

# 2021 Practical Parallel Computing (実践的並列コンピューティング) No. 3

Part1: OpenMP (1)  
Apr 19, 2021

Toshio Endo  
School of Computing & GSIC  
[endo@is.titech.ac.jp](mailto:endo@is.titech.ac.jp)





# Overview of This Course

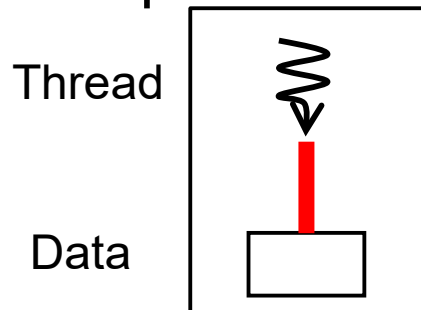
- Part 0: Introduction
  - 2 classes
- Part 1: **OpenMP** for shared memory programming
  - 4 classes **← We are here (1/4)**
- Part 2: **GPU** programming
  - OpenACC and CUDA
  - 4 classes
- Part 3: **MPI** for distributed memory programming
  - 3 classes



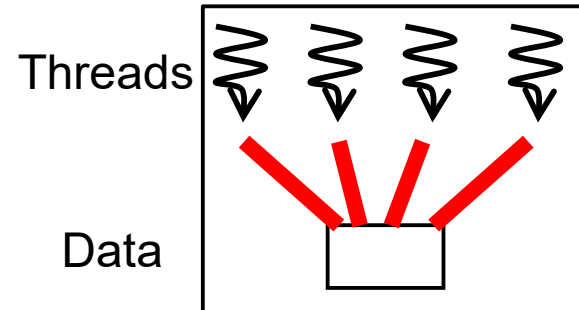
# What is OpenMP?

- One of programming APIs based on **shared-memory** parallel model
  - Multiple threads work cooperatively
  - Threads can share data

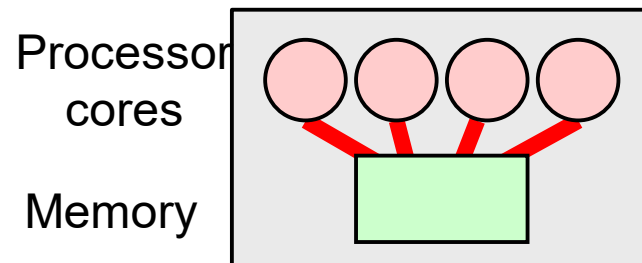
Simple C software

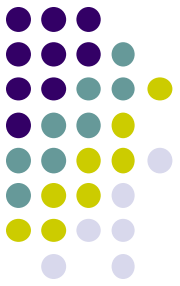


OpenMP software



Hardware





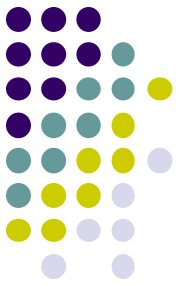
# OpenMP Programs Look Like

- OpenMP defines extensions to C/C++/Fortran
- Directive syntaxes & library functions
  - Directives look like: `#pragma omp ~~~`

```
int a[100], b[100], c[100];  
int i;  
#pragma omp parallel for  
for (i = 0; i < 100; i++) {  
    a[i] = b[i]+c[i];  
}
```

An example of OpenMP  
*directive*

In this case, a directive has  
an effect on the following  
block/sentence



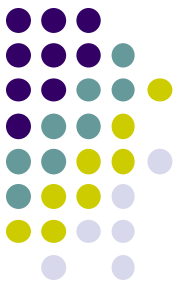
# Sample Programs

- [/gs/hs1/tga-ppcomp/21/](#) directory
  - You have to be a member of [tga-ppcomp](#) group
  - There are sub-directories per sample
- Samples related to today's class
  - [hello-omp](#)
  - matrix multiplication
    - [mm](#): sequential version
    - [mm-omp](#): OpenMP version



