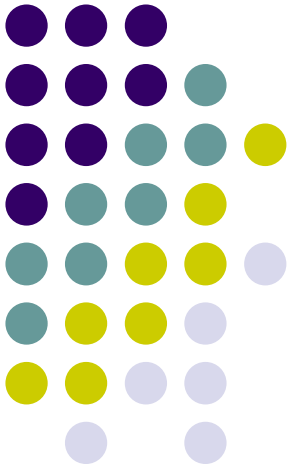
2017 Practical Parallel Computing (実践的並列コンピューティング) No. 9

Distributed Memory Parallel Programming with MPI (3)



Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

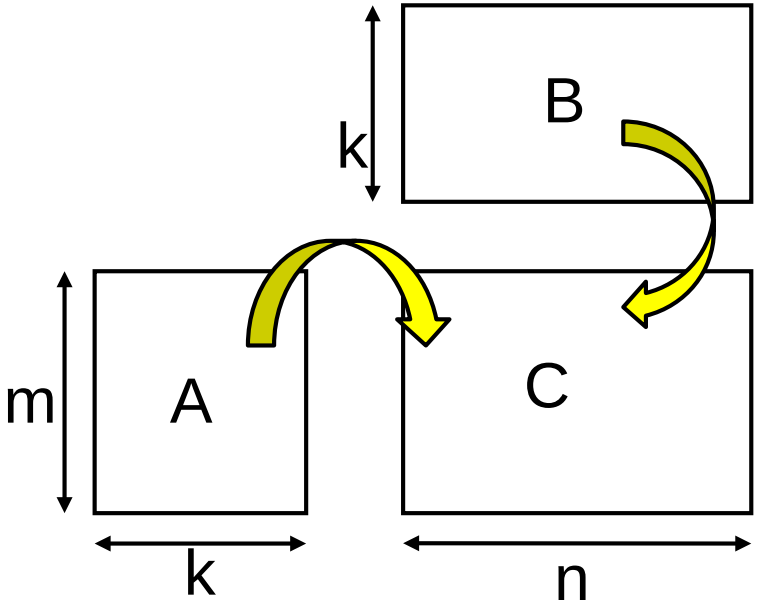
“mm” sample: Matrix Multiply (Revisited, related to [M2])

MPI version available at ~endo-t-ac/ppcomp/17/mm-mpi/

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

C: a (m × n) matrix

C ← A × B

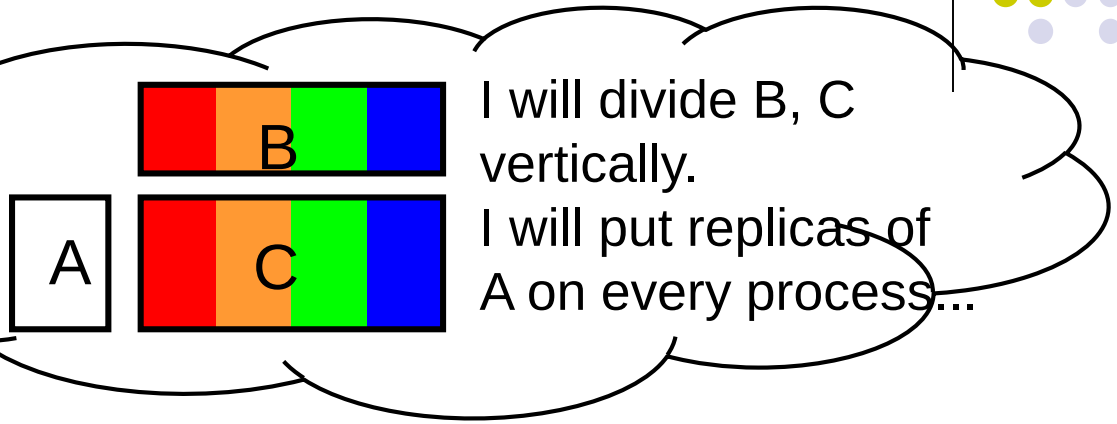


- Algorithm with a triple for loop
- Supports variable matrix size.
  - Each matrix is expressed as a 1D array by *column-major* format

