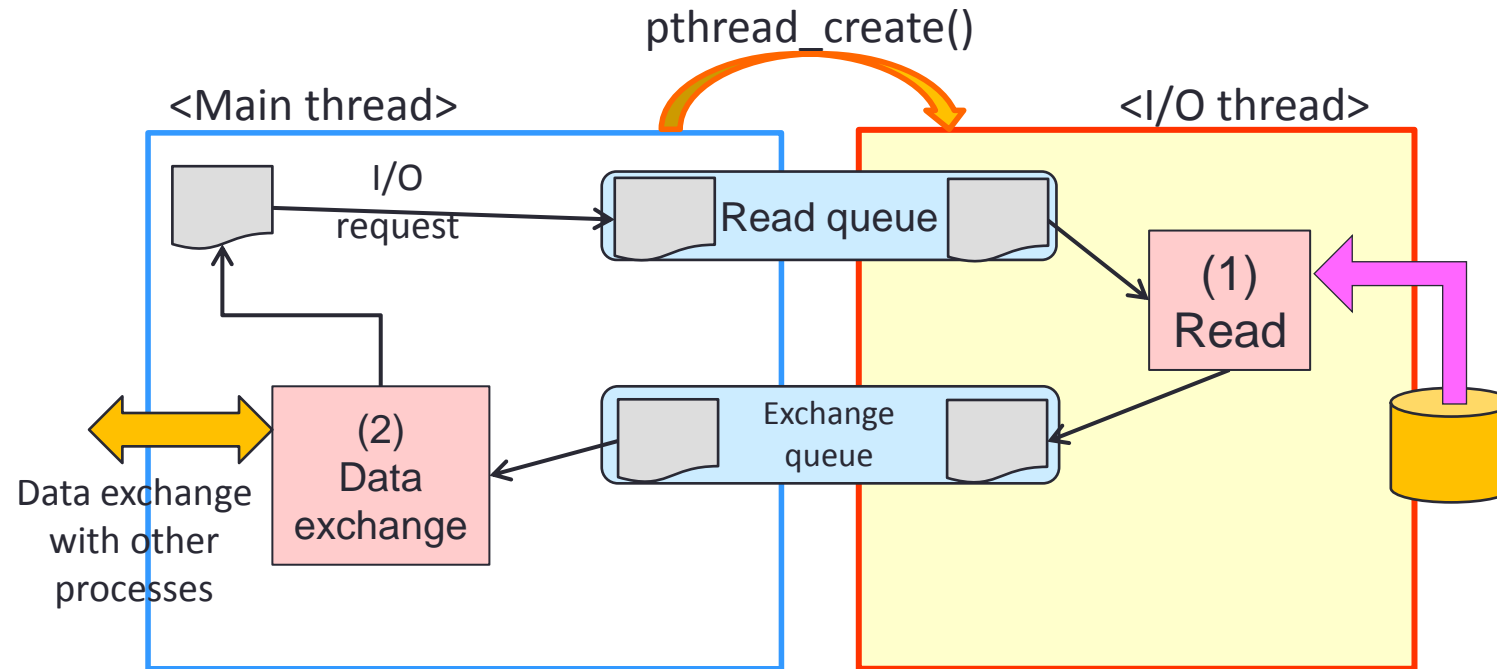


Overlapping



Pipelined Two-Phase I/O

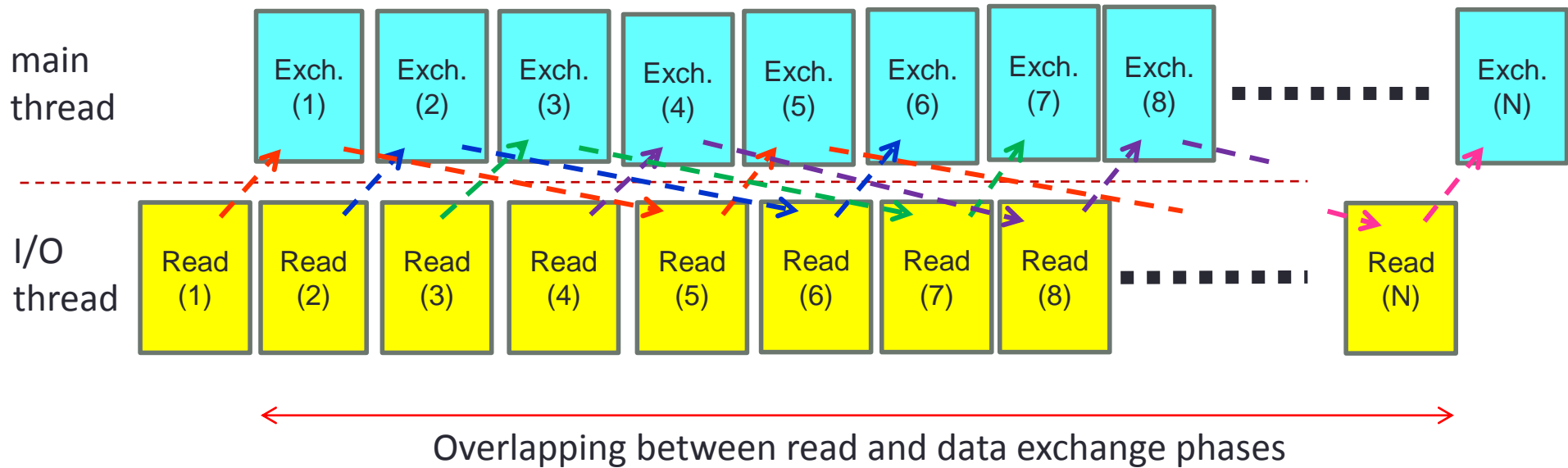
Multithreaded Two-Phase I/O (2)



- Overlapping strategy
 - Main thread: Data exchanges
 - I/O thread: File I/O
 - Synchronization by using shared queues (read and exchange queues)
