Practical Parallel Computing (実践的並列コンピューティング)

Part3: MPI (4)
May 30, 2021

Toshio Endo

School of Computing & GSIC

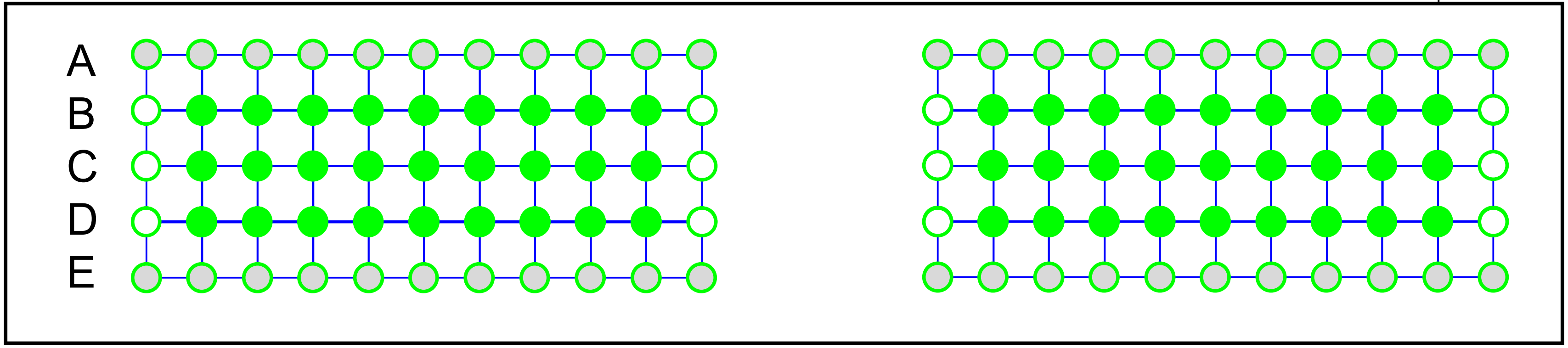
endo@is.titech.ac.jp

Improving MPI diffusion by Overlapping of Communication

related to [M1], but optional

Overview of MPI “diffusion”

(See MPI (2) Slides)



```
for (t = 0; t < nt; t++) {
  if (rank > 0) Send B to rank-1
  if (rank < size-1) Send D to rank+1
  if (rank > 0) Recv A from rank-1
  if (rank < size-1) Recv E from rank+1
  Computes points in rows B-D
  Switch old and new arrays
}
```

(1) Communication in “old” array

(2) Computation “old” array ⇒ “new” array

Actually this should be fixed to avoid deadlock

Considering Performance of MPI Programs

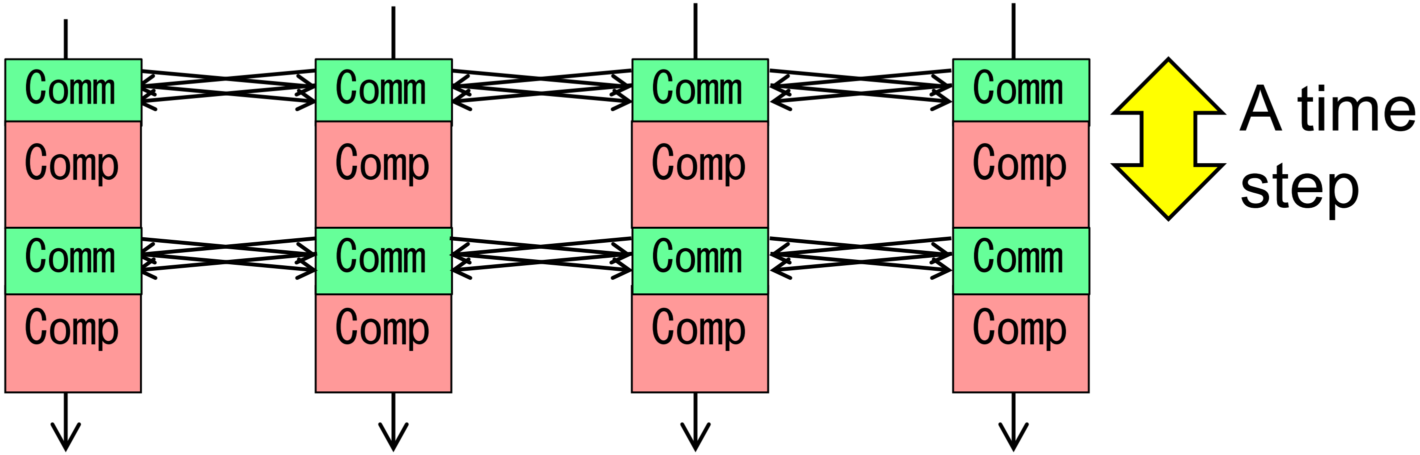
(Simplified) Execution time of an MPI program =

Computation time ← including memory access

+ Communication time ← including congestion

+ Others ← load imbalance, I/O…

Behavior of stencil computations on MPI



Computation Time & Communication Time (1)

How are they determined? (very simplified discussion)

1. Aspect of software

Computation time

- Longer if computation costs are larger
  - O(mnk/p) in matmul,
  - O(NX NY NT/p) in diffusion

per process

Communication time

- Longer if communication costs are larger
  - O(mk) in memory reduced matmul
  - O(NX NT) in diffusion

per process

Computation Time & Communication Time (2)

