

2019

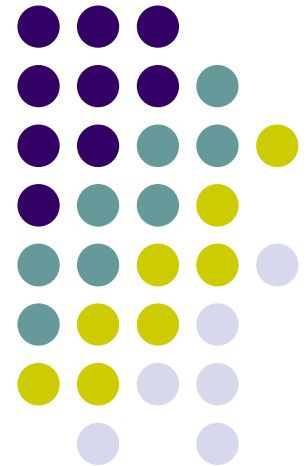
Practical Parallel Computing (実践的並列コンピューティング) No. 7

Distributed Memory Parallel
Programming with MPI (1)

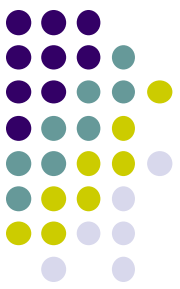
Toshio Endo

School of Computing & GSIC

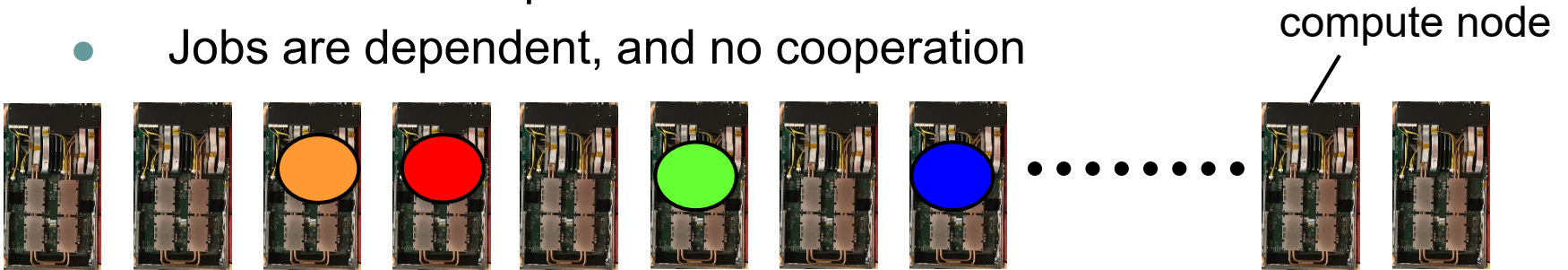
endo@is.titech.ac.jp



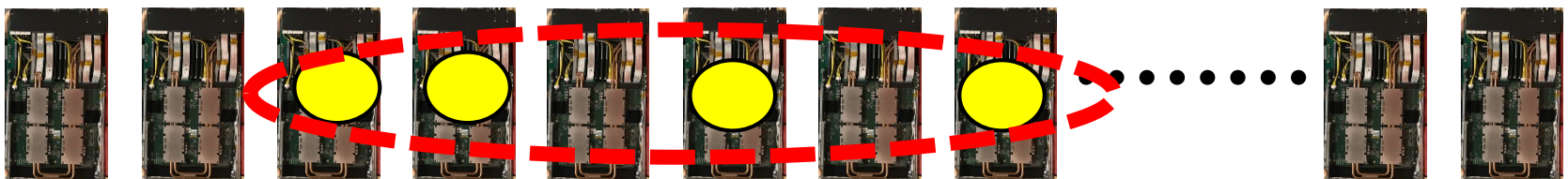
How to Use Many Nodes in Supercomputers



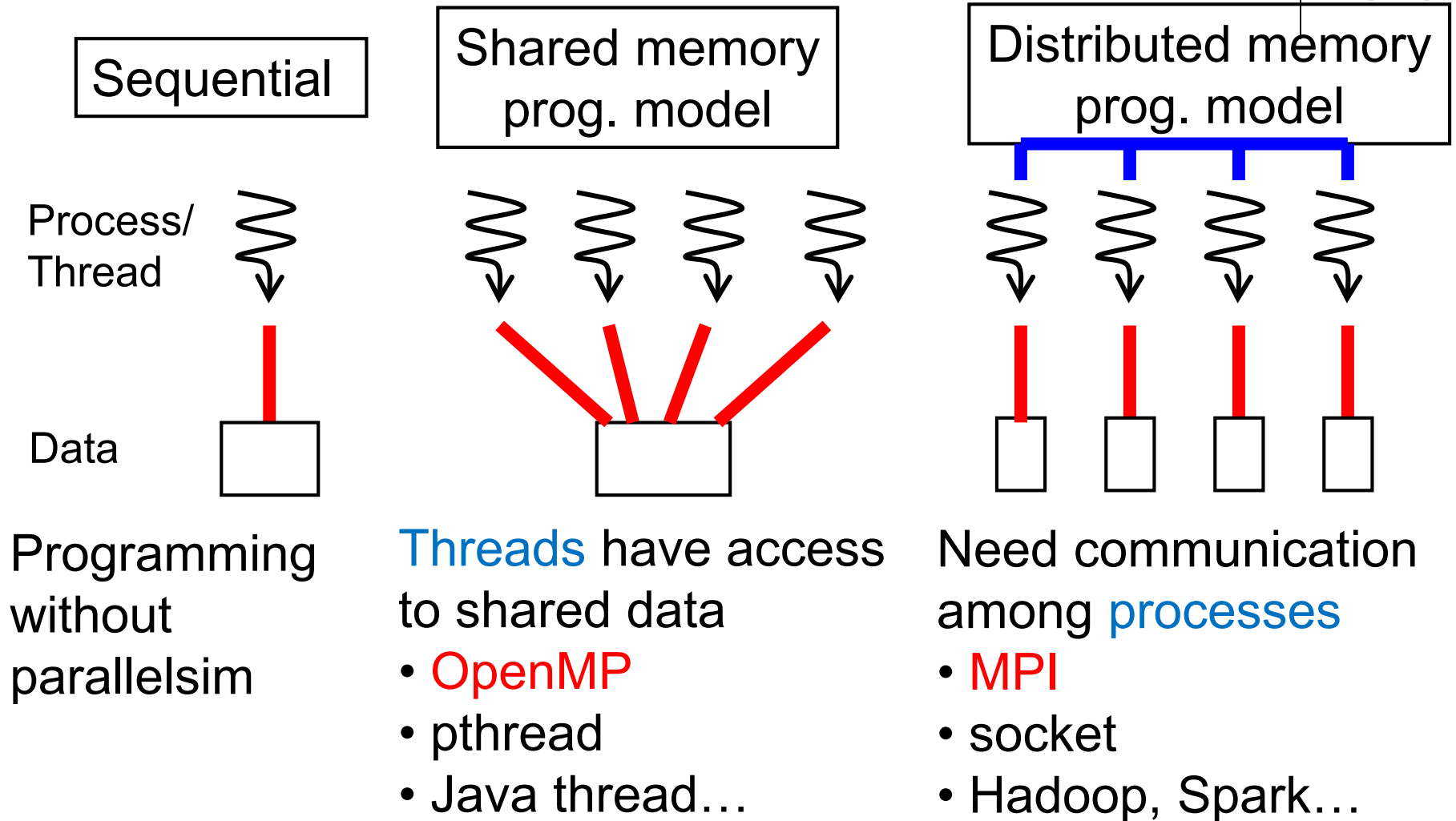
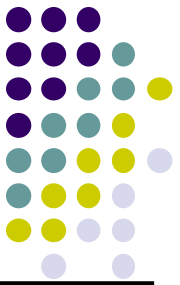
1. Submit several jobs into job scheduler
 - cf) Program executions with different parameters → Parameter Sweep
 - Jobs are dependent, and no cooperation



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



Classification of Parallel Programming Models



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 😞

Sample MPI Programs on TSUBAME (case of OpenMPI)



Samples at [~endo-t-ac/ppcomp/19/mpitest/](https://endo-t-ac/ppcomp/19/mpitest/)
[~endo-t-ac/ppcomp/19/mm-mpi/](https://endo-t-ac/ppcomp/19/mm-mpi/) on TSUBAME
Please copy them to your directory as usual

- Preparation for MPI environment
 - `module load cuda openmpi`
 ↖ `for module dependency` ☹

Now you can use `mpicc` command, until you log-out from TSUBAME

- MPI programs are compiled with `mpicc` command
 - In sample directories, “make” command will be ok
- Execution ↖ `Number of processes`
 - `mpirun -n 2 ./mpitest`
 - Executed on login nodes. Please use `qsub` usually



Submit an MPI Job (case of OpenMPI)

