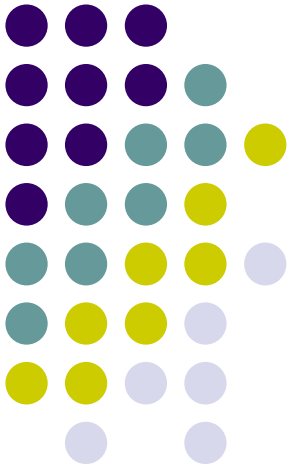
2019 Practical Parallel Computing (実践的並列コンピューティング) No. 10

Distributed Memory Parallel Programming with MPI (4)

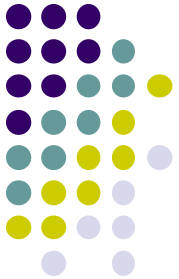


Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

Considering Performance of MPI Programs



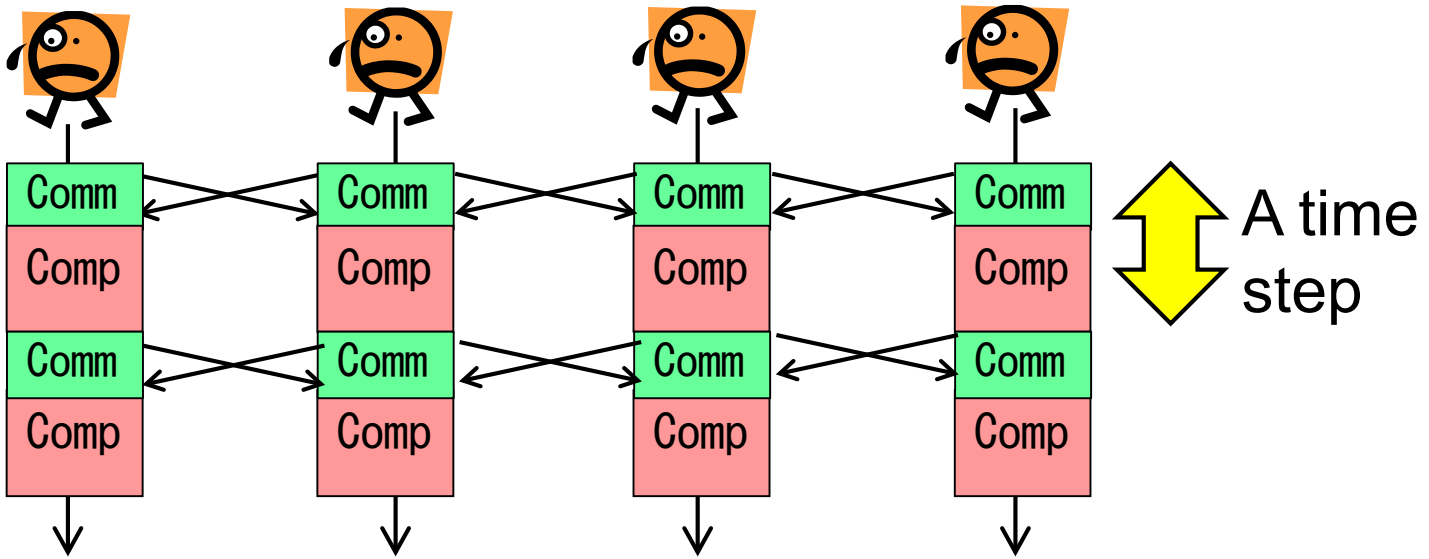
(Simplified) Execution time of an MPI program =

Computation time ← including memory access

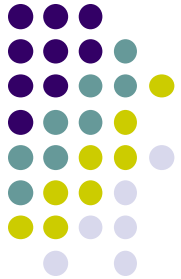
\+ Communication time ← including congestion

\+ Others ← load imbalance, I/O…

Behavior of "diffusion" on MPI



Computation Time & Communication Time (1)



How are they determined? (very simplified discussion)

1. Aspect of software

Computation time ∝ computation costs per process

Communication time ∝ communication costs per process

	Computation costs (per process)	Communication costs (per process)
diffusion	O(NX NY NT / p)	O(NX NT)
mm	O(mnk / p)	0
mm (memory reduced)	O(mnk / p)	O(mk)

per process

※ Communication costs depend on data distribution methods
The table shows representative examples

Computation Time & Communication Time (2)