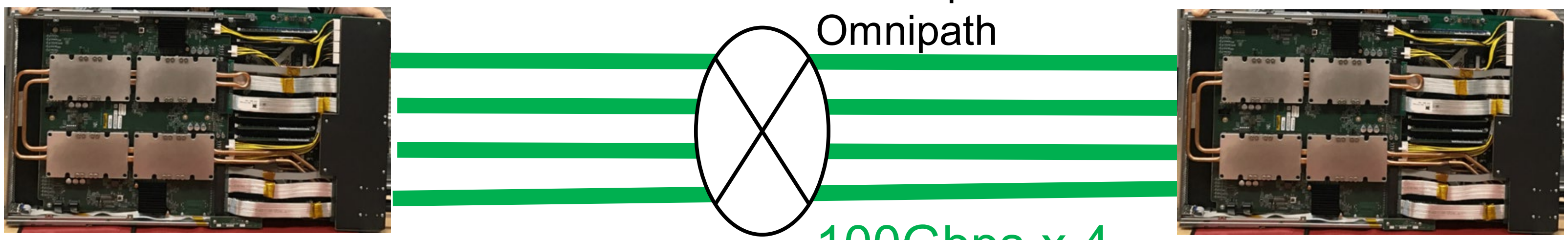
2. Aspect of hardware

Computation time

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

Communication time

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



100Gbps Omnipath

425GFlops x 2CPUs
= 850GFlops

100Gbps x 4
=400Gbps

425GFlops x 2CPUs
= 850GFlops

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

Parameters for Network Speed

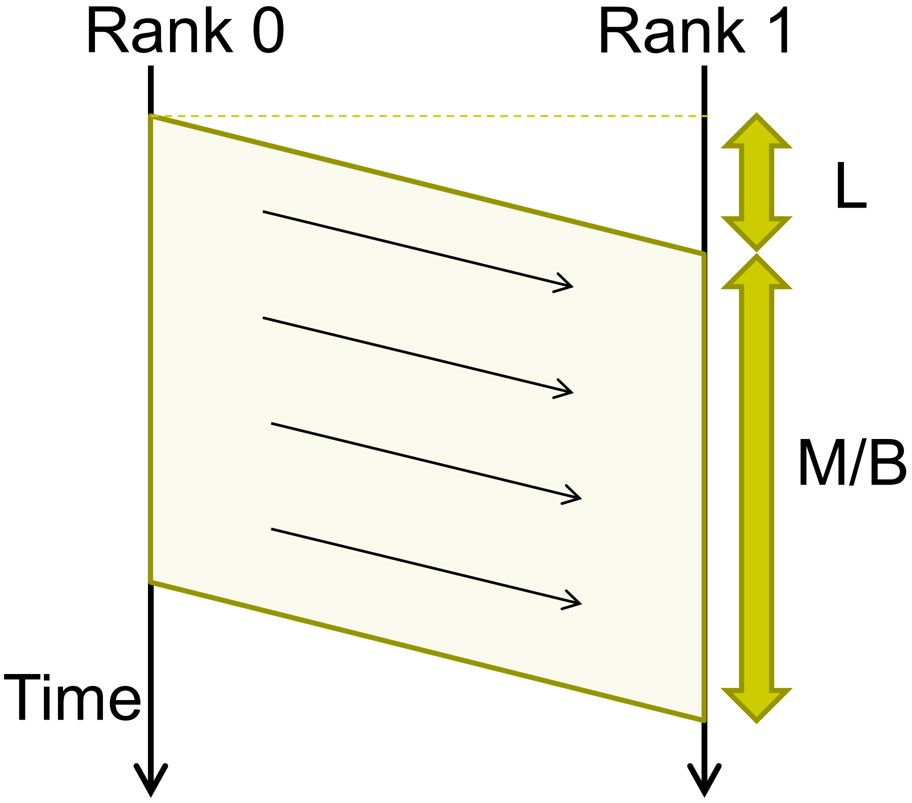
What parameters describes network speed?

- 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, a few us

※ Additionally, communication time may suffer from effects of network topology: how nodes/switches are connected to each other

Bandwidth and Latency

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



$$T = M / B + L$$

T: Communication time

M: Data size

B: Bandwidth

L: Network latency

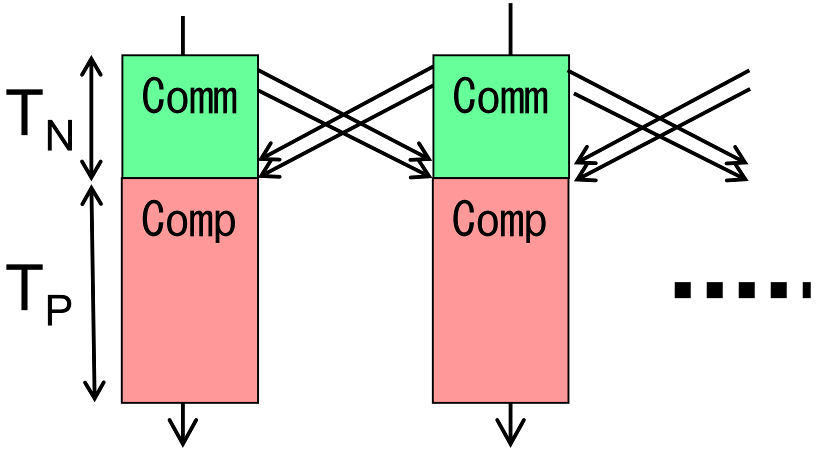
※ Be aware of difference between “Byte” and “bit”: 1Byte=8bit

※ In some contexts, T, not L, may be called “latency”

Idea of Overlapping

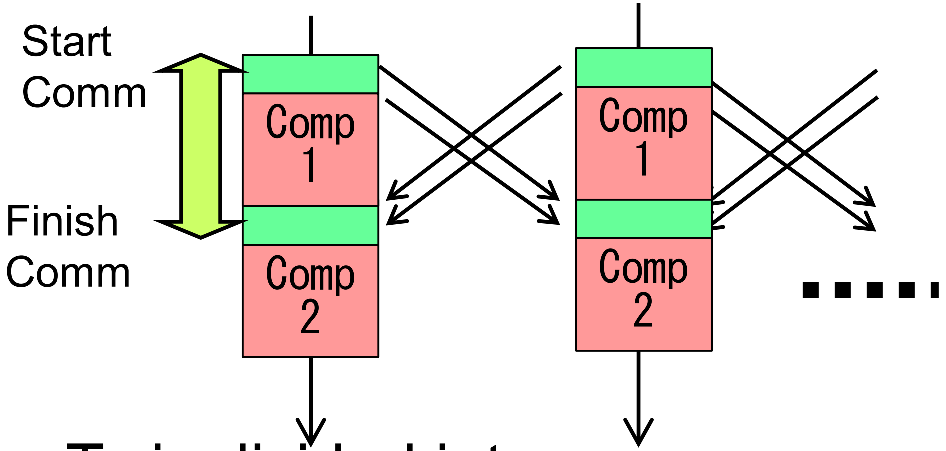
If "some computations" do not require contents of message, we may start them beforehand

Without overlap



$$T=T_N+T_P$$

With overlap



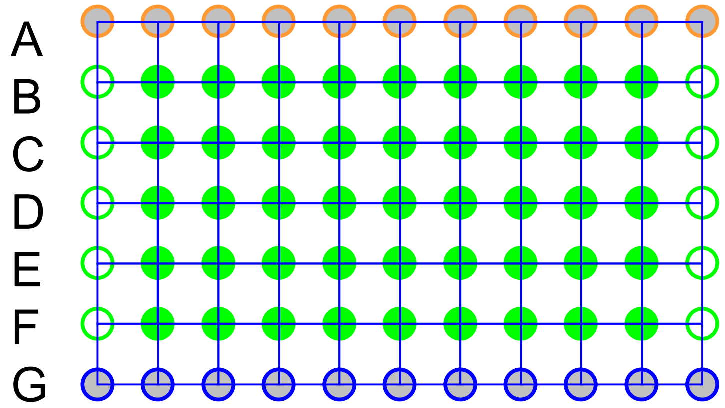
T_P is divided into

- 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 requied)

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

For such purposes, non-blocking communications (MPI_Isend, MPI_Irecv…) are helpful again

Implementation without Overlapping (Not Fast!)