- Here program name is “a.out”. We are going to execute it with 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

mpirun -n 8 -npnode 4 ./a.out a b
```

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`

`qsub -g tga-ppcomp job.sh`
(if you use the group)

Notes on Job Submission

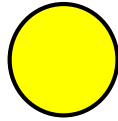


- Please specify **maximum run time (h_rt)** properly
 - If h_rt is larger than 0:10:00, you need to specify “TSUBAME group name” (charged/有料)
`qsub -g tga-ppcomp job.sh`
 - Use tga-ppcomp group only for this lecture
(tga-ppcompグループは、本授業の課題とそのテスト専用に使ってください)
- Without TSUBAME group, you can only use ≤ 2 nodes
(グループ無しの無料利用は2ノードまで)
 - If you use “-l f_node=2”, you can use ≤ 56 cores
 - If number of nodes > 2 , group name is required (and charged)

For the assignments:

- Please use ≤ 2 nodes, basically
- If you want, you can try ≤ 448 cores (-l f_node=16), but do not consume TSUBAME points too much

Nodes, Cores, MPI Processes



```

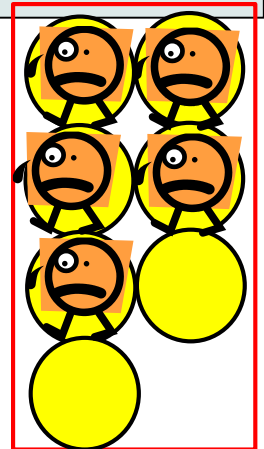
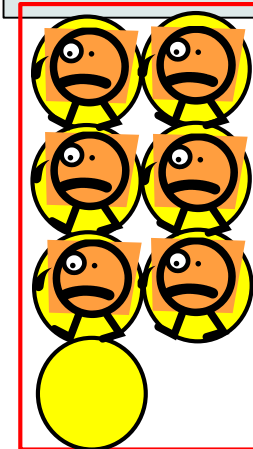
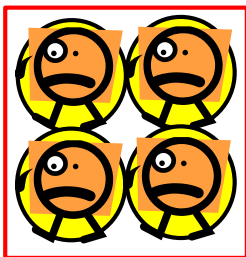
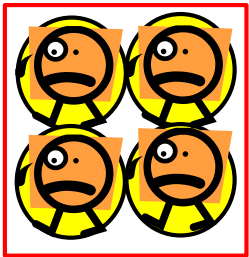
:
#$ -l q_core=2
:
mpirun -n 8 -npnode 4
...
    
```

```

:
#$ -l s_core=8
:
mpirun -n 8 -npnode 1
...
    
```

```

:
#$ -l q_node=2
:
mpirun -n 11 -npnode 6
...
    
```



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

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

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
```

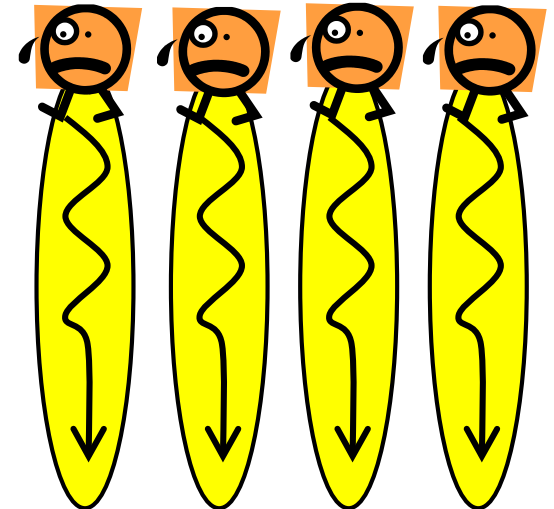
```
    (Computation/communication)
```

```
    MPI_Finalize();
```

```
}
```

