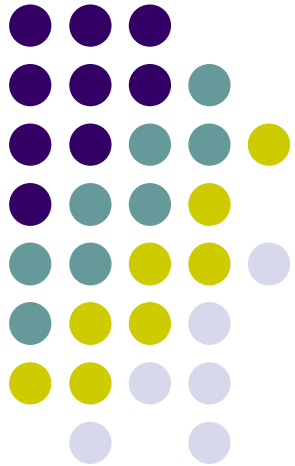
2019 Practical Parallel Computing (実践的並列コンピューティング) No. 9

Distributed Memory Parallel Programming with MPI (3)



Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

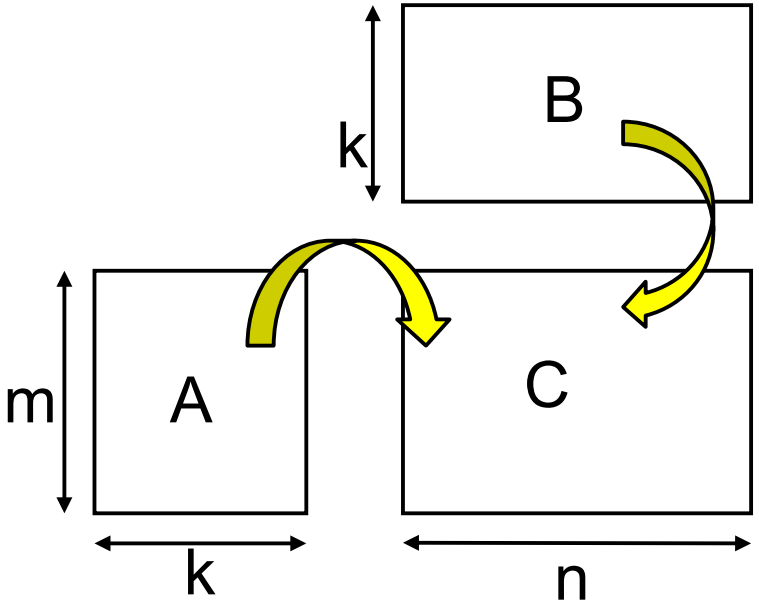
“mm” sample: Matrix Multiply (Revisited, related to [M2])

MPI version available at ~endo-t-ac/ppcomp/19/mm-mpi/

A: a (m × k) matrix, B: a (k × n) matrix

C: a (m × n) matrix

C ← A × B

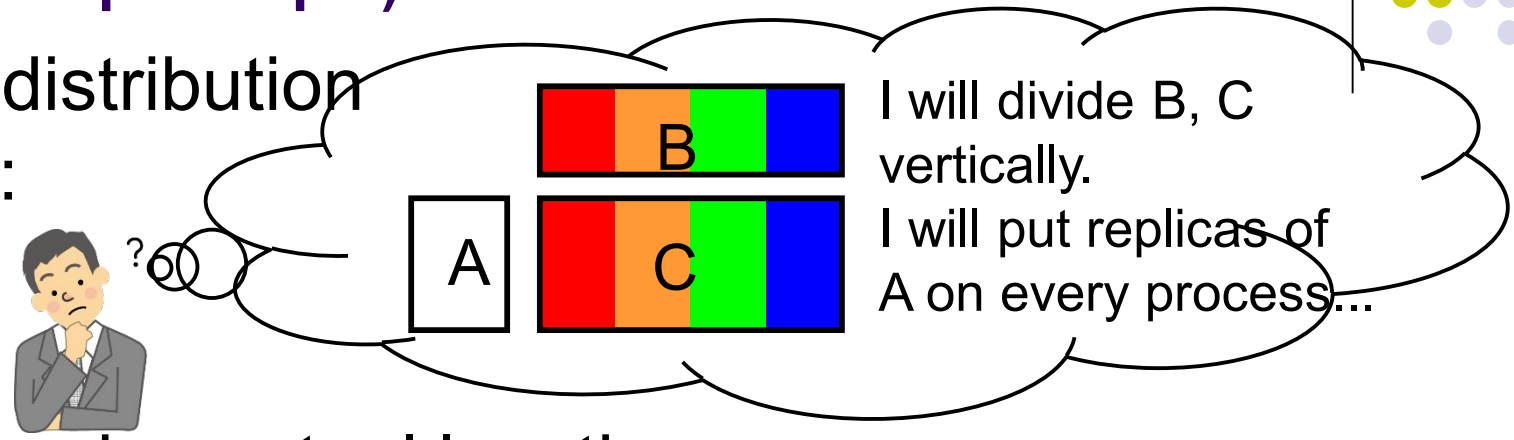


- Algorithm with a triple for loop
- Supports variable matrix size.
  - Each matrix is expressed as a 1D array by *column-major* format