- Further optimization
 - Multiple I/O requests in shared queues
 - The number of slots can be managed by `MPI_Info_set` with a key-value pair.

Multithreaded Two-Phase I/O (3)

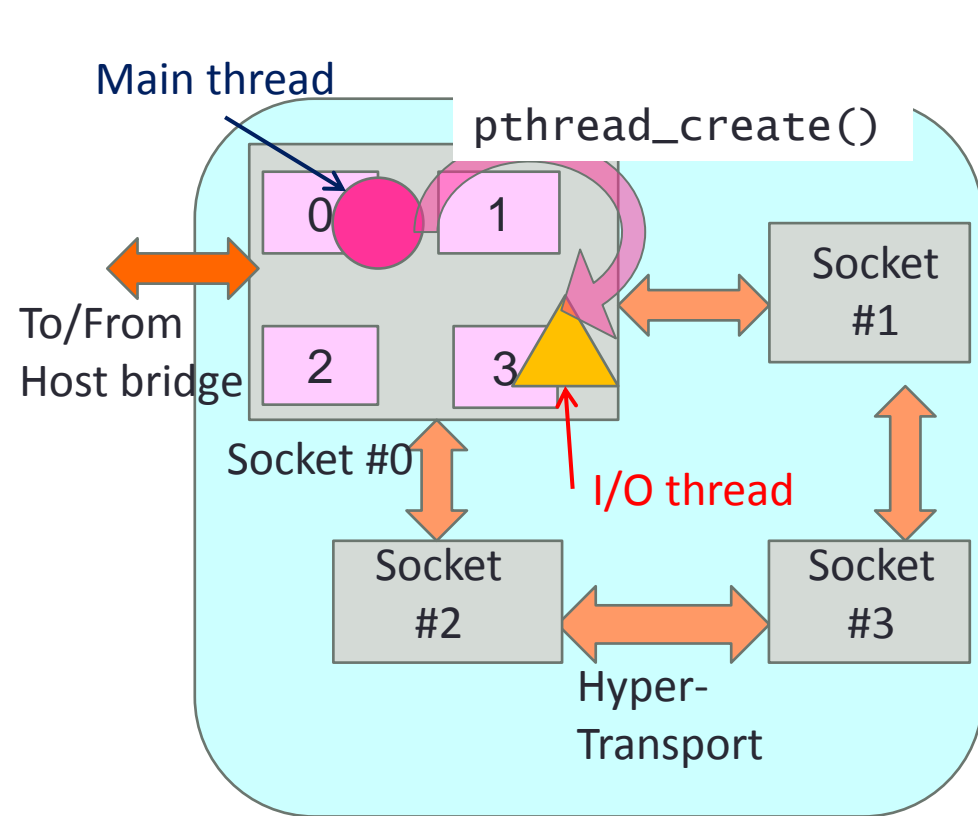
- Two-Phase I/O flow in collective read operations



