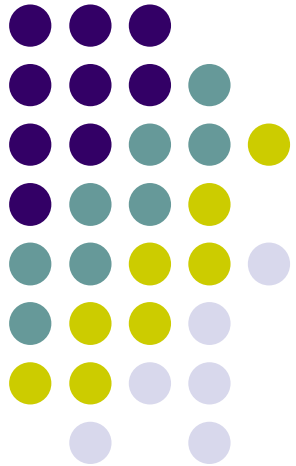
Practical Parallel Computing (実践的並列コンピューティング)



Part 3: MPI

No 1: MPI Introduction

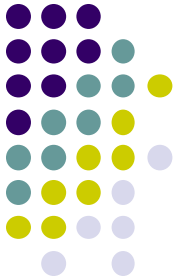
May 23, 2024

Toshio Endo

School of Computing & GSIC

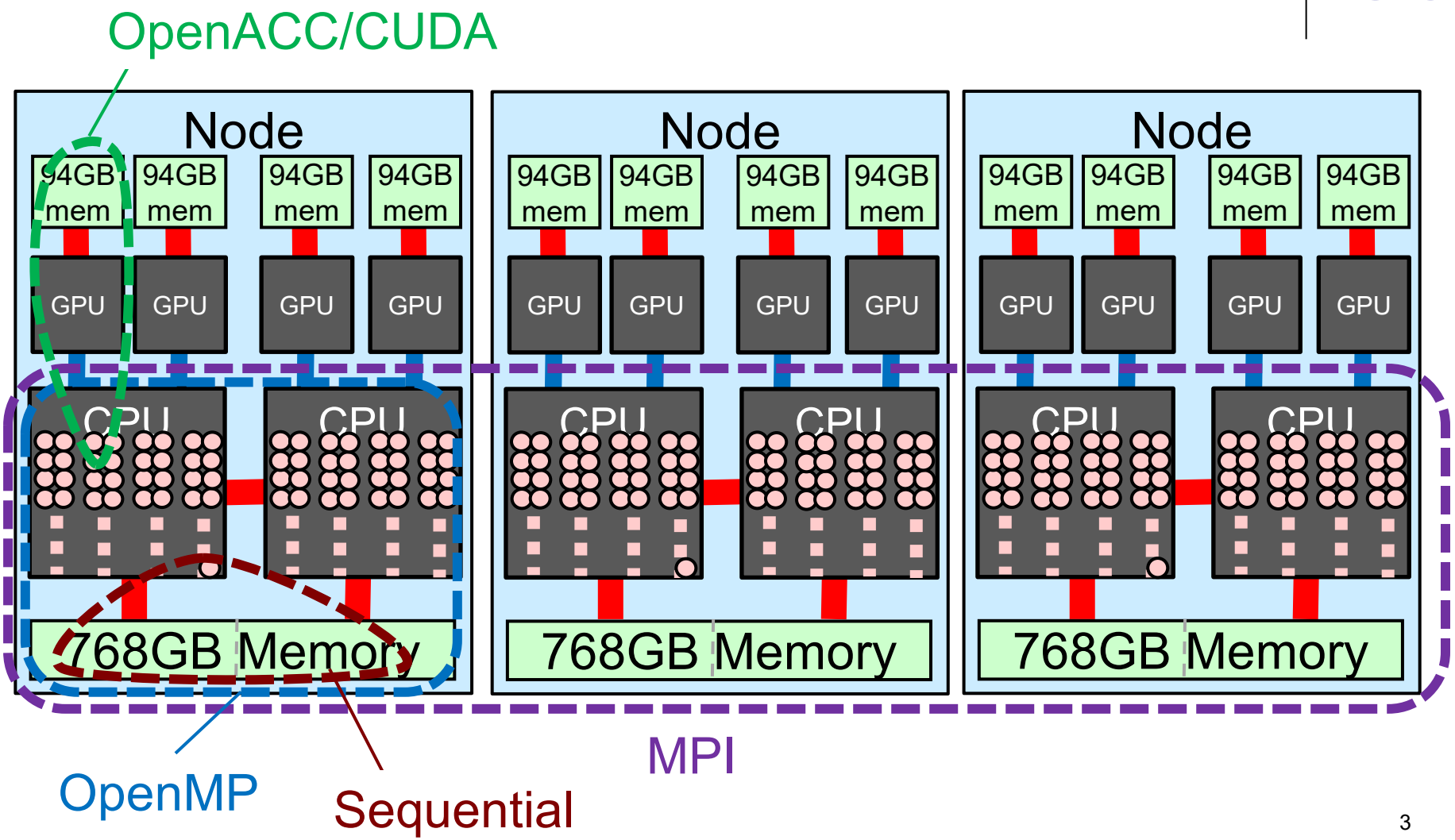
endo@is.titech.ac.jp

Overview of This Course

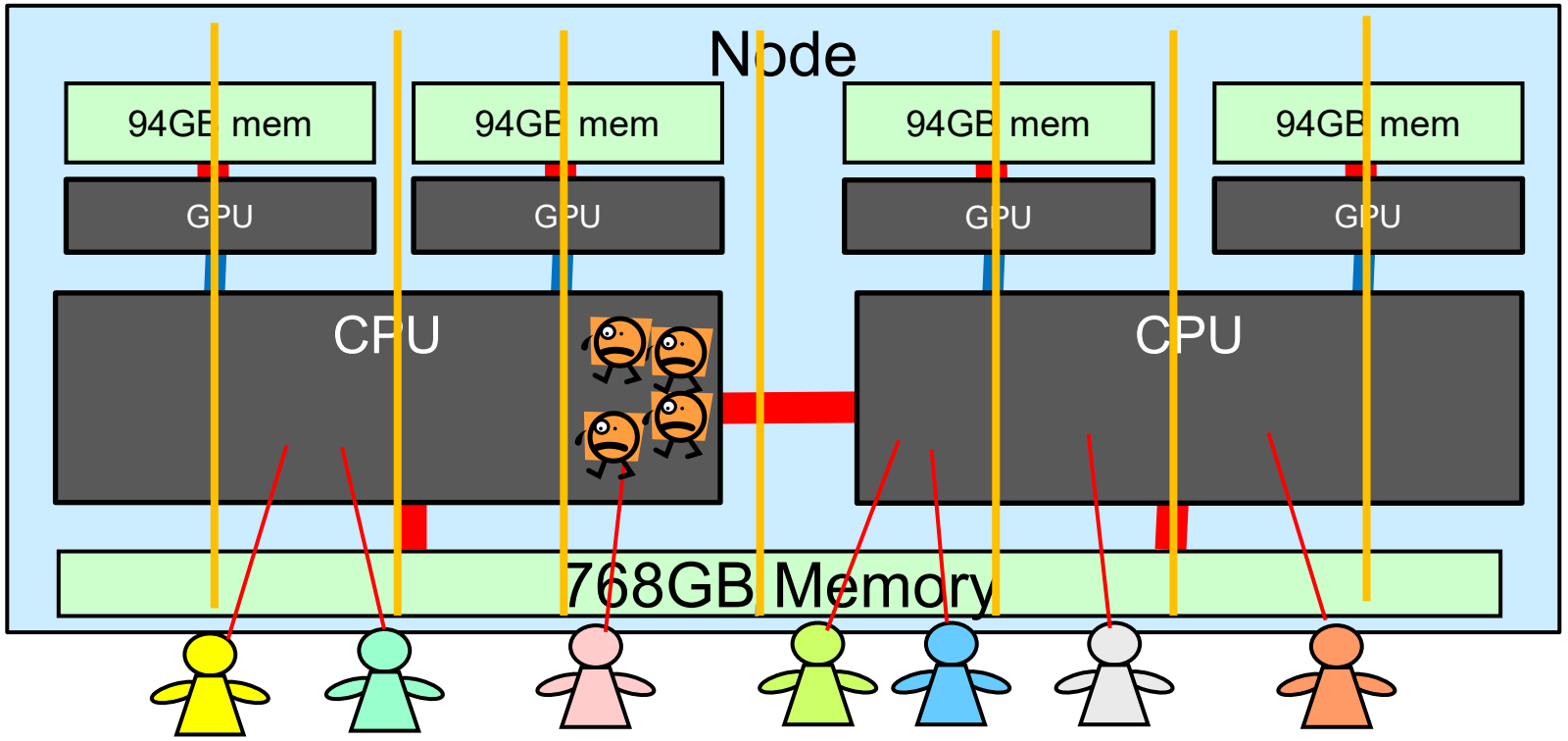


- Part 0: Introduction
  - 2 classes