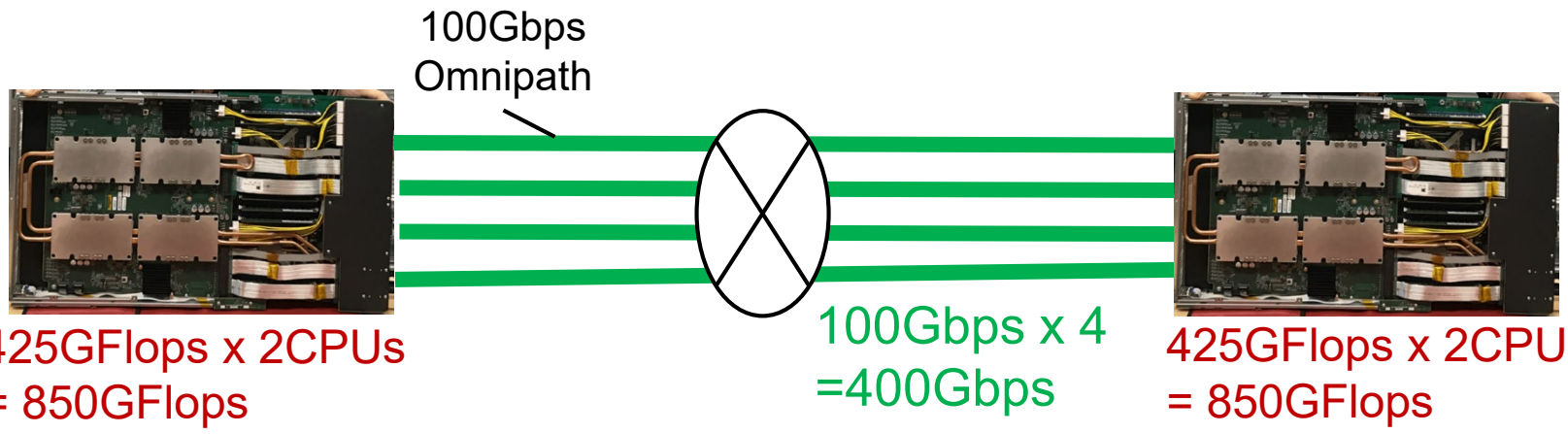
2. Aspect of hardware

Computation time

- Shorter if processor speed is faster
  - 850GFlops per node on TSUBAME3

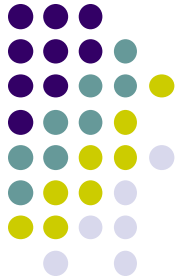
Communication time

- Shorter if network speed is faster
  - 400Gbps per node on TSUBAME3



Speed of actual software is slower than the “peak” performance

Parameters for Network Speed



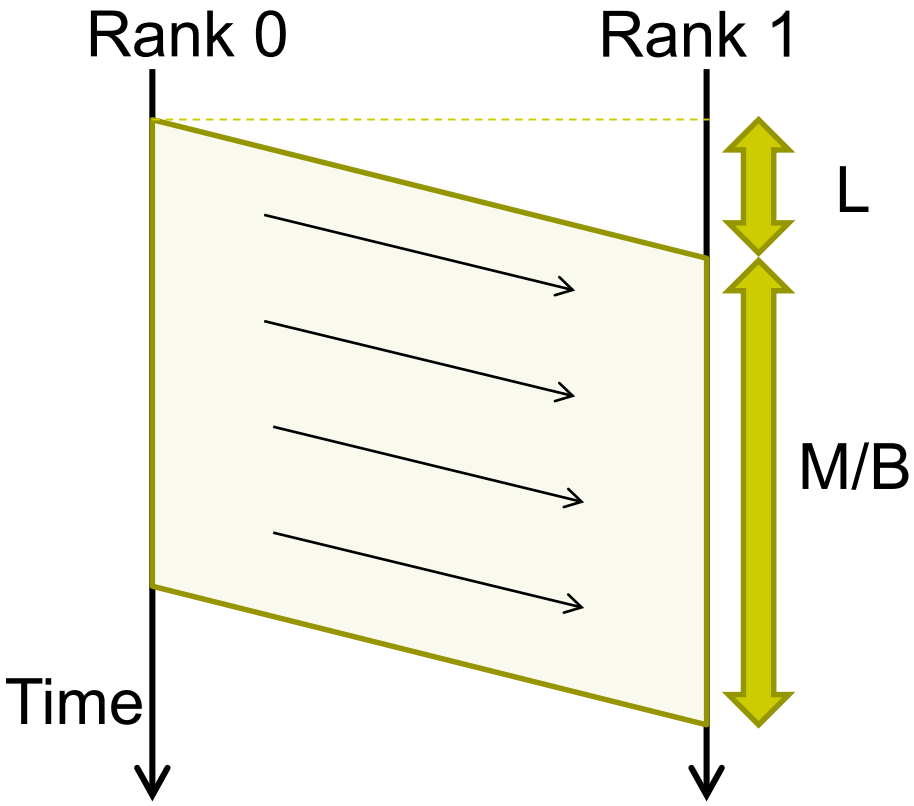
What parameters describes network speed?