- Exch. : Data exchange phase
- Read: File read phase

CPU Core Affinity Management in Multithreaded Two-Phase I/O

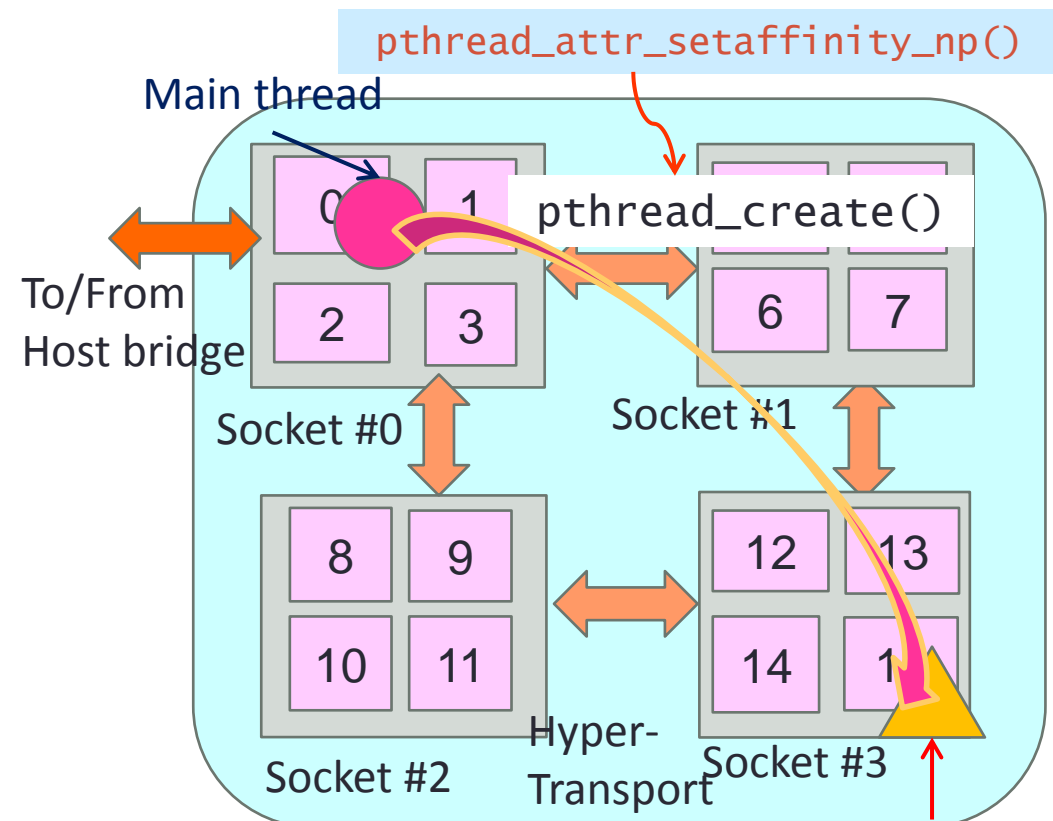
- Prevention of race condition between threads
 - Management by using `pthread_attr_setaffinity_np()`
 - Core management via `MPI_Info` object



main thread: Core-0

I/O thread : Core-3 (scheduler dependent)

Race condition in the same CPU socket



main thread: Core-0

I/O thread : Core-15

Prevention of race condition by deploying threads on different CPU sockets

Performance Evaluation (1)

● Evaluation on T2K Open Supercomputer (T2K-Toudai)

Specifications of a PC node of the T2K-Toudai (Special node group: 32 nodes in total)

CPU	4 x AMD Opteron (Barcelona, 2.3 GHz, 4 cores)
Memory	32 GiB (4 x 8 GiB)
Interconnect	Myrinet 10Gbps (IP over Myrinet)
OS	Linux kernel 2.6.18-53 with glibc version 2.5
MPI library	MPICH2 ver.1.4.1 with our implementation
HDF5 library	HDF5 version 1.8.10
Parallel file system	Lustre ver. 1.8.1 (1 MDS, 4 OSTs)