- Part 1: OpenMP for shared memory programming
  - 4 classes
- Part 2: GPU programming
  - 4 classes ← We are here (1/4)
  - OpenACC (1.5 classes) and CUDA (2.5 classes)
- Part 3: MPI for distributed memory programming
  - 4 classes ← We are here (1/4)

Parallel Programming Methods on TSUBAME



On TSUBAME Interactive Node



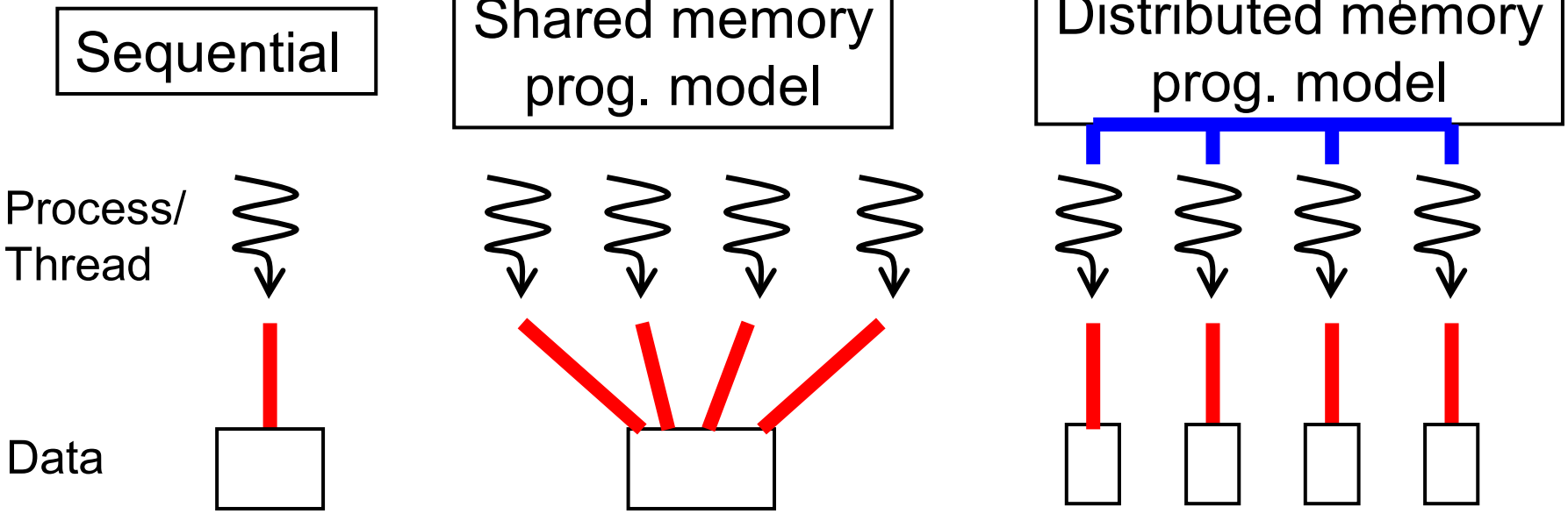
In this lecture, you are using an interactive node

