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

Part 3: MPI

No 1: MPI Introduction

May 18, 2023

Toshio Endo

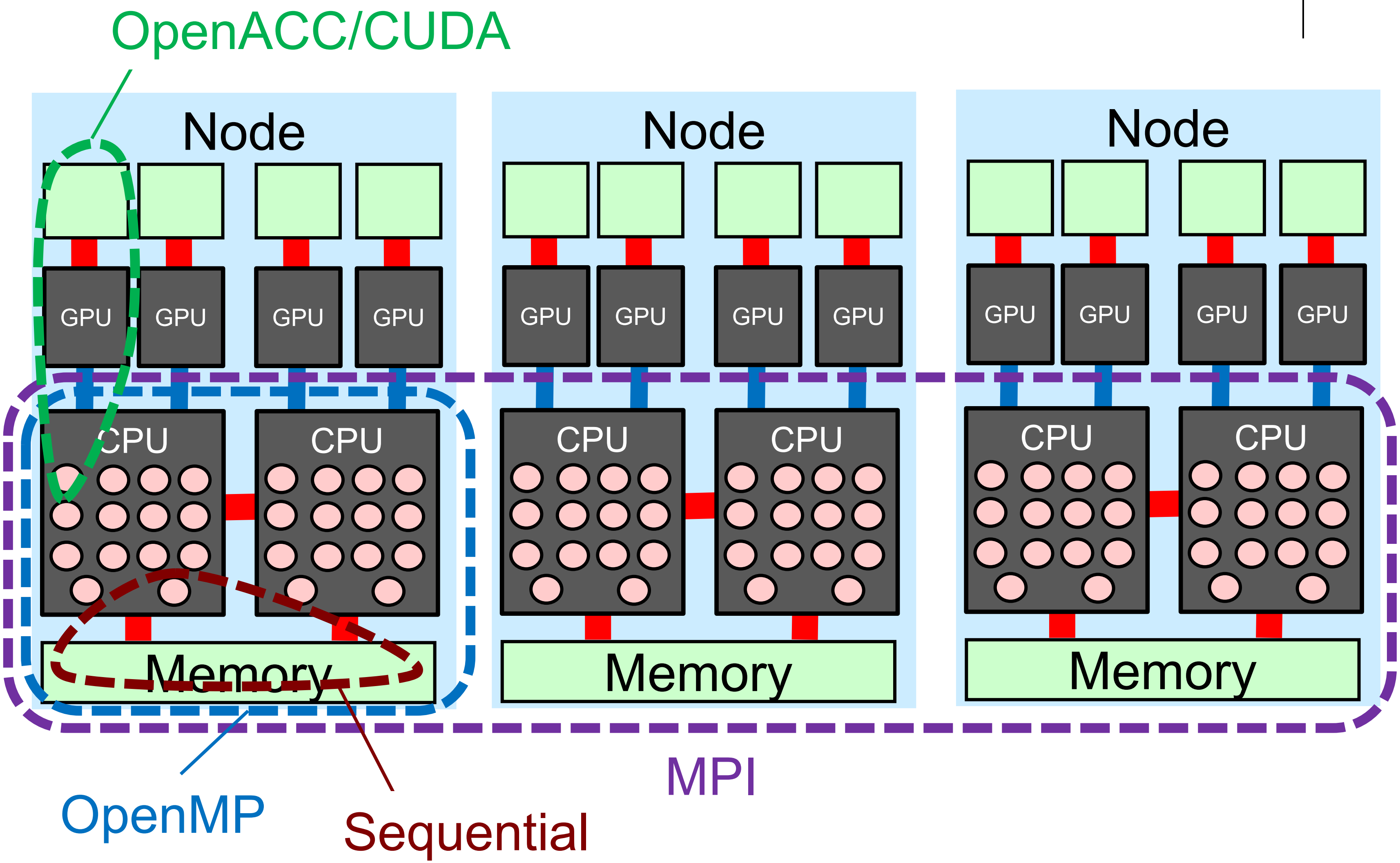
School of Computing & GSIC

endo@is.titech.ac.jp

Overview of This Course

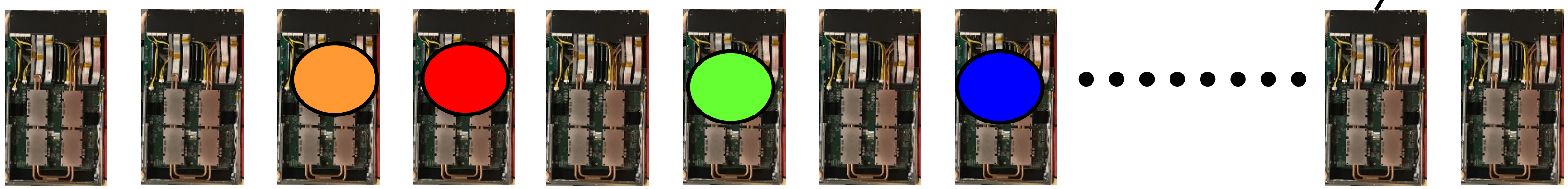
- Part 0: Introduction
  - 2 classes
- Part 1: OpenMP for shared memory programming
  - 4 classes
- Part 2: GPU programming
  - 4 classes ← We are here (1/4)
  - OpenACC (1.5 classes) and CUDA (2.5 classes)
- Part 3: MPI for distributed memory programming
  - 4 classes ← We are here (1/4)