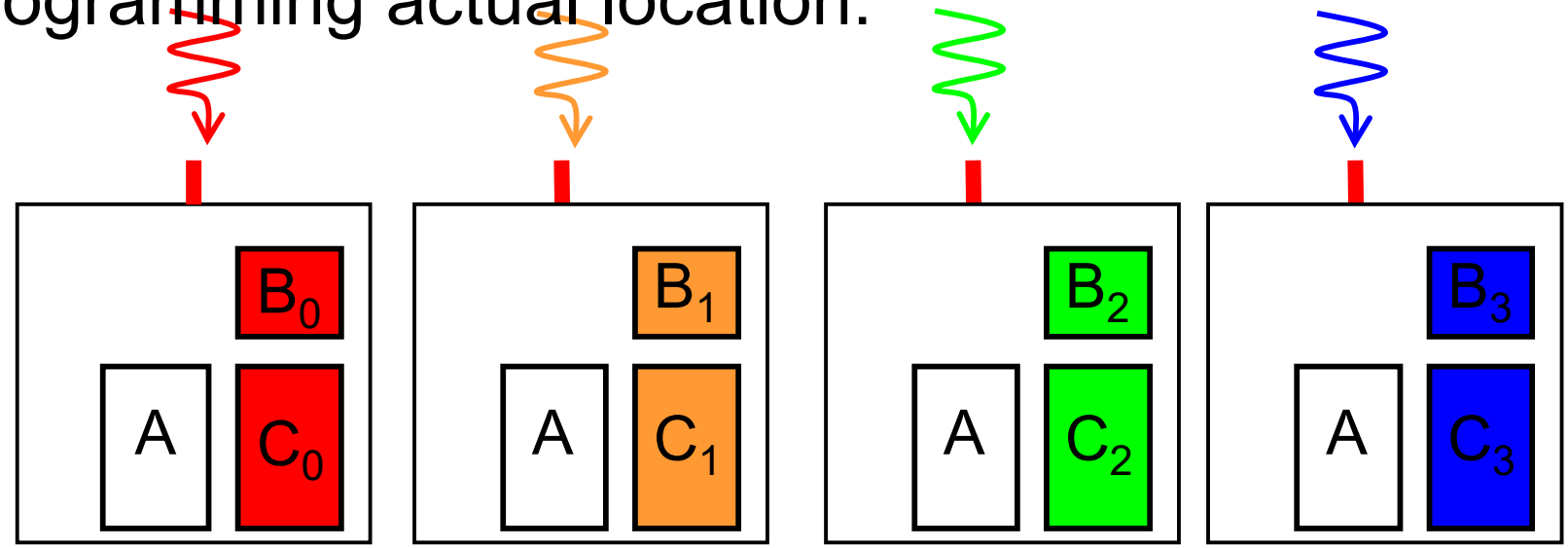
Execution: mpirun –n [#proc] ./mm [m] [n] [k]

Programming Data Distribution
(for mm-mpi sample)

Design distribution method:

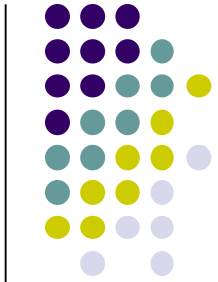


Programming actual location:



This is not a unique solution. How about other solutions?

Discussion on Considering Data Distribution

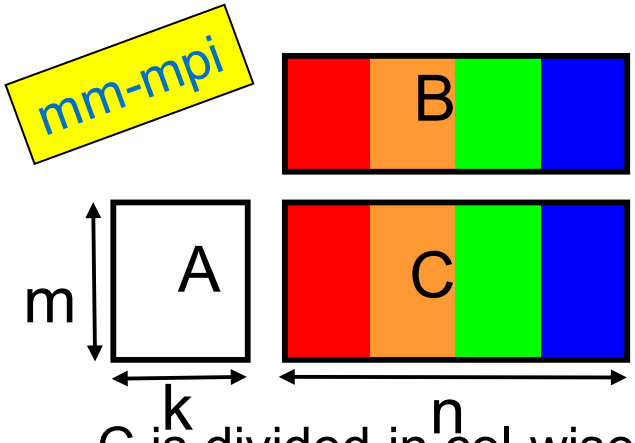


- Choice of data distribution may affect to
  - Communication cost
  - Memory consumption cost
  - (Sometimes, computation cost)

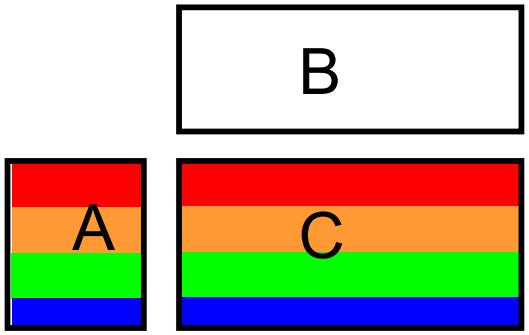
- Smaller cost is better

Other Data Distribution Methods?