- 1/8 node has 24 CPU cores

Multiple processes are invoked on a interactive node

If you want to use multiple nodes, use the job scheduler (ppcomp-sub slides)

Classification of Parallel Programming Models

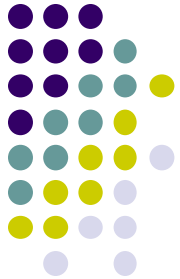


Programming without parallelsim

Threads have access to shared data
- OpenMP
- pthread
- Java thread…

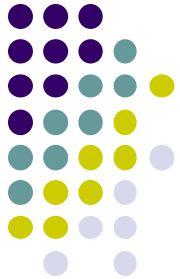
Need communication among processes
- MPI
- socket
- Hadoop, Spark…

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
  - Intel MPI
  - OpenMPI ← OpenMPI ≠ OpenMP ☹
  - ...

First MPI Sample



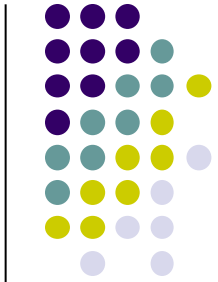
- /gs/bs/tga-ppcomp/24/hello-mpi

```
[make sure that you are at a interactive node (r7i7nX) ]
module load intel-mpi    [Do once after login]
cd ~/ppc24
cp -r /gs/bs/tga-ppcomp/24/hello-mpi  .
cd hello-mpi
make
[An executable file “hello” is created]
mpiexec -n 4 ./hello
```

Number of processes

Name of program (using options are ok)

Compiling and Executing MPI Programs



Case of Intel MPI library on TSUBAME4.0

- To compile
  - module load intel-mpi, and then use mpicc
  - For sample programs, “make” command works
- To execute
  - mpiexec -n 4 ./hello

Number of processes

↑ These methods uses 1 (current) node.

An MPI Program Looks Like

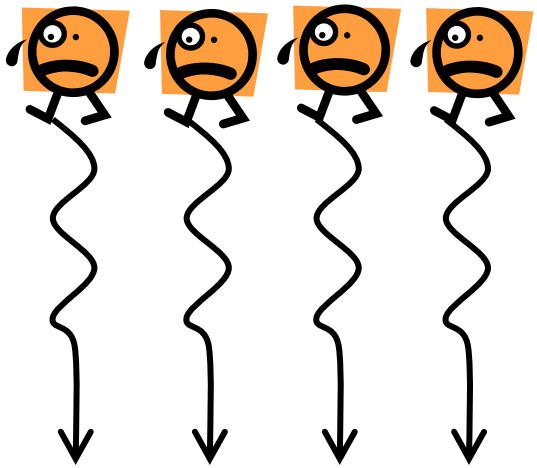
```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
   MPI_Init(&argc, &argv);
   (do something)
   MPI_Finalize();
}
```