- Bandwidth : Data amounts that network can transport per unit time → Larger is better
  - bps: X bits per second
  - B/s: X Bytes per second
  - On TSUBAME3, 400Gbps = 50GB/s per node
- Network latency : Time to transport minimum data (1bit, for example) → Smaller is better
  - On TSUBAME3, <10us

[Q] Is “latency” reciprocal of “bandwidth”?
→ No, because data are transported in “pipe-lined” style

Model of Communication Time

Illustration of communication of data size M



$$T = M / B + L$$

T: Communication time

M: Data size

B: Bandwidth

L: Network latency

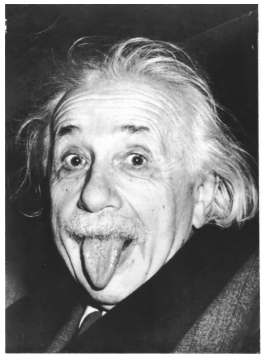
※ Be aware of difference between "Byte" and "bit": 1Byte=8bit

※ Actually it is more complex for effects of network topology, congestion, packet size, error correction…

Why Latency L > 0?

1. Overhead when data passes network switches

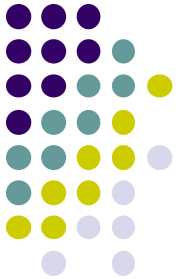
2. Software overhead
   - Cf) Socket library and MPI library perform data copy internally



3. Transfer speed of data cannot exceed speed of light! (3×10^8 m/s)

Considering T = M / B + L,
batching communication may improve communication time
cf) Sending 1Gbytes at once is much faster than sending 1Kbytes for 1,000,000 times

How to Improve Performance of MPI Programs?



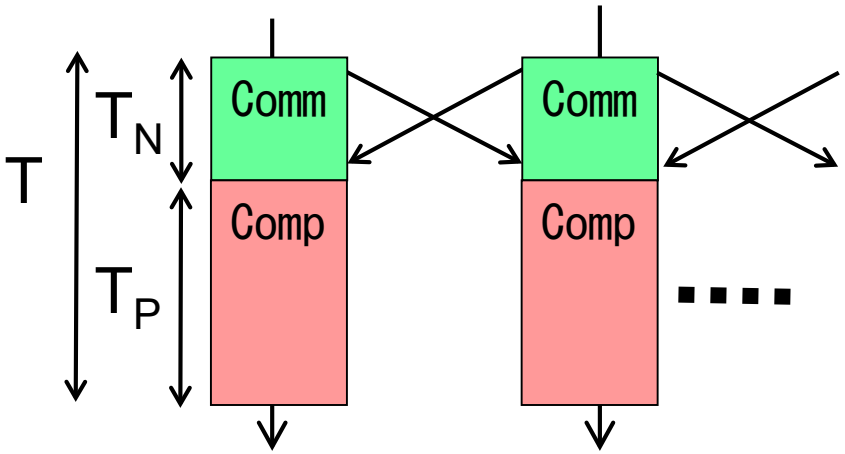
- To reduce computation time
  - Reduce computation amount
  - Use cache memory efficiently
- To reduce communication time
  - Reduce communication amount
  - Batch communication
  - Using collective communication is also good
- To reduce other time
  - Improve load balancing
  - Reconsider I/O
- To overlap computation and communication

Idea of Overlapping

If "some computations" do not require contents of message, we may start them beforehand

Case of diffusion

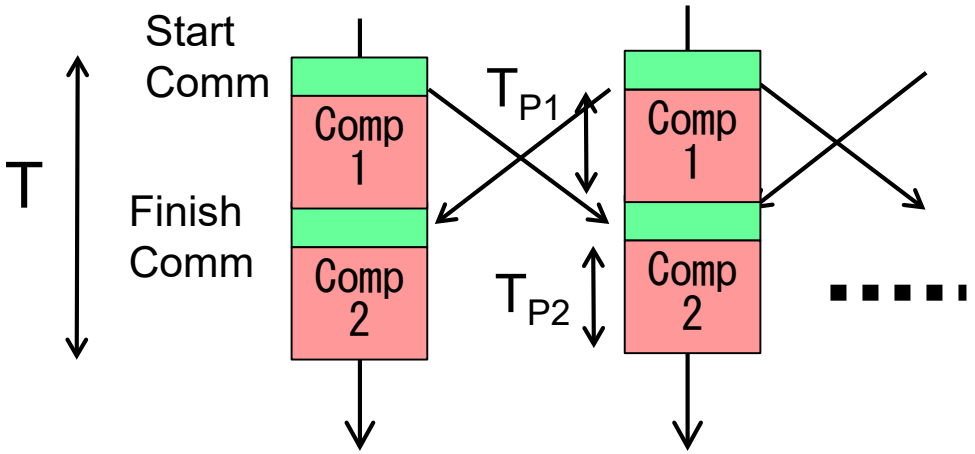
Without overlap



T: Execution time of 1 step
T_N: Communication time
T_P: Computation time ∝NX NY/p