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

T_N

T_P

$$T=T_N+T_P$$

Implementation with Overlapping

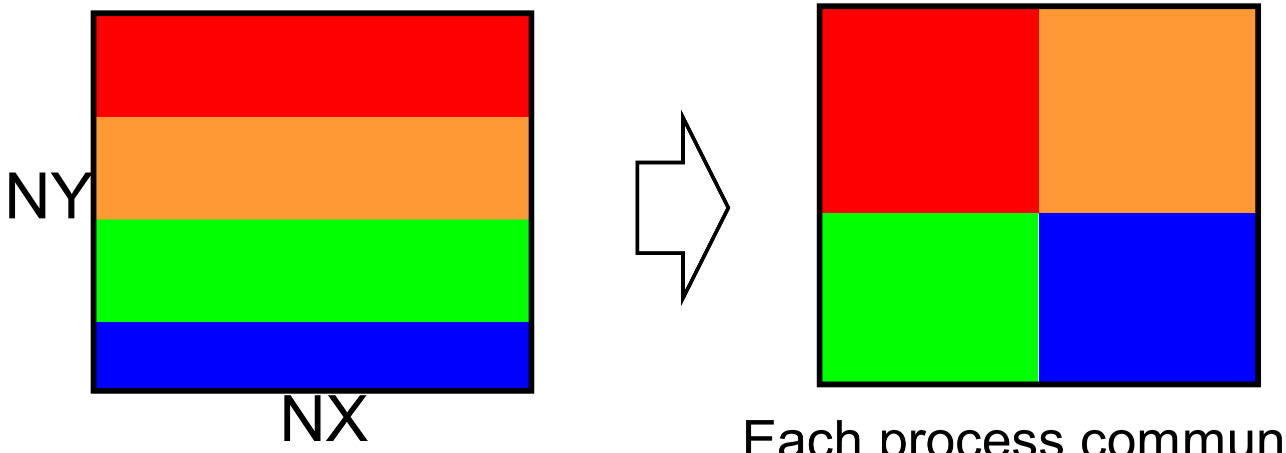
```
for (t = 0; t < nt; t++) {
   Start Send B to rank-1, Start Send F to rank+1
    (MPI_Isend)
   Start Recv A from rank-1, Start Recv G from
    rank-1 (MPI_Irecv)
  Compute rows C--E
   Waits for finishing all communications
    (MPI_Wait)
  Compute rows B, F
  Switch old and new arrays
}
```

computations are divided

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

Another Improvement: Reducing Communication Amounts

Multi-dimensional division may reduce communication



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

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

per 1 process, 1 iteration

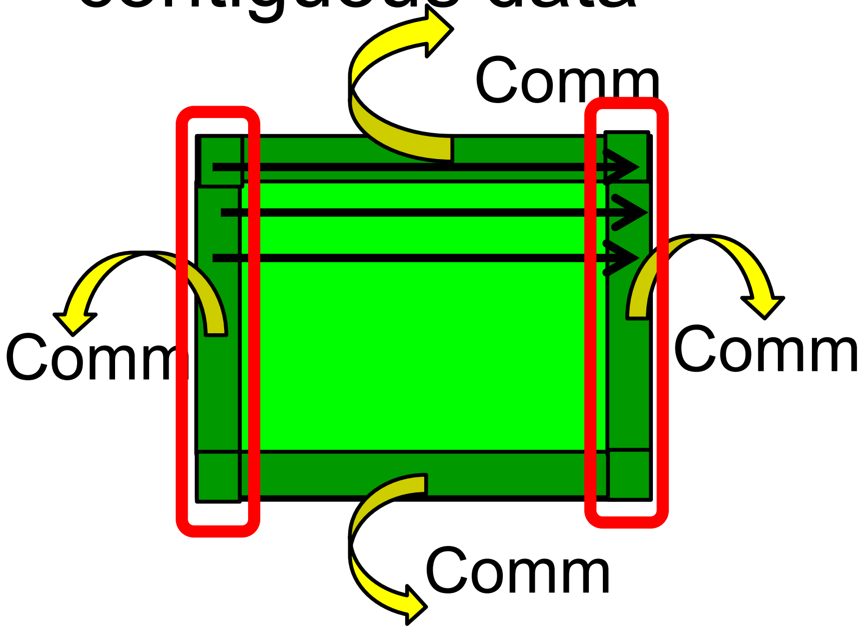
- Comp: O(NY NX/p)
- Comm: O((NY+NX)/$p^{1/2}$)