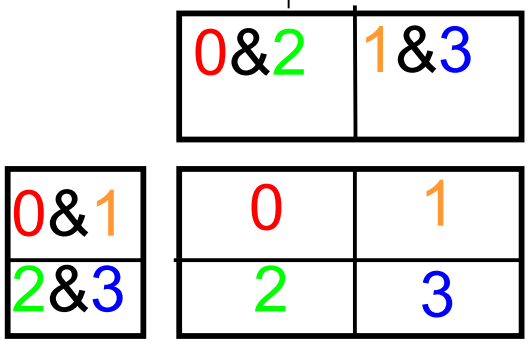
- $C_{i,j}$ requires *i*-th row of A and *j*-th column of B



C is divided in col-wise
⇒ Similarly B
A is replicated



C is divided in row-wise
⇒ Similarly A
B is replicated



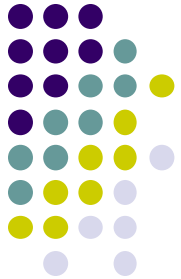
C is divided in 2D
⇒ A:row-wise + replica
B:col-wise + replica

Total Comm.	0 (※)	0 (※)	0 (※)
Totel Mem.	O(mkp+nk+mn)	O(mk+nkp+mn)	O(mkp$^{1/2}$+nkp$^{1/2}$+mn)

p: the number of processes
(※) If initial matrix is owned by one process, we need communication before computation

Among them, the third version has lowest memory consumption

Reducing Memory Consumption Further



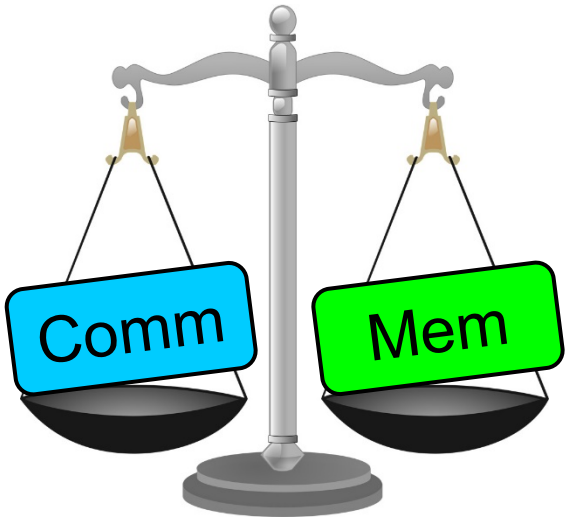
- Even in the third version, memory consumption is $O(mkp^{1/2}+nkp^{1/2}+mn)$ > $O(mk+nk+mn)$ (theoretical minimum)
- If p=10000, we consume 100x larger memory ☹

→ we cannot solve larger problems on supercomputers

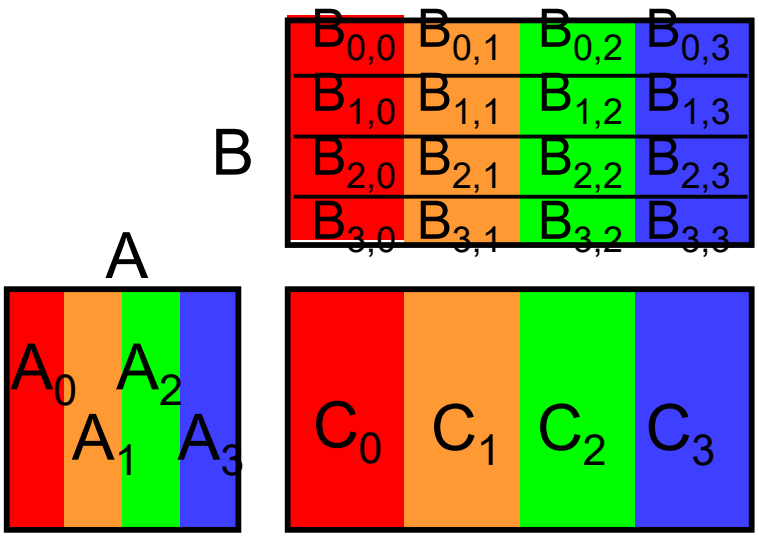
- To reduce memory consumption, we want to eliminate replica!

→ But this increases communication costs

Trade-off: a balance achieved between two desirable but incompatible features



Data Distribution with Less Memory Consumption



Not only B/C, but A is divided among all processes
(In this example, column-wise)
⇒ We need communication!