$$T=T_N+T_P$$

With overlap



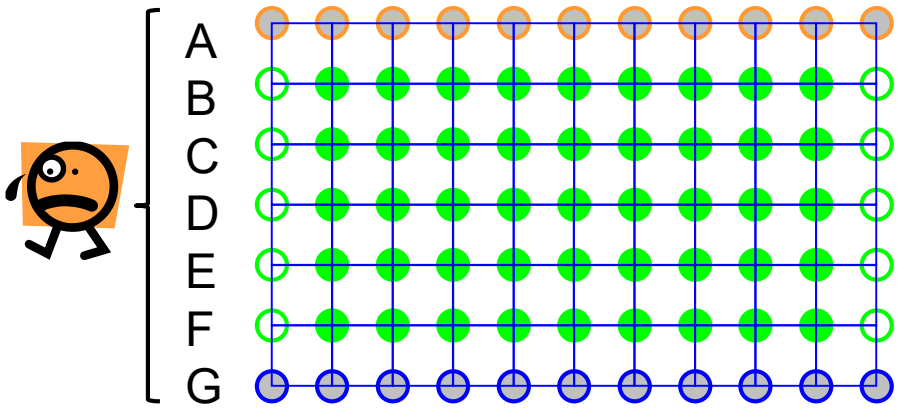
T_P is divided into

- T_{P1}: can be overlapped
- T_{P2}: cannot be overlapped

$$T=\max(T_N,T_{P1})+T_{P2}$$

Overlapping in Stencil Computation (related to [M1], but not mandatory)

When we consider data dependency in detail, we can find computations that do not need data from other processes



Rows C, D, E do not need data from other processes

→ They can be computed without waiting for finishing communication

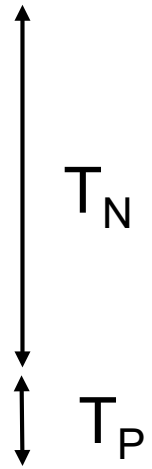
On the other hand, rows B, F need received data

Non-blocking communications (MPI_Isend, MPI_Irecv…) are used for 2 purposes

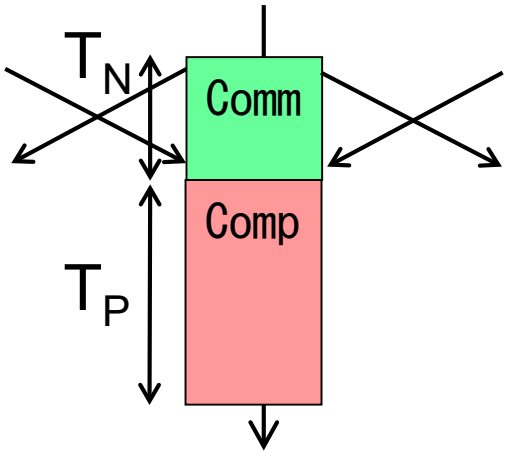
1. To avoid deadlock problem (see No.8 slides)
2. To overlap

Diffusion Algorithm without Overlapping

```
for (t = 0; t < nt; t++) {
  Start to Send B to rank-1 (MPI_Isend)
  Start to Send F to rank+1 (MPI_Isend)
  Start to Recv A from rank-1 (MPI_Irecv)
  Start to Recv G from rank-1 (MPI_Irecv)
  Finish all communications (MPI_Wait x 4)
  Compute rows B--F
  Switch old and new arrays
}
```



※ This algorithm is different from one in No.8 slide; both avoid deadlock

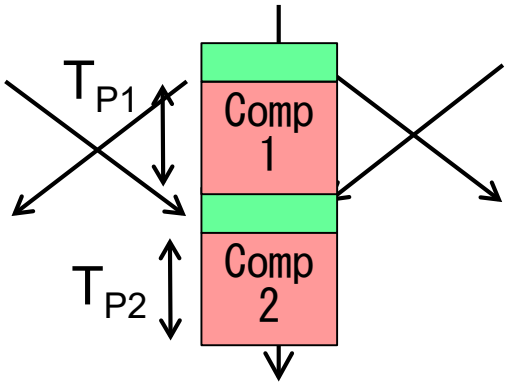


$$T=T_N+T_P$$

Diffusion Algorithm with Overlapping

```
for (t = 0; t < nt; t++) {
  Start to Send B to rank-1 (MPI_Isend)
  Start to Send F to rank+1 (MPI_Isend)
  Start to Recv A from rank-1 (MPI_Irecv)
  Start to Recv G from rank-1 (MPI_Irecv)
  Compute rows C - E (TP1)
  Finish all communications (MPI_Wait x 4)
  Compute rows B, F (TP2)
  Switch old and new arrays
}
```