- I/O performance evaluation by using a simple parallel HDF5 test code in an HDF5 source code distribution with our some modifications
 - Due to some unsupported HDF5 APIs in the newest IOR benchmark, we gave up to use the IOR.
- Thread deployment on CPU cores
 - ✓ CPU core management by using `MPI_Info_set()` in the HDF5 test program with some key-value pairs
- Eliminating file cache effect by remounting the Lustre file system prior to every I/O operation run

Performance Evaluation (2)

- HDF5 test program (pseudo code)
 - About 2.2 GiB data set was read in collective manner.
 - 2-dimensional data : 24,320 x 24,320 with integer data type
 - 32 MPI processes (1 process/node)

```
...
dataset1 = H5Dopen2(fid1, "Data1",
    H5P_DEFAULT);
...
file_daspace = H5Dget_space(dataset1);
...
ret = H5Sselect_hyperslab(file_daspace, ...);
...
mem_daspace = H5Screate_simple(...);
...
xfer_plist = H5Pcreate(H5P_DATASET_XFER);
...
ret = H5Pset_dxpl_mpio(xfer_plist,
    H5FD_MPIO_COLLECTIVE);
...
```

Preparation of collective read operations

```
io_time = MPI_Wtime();

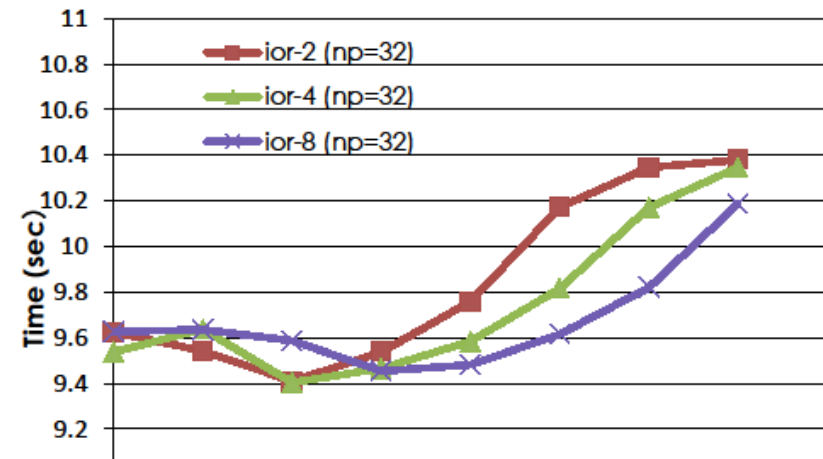
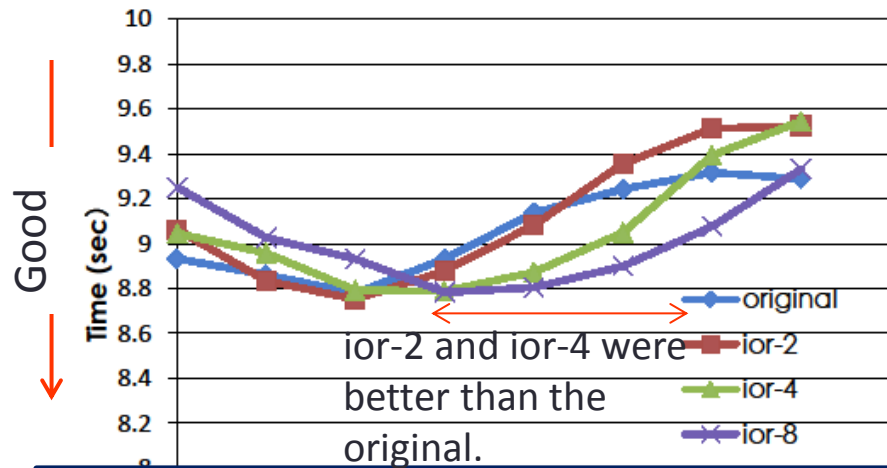
ret = H5Dread(dataset1, H5T_NATIVE_INT,
    mem_daspace, file_daspace,
    xfer_plist, &(amp;data_array1[0][0]));

io_time = MPI_Wtime() - io_time;
...
H5Sclose(file_daspace);
H5Sclose(mem_daspace);
H5Pclose(xfer_plist);
H5Dclose(dataset1);
...
```