# Using hello-omp Sample

```
[make sure that you are at a interactive node (r7i7nX) ]  
cd ~/t3workspace [Example in web-only route]  
cp -r /gs/hs1/tga-ppcomp/21/hello-omp .  
cd hello-omp  
make  
[this creates an executable file "hello"]  
./hello
```



# Compiling OpenMP Programs

All famous compilers support OpenMP (fortunately☺), but require different options (unfortunately☹)

- gcc
  - `-fopenmp` option in compiling and linking
- PGI compiler (included by NVIDIA HPC SDK)
  - `module load nvhpc`, and then use `pgcc`
  - `-mp` option in compiling and linking
- Intel compiler
  - `module load intel`, and then use `icc`
  - `-openmp` option in compiling and linking

Also see outputs of “make” in OpenMP sample directory



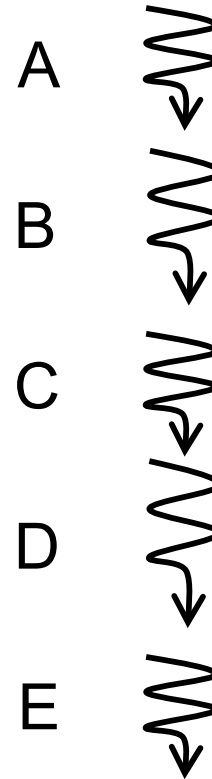
# A Sequential Example

```
int main()
{
    A;

    {
        B;
    }
    C;

    D;
    E;
}
```

Flow of  
execution





# Basic Parallelism in OpenMP: Parallel Region



```
#include <omp.h>
```

```
int main()  
{
```

```
    A;
```

```
    #pragma omp parallel
```

```
    {
```

```
        B;
```

```
    }
```

```
    C;
```

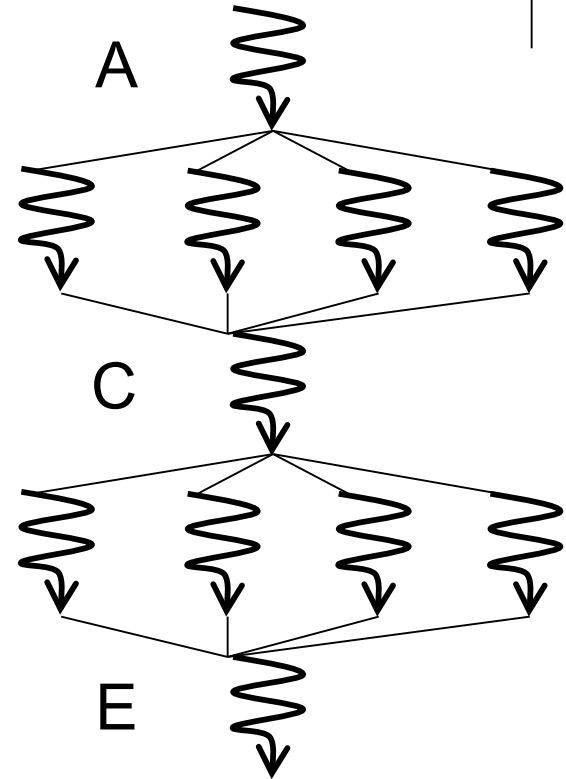
```
    #pragma omp parallel
```

```
    D;
```

```
    E;
```

```
}
```

Parallel  
region



Sentence/block immediately after **#pragma omp parallel** is called **parallel region**, executed by multiple threads

- Here a “block” is a region surrounded by braces `{ }`
- Functions called from parallel region are also in parallel region



# Number of Threads

- Specify number of threads by **OMP\_NUM\_THREADS** environment variable (this is done out of program)
  - cf) export OMP\_NUM\_THREADS=7  
in command line
  - In default, number of cores (including HyperThreads) are used. On an interactive node,  $7 \times 2 = 14$
- Obtain number of threads
  - cf) `n = omp_get_num_threads();`
- Obtain “my ID” of calling thread
  - cf) `id = omp_get_thread_num();`
    - $0 \leq id < n$  (total number)