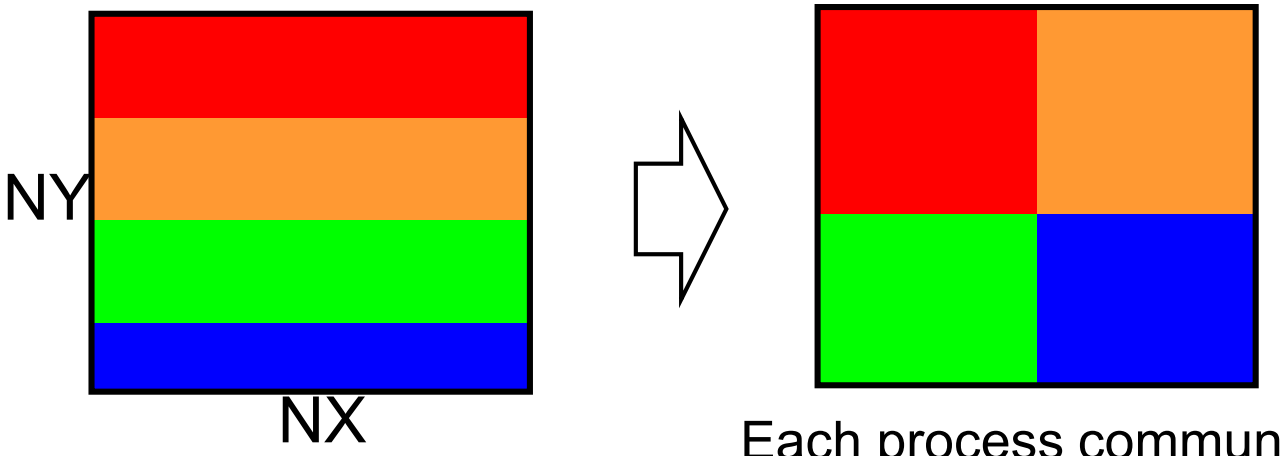
Computations are divided into 2 parts



$$T=\max(T_N,T_{P1})+T_{P2}$$

Another Improvement: Reducing Communication Amounts

Multi-dimensional (MD) division can reduce communication



Each process communicate with upper/lower/right/left processes

- Comp: O(NY NX/p)
- Comm: O(NX)

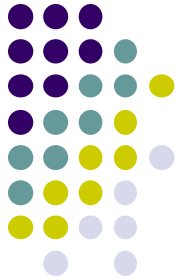
per 1 process, 1 iteration

- Comp: O(NY NX/p)
- Comm: O((NY+NX)/$p^{1/2}$)

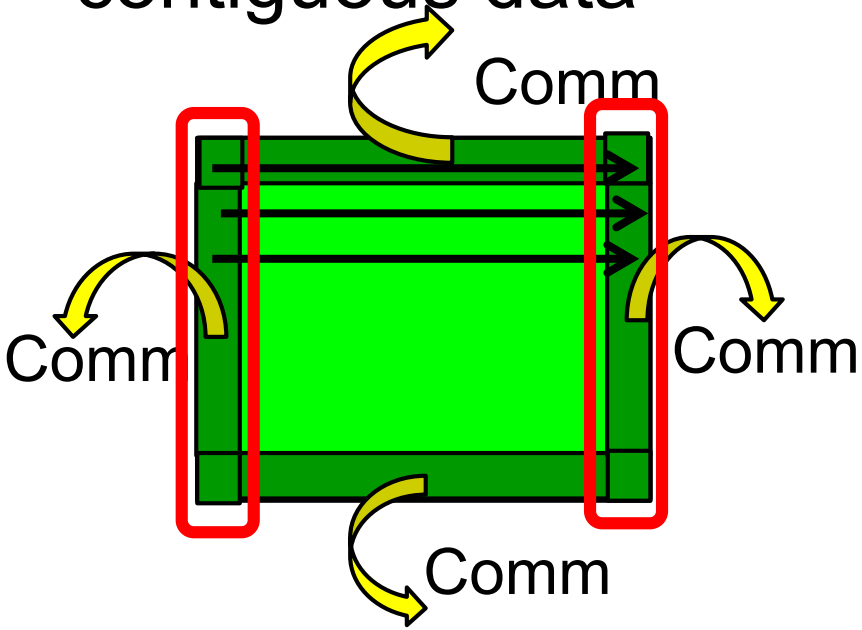
per 1 process, 1 iteration

→ Comm is reduced

Multi-dimensional division and Non-contiguous data (1)



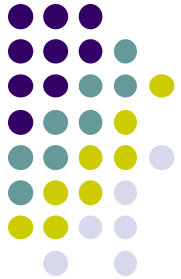
- MD division may need communication of non-contiguous data



In Row-major format, we need send/recv of non-contiguous data for left/right borders

But “fragmented communication” degrades performance! (since Latency > 0)
How do we do?