Algorithm

Step 0 :

P_0 sends A_0 to all other processes

Every process P_r computes

C_r += A_0 × $B_{0,r}$

Step 1 :

P_1 sends A_1 to all other processes

Every process P_r computes

C_r += A_1 × $B_{1,r}$

:

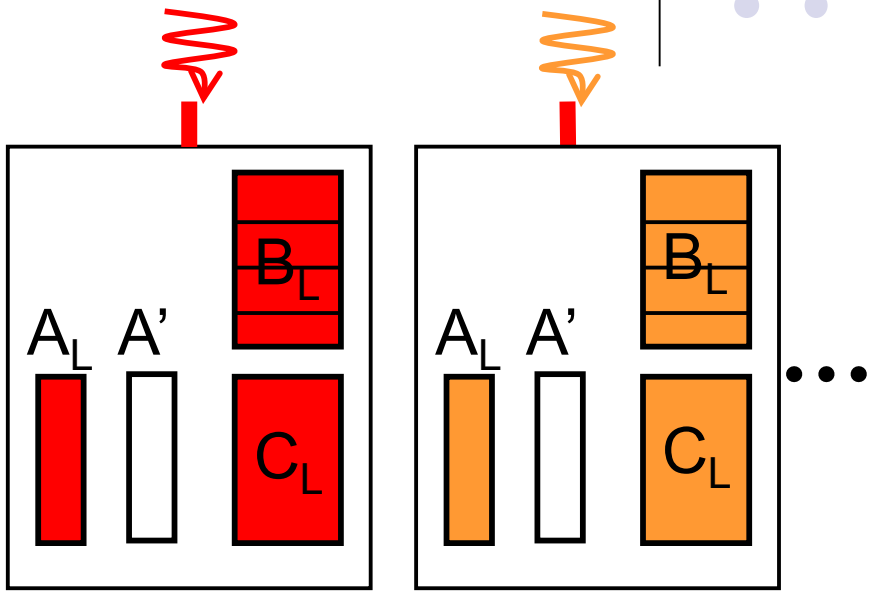
Repeat until *Step (p-1)*

Total Comm: O(mkp) Total Mem: O(mk+nk+mn)

Actual Data Distribution of Memory Reduced Version

Every process has partial A, B, C

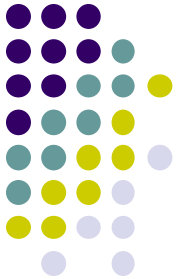
- A_L on process r ⇔ A_r
- B_L on process r ⇔ B_r
- C_L on process r ⇔ C_r



- Additionally, every process should prepare a receive buffer ➔ A' in the figure
  - A' (instead of A) is used for arguments of MPI_Recv()
  - On receivers, A' is used for computation

[Q] What if a process uses A_L for MPI_Recv() ?

Programming Memory Reduced Matrix Multiplication

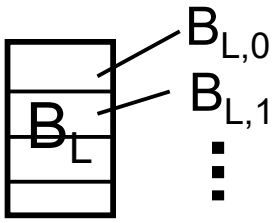


On every process r:

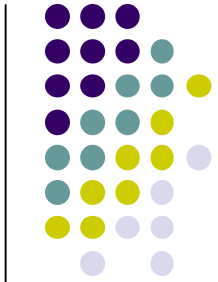
```
for (i = 0; i < size; i++) {   // size: number of processes
    if (r == i) {
        for (dest = 0; dest < size; dest++)
            if (dest != r) MPI_Send(A_L, …, dest, …);
    } else
        MPI_Recv(A', …, i, …);

    if (r == i)
        Compute C_L += A_L × B_L,i
    else
        Compute C_L += A' × B_L,i
}
```

P_i sends its A_L to all other processes



Improvements of Memory Reduced Version



Followings are options (NOT mandatory) in assignments [M2]

1. To use collective communications (explained hereafter)

2. To use SUMMA: scalable universal matrix multiplication algorithm
   - See http://www.netlib.org/lapack/lawnspdf/lawn96.pdf
   - Replica is eliminated, and matrices are divided in 2D

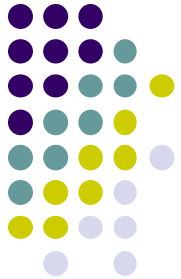
Peer-to-peer Communications

- Communications we have learned are called peer-to-peer communications
- A process sends a message. A process receives it

※ MPI_Irecv, MPI_Isend also does peer-to-peer communications

	Blocking	Non-Blocking
Peer-to-Peer	MPI_Send, MPI_Recv…	MPI_Isend, MPI_Irecv…
Collective	MPI_Bcast, MPI_Reduce…	(MPI_Ibcast, MPI_Ireduce…)