← Initialize MPI

← Finalize MPI

Case with
4 processes

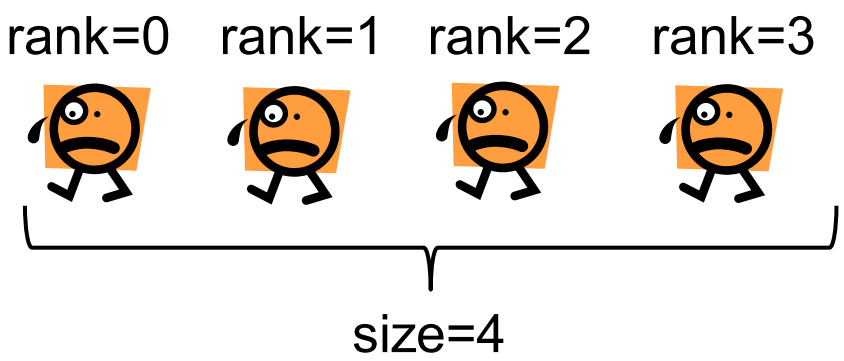


ID of Each MPI Process

- 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



“mm” sample: Matrix Multiply

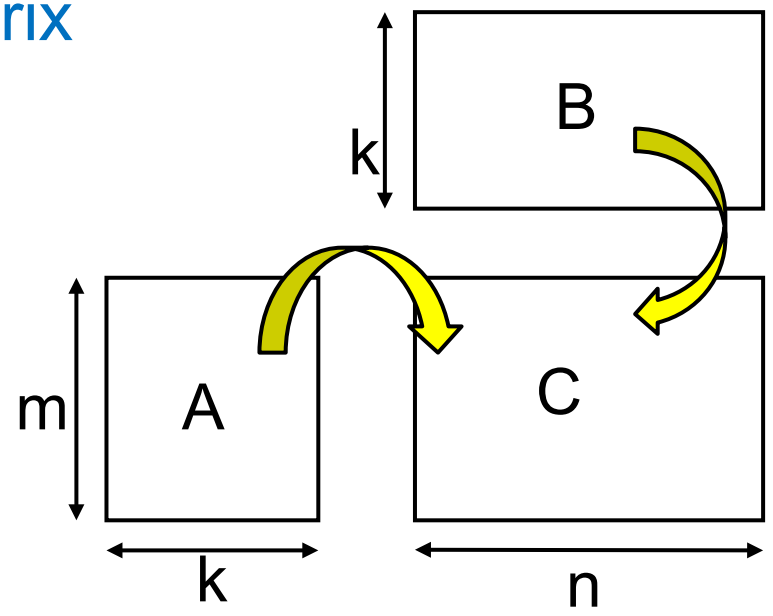
MPI version available at /gs/bs/tga-ppcomp/24/mm-mpi/

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

C: a (m × n) matrix

C ← A × B

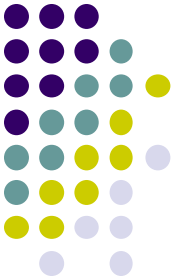
- Algorithm with a triple for loop
- Supports variable matrix size.
  - Each matrix is expressed as a 1D array by *column-major* format



Execution:

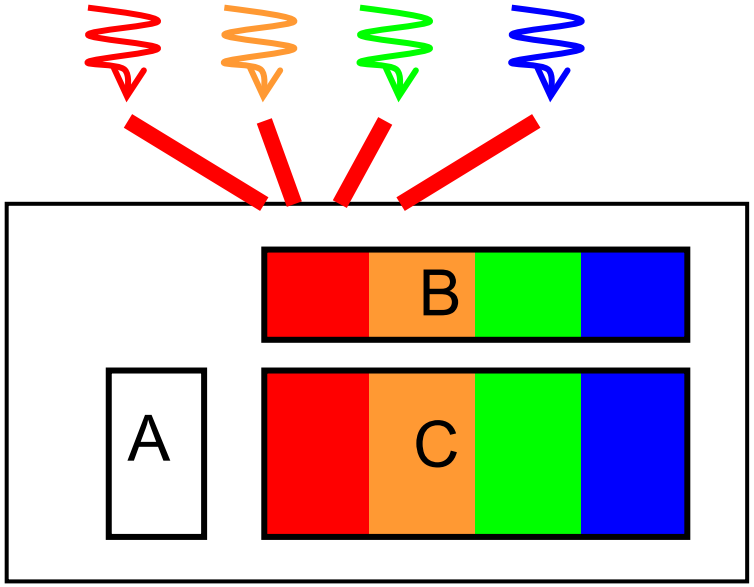
mpiexec -n [np] ./mm [m] [n] [k]

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



Shared memory with OpenMP:

Programmers consider how computations are divided

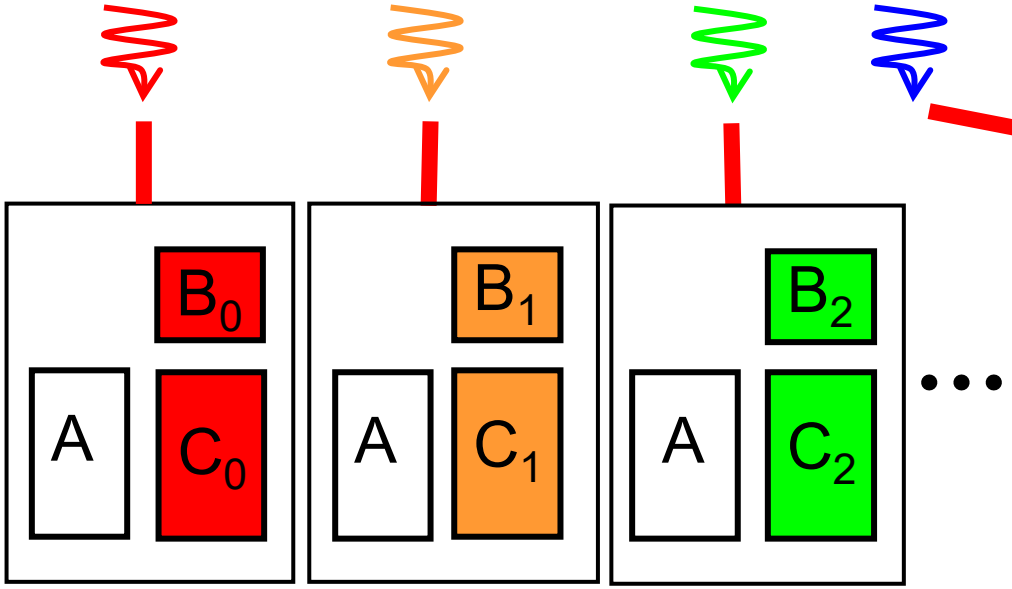


In this case, matrix A is accessed by all threads

→ Programmers do not have to know that

Distributed memory with MPI:

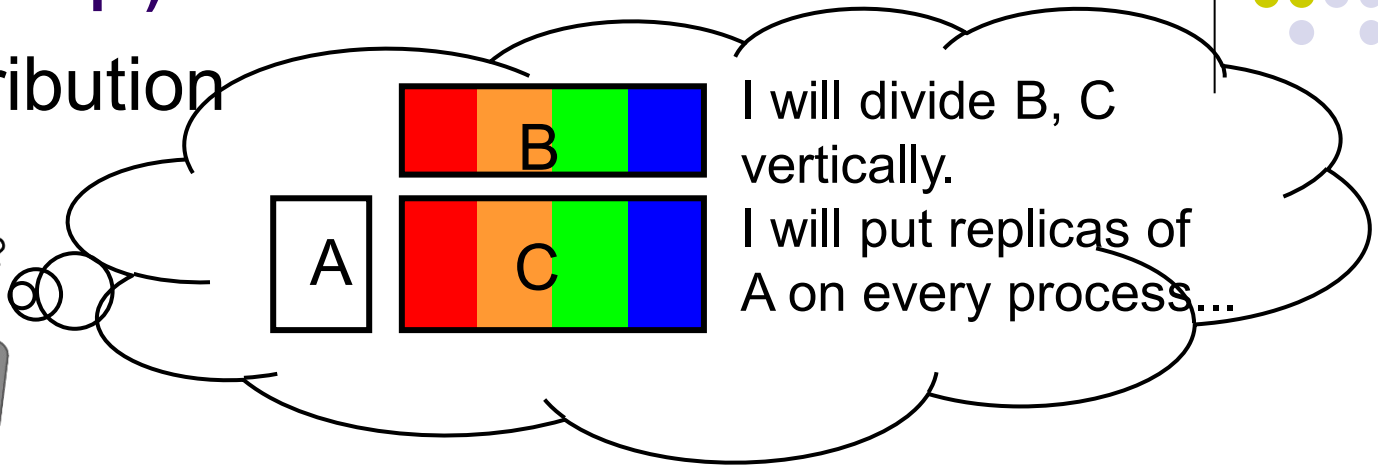
Programmers consider how data and computations are divided



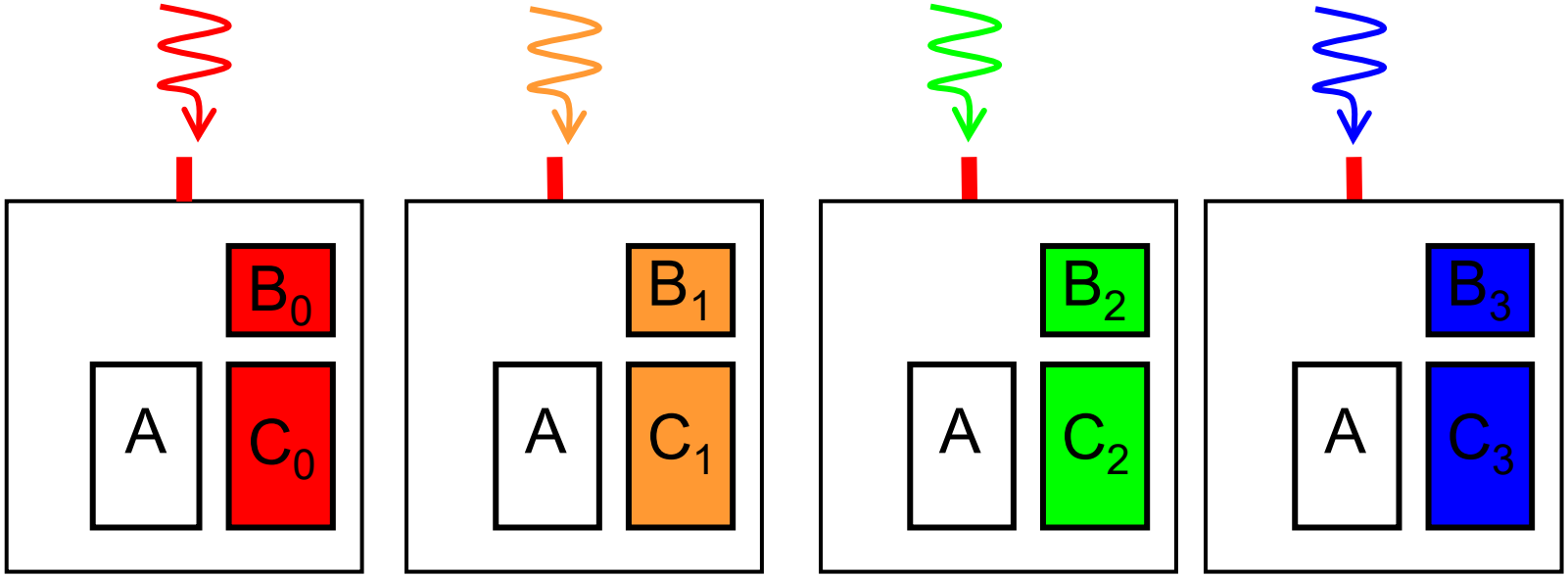
Programmers have to design which data is accessed by each process

Programming Data Distribution
(case of mm-mpi)

Design distribution method:



Programming actual location:



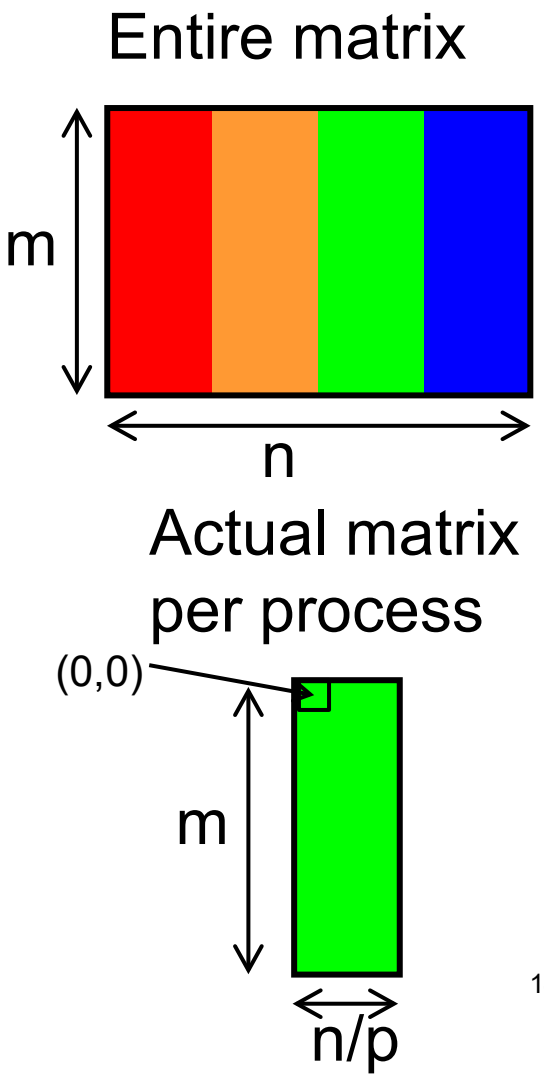
Programming Actual Data Distribution

- 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*(n/p)*sizeof(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 ⇔

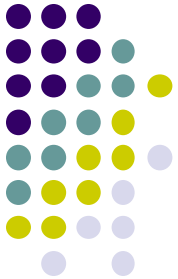
      (i, n/p*r + j) element in entire matrix

      local index

      global index

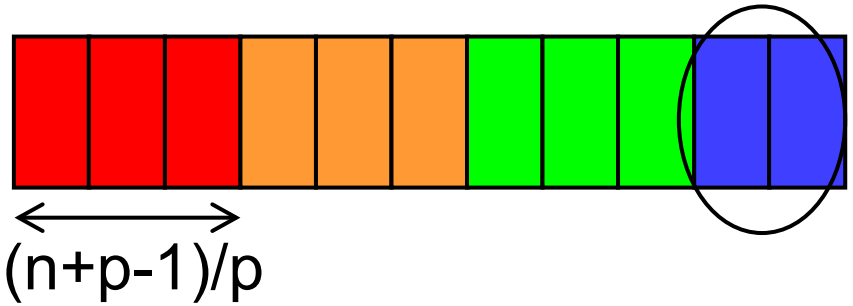


Considering Indivisible Cases



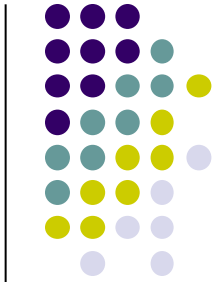
- What if data size n is indivisible by p?
- We let n=11, p=4
  - How many data each process take?
  - n/p = 2 is not good (C division uses round down). Instead, we should use round up division
  - → (n+p-1)/p = 3 works well

Note that the "final" process takes less than others



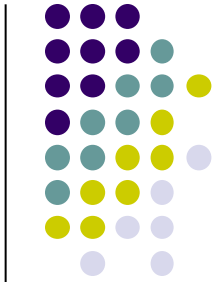
See divide_length() function in mm-mpi/mm.c
It calculates the range the process should take
→ Outputs are first index s and last index e