Multi-dimensional division and Non-contiguous data (2)



Solution (1):

- Before sending, copy non-contiguous data into another contiguous buffer
- After receiving, copy contiguous buffer to non-contiguous area

Solution (2):

- Use MPI_Datatype
  - Skipped in the class; please use Google

Both solutions suffer from costs for access to non-contiguous data

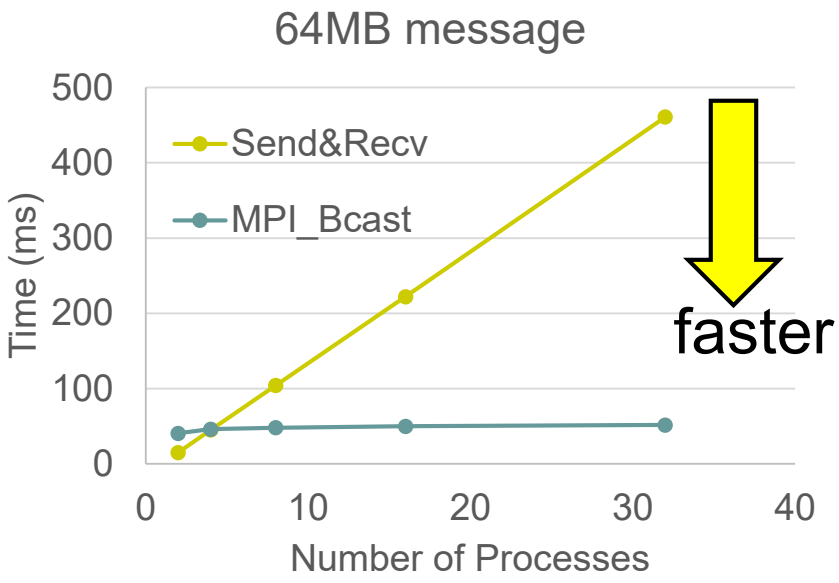
→ MD division tends to be slower than theory ☹

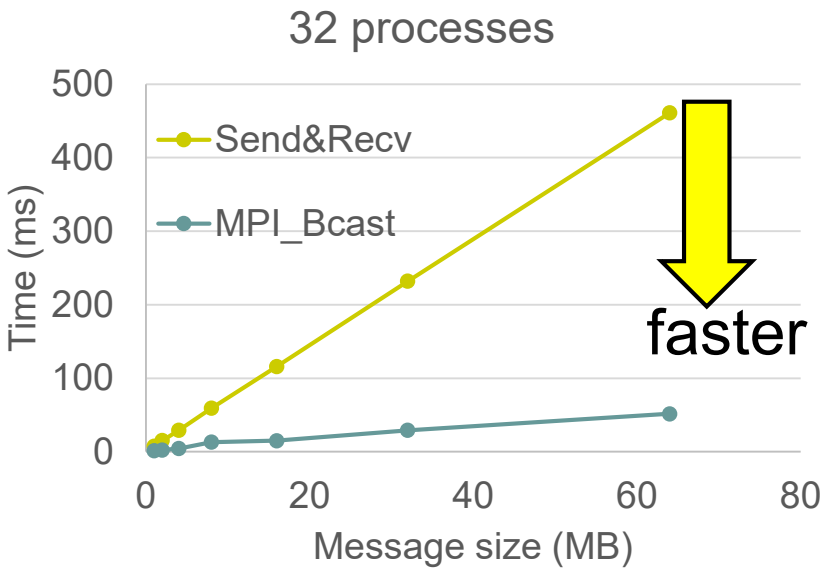
On Collective Communications

- Comparing MPI_Bcast and MPI_Send&Recv

  1 process per node is invoked on TSUBAME2

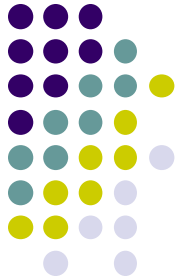
  In the latter, rank 0 called MPI_Send for p-1 times to other processes