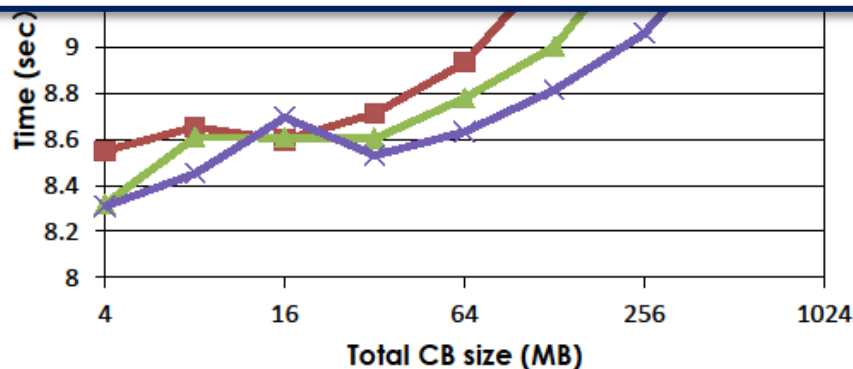
Measurement of collective read operations for a target dataset

Performance Evaluation (3)

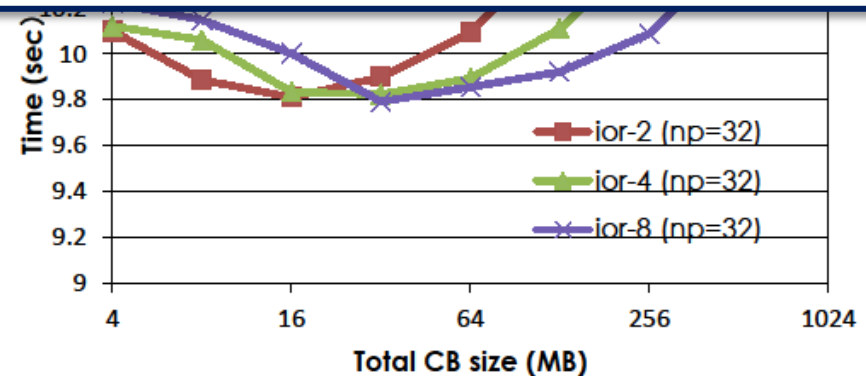
- I/O times in collective read by using PHDF5 (32 MPI processes)



- An increase in the number of I/O requests led to performance improvements in every case.
- Up to 7% minimization was achieved relative to the original TP-I/O.
- Only the case (c) minimized up to 8% relative to the case (a) in each.
- I/O times in the cases (b) and (d) were longer than the original case (a).



(c) core management, main:0, I/O:15



(d) core management, main:15, I/O:0

Performance Evaluation (4)

● Analysis of internal operation times of Two-Phase I/O

◆ TP-I/O for collective read consists of

1. Contiguous read operations
2. Data exchanges

In order to examine what was going on inside the TP-I/O

◆ Time measurement for each operations

- ✓ read and data exchanges

◆ Calculation of I/O times by using the measured internal times

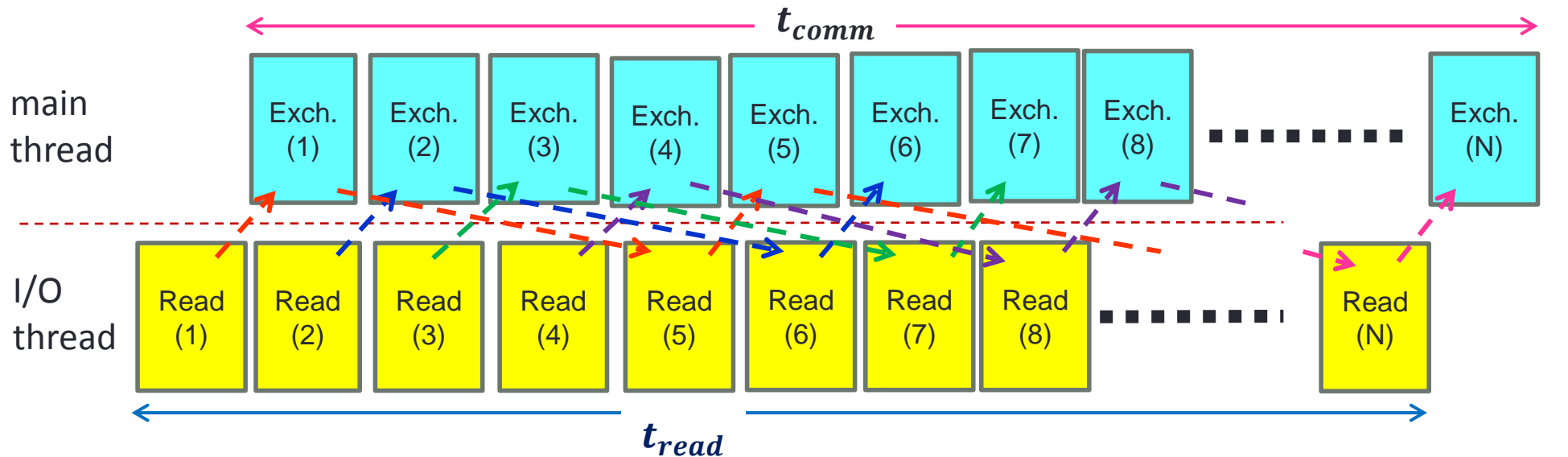
- ◆ Comparison with the measured times to examine overlapping effects and so forth.