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

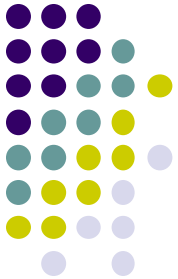
Differences from OpenMP



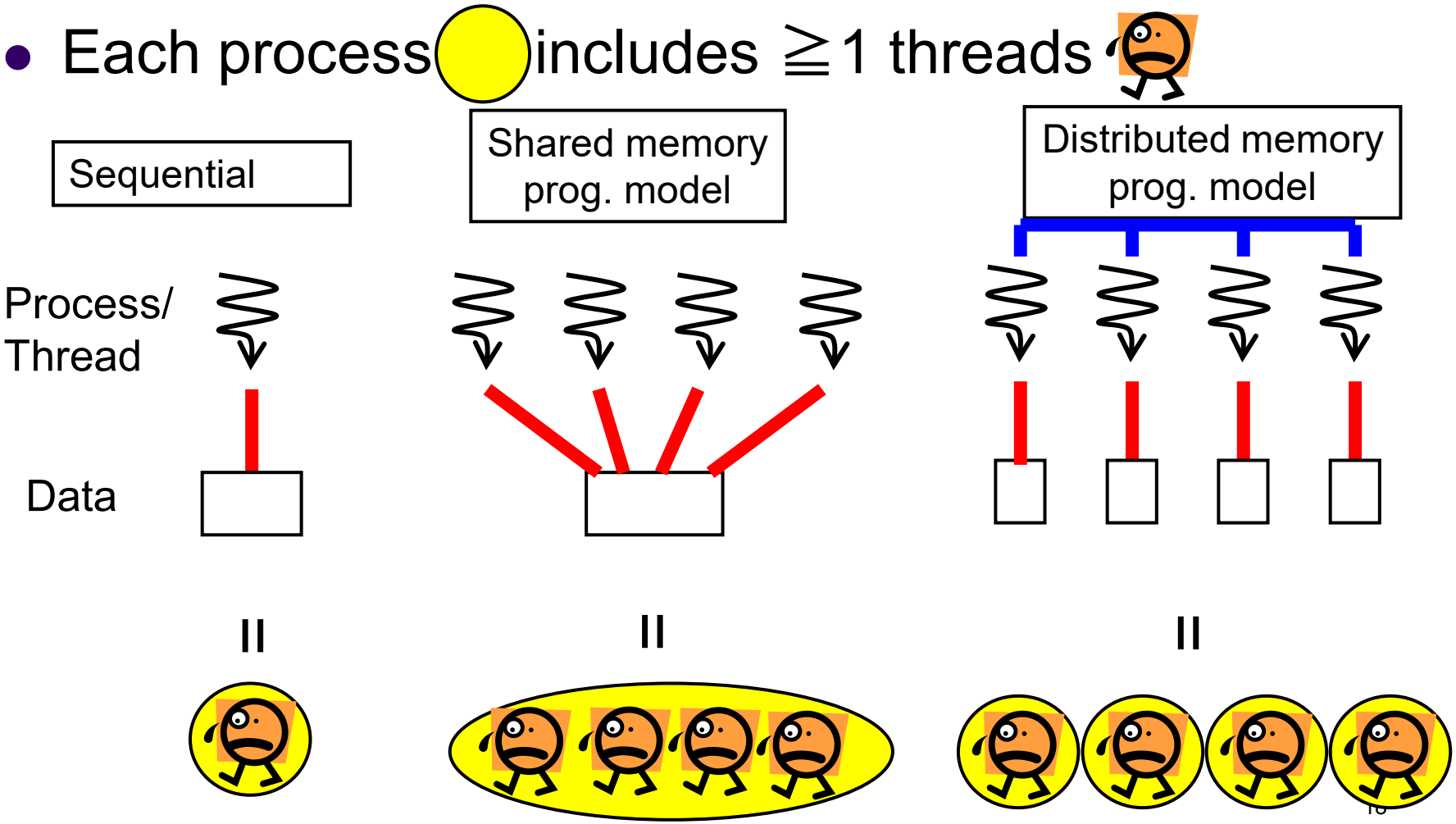
In MPI,

- An execution consists of multiple processes
  - We can use multiple nodes ☺
  - The number of running processes is basically constant
- No variables are shared
  - All variables, including global variables, are private!
- If we want to share data between processes, 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 ☹

Differences between Processes and Threads (1)



- Each process includes ≧1 threads



Differences between Processes and Threads (2)

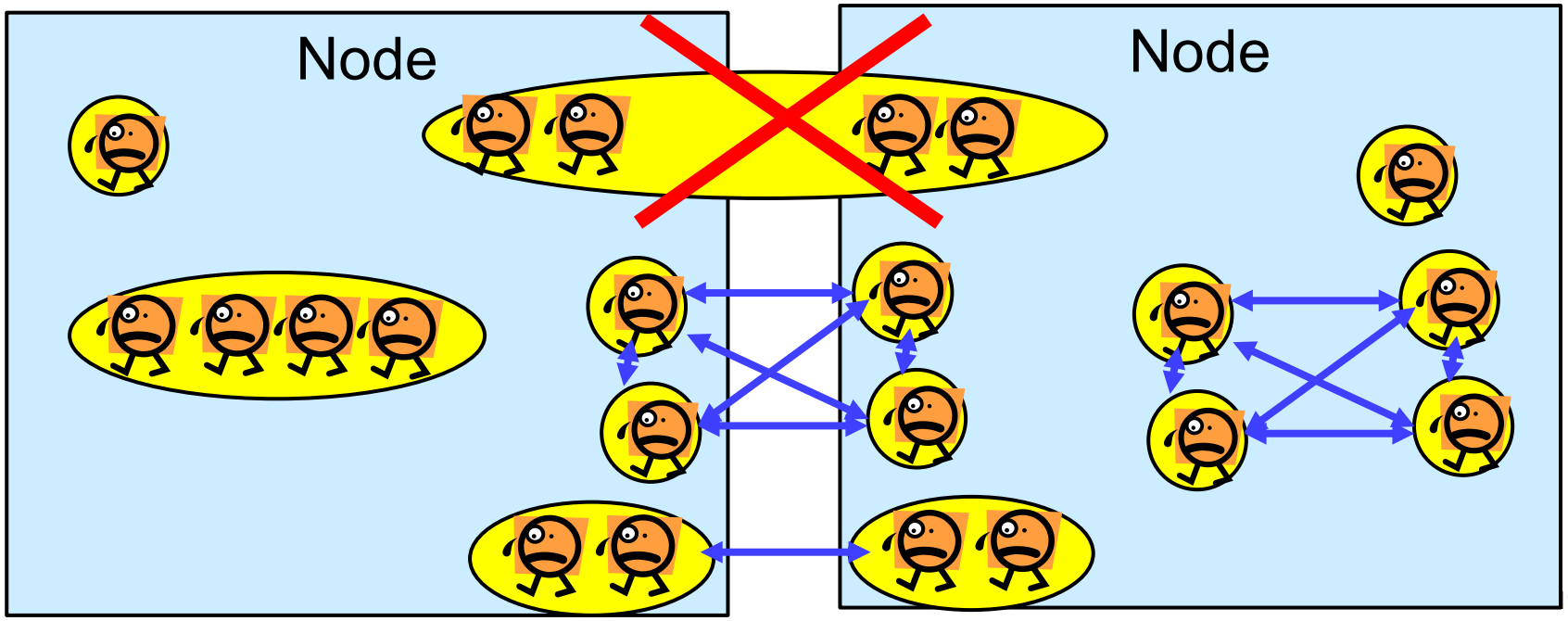
A process has its “virtual address space”