Parallel Programming Methods on TSUBAME



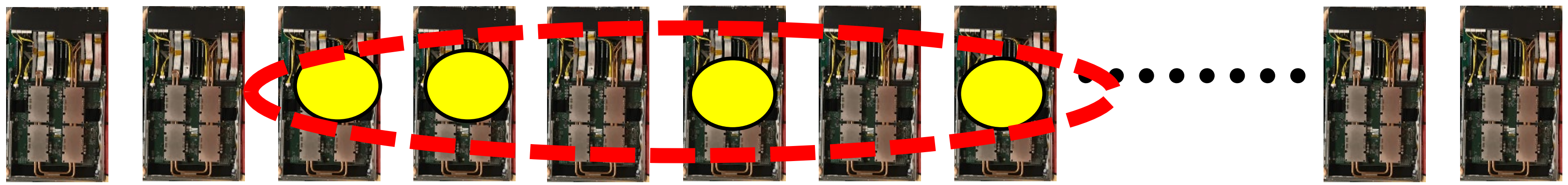
How Multiple Nodes are Used

1. Submit several jobs into job scheduler
   - cf) Program executions with different parameters → Parameter Sweep
   - Jobs are dependent, and no cooperation

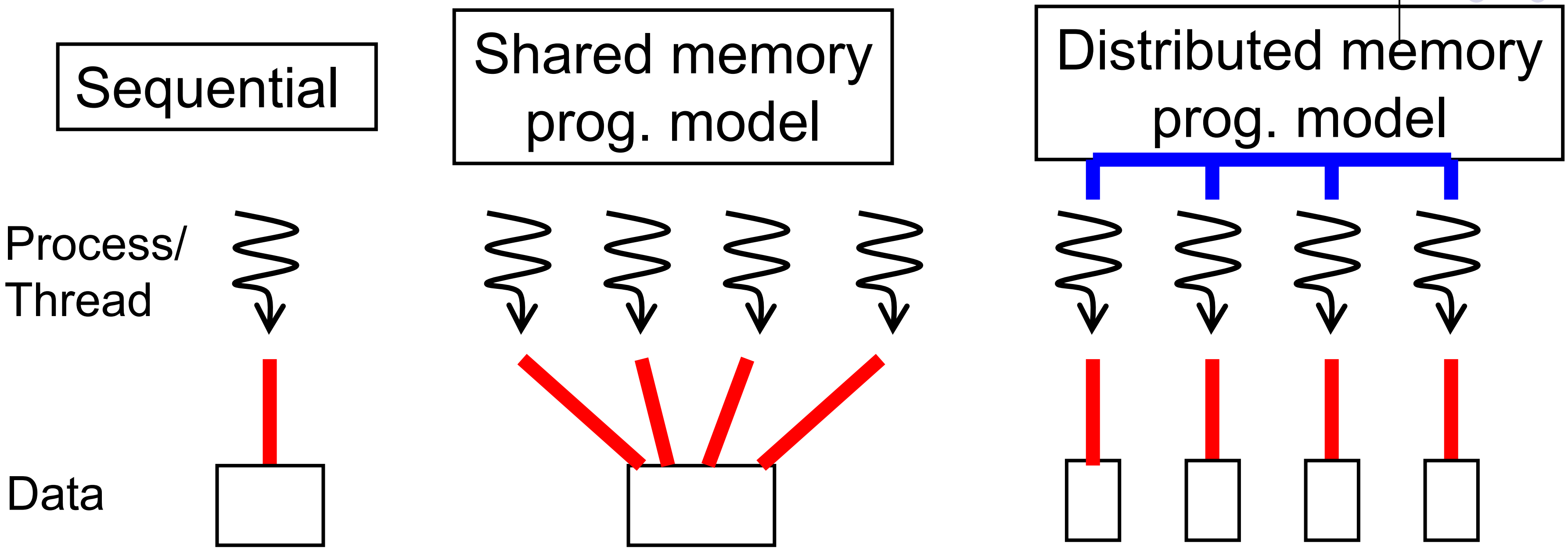


2. Use distributed memory programming → A single job can use multiple nodes
   - Socket programming, Hadoop, Spark…
   - And MPI

Part 3



Classification of Parallel Programming Models



Programming without parallelsim

Threads have access to shared data

- OpenMP
- pthread
- Java thread…

Need communication among processes

- MPI
- socket
- Hadoop, Spark…

MPI (message-passing interface)

- Parallel programming interface based on distributed memory model
- Used by C, C++, Fortran programs
  - Programs call MPI library functions, for message passing etc.
- There are several MPI libraries
  - OpenMPI (default) ← OpenMPI ≠ OpenMP ☹
  - Intel MPI, SGI MPE, MVAPICH, MPICH…

Differences from OpenMP

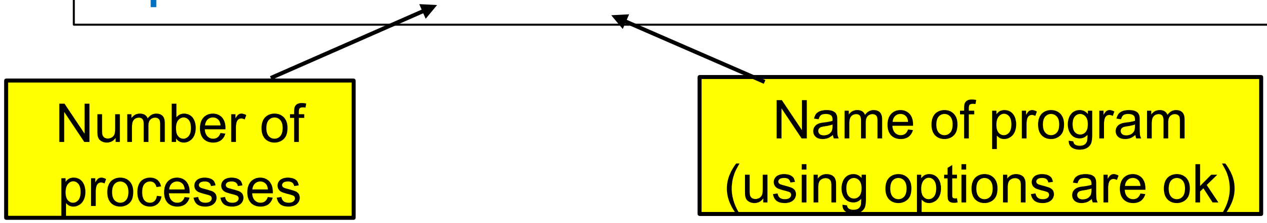
In MPI,

- An execution consists of multiple processes (not threads)
  - We can use multiple nodes ☺
  - The number of running processes is basically constant
- No variables are shared. Instead message passing is used
  - Data distribution has to be programmed
- No smart syntaxes such as “omp for” or “omp task” ☹
  - Task distribution has to be programmed ☹