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; you may use Google

Performance of Collective Communication

also in MPI (3) slides

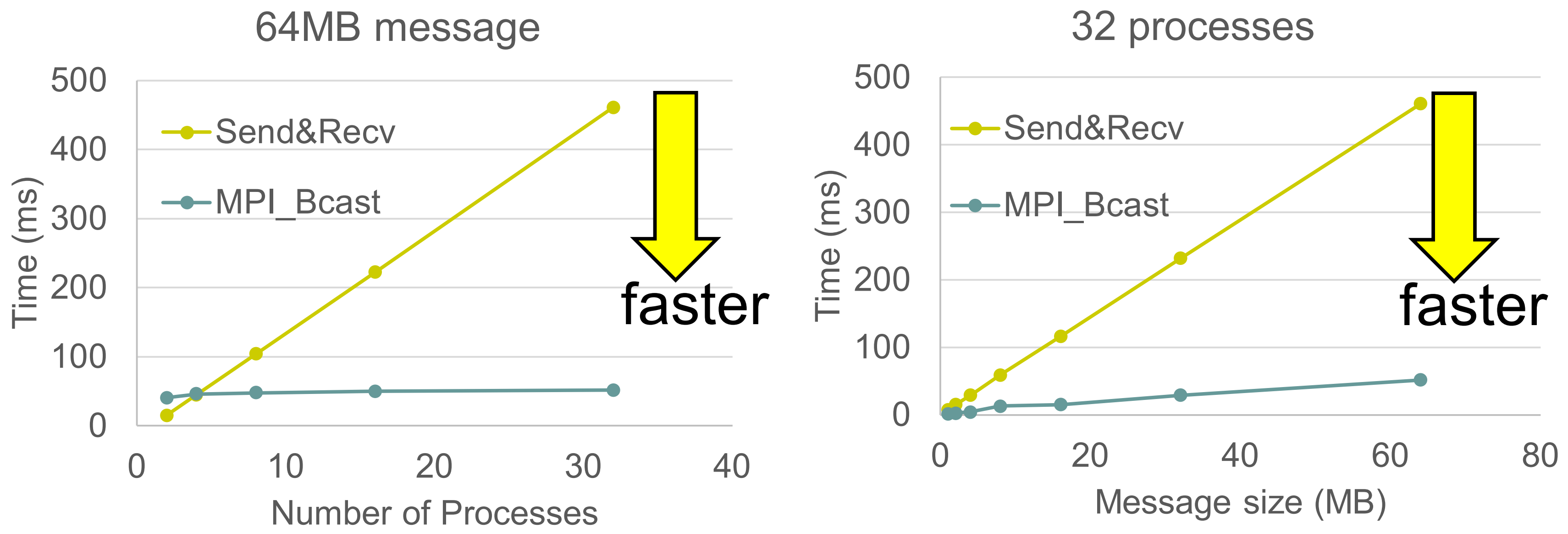
It is Better to Use Collective Communications if Appropriate

- Comparing MPI_Bcast and MPI_Send&Recv

  1 process per node is invoked (to measure network)

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

measured on TSUBAME2

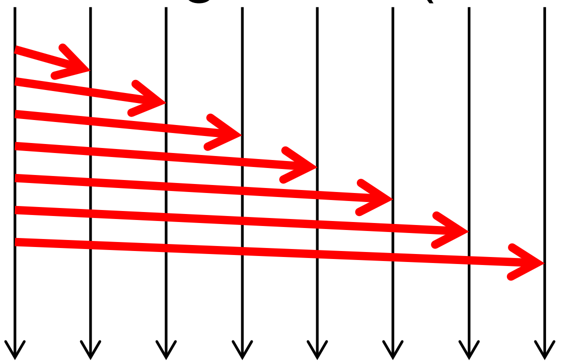


- In most cases, MPI_Bcast is faster

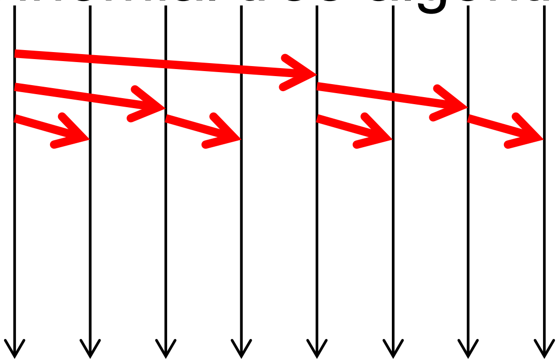
Why are Collective Communications Fast?

- Since MPI library uses scalable communication algorithms
  - Case of broadcast:

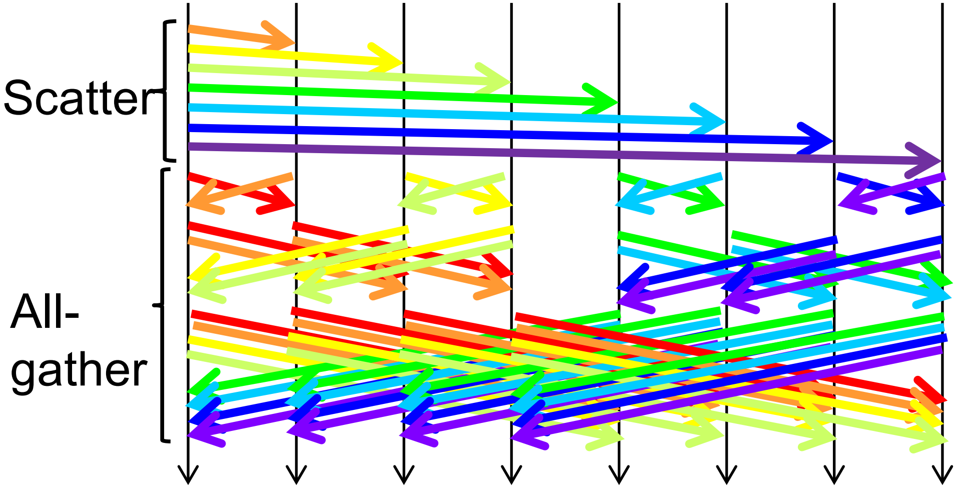
Flat tree algorithm (slow)