$$\max(t_{comm}, t_{read}) \cdot (1 - S_{CB} / S_{data}) + (t_{comm} + t_{read}) \cdot S_{CB} / S_{data}$$

$$\left\{ \begin{array}{l} t_{comm} : \text{mean communication time} \\ t_{read} : \text{mean read time} \\ S_{CB} : \text{CB size} \\ S_{data} : \text{Size of accessed region per process} \end{array} \right.$$

Performance Evaluation (5)

● I/O time calculation model



Exchange time / cycle : $t_{comm} / (S_{data} / S_{CB})$

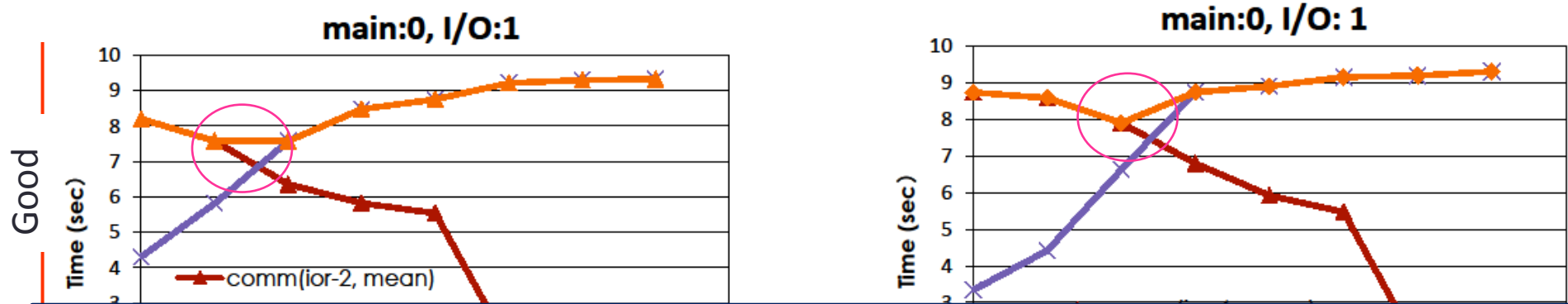
Read time / cycle : $t_{read} / (S_{data} / S_{CB})$

$$\frac{t_{read}}{(S_{data} / S_{CB})} + (S_{data} / S_{CB} - 1) \cdot \max\left(\frac{t_{read}}{S_{data} / S_{CB}}, \frac{t_{comm}}{S_{data} / S_{CB}}\right) + \frac{t_{comm}}{S_{data} / S_{CB}}$$

$$= \max(t_{read}, t_{comm}) \cdot (1 - S_{CB} / S_{data}) + (t_{comm} + t_{read}) \cdot S_{CB} / S_{data}$$

Performance Evaluation (6)