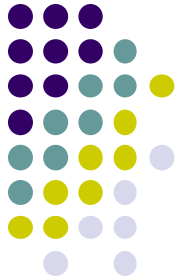
Collective Communications (Group Communications)



- Collective communications involves many processes
  - MPI provides several collective communication patterns
    - Bcast, Reduce, Gather, Scatter, Barrier・・・
  - All processes must call the same communication function

→ Something happens for all of them

One of Collective Communications: Broadcast by MPI_Bcast

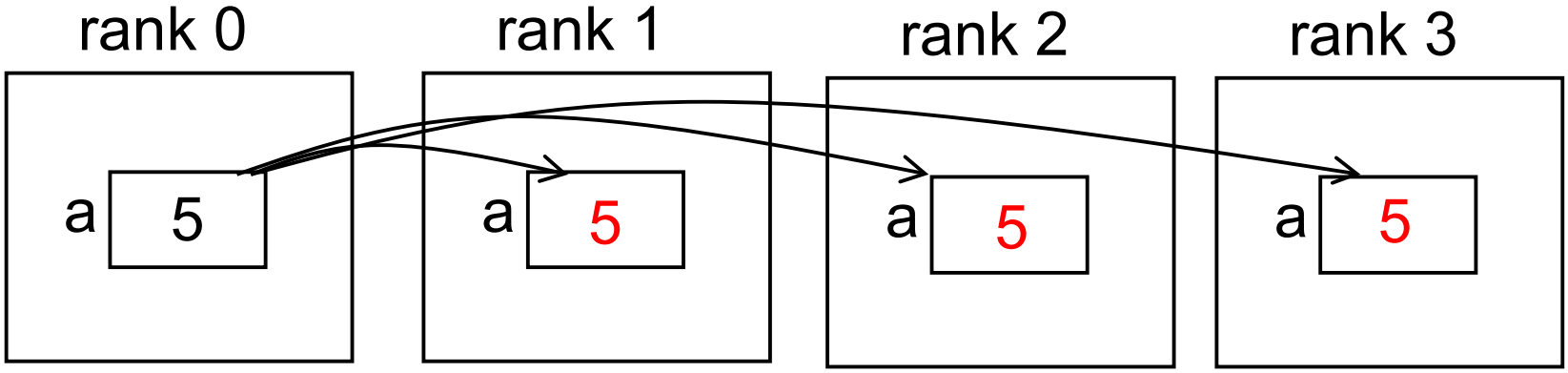


cf) rank 0 has “int a” (called root process). We want to send it to all other processes

```
MPI_Bcast(&a, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

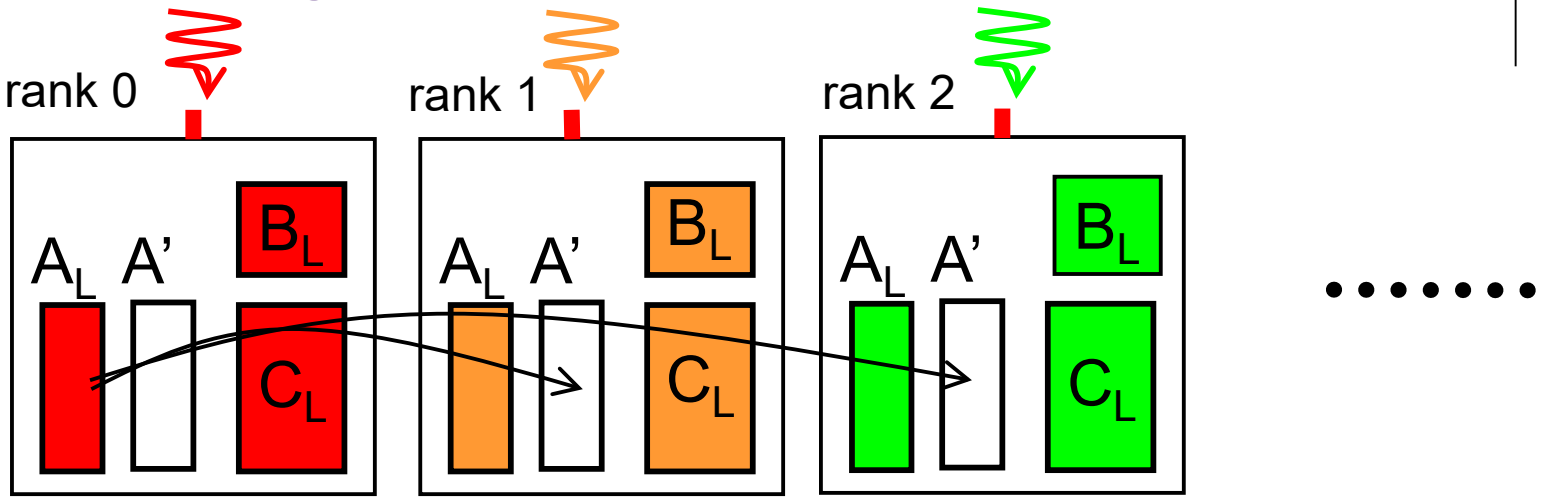
- All processes (in the communicator) must call MPI_Bcast(), including rank 0

→ All other process will receive the value on memory region a



※ What is the role of 1st argument?
it is “input” on the root process, and “output” on other processes

MPI_Bcast Can Be Used in Memory Reduced MM



- In Step i, rank i becomes the root
- It sends A_L to all other processes

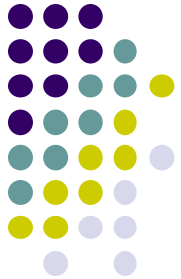