Binomial tree algorithm

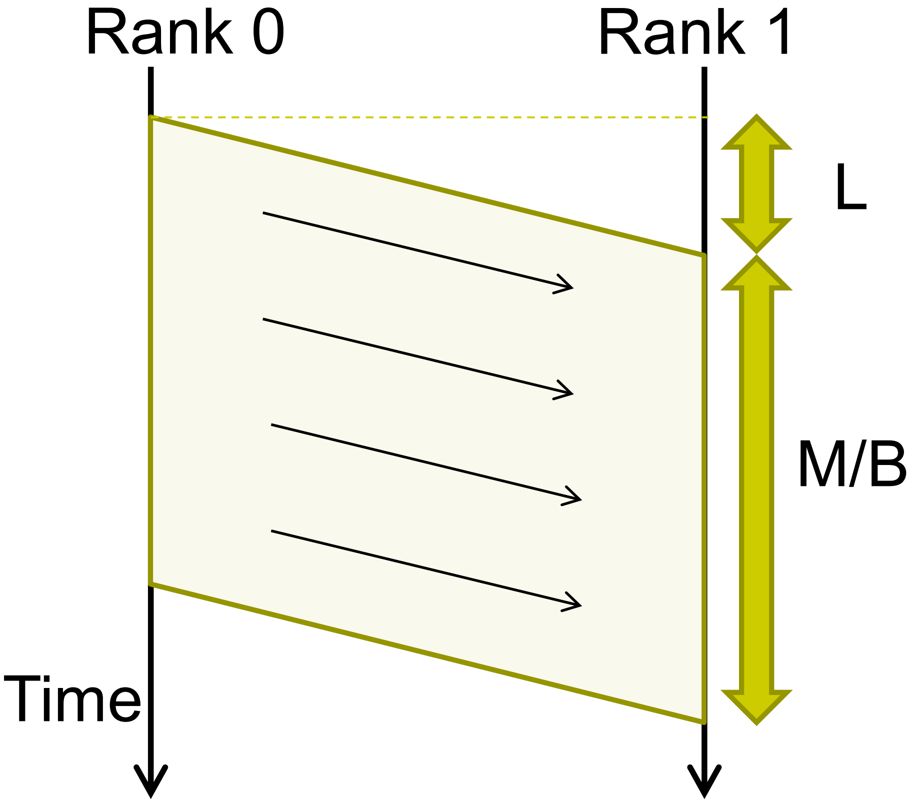


Scatter&Allgather algorithm



(again) Model of Communication Time

Illustration of peer-to-peer communication of data size M



$$\boxed{T = M / B + L}$$

T: Communication time

M: Data size

B: Bandwidth

L: Network latency

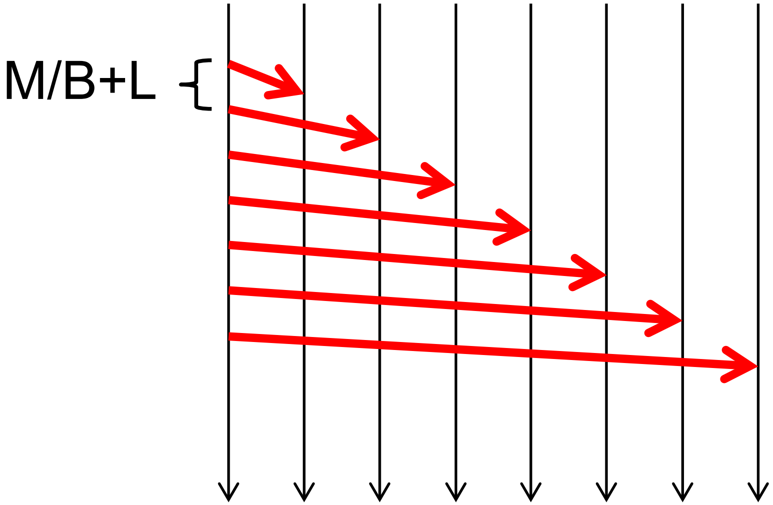
※ Be aware of difference between "Byte" and "bit": 1Byte=8bit

※ Actually it is more complex for process's place, effects of network topology, congestion, packet size…

Cost Model of Broadcast Algorithms

- Case of “broadcast” of size M data
  - p: number of processes, B: network bandwidth, L: network latency