# Outputs of hello-omp

Before the parallel region

```
Hello OpenMP World
I'm 8-th thread out of 14 threads
I'm 6-th thread out of 14 threads
I'm 9-th thread out of 14 threads
I'm 1-th thread out of 14 threads
I'm 0-th thread out of 14 threads
I'm 7-th thread out of 14 threads
:
Good Bye OpenMP World
```

Inside the parallel region,  
each thread prints a message  
for several (5) times

omp\_get\_num\_threads()

omp\_get\_thread\_num()

After the parallel region

# Executing a Sample with Various Number of Threads



*[make sure that there is an executable file “hello”]*

```
export OMP_NUM_THREADS=1  
./hello
```

```
export OMP_NUM_THREADS=4  
./hello
```

```
export OMP_NUM_THREADS=7  
./hello
```

```
export OMP_NUM_THREADS=14  
./hello
```

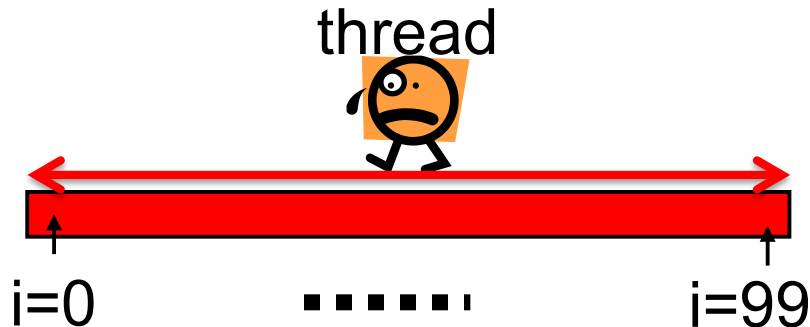
# How Can We Make a Program Faster?



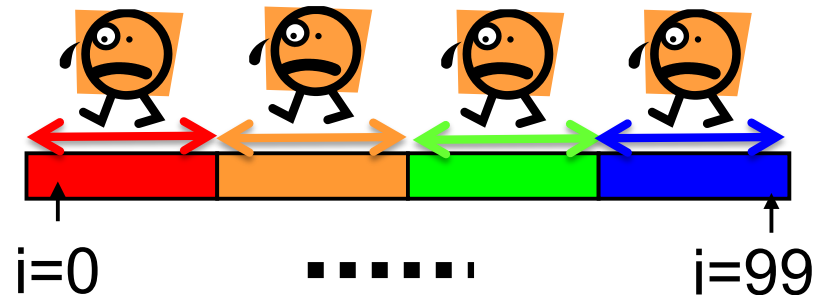
```
for (i = 0; i < 100; i++) { some computation; }
```

assumption: 100 tasks are independent with each other

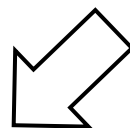
Only with one thread



With 4 threads



thread 0: for ( $i = 0$  ;  $i < 25$ ; ...  
thread 1: for ( $i = 25$ ;  $i < 50$ ; ...  
thread 2: for ( $i = 50$ ;  $i < 75$ ; ...  
thread 3: for ( $i = 75$ ;  $i < 100$ ; ...



OpenMP has a syntax to do this smarter

# #pragma omp for for Easy Parallel Programming



“for” loop with simple forms can parallelized easily

```
{  
#pragma omp parallel  
{  
    int i;  
#pragma omp for  
    for (i = 0; i < 100; i++) {  
        a[i] = b[i]+c[i];  
    }  
}  
}
```

#pragma omp for must be

- inside a parallel region
- right before a “for” loop

→ Computations in the loop are distributed among threads (work distribution)