➔ Threads in the same process share data on memory

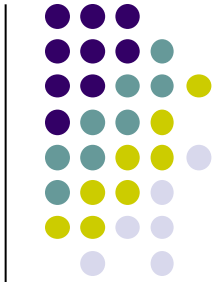
➔ A Process cannot see data of other processes

- One process cannot span multiple nodes (even with multiple threads)

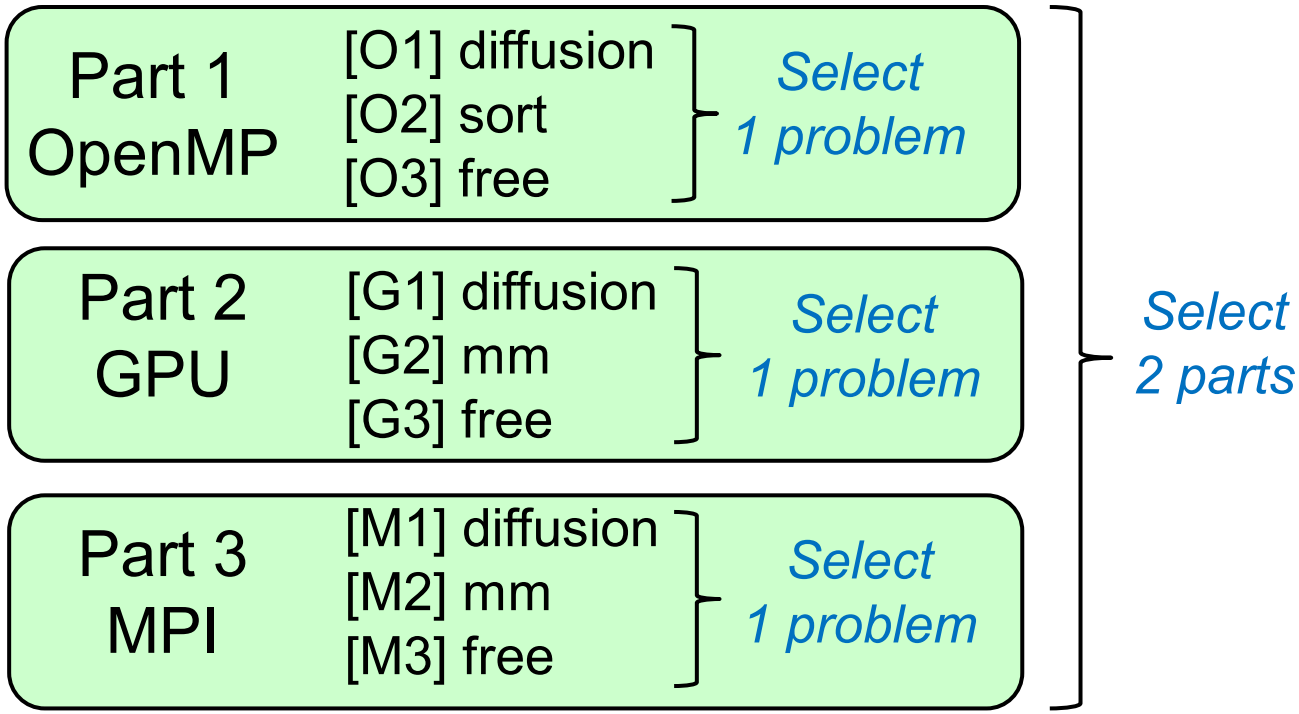
⇔ Multiple processes are needed

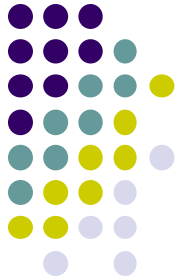


Assignments in this Course



- 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: June 13 (Thursday)

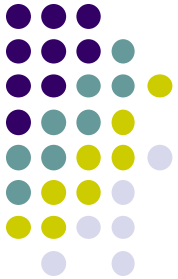
[M1] Parallelize “diffusion” sample program by MPI.

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

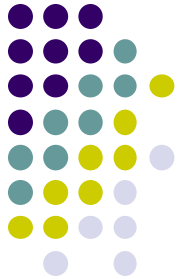


[M2] Improve “mm-mpi” sample in order to reduce memory consumption

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

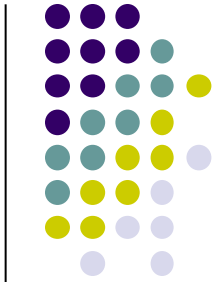
Assignments in MPI Part (3)



[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)

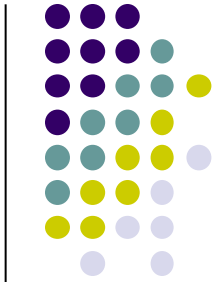


- Submit the followings via T2SCHOLA

  (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
    - To use multiple nodes, 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 27: Part 3 (2)
  - May 30: Part 3 (3)
  - June 3: Part 3 (4): (Short) class + TSUBAME4 tour
    - If you come to G2-202, Suzukake-dai, you can see TSUBAME4