← Finalize MPI

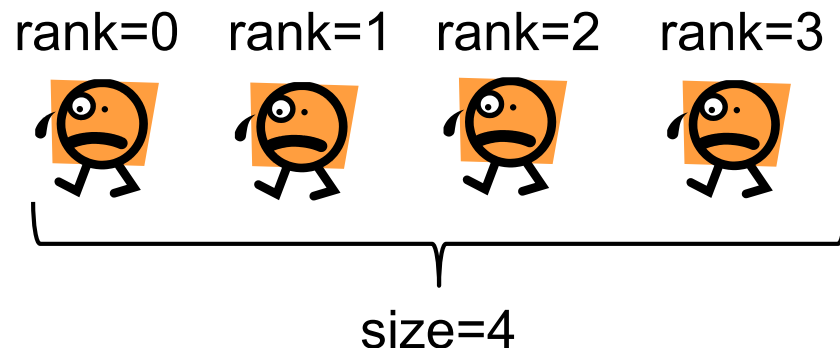
If number of
processes=4





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
 - $0 \leq \text{rank} < \text{size}$
 - The rank is used as target of message passing



“mm” sample: Matrix Multiply



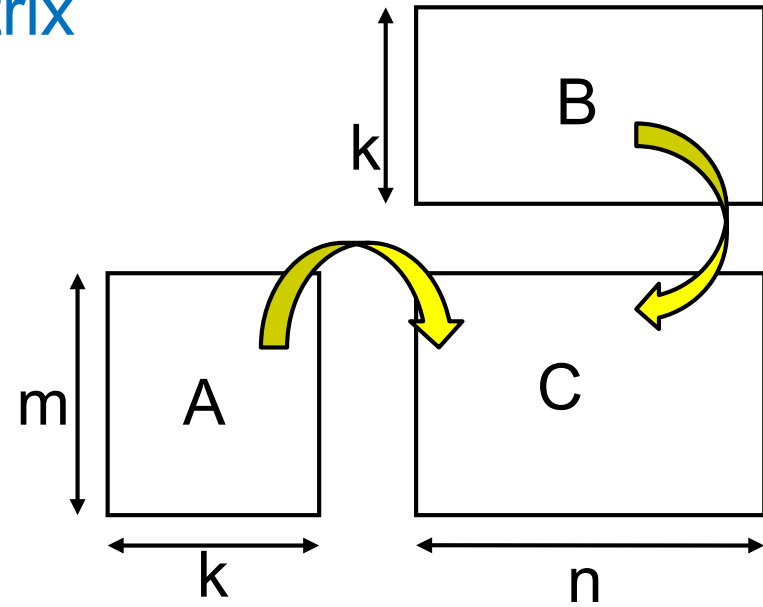
MPI version available at [~endo-t-ac/ppcomp/19/mm-mpi/](https://endo-t-ac/ppcomp/19/mm-mpi/)

A: a $(m \times k)$ matrix, B: a $(k \times n)$ matrix

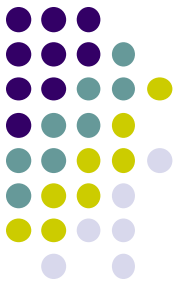
C: a $(m \times n)$ matrix

$$C \leftarrow A \times B$$

- 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 [np] -npernode [nn] ./mm [m] [n] [k]`

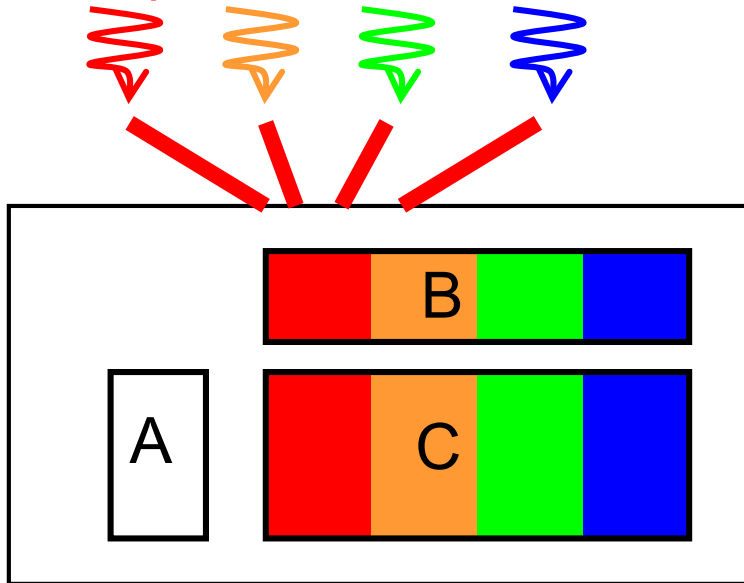


Why Distributed Programming is More Difficult (case of mm-mpi)