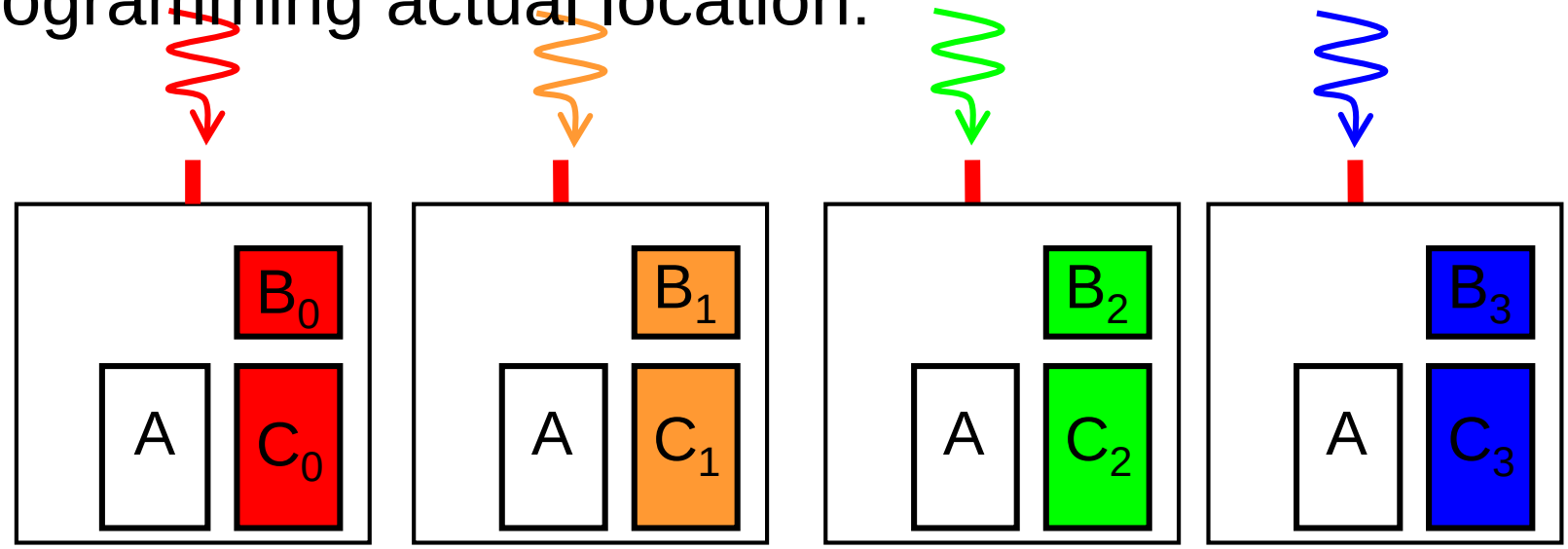
- Execution: mpirun –np [np] ./mm [m] [n] [k]

Programming Data Distribution
(for mm-mpi sample)

Design distribution method:



Programming actual location:

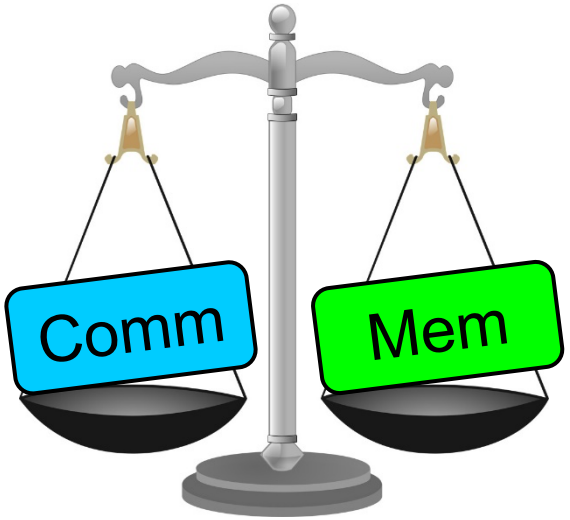


This is not a unique solution. How about other solutions?

Discussion on Considering Data Distribution

- Data distribution may affect to
  - Communication costs
  - Memory consumption
  - (Sometimes, computation costs)

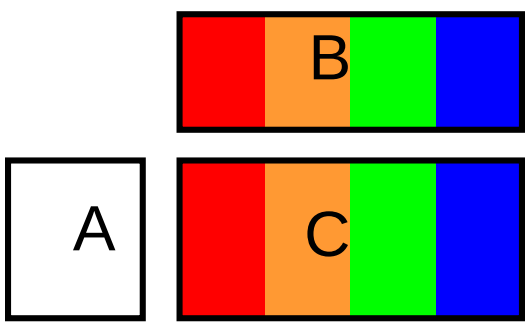
- Smaller is better
  - To reduce communication costs, we should put “dependent” data on the same processes