First MPI Sample

- /gs/hs1/tga-ppcomp/23/hello-mpi

```
[make sure that you are at a interactive node (r7i7nX) ]
module load cuda openmpi     [Do once after login]
cd ~/t3workspace             [In web-only route]
cp -r /gs/hs1/tga-ppcomp/23/hello-mpi  .
cd hello-mpi
make
[An executable file “hello” is created]
mpiexec -n 7 ./hello
```



Number of processes

Name of program (using options are ok)

Compiling and Executing MPI Programs

Case of OpenMPI library on TSUBAME3.0

- To compile
  - module load cuda openmpi, and then use mpicc
  - For sample programs, “make” command works

Required for module dependency☹

- To execute
  - mpiexec -n 7./hello

Number of processes

↑ These methods uses 1 (current) node.

When We Want to Use More Processes than CPU Cores

- An interactive node has 7 CPU cores (= 14 Hyperthreads)
- If number of processes is larger than CPU cores, we need "--oversubscribe" option of OpenMPI

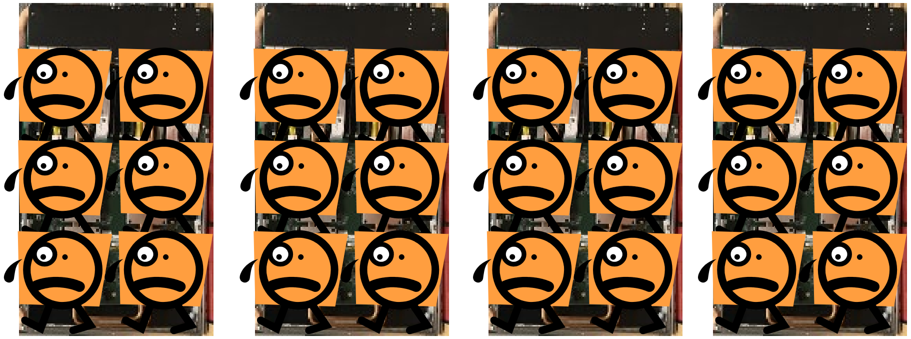
  - mpiexec -n 7./hello
  - mpiexec -n 14 --oversubscribe ./hello

Needed when we use processes >physical cores (7, in an interactive node)

From Single Node To Multiple Nodes

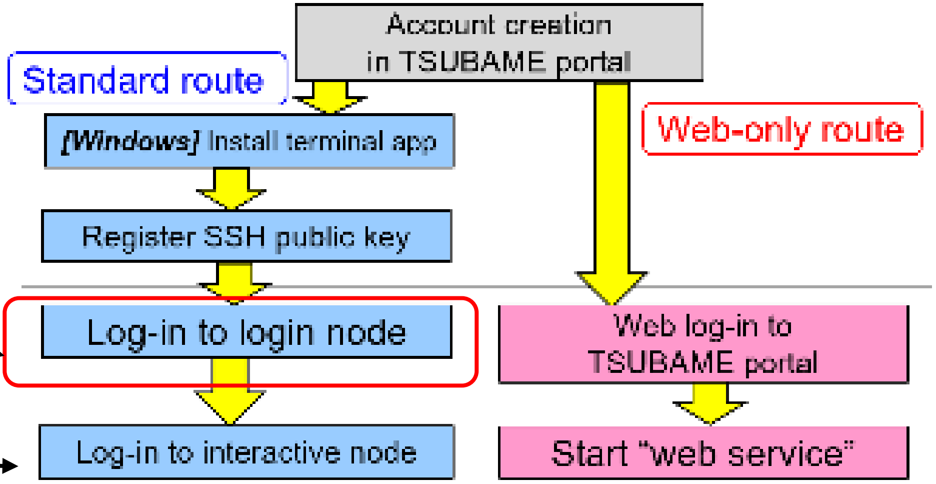
- So far, we used a single node
  - An interactive node with 7 cores = 14 Hyperthreads
  - For reports, a single node is enough
- MPI can support multiple nodes!

To use multiple nodes on TSUBAME3.0, we need "job submission" !