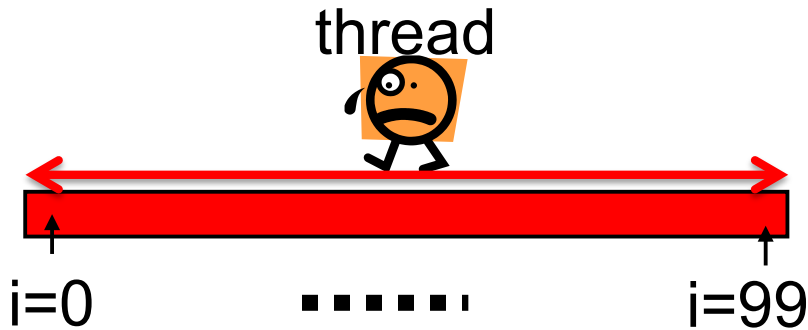
- With 4 threads, each thread take  $100/4=25$  iterations → speed up!!
  - Indivisible cases are ok, such as 7 threads

- Abbreviation: omp parallel + omp for = omp parallel for

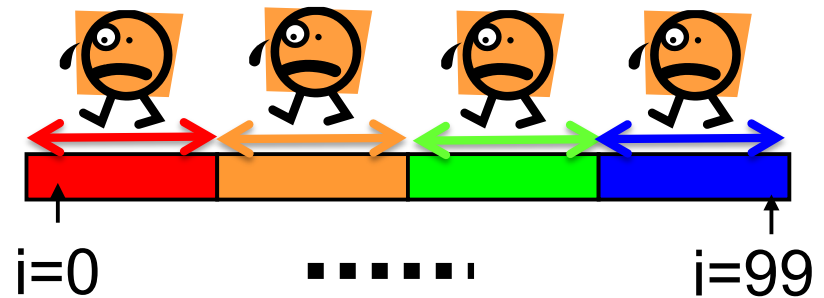
# Why “omp for” Reduces Execution Time



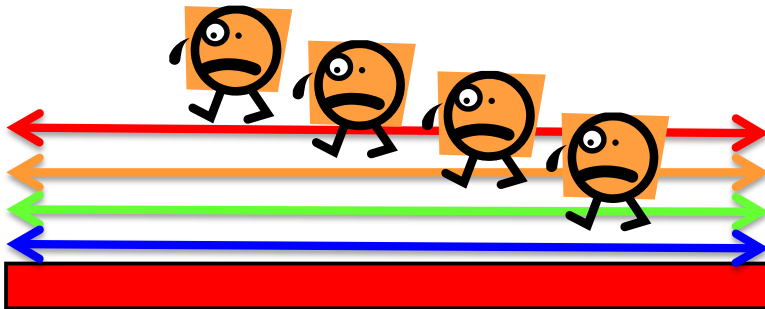
Only with one thread



With several threads



- What if we use “omp parallel”, but **forget** to write “omp for”?