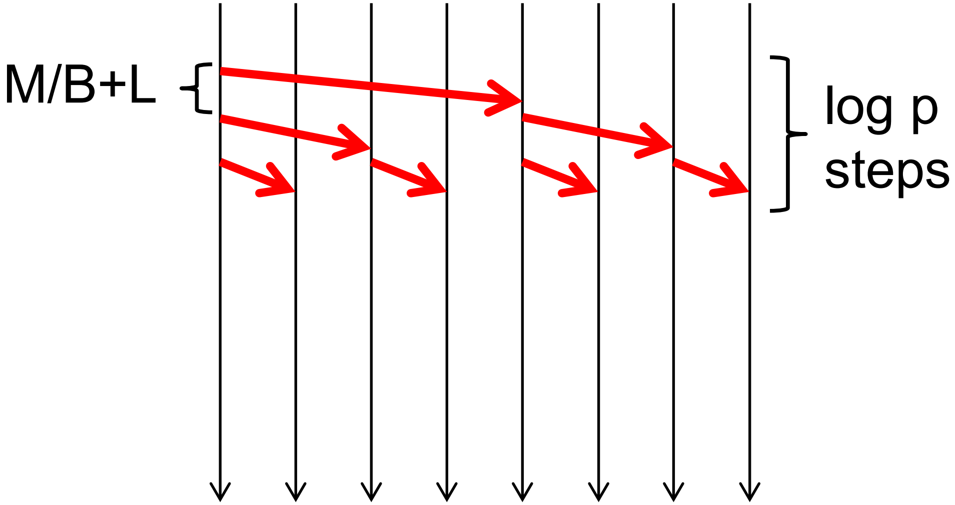
Flat tree algorithm



$p(M/B+L)$

→ Slow

Binomial tree algorithm



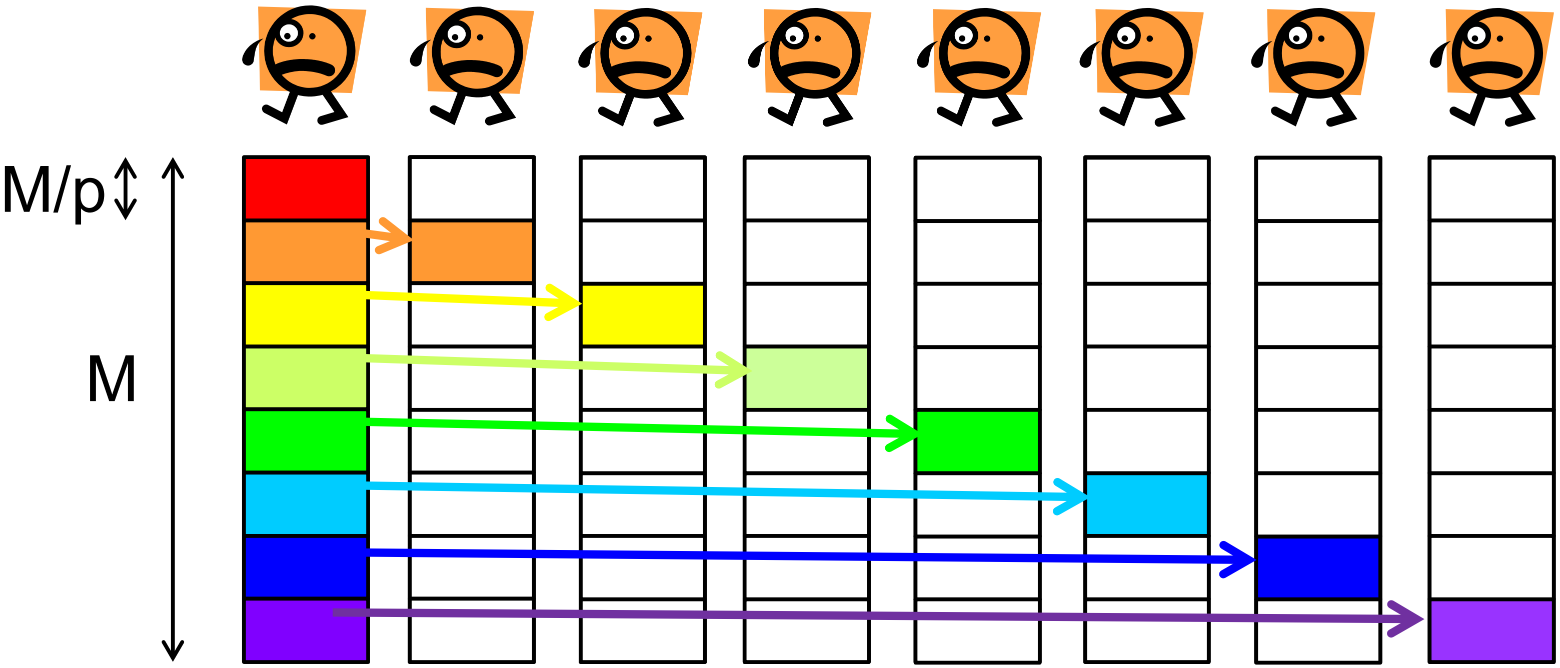
$(\log p)(M/B+L)$

※difference between p-1 and p is ignored

Broadcast by Scatter&Allgather (1)

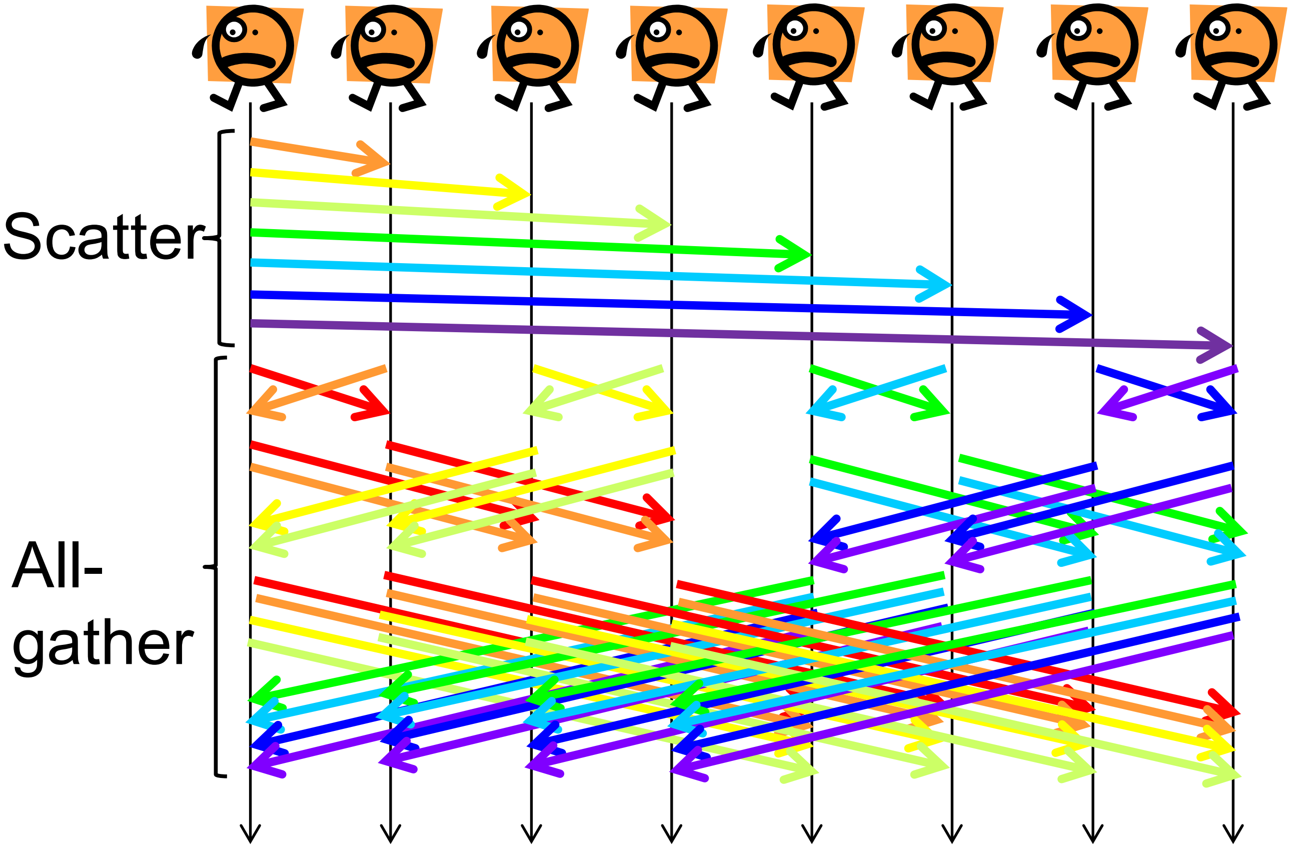
- Scatter&Allgather algorithm
  1. The root process divide the message into p parts
  2. Scatter
  3. Allgather

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



Broadcast by Scatter&Allgather (2)

- Scatter&Allgather algorithm
  1. The root process divide the message into p parts
  2. Scatter
  3. Allgather



$$\underbrace{\overbrace{M/B + pL}^{\text{Scatter}} + M/B + (\log p)L}_{\text{All-gather}}$$