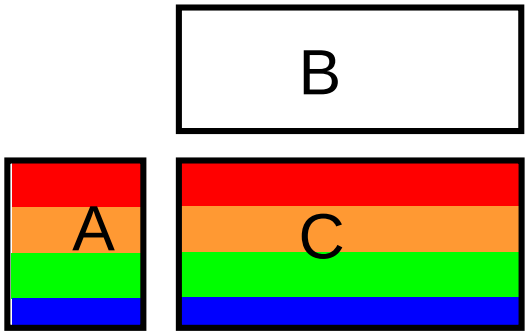
But there may be tradeoffs among them

Reconsidering Data Distribution of mm-mpi

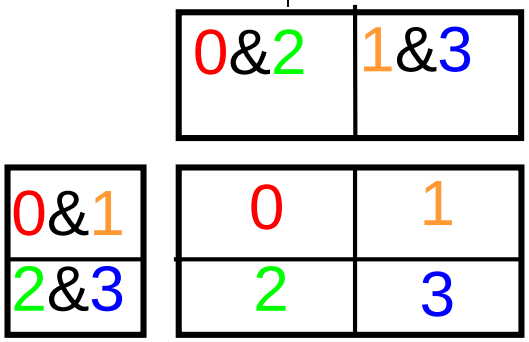
- Getting $C_{i,j}$ requires *i*-th row of A and *j*-th column of B



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. Is it sufficient?

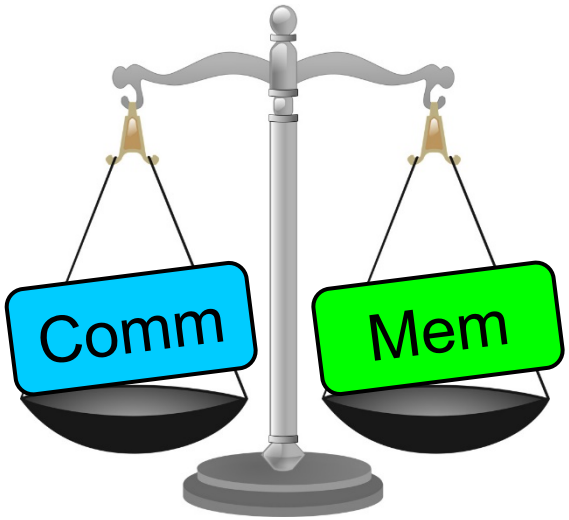
Reducing Memory Consumption

- 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 ☹

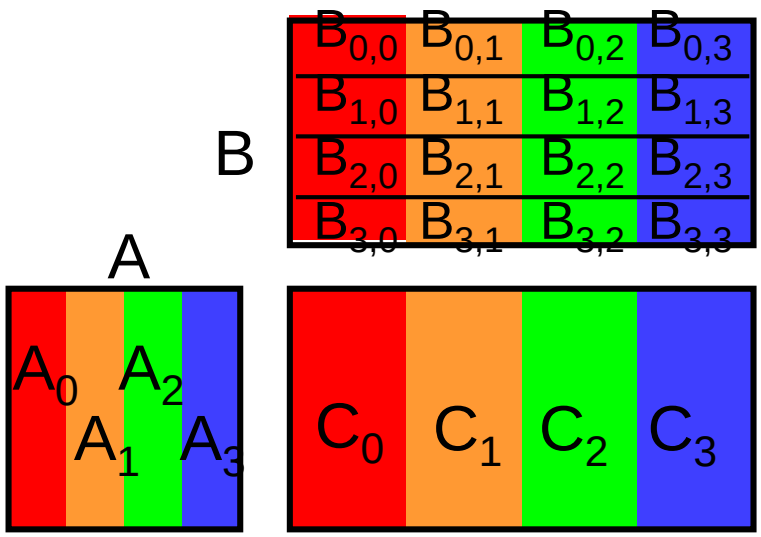
→ we cannot solve larger problems on supercomputers

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

→ But this increases communication costs



Data Distribution with Less Memory Consumption



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

Algorithm P_i = Process i

Step 0:

P_0 sends A_0 to all other processes

Every process P_i computes

C_i += A_0 × $B_{0,i}$

Step 1:

P_1 sends A_1 to all other processes

Every process P_i computes

C_i += A_1 × $B_{1,i}$

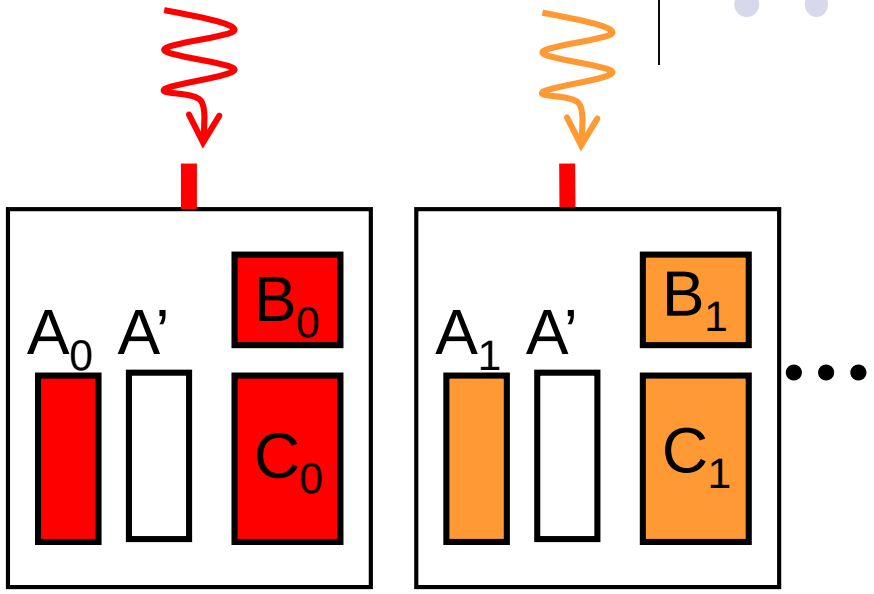
:

Repeat until Step (p-1)

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

Actual Data Distribution of Memory Reduced Version

- Basically, every process has partial A, B, C



- Additionally, every process should prepare a receive buffer (A’ in the figure)
  - A’ is used for arguments of MPI_Recv()

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

Improvements of Memory Reduced Version

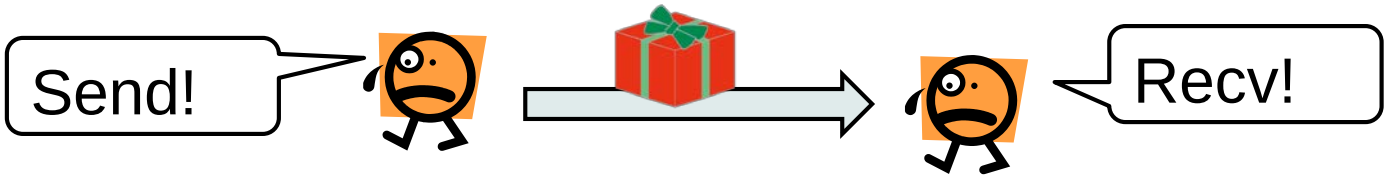
Followings are not mandatory in assignments [M2]

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

2. Use collective communications (described hereafter)

Peer-to-peer Communications

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



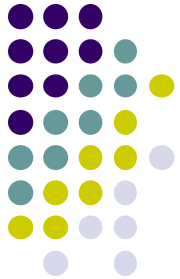
※ MPI_Irecv, MPI_Isend also does peer-to-peer communications

Collective 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

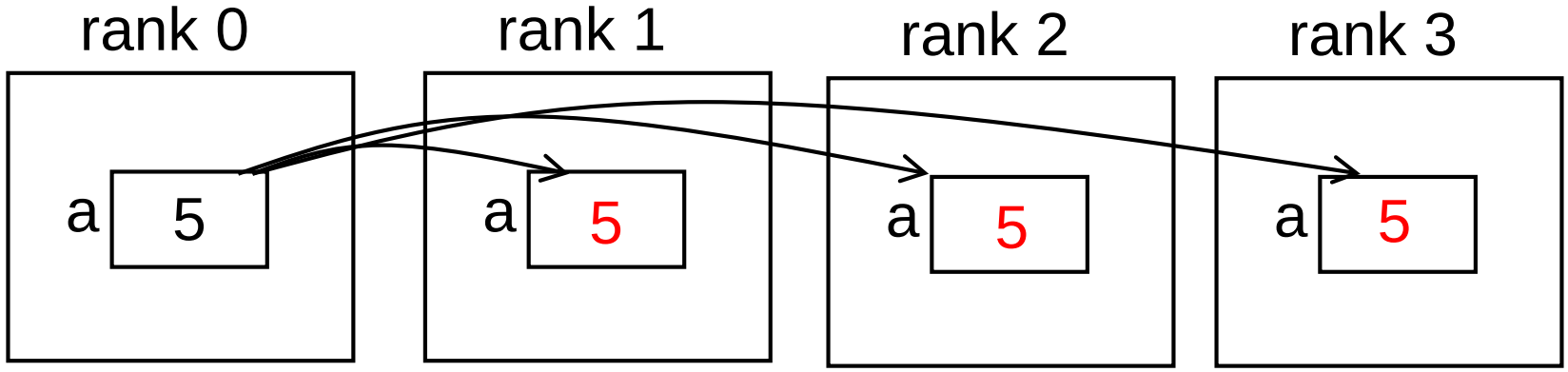


cf) rank 0 has “int a” (called root process). We want to send it to all other processes