Notes on Job Submission in "Standard route"

- On an interactive node via "standard route", qsub/qstat commands may not be found
- Please use qsub/qstat on a login node
  - (out of iqrsh)



Account creation in TSUBAME portal

Standard route

[Windows] Install terminal app

Register SSH public key

Log-in to login node

Log-in to interactive node

Web-only route

Web log-in to TSUBAME portal

Start "web service"

Use qsub @login0 or @login1

qsub fails @r7i7nX

Submit an MPI Job
(case of OpenMPI)

- We are going to execute a job using

  4 processes × 2 nodes = 8 processes

(1) Make a script file: job.sh

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

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

mpiexec -n 8 -npernode 4 ./hello
```

4core node x 2

Module preparation

Number of processes

Number of processes per node

Program name (and option)

(2) Submit the job with “qsub”

```
qsub job.sh
```

↑≦0:10:00 and ≦2nodes

```
qsub -g tga-ppcomp job.sh
```

↑ >0:10:00 or ≧3nodes

Be careful for TSUBAME point

Notes in This Lecture

(also shown in 1-2 slides)

- First, please consider usage of interactive node (web usage/iqrsh)
- まずはインタラクティブノードの利用を検討してください (web usage/iqrsh)

- If necessary for reports, you can use up to 18,000 points in total per student. For more, please ask Endo
- 本講義のレポートの作成の目的で、一人あたり合計で18,000ポイントまで利用を認めます。より必要な場合は遠藤へ相談を
  - 18,000 points ≒ F_node x 5 hours
  - You can check point consumption on TSUBAME portal

- The TSUBAME group name is tga-ppcomp

Users need to follow the rules at www.t3.gsic.titech.ac.jp

利用時には www.t3.gsic.titech.ac.jp に示される規則を守る必要があります

Resource Types on TSUBAME3.0

- Choose one of resource types (number of cores, mainly)
  - It is like “instance types” in cloud systems
  - Please specify “proper” one

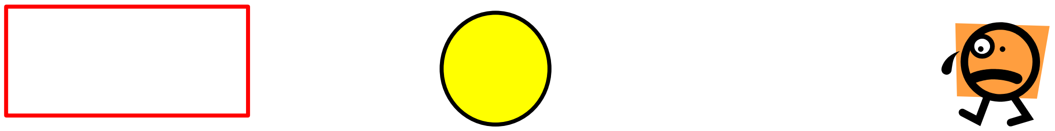
Resource type name	Physical CPU cores	Memory (GB)	GPUs
f_node	28	240	4
h_node	14	120	2
q_node	7	60	1
q_core	4	30	0
s_core	1	7.5	0
s_gpu	2	15	1

← Same as an interactive node

#$ -l [resource_type] = [Number]

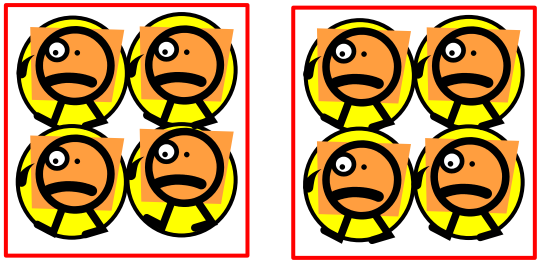
Example:

#$ -l s_core=1 ← The minimum resource allocation



Nodes, Cores, MPI Processes

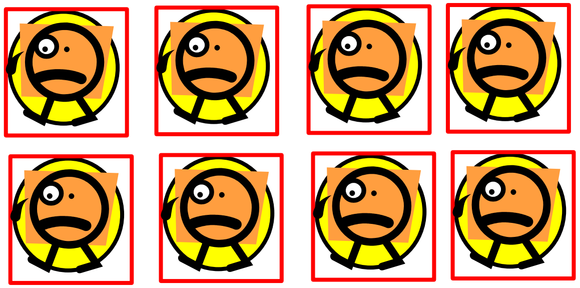
```
        :
#$ -l q_core=2
        :
mpirun -n 8 -npernode 4
  …
```



2 (virtual) nodes are prepared
Each node has 4 cores (q_core)

4 processes are created per node. Totally 8 are created
→ 2 nodes are used

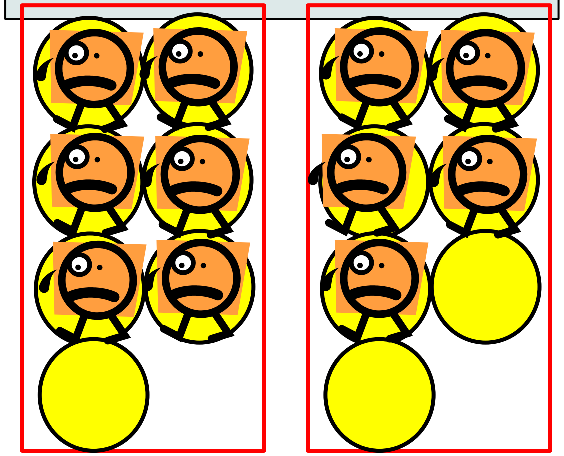
```
        :
#$ -l s_core=8
        :
mpirun -n 8 -npernode 1
  …
```



8 (virtual) nodes are prepared
Each node has 1 cores (s_core)

1 processes are created per node. Totally 8 are created
→ 8 nodes are used

```
        :
#$ -l q_node=2
        :
mpirun -n 11 -npernode 6
  …
```



2 (virtual) nodes are prepared
Each node has 7 cores (q_node)

6 processes are created per node. Totally 11 are created
→ 2 nodes are used
(There are idle cores)

An MPI Program Looks Like

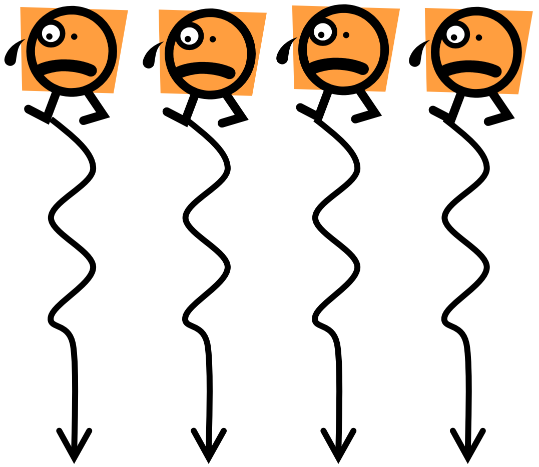
```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
   MPI_Init(&argc, &argv);← Initialize MPI