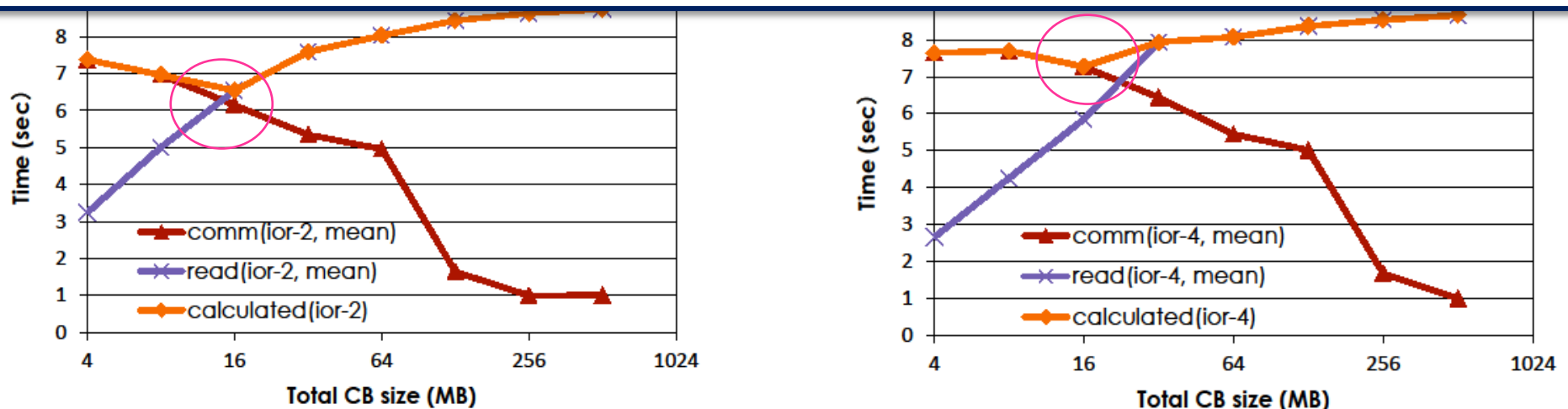
- Analysis of internal operations in TP-I/O with CPU core affinity



- Calculated I/O times in the cases (c) and (d) were shorter than those in the cases (a) and (b), respectively.



- CPU core management had an impact in I/O performance improvements.



(c) 2 I/O requests (main:0, I/O:15)

(d) 4 I/O requests (main:0, I/O:15)

Related Work

- P. Dickens and R.Thakur, “Improving Collective I/O Performance Using Threads,” Proc. of 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, pp.38-45 (1999)
- X. Ma et al., “Improving MPI-IO Output Performance with Active Buffering Plus Threads,” Proc. of IPDPS’03 (2003)
 - ✓ Both works showed excellent overlapping by using multithreaded scheme, however they overlapped **computations** with file I/O.
 - ✓ While our approach is addressing to overlap Two-Phase I/O operations.
- J.G. Blas et al, “View-Based Collective I/O for MPI-IO”,Proc. of CCGRID2008, pp. 409-416 (2008)
 - ✓ Optimization of the TP-IO by eliminating extra communication cost which generates file and memory access patterns on demand

Related Work (cont'd)

- J.G. Blas et al, “Implementation and Evaluation of File Write-Back and Prefetching for MPI-IO over GPFS”, IJHPCA, Vol. 24, pp.78-92 (2010)
 - ✓ GPFS is used to have background writing and read-ahead operations to overlap with data exchanges.
 - ✓ Multithread scheme is used, however this work is dependent on GPFS.
- J.P. Prost et al., “MPI-IO/GPFS, An Optimized Implementation of MPI-IO on Top of GPFS,” Proc. of SC2001, p. 58 (2001)
 - ✓ Overlapping of I/O operations with data exchanges is realized by using double buffering scheme.
 - ✓ Our proposal is similar to this work, however this idea is tightly coupled with GPFS functions, while our proposal is independent of underlying file system at least a Pthreads library is available.

Concluding Remarks

- Proposal of CPU core affinity management scheme in multithreaded TP-IO
 - CPU core affinity management in multithreaded operations
 - Prevention of race condition on the same CPU
- Performance evaluation
 - Good performance improvements
 - Internal file read and communication times were minimized by deploying an I/O thread on a different CPU, being apart from CPU where a main thread was working.
 - 7% improvements relative to the original TP-IO
 - 8% improvements relative to the multithreaded TP-IO without CPU core affinity management
- Future work
 - Collective write implementation
 - Further examinations about resource utilization by monitoring some system specific resources such as CPU and I/O systems
 - Utilization of an affinity management library such as hwloc or likwid