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

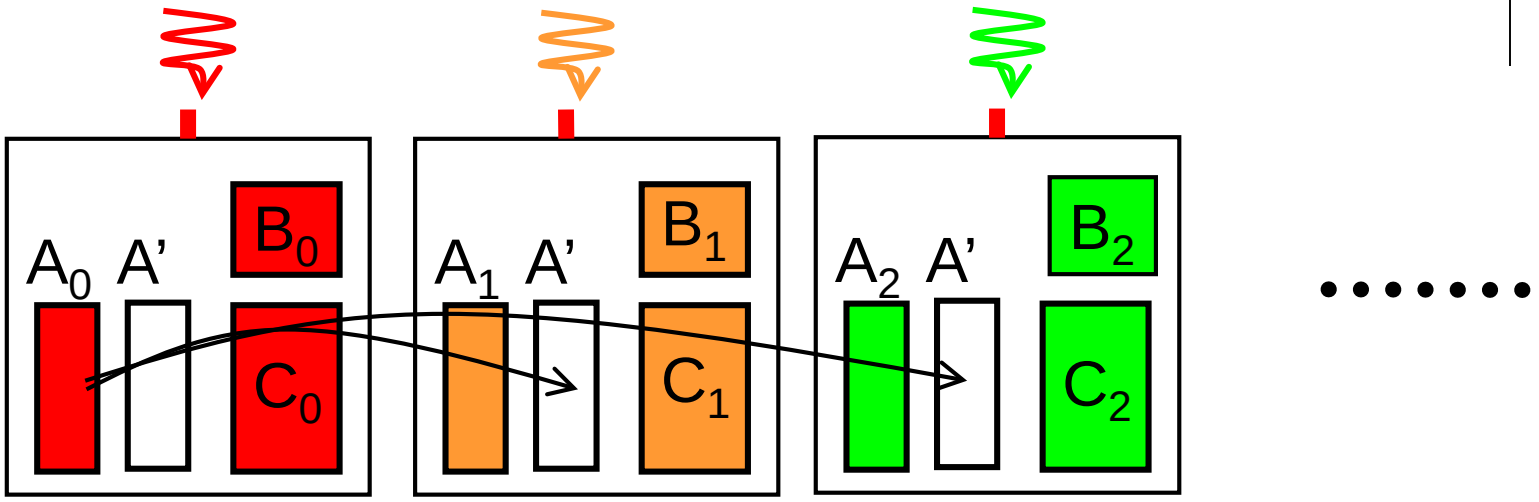
- All processes (in the communicator) must call MPI_Bcast(), including rank 0

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



※ 1st argument has a bit complicated role;
it is “input” on the root process, and “output” on other processes

MPI_Bcast Can Be Used in Memory Reduced MM



- In Step r, Process r becomes the root
- It sends A_r to all other processes

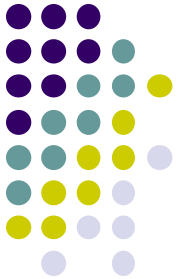
→ This is "broadcast" pattern. We can use MPI_Bcast!

Note: Root wants to send A_r. Others want to receive to A' → Different pointers

Solution 1:
Rank r copies A_r to A'
and then MPI_Bcast(A', …);

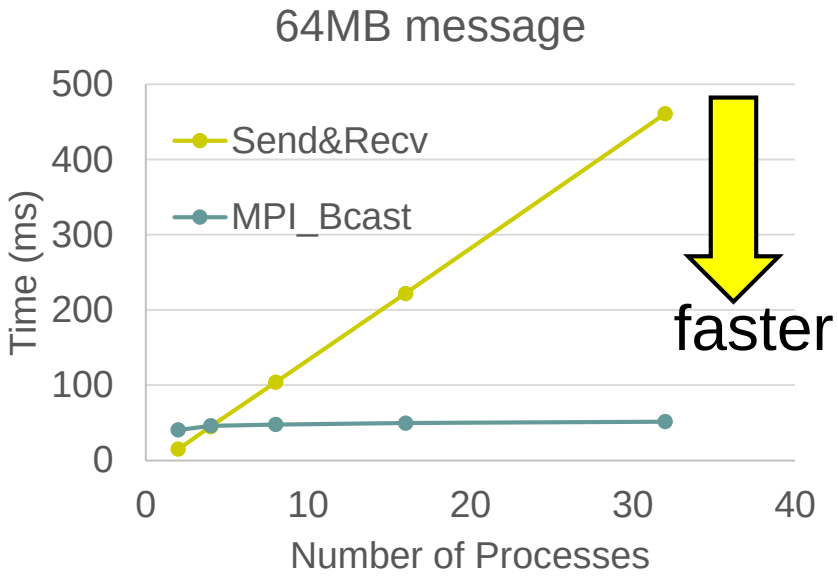
Solution 2:
if (I am rank r) {MPI_Bcast(A_r, …); }
else {MPI_Bcast(A', …); }