   (Computation/communication)

   MPI_Finalize();        ← Finalize MPI
}
```

If number of processes=4

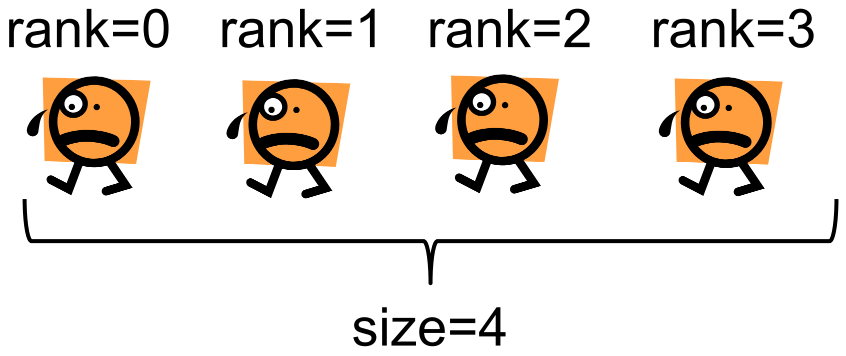


ID of Each MPI Process

- Each process has its ID (0, 1, 2…), called rank
  - `MPI_Comm_rank(MPI_COMM_WORLD, &rank);`

  → Get its rank
  - `MPI_Comm_size(MPI_COMM_WORLD, &size);`

  → Get the number of total processes
  - 0 ≦ rank < size
  - The rank is used as target of message passing



rank=0 rank=1 rank=2 rank=3

size=4

“mm” sample: Matrix Multiply

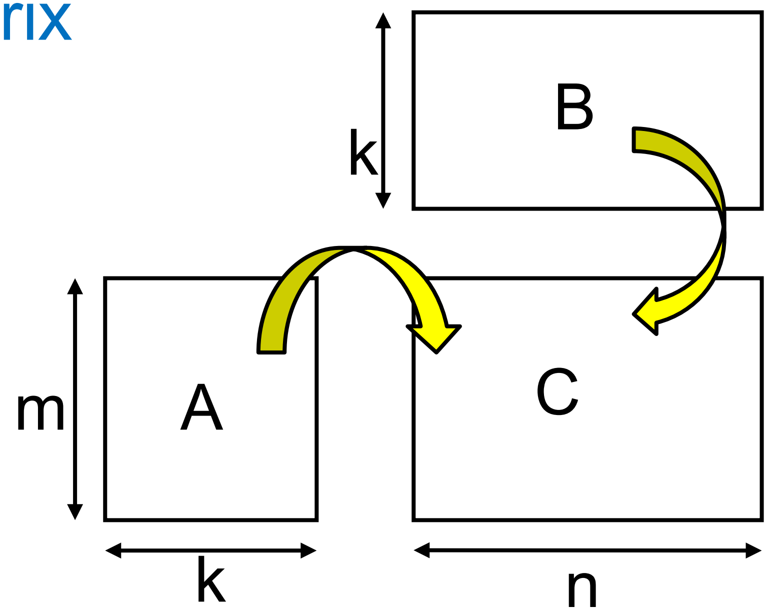
MPI version available at /gs/hs1/tga-ppcomp/23/mm-mpi/

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



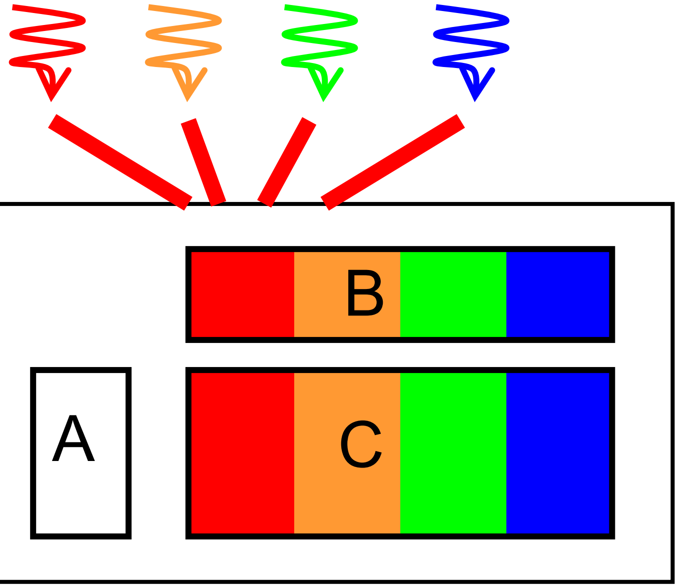
Execution:

```
mpiexec -n [np] ./mm [m] [n] [k]  (interactive)
mpiexec -n [np] -npernode [nn] ./mm [m] [n] [k]   (in job script)
```

Why Distributed Programming is More Difficult (case of mm-mpi)

Shared memory with OpenMP:

Programmers consider how computations are divided

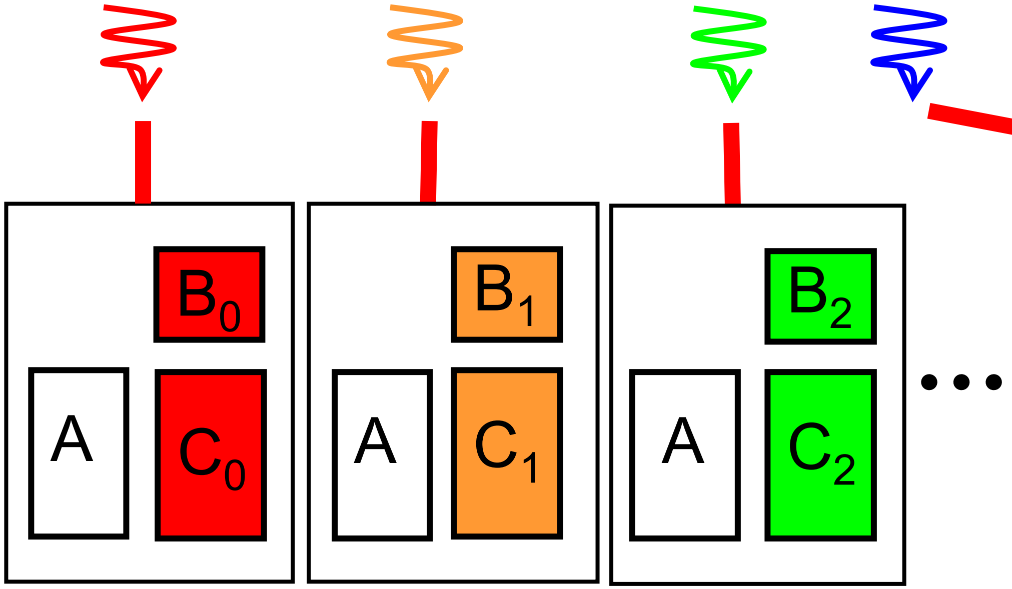


In this case, matrix A is accessed by all threads

→ Programmers do not have to know that

Distributed memory with MPI:

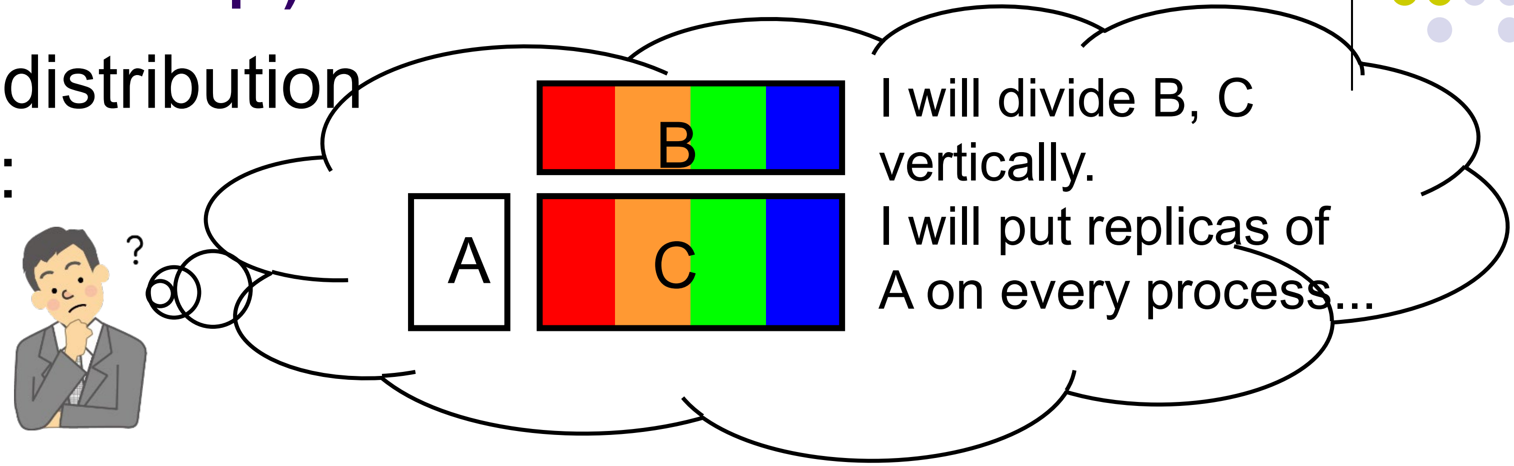
Programmers consider how data and computations are divided



Programmers have to design which data is accessed by each process

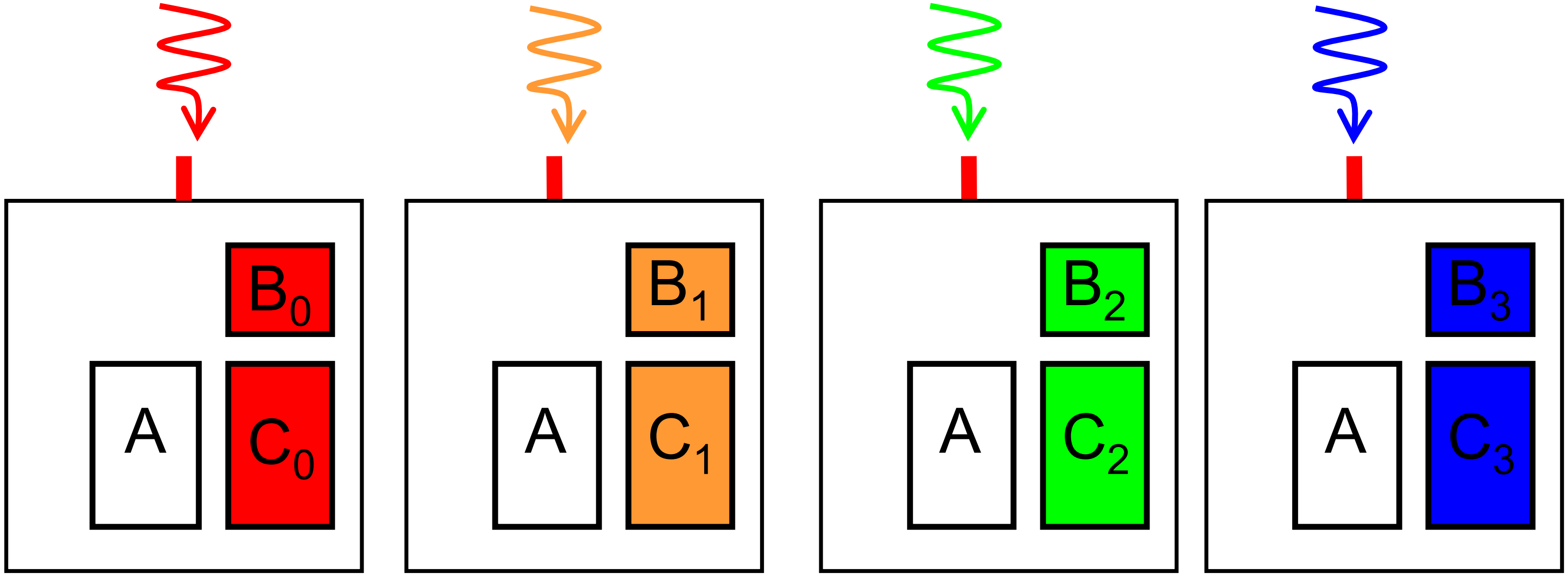
Programming Data Distribution
(case of mm-mpi)

Design distribution method:



I will divide B, C vertically.
I will put replicas of A on every process...

Programming actual location:

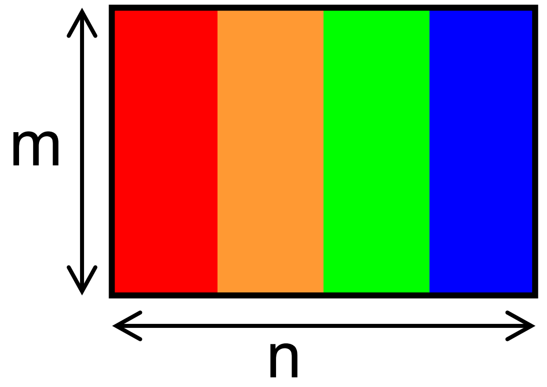


Programming Actual Data Distribution

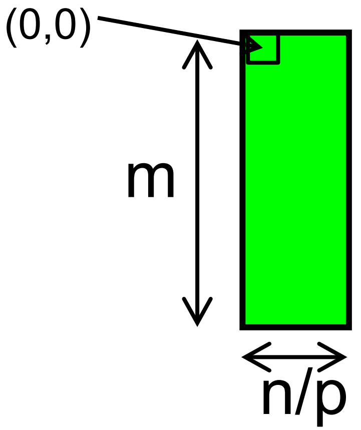
- We want to distribute a *m* × *n* matrix among *p* processes
  - We assume n is divisible by p
- Each process has a partial matrix of size m × (n/p)
  - We need to "malloc" m*(n/p)*sizeof(data-type) size
  - We need to be aware of relation between partial matrix and entire matrix
    local index
    - (i,j) element in partial matrix owned by Process r ⇔

      (i, n/p*r + j) element in entire matrix
      global index

Entire matrix



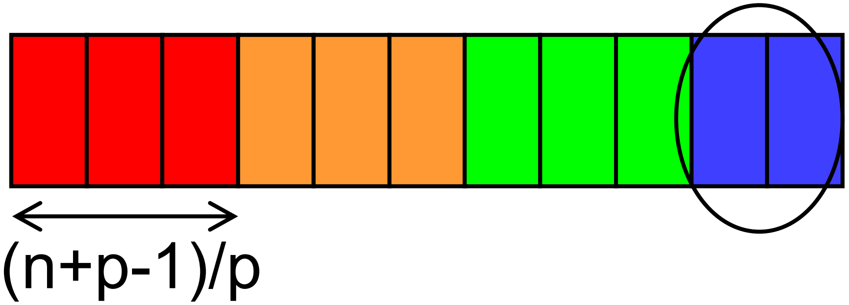
Actual matrix per process



Considering Indivisible Cases

- What if data size n is indivisible by p?
- We let n=11, p=4
  - How many data each process take?