→ This is "broadcast" pattern. We can use MPI_Bcast!

Note: Root wants to send A_L. Others want to receive data into A'
→ Different pointers

Solution 1:
if (I am rank i) copies A_L to A'
MPI_Bcast(A', …);

Solution 2:
if (I am rank i) {MPI_Bcast(A_L, …); }
else {MPI_Bcast(A', …); }

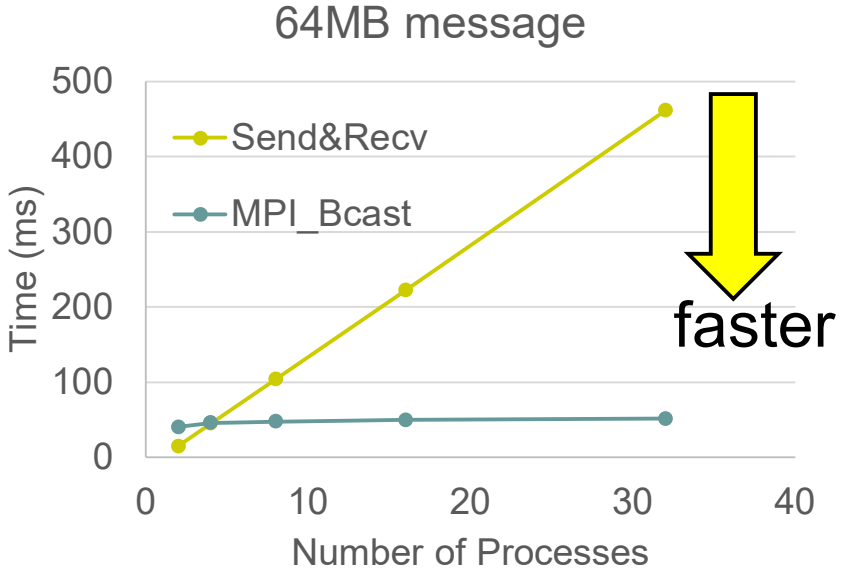
“Do I Really Need to Learn New Functions?”

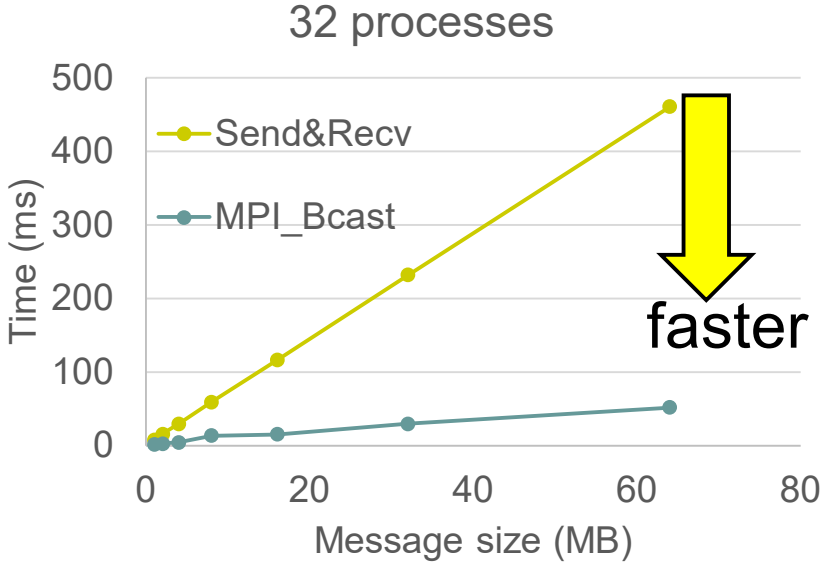


- You can still use MPI_Send/MPI_Recv multiple times, but collective functions are often faster

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

measured on TSUBAME2





- MPI_Bcast are faster, especially when p is larger !
- The reason is MPI uses “scalable” communication algorithms

  cf) http://www.mcs.anl.gov/~thakur/papers/mpi-coll.pdf

Reduction by MPI_Reduce

cf) Every process has “int a”. We want the sum of them

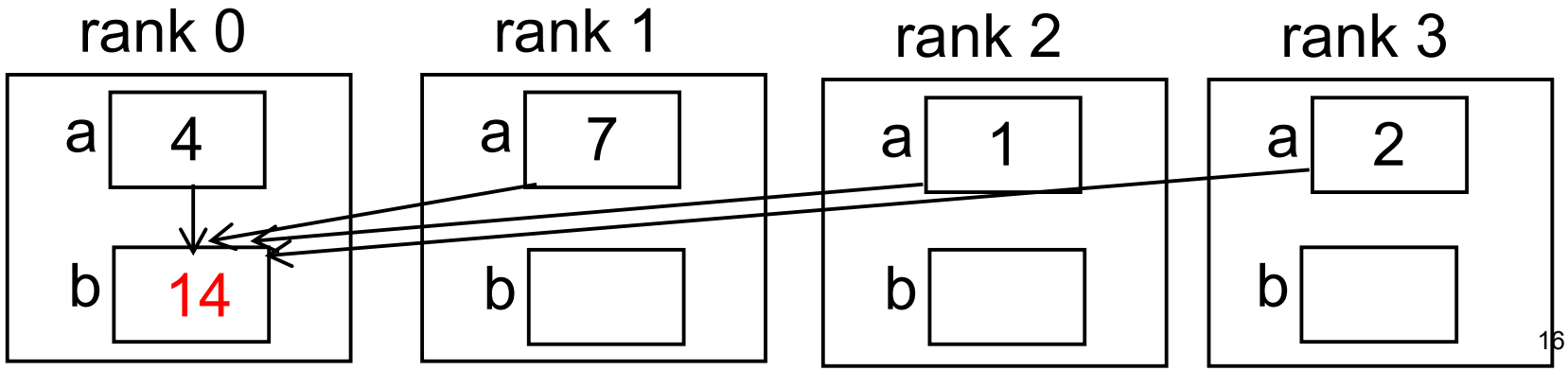
```
MPI_Reduce(&a, &b, 1, MPI_INT, MPI_SUM, 0,
           MPI_COMM_WORLD);
```

operation *root process*

- Every process must call MPI_Reduce()

→ The sum is put on b on root process (rank 0 now)