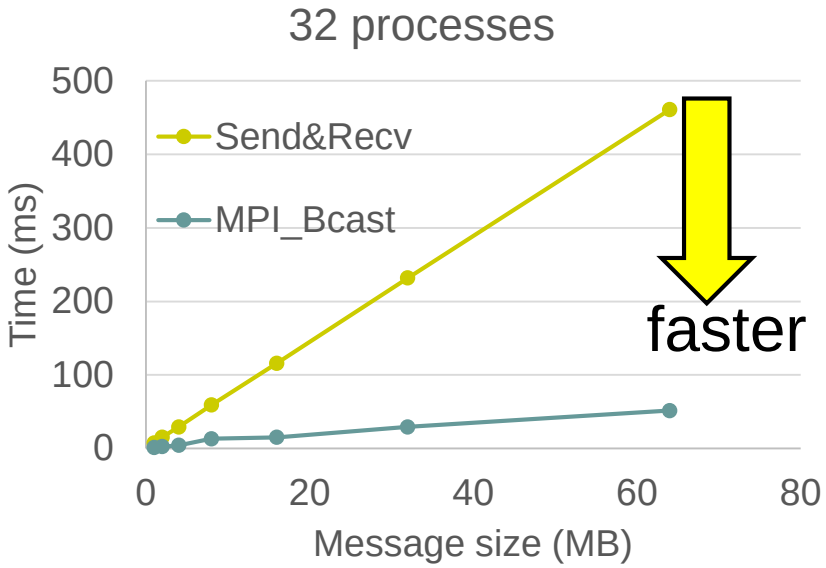
“Do I Really Need to Learn New Functions?”



- Yes, you can use MPI_Send/MPI_Recv multiple times instead of MPI_Bcast, but…

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





- In most cases, MPI_Bcast is 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 to sum of them.

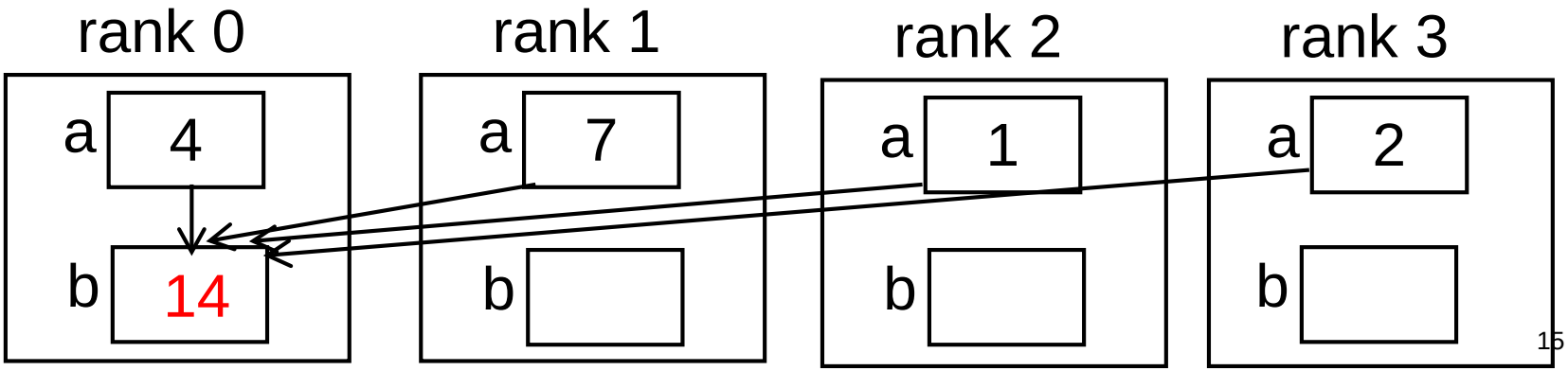
```
MPI_Reduce(&a, &b, 1, MPI_INT, MPI_SUM, 0,
           MPI_COMM_WORLD);
```

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.