Every thread would work for all iterations

→ No speed up ☹️

→ Answer will be wrong ☹️

# “mm” sample: Matrix Multiply



Available at </gs/hs1/tga-ppcomp/21/mm/>

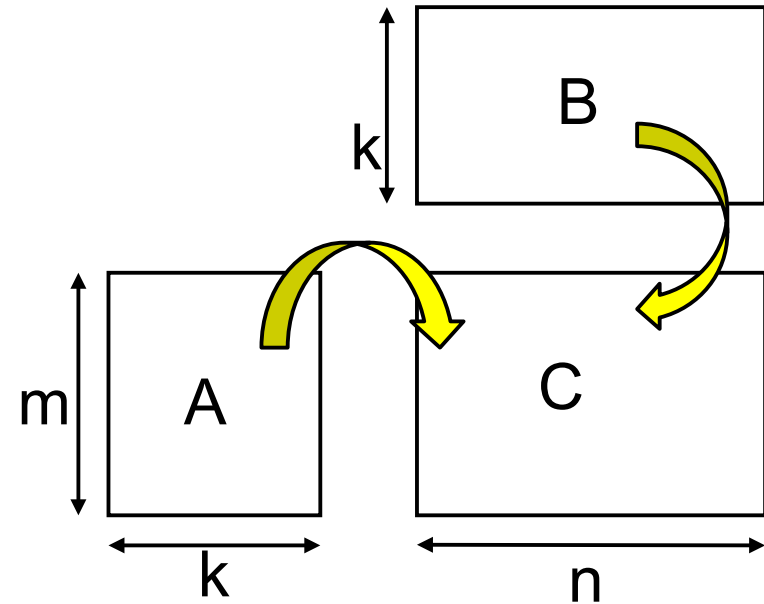
A: a  $(m \times k)$  matrix

B: a  $(k \times n)$  matrix

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

$C \leftarrow A \ B$

- This sample supports variable matrix sizes
- Execution: `./mm [m] [n] [k]`



```
for (j = 0; j < n; j++) {  
    for (l = 0; l < k; l++) {  
        for (i = 0; i < m; i++) {  
            C[i+j*ldc] += A[i+l*lda] * B[l+j*ldb];  
        } } }  
}
```



# OpenMP Version of mm (mm-omp)



- There are 3 loops. Here, j loop is parallelized

```
#pragma omp parallel private(i,l)
#pragma omp for           ← j loop is parallelized
for (j = 0; j < n; j++) {
    for (l = 0; l < k; l++) {
        for (i = 0; i < m; i++) {
            C[i+j*ldc] += A[i+l*lda] * B[l+j*ldb];
        } } }
```