- Operation is one of MPI_SUM, MPI_PROD(product), MPI_MAX, MPI_MIN, MPI_LAND (logical and), etc.



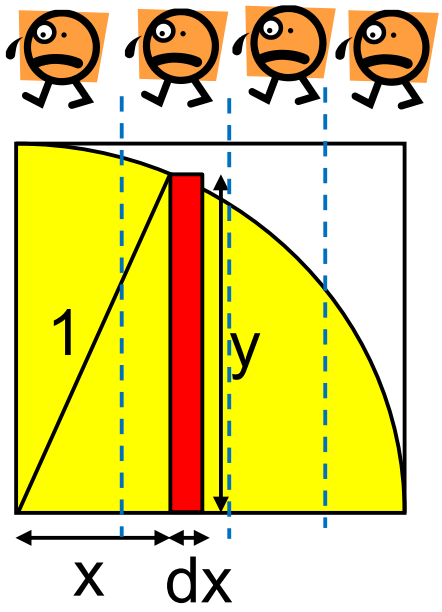
MPI Version of “pi” Sample

~endo-t-ac/ppcomp/19/pi-mpi/

- Execution : ./pi [n]
  - n: Number of division
  - Cf) ./pi 100000000

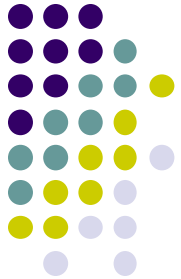
- We divide *n* tasks among processes and calculate total yellow area

1. Each process calculates local sum
2. Rank 0 obtains the final sum by MPI_Reduce

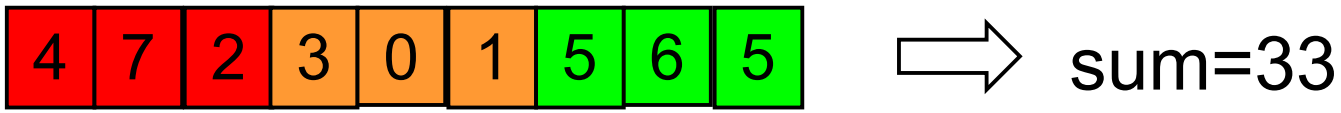


dx = 1/n
y = sqrt(1-x*x)

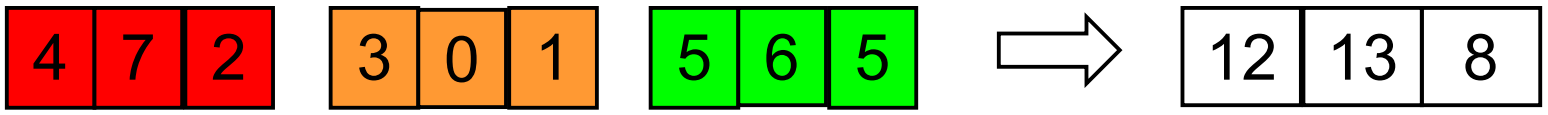
Note: Differences with “omp for reduction” in OpenMP



- Syntaxes are completely different
- Computations are also different
  - #pragma omp for reduction(…) in OpenMP
    - Do “sum += a[i]” in parallel for loop with reduction(+:s)



  - MPI_Reduce(…) in MPI
    - If each input is an array, output is also an array
    - Operations are done for each index

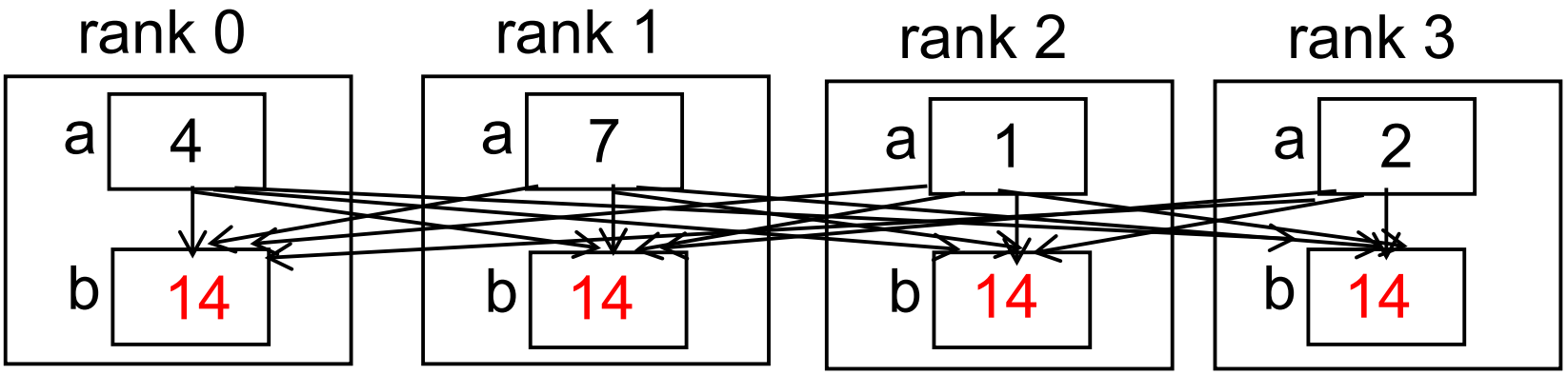


MPI_Allreduce

- Allreduce = Reduction + Bcast

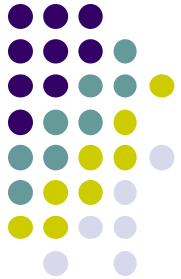
```
MPI_Allreduce(&a, &b, 1, MPI_INT, MPI_SUM,
              MPI_COMM_WORLD);
```

  - The sum is put on b on all processes



Important communication pattern for distributed deep learning → Google "allreduce deep learning"

MPI_Barrier

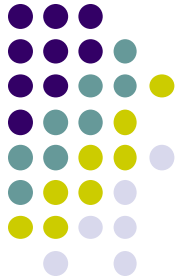


- Barrier synchronization: processes are stopped until all processes reach the point

```
MPI_Barrier(MPI_COMM_WORLD);
```

  - Used in sample programs, to measure execution time more precisely

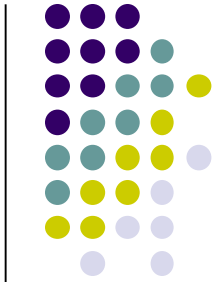
Other Collective Communications



- MPI_Scatter
  - An array on a process is “scattered” to all processes
  - cf) Process 0 has an array of length 10,000. There are 10 processes. The array is divided to parts of length 1,000 and scatterd
- MPI_Gather
  - Data on all processes are “gathered” to the root process.
  - Contrary to MPI_Scatter
- MPI_Allgather
  - Similar to MPI_Gather. Gathered data are put on all processes

:

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

- MPI (4)
  - Discussion on performance of MPI programs

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