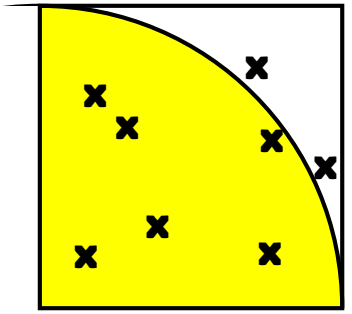


MPI Version of “pi” Sample

- pi-mpi sample

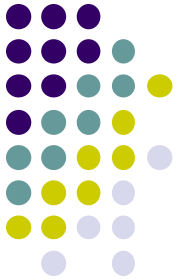
  ~endo-t-ac/ppcomp/17/pp-mpi/

  - We need to get “total number” of points in yellow area → Reduction

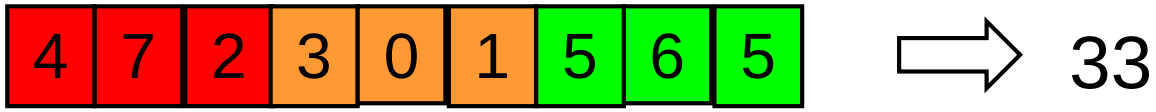


1. Each process calculates local sum
2. Rank 0 obtains the final sum by MPI_Reduce

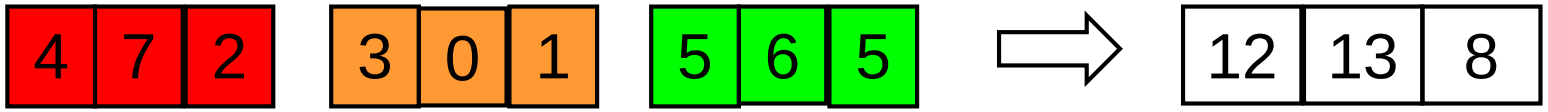
Difference with “omp for reduction” in OpenMP



- First, syntaxes are completely different
- In OpenMP



- In MPI
  - If each input is an array, output is also an array
  - Operations are done for each index

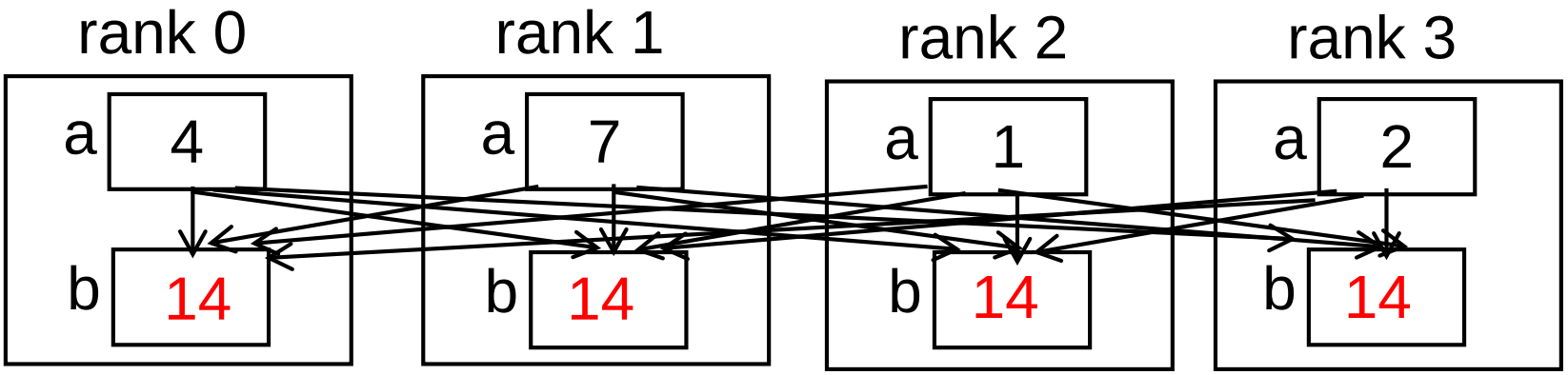


MPI_Allreduce

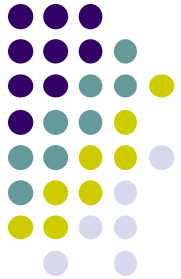
- Allreduce = Reduction + Bcast

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

  - The sum is put on b on all processes



MPI_Barrier

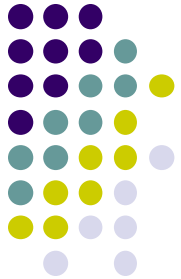


- Barrier synchronization: processes are stopped until all processes reach the point