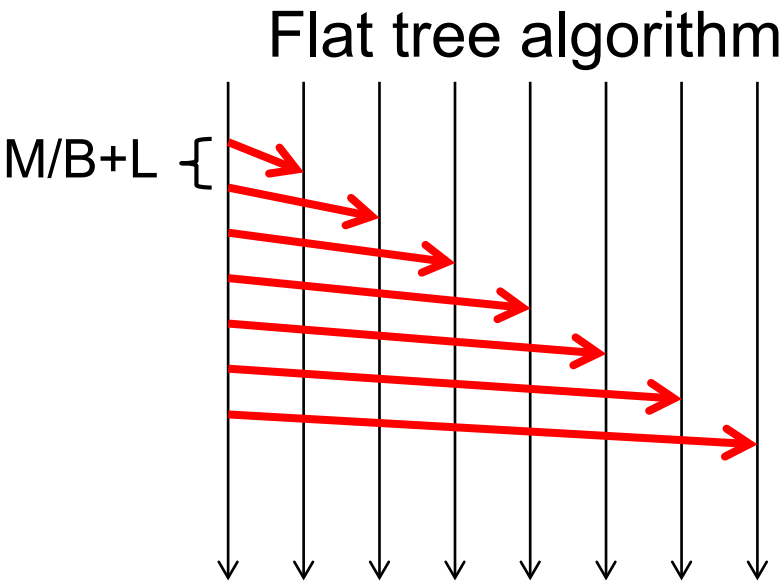


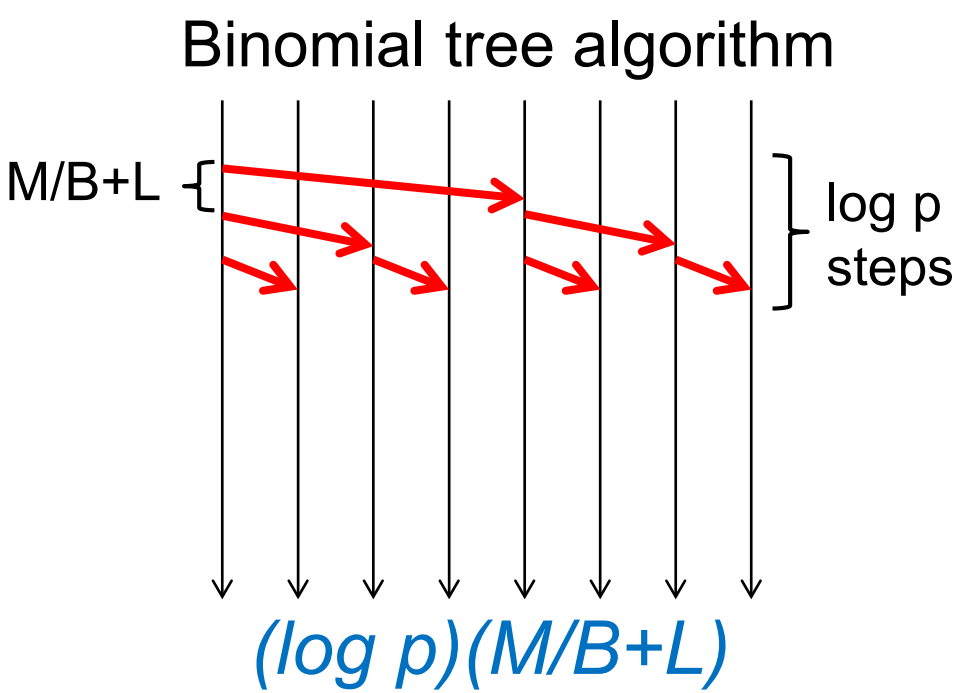
- In most cases, MPI_Bcast is faster

Why are Collective Communications Fast?



- Since MPI library uses scalable communication algorithms
- Case of "broadcast" of size M data
  - p: number of processes, B: network bandwidth, L: network latency