  - n/p = 2 is not good (C division uses round down). Instead, we should use round up division
  - → (n+p-1)/p = 3 works well

Note that the "final" process takes less than others



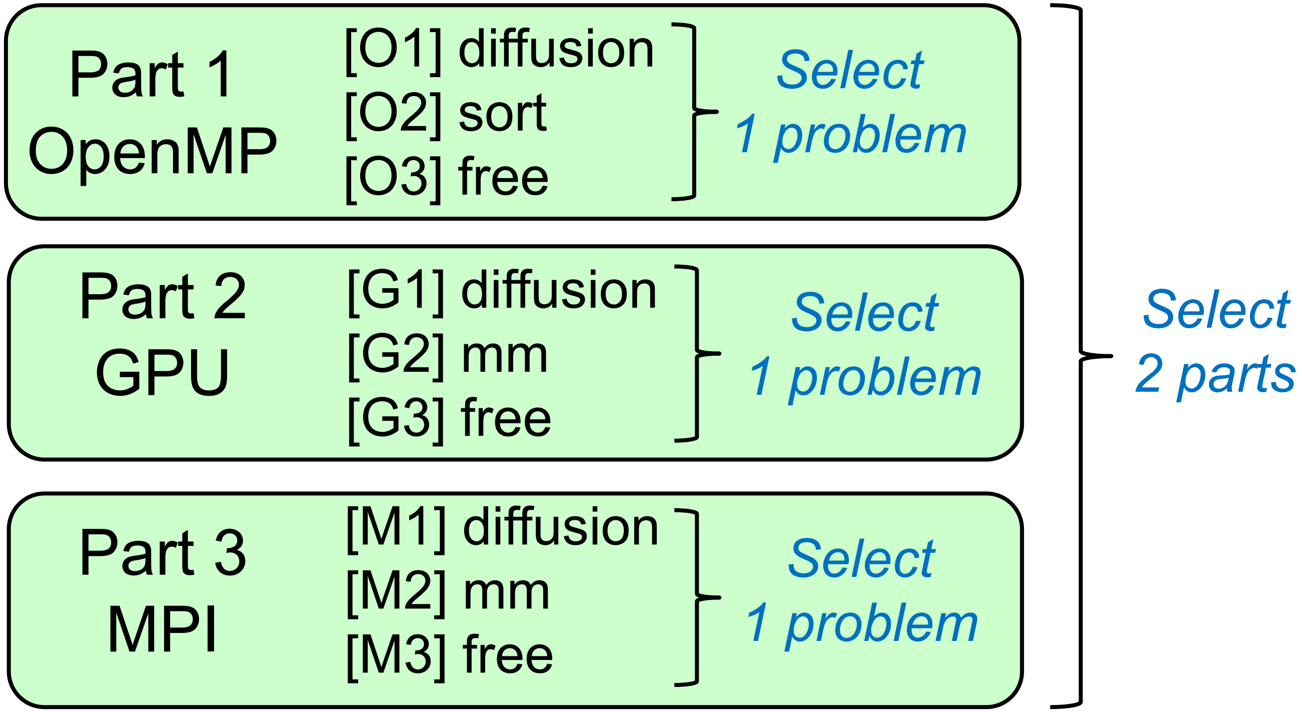
See divide_length() function in mm-mpi/mm.c
It calculates the range the process should take
→ Outputs are first index s and last index e

Notes in Time Measurement

- In mm-mpi, gettimeofday() is used for time measurement
- For accurate measurement, we should call MPI_Barrier(MPI_COMM_WORLD) before measurement
  - This synchronizes all processes
  - All processes need to call this

Assignments in this Course

- There is homework for each part. Submissions of reports for 2 parts are required



Part 1
OpenMP

[O1] diffusion
[O2] sort
[O3] free

Select 1 problem

Part 2
GPU

[G1] diffusion
[G2] mm
[G3] free

Select 1 problem

Part 3
MPI

[M1] diffusion
[M2] mm
[M3] free

Select 1 problem

Select 2 parts

Assignments in MPI Part (1)

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

Due date: June 12 (Monday)

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

- Do not forget to change Makefile and job.sh appropriately
- Use deadlock-free communication
  - see neicomm_safe() in neicomm-mpi sample

Optional :

- To make array sizes (NX, NY) variable parameters
- To consider the case with NY is indivisible by p
  - see divide_length() in mm_mpi sample
- To improve performance further. Blocking, 2D division, etc

Assignments in MPI Part(2)

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

Optional:

- To consider indivisible cases
- To try advanced algorithms, such as SUMMA
  - the paper “*SUMMA: Scalable Universal Matrix Multiplication Algorithm*” by Van de Geijn
  - http://www.netlib.org/lapack/lawnspdf/lawn96.pdf

Assignments in MPI Part (3)

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

- cf) A problem related to your research
- More challenging one for parallelization is better
  - cf) Partial computations have dependency with each other

Notes in Report Submission (1)

- Submit the followings via T2SCHOLA

  (1) A report document

  - PDF, MS-Word or text file
  - 2 pages or more
  - in English or Japanese (日本語もok)

  (2) Source code files of your program

  - Try “zip” to submit multiple files

Notes in Report Submission (2)

The report document should include:

- Which problem you have chosen
- How you parallelized
  - It is even better if you mention efforts for high performance or new functions
- Performance evaluation on TSUBAME
  - With varying number of processes
    - On an interactive node,
      - 1 ≦ P ≦ 7 (P=number of processes)
      - If you try P > 7, use "mpiexec -n [P] --oversubscribe …"
    - To use more CPU cores, you need to do "job submission" (optional)
  - With varying problem sizes
  - Discussion with your findings
  - Other machines than TSUBAME are ok, if available

Next Class

- MPI (2)
  - Basic message passing
  - How to parallelize diffusion sample with MPI
    - Related to [M1]

- Planned schedule
  - May 22: Part 3 (2)
  - May 25: cancelled (休講) & Due for Part2 assignment
  - May 29: Part 3 (3)
  - June 1: Part 3 (4)