  `MPI_Barrier(MPI_COMM_WORLD);`

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

Other Collective Communications

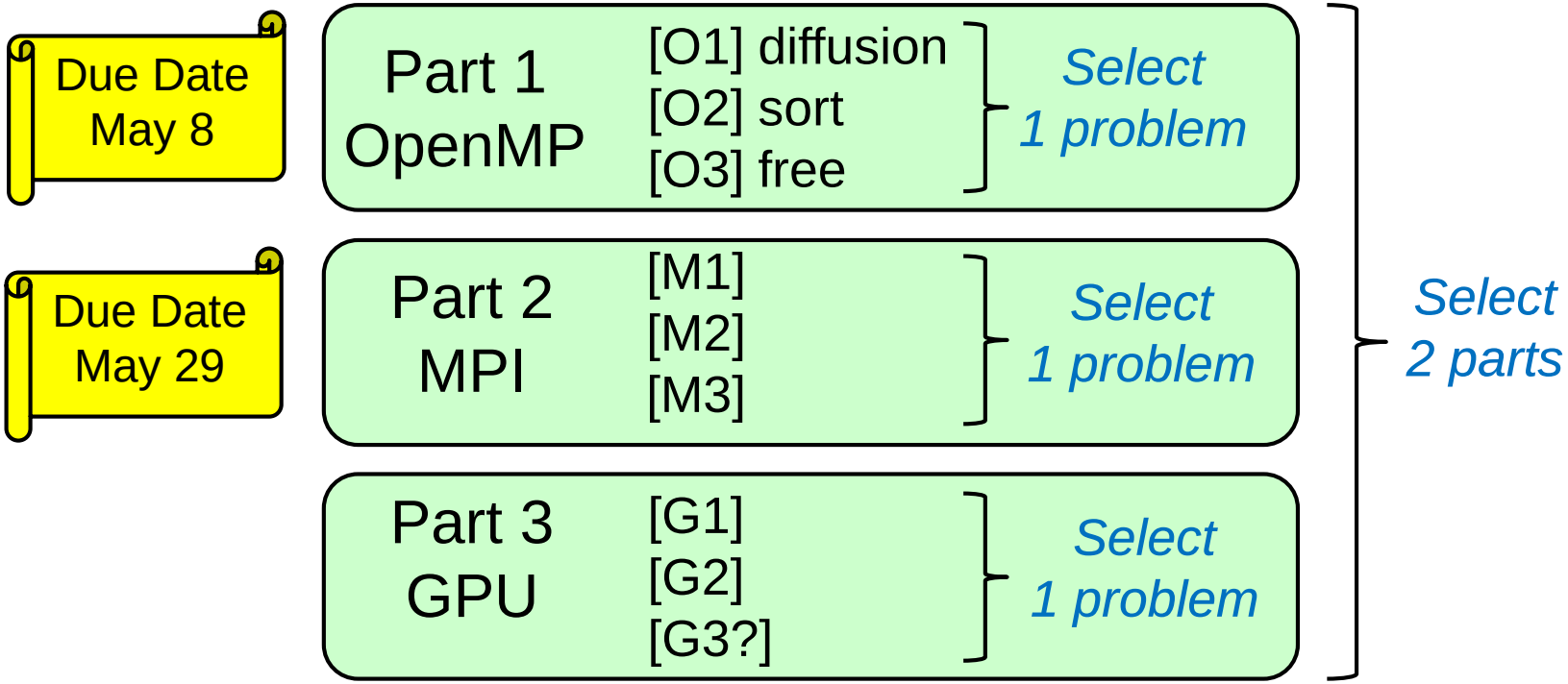


- MPI_Scatter
  - An array on a process is “scattered” to all processes
  - cf) Process 0 has an array of length 10,000. There are 10 processes. The array is divided to parts of length 1,000 and scatterd
- MPI_Gather
  - Data on all processes are “gathered” to the root process.
  - Contrary to MPI_Scatter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

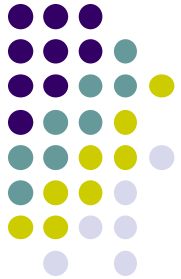
:

Assignments in this Course

- There is homework for each part. Submissions of reports for 2 parts are required
- Also attendances will be considered



Assignments in MPI Part (Abstract)



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

Due date: May 29 (Monday)

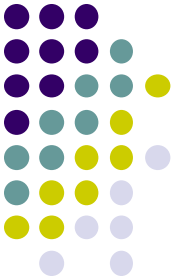
[M1] Parallelize “diffusion” sample program by MPI.

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

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

For more detail, please see Apr 27 slides or OCW-i.

Next Class on May 15



- May 11 is cancelled
- MPI (4)
  - Discussion on performance of MPI programs

- Do not forget: Due date of assignments of Part 1 is May 8 (today)