Shared memory with OpenMP:

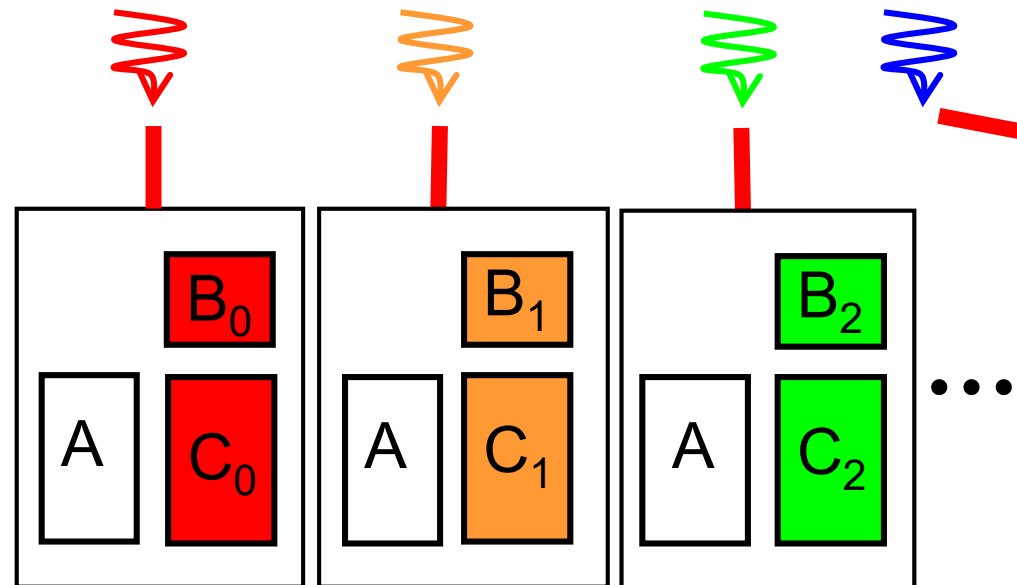
Programmers consider how **computations** are divided



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

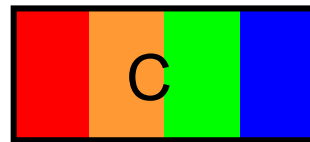


Programmers **have to design** which data is accessed by each process

Programming Data Distribution

(case of mm-mpi)

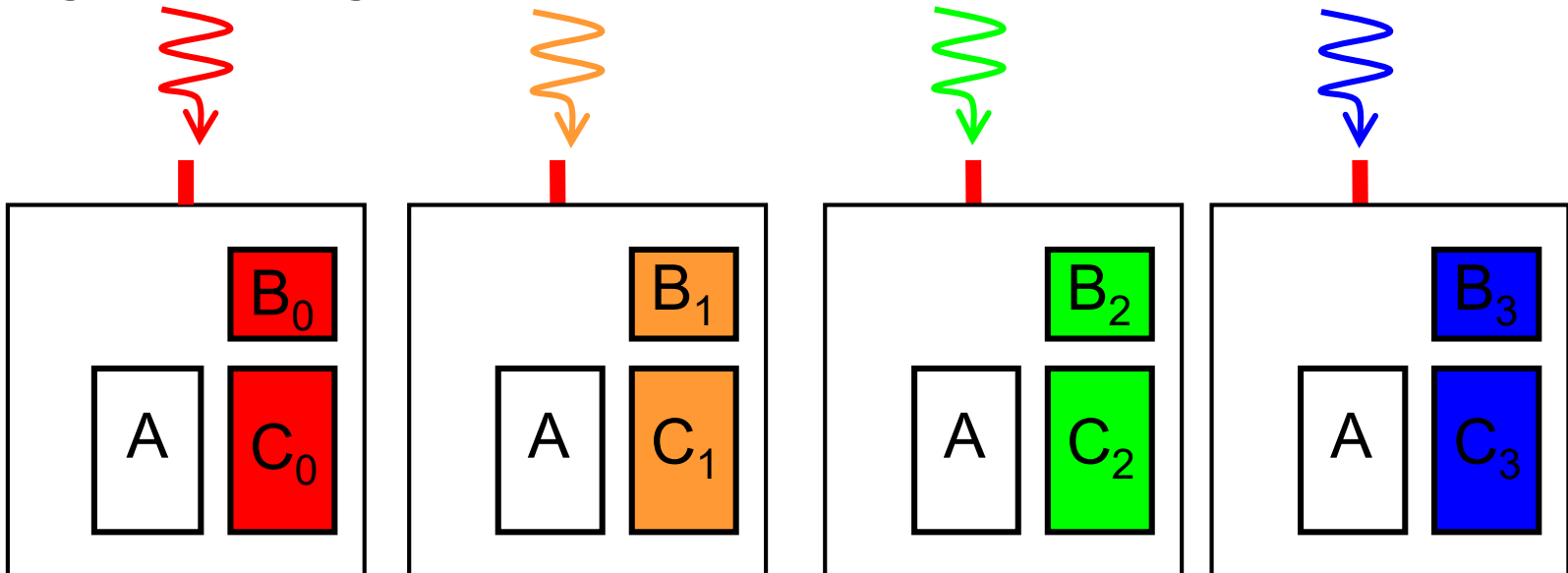
Design distribution method:



I will divide B, C vertically.

I will put replicas of A on every process...

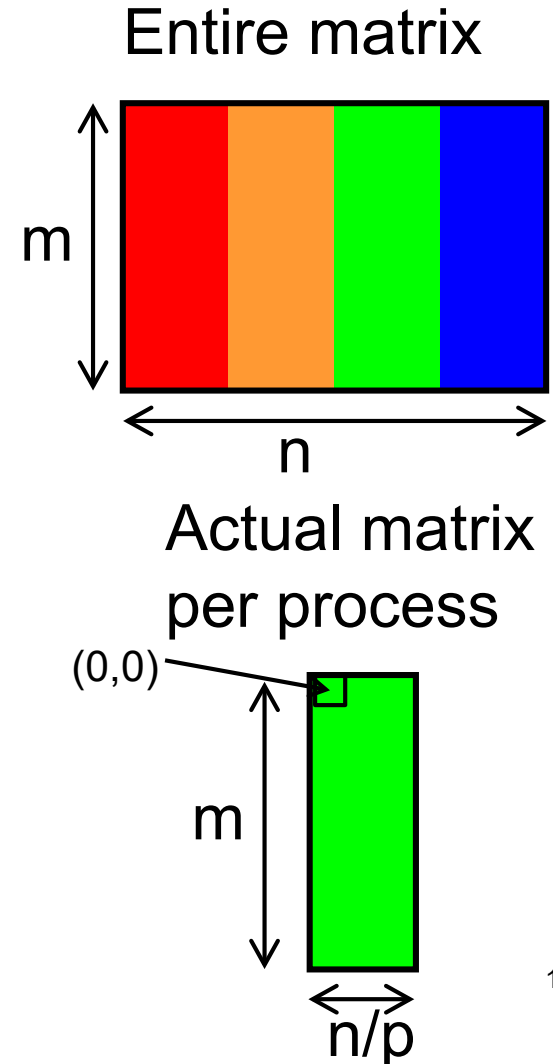
Programming actual location:



Programming Actual Data Distribution



- We want to distribute a $m \times n$ matrix among p processes
 - We assume n is divisible by p
- Each process has a partial matrix of size $m \times (n/p)$
 - We need to “malloc”
 $m \times (n/p) \times \text{sizeof}(\text{data-type})$ size
 - We need to be aware of relation between partial matrix and entire matrix
 - (i, j) element in partial matrix owned by Process $r \Leftrightarrow (i, n/p \times r + j)$ element in entire matrix