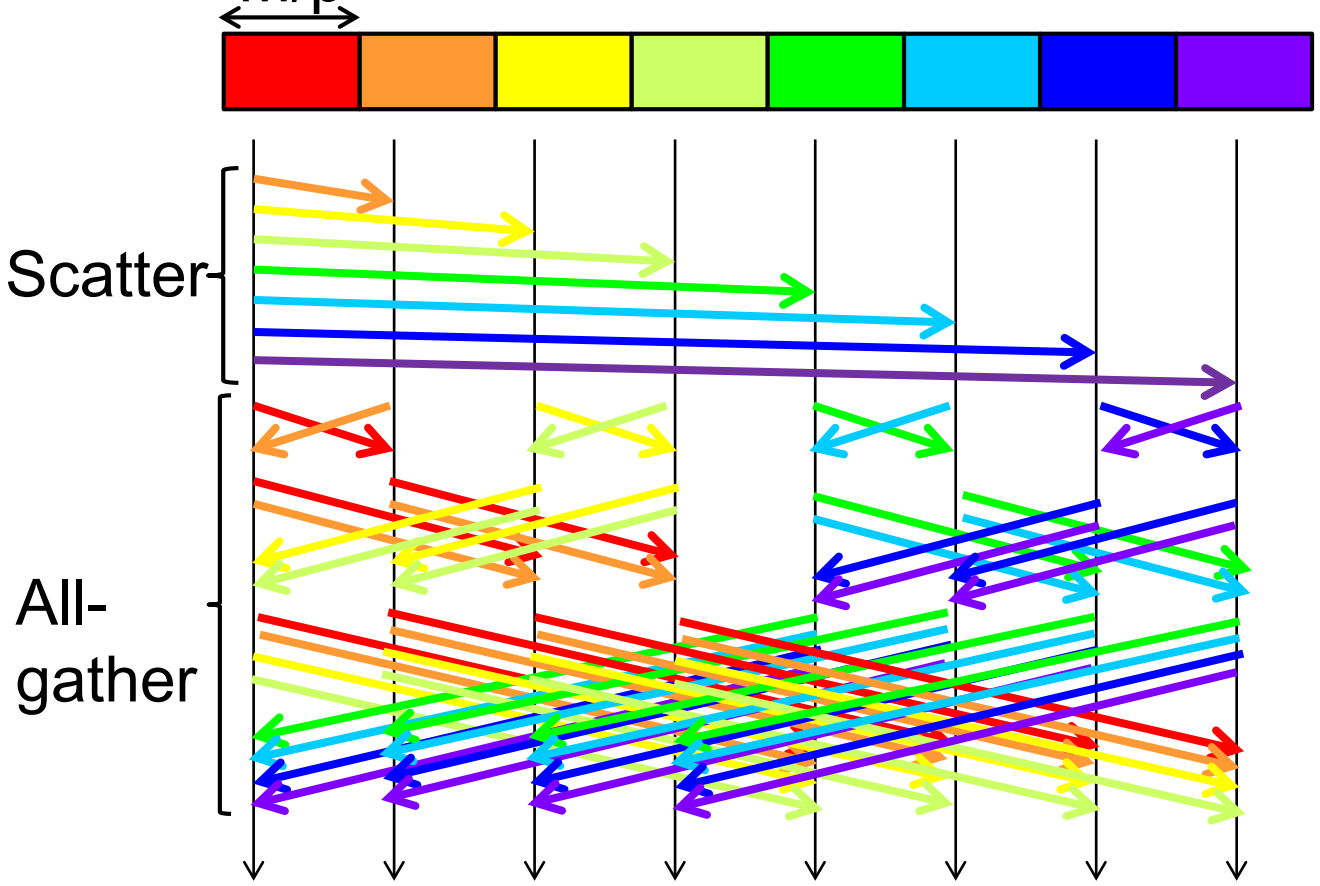




One of Scalable Broadcast Algorithms

- Scatter&Allgather algorithm
  - Message is divided into p parts
  - Better than “binomial tree” if M is larger

R. Thakur and W. Gropp. Improving the performance of collective operations in mpich. EuroPVM/MPI conference, 2003.



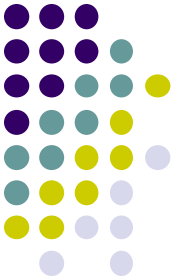
Scatter

$M/B + pL +$

$M/B + (\log p)L$

All-gather

Comparison of Broadcast Algorithms

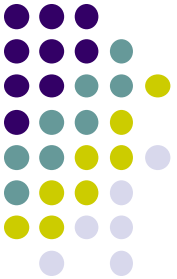


- Consider two extreme cases
  - If M is sufficiently large: *M/B+L → M/B*
  - If M is close to zero: *M/B+L → L*

	Flat Tree	Binomial Tree	Scatter& All-gather
Cost (General)	p(M/B+L)	(log p) (M/B+L)	2M/B + (p + log p)L
Cost with very large M	p M/B	(log p) M/B	2 M/B → Fastest
Cost with very small M	p L	(log p) L → Fastest	(p + log p) L

Many MPI libraries implement multiple algorithms
They switch them automatically according to message size M ☺

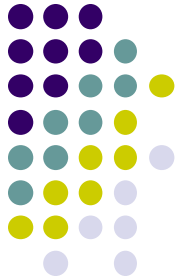
Where We are Now



- We have finished
  - Part 1: OpenMP for shared memory parallel programming
  - Part 2: MPI for distributed memory parallel programming

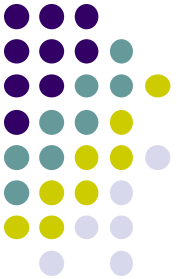
- Why are “parallel programs” slower than expectation?
  - “p times speed-up with p processor cores” (linear scaling) is ideal, but…
  - Parallel software is often less scalable

How Should We Tackle Performance Limiting Factors?



- It is important to know “why it is slow now”
- Consider what should be measured in order to specify current problem
  - Measuring time part by part may be helpful
  - Comparing computation time and communication time separately
  - Comparing 1-node performance and multi-node performance may be helpful
- It is good to use knowledge of computer hardware

Assignments in MPI Part (Abstract)



Choose *one of* [M1]—[M3], and submit a report

Due date: May 30 (Thursday)

[M1] Parallelize “diffusion” sample program by MPI.

[M2] Improve mm-mpi sample in order to reduce memory consumption.

[M3] (Freestyle) Parallelize *any* program by MPI.

For more detail, please see No. 7 slides or OCW-i.

Next Class

- Part 3 starts
  - GPU parallel programming
    - OpenACC is planned

Information

Lecture

- Slides are uploaded in OCW
  - www.ocw.titech.ac.jp → search “2019 practical parallel computing”
- Assignments information/submission site are in OCW-i
  - Login portal.titech.ac.jp → OCW/OCW-i
- Inquiry
  - ppcomp@el.gsic.titech.ac.jp
- Sample programs
  - Login TSUBAME, and see ~endo-t-ac/ppcomp/19/ directory

TSUBAME

- Official web including Users guide
  - www.t3.gsic.titech.ac.jp
- Your account information
  - Login portal.titech.ac.jp → TSUBAME portal