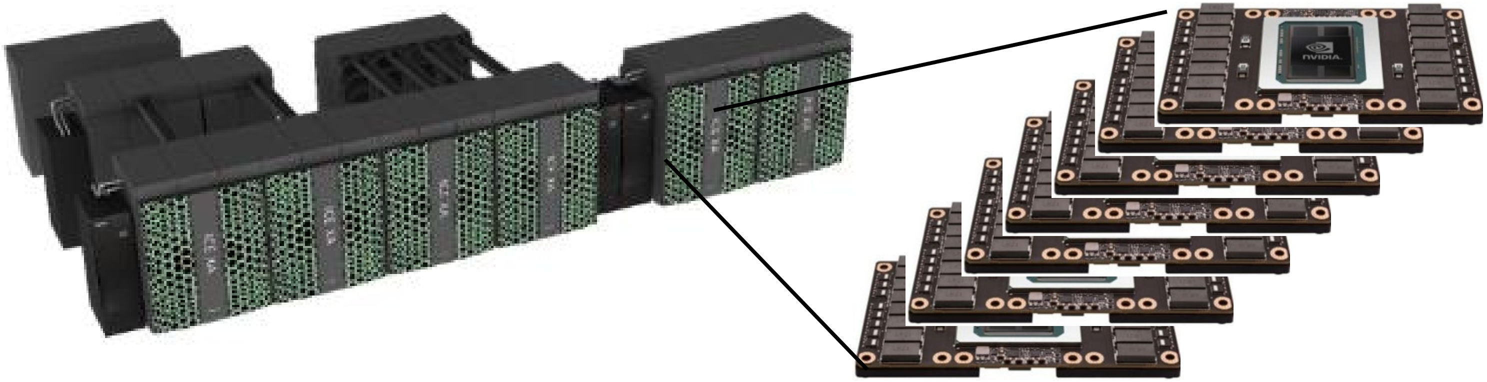
Comparison of Broadcast Algorithms

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

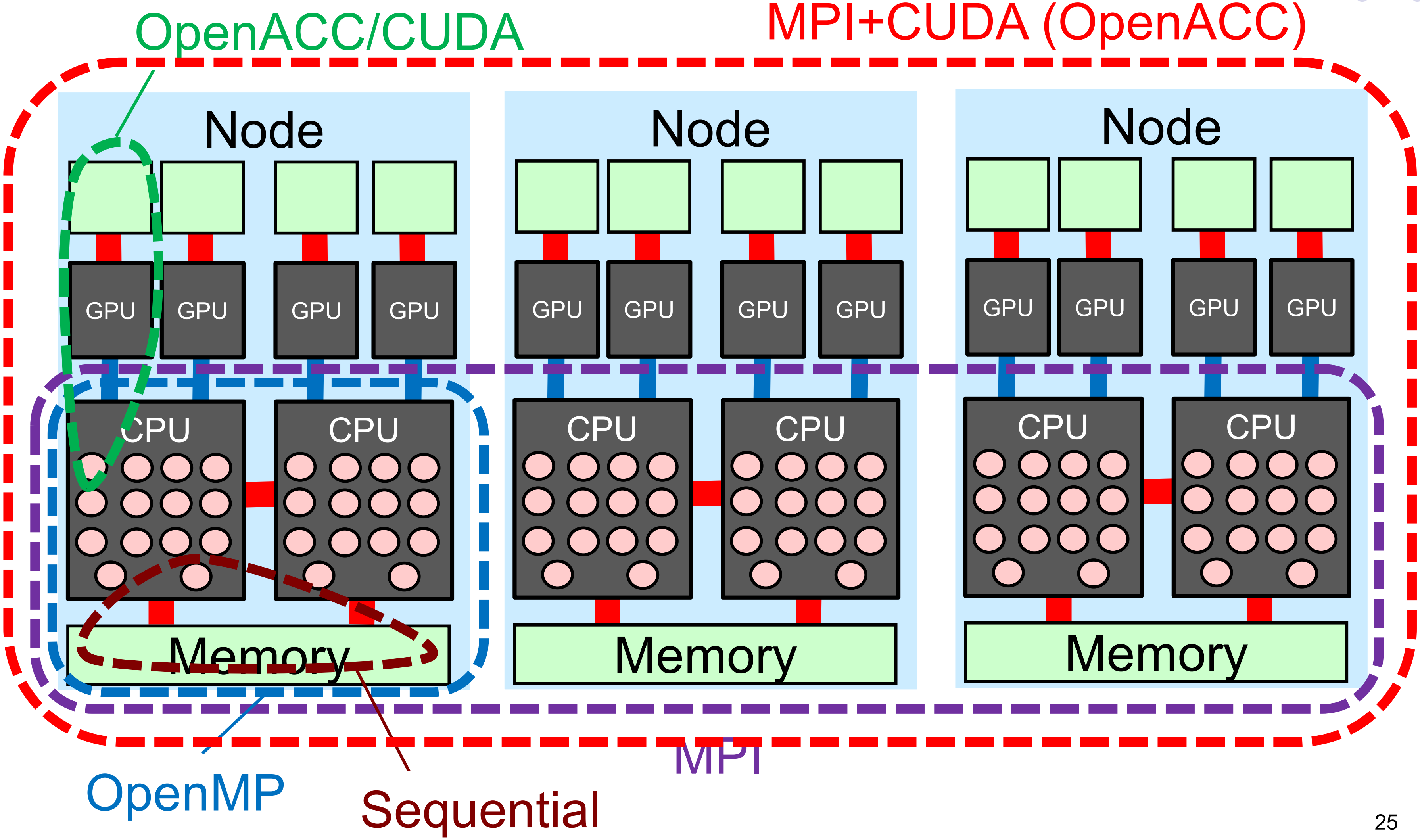
	Flat Tree	Binomial Tree	Scatter& Allgather
Cost (General)	p(M/B+L)	(log p) (M/B+L)	2M/B + (p + log p)L
Cost with very large M	p M/B	(log p) M/B	2 M/B → Fastest
Cost with very small M	p L	(log p) L → Fastest	(p + log p) L