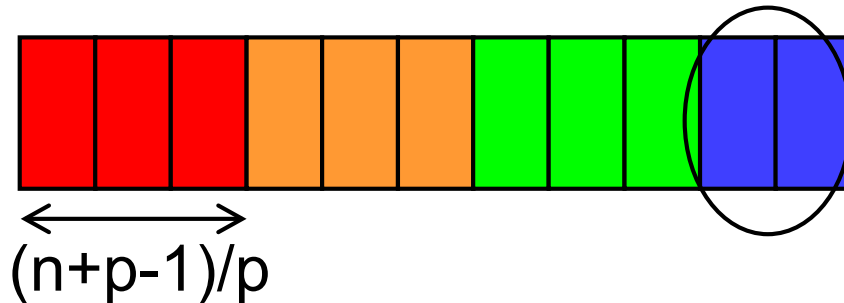




What is Done for Indivisible Cases

- 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

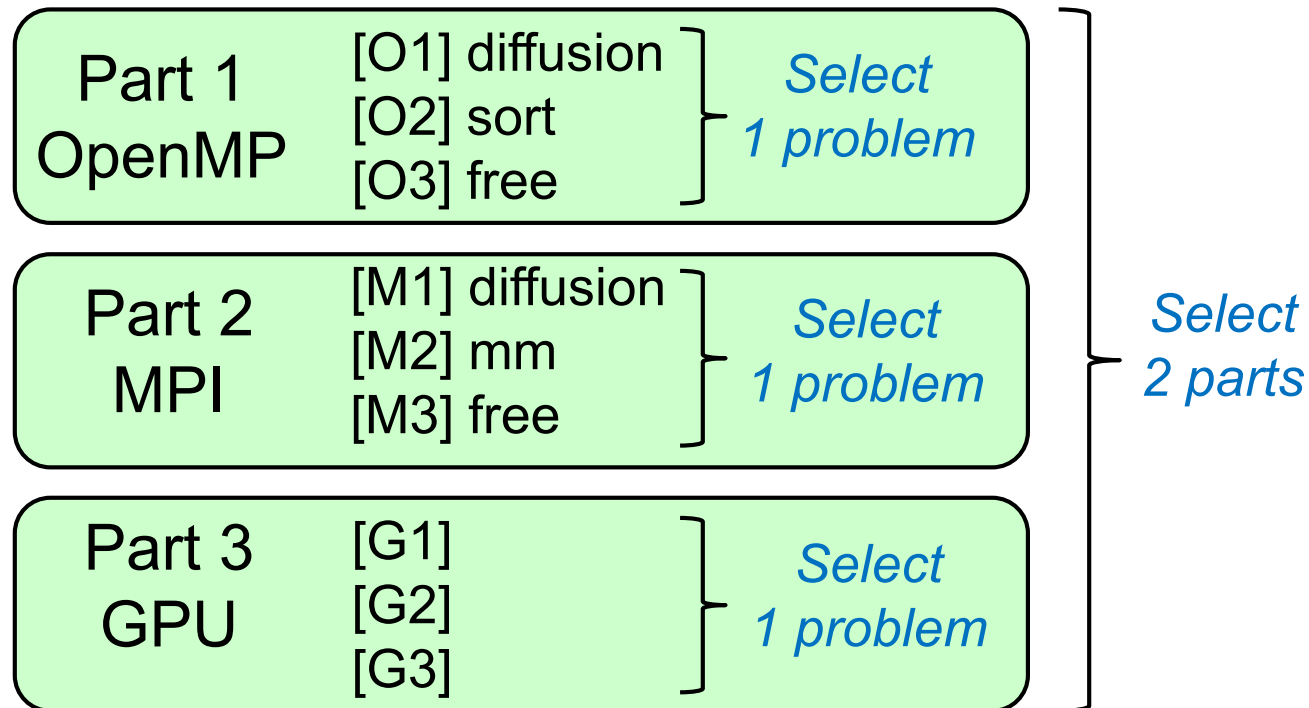


See `divide_length()` function in `mm-mpi/mm.c`
It calculates the range the process should take
(first index s and last index e)

Assignments in this Course



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





Assignments in MPI Part (1)

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

Due date: May 30 (Thursday)

[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` 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



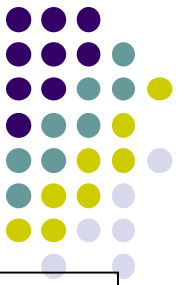
Notes in Submission

- Submit the followings via **OCW-i**
 - (1) **A report document**
 - A PDF or MS-Word file, 2 pages or more
 - in English or Japanese (日本語もok)
 - (2) **Source code files** of your program
 - If you use multiple files, you can use “.zip” or “.tgz”
- Report 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 processor cores
 - With varying problem sizes (if possible)
 - Discussion with your findings
 - Other machines than TSUBAME are ok, if available



Next Class: on **May 9 (Thu)**

- MPI (2)
 - How to parallelize diffusion sample with MPI



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