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

Using Multiple GPUs with MPI+CUDA



Parallel Programming Methods on TSUBAME



Using Multiple GPUs

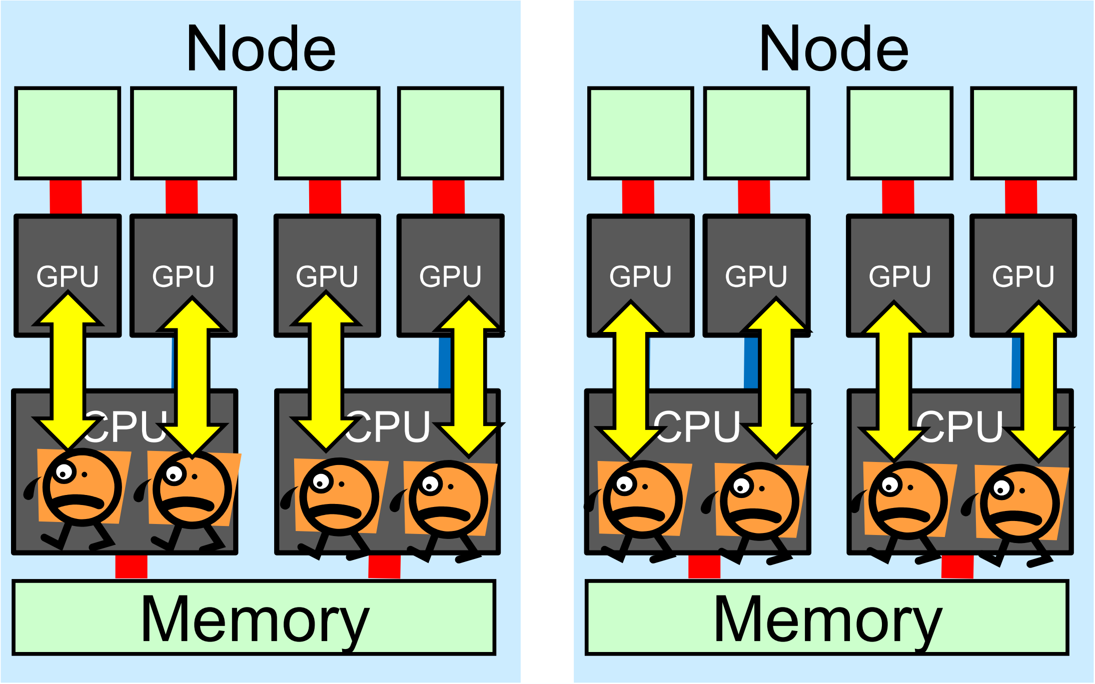
- GPUs on a single node
  - Up to 4 GPUs on TSUBAME3.0 f_node
  - OpenMP + CUDA
    - 1 thread uses 1 GPU
  - 1 thread switches multiple GPUs
    - cudaSetDevice() is called many times
- GPUs on multiple nodes
  - MPI + CUDA
    - 1 process uses 1 GPU (mm-mpi-cuda sample)

Using Multiple GPUs with MPI

- Basic idea:

  (1) Start processes on multiple nodes by MPI

  (2) Each process uses its local GPU by CUDA



Sample: /gs/hs1/tga-ppcomp/22/mm-mpi-cuda/

Compiling mm-mpi-cuda Sample

```
module load cuda openmpi    [Do once after login]
cd ~/t3workspace            [In web-only route]
cp -r /gs/hs1/tga-ppcomp/22/mm-mpi-cuda  .
cd mm-mpi-cuda
make
[An executable file “mm” is created]
```

In this Makefile,

- nvcc is used as the compiler
- mpic++ is used as the linker, with CUDA libraries

Note: This may not work on other systems or future TSUBAME

Executing mm-mpi-cuda

- Interactive use is only for one node

→ To use multiple nodes, job submission is required

In standard route,
Use qsub on the
login node

qsub job2q.sh ➔ q_node (1GPU) x 2 are used ➔ 2GPUs in total

qsub job2f.sh ➔ f_node (4GPU) x 2 are used ➔ 8 GPUs in total

job2f.sh

```
#!/bin/sh
#$ -cwd
#$ -l f_node=2
#$ -l h_rt=0:10:00

. /etc/profile.d/modules.sh
module load cuda openmpi

mpiexec -n 8 -npernode 4 -x LD_LIBRARY_PATH ./mm 2048 2048 2048
```

8 processes, 4 processes per node

To avoid scheduler's problem

Issues in Multiple GPUs per node

- f_node or h_node has multiple GPUs (4 or 2)
  - In default, all processes use “GPU 0” on the node → slow ☹

- Each process should use distinct GPUs

→ In mm.cu, cudaSetDevice(int dev) is called first
  - specifies the GPU to be used
  - dev: GPU number in the node (0, 1, 2…)
    - In this sample, GPU number (rank % num of devices) is used

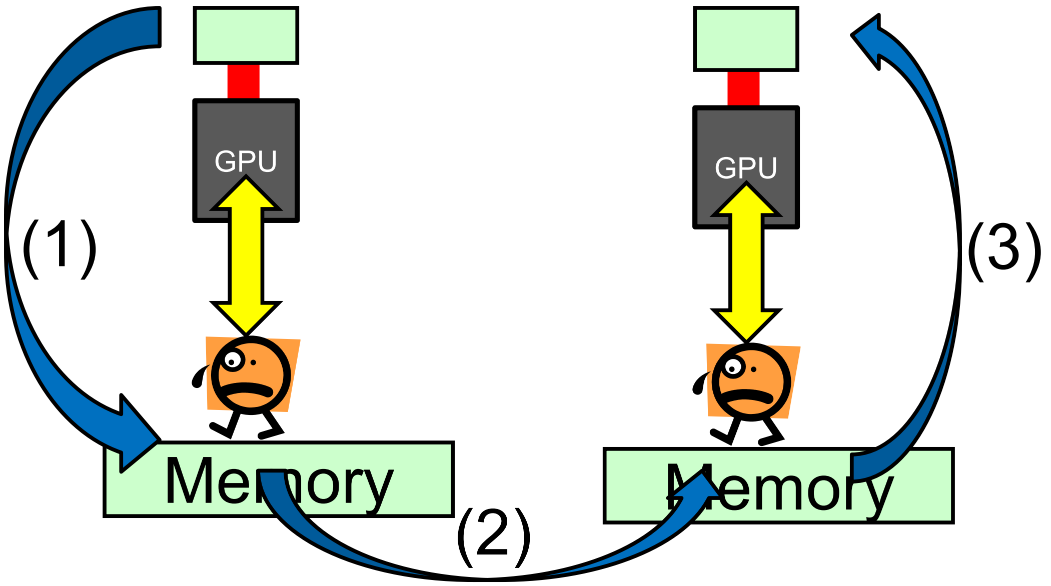
Data Transfer

- mm sample does not use communication
- If we want to do, the basic method is

  (1) Copy data on GPU memory to CPU (cudaMemcpy)

  (2) Transfer between processes (MPI_Send/MPI_Recv)

  (3) Copy data on CPU memory to GPU (cudaMemcpy)



NOTE:
Recent MPI supports
GPU direct,
to direct communication
on GPU memory

We Have Learned

- Part 1: Shared memory parallel programming with OpenMP
- Part 2: GPU programming with OpenACC and CUDA
- Part 3: Distributed memory parallel programming with MPI

Many common strategies towards faster software:

- To understand source of bottleneck
- Reducing computation and communication
- Overlapping computation and communication
- To understand property of architecture

Assignments in MPI Part (Abstract)

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

Due date: June 9 (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 MPI (1) slides on May 19

- Thank you for participating in practical parallel computing

Let’s enjoy high performance computing!