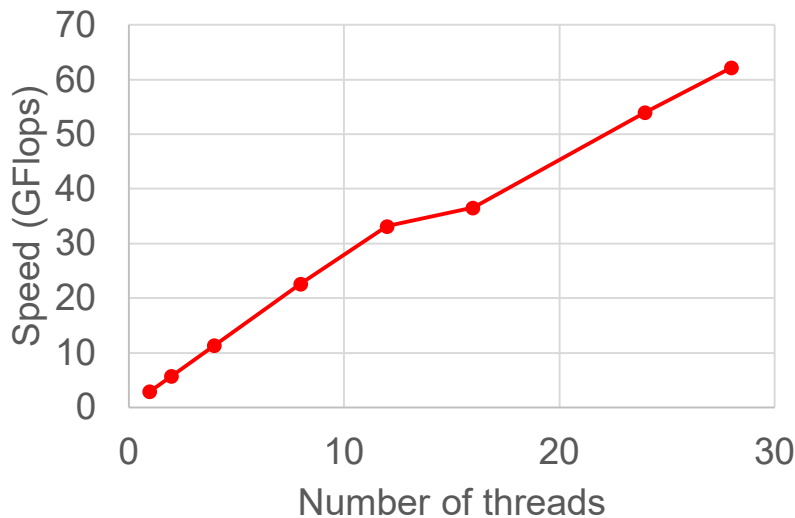
- “private” option is explained later

# Performance of mm sample

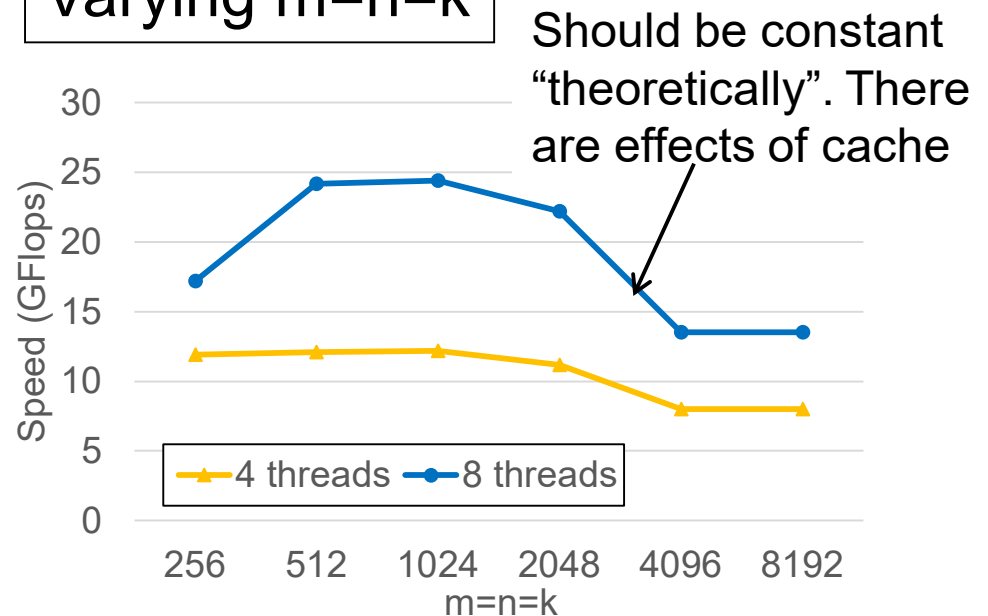


- A TSUBAME3 node (Xeon E5-2680 v4 x2 = 28core)
- Speed is (2mnk/t)

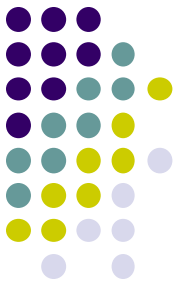
m=n=k=2048,  
Varying # of threads



8 threads,  
Varying m=n=k



# Shared Variables & Private Variables (1)



While OpenMP uses “shared memory model”, **not all are shared**

In default, variables are classified as follows

- Variables declared **out of** parallel region  $\Rightarrow$  **Shared variables**
- Global variables  $\Rightarrow$  **Shared variables**
- Variables declared **inside** parallel region  $\Rightarrow$  **Private variables**

```
{
    int s = 1000;
    #pragma omp parallel
    {
        int i;
        i = func(s, omp_get_thread_num());
        printf( "%d\n" , i);
    }
}
```

**shared**

**private**

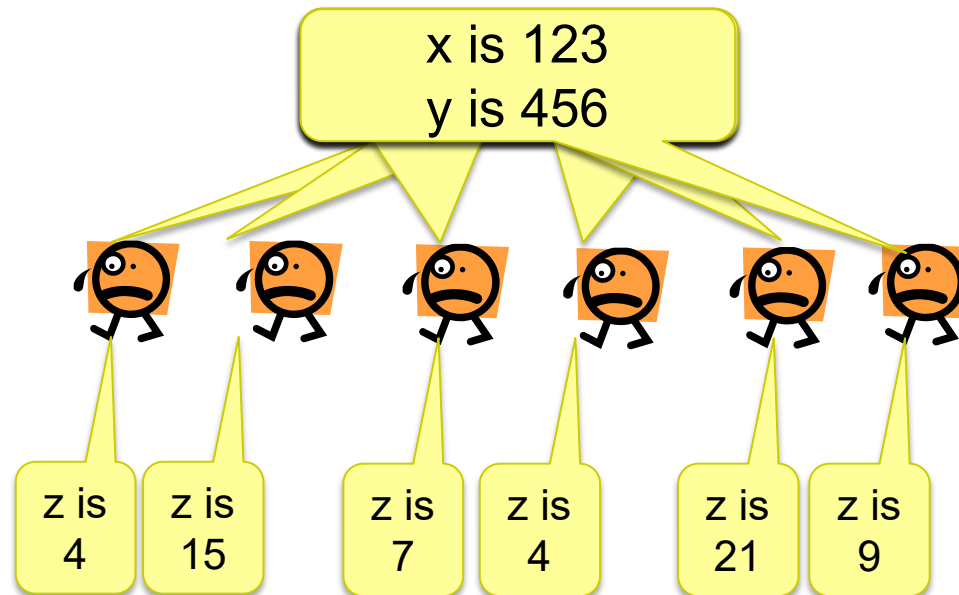
```
int func(int a, int b)
{
    int rc = a+b;
    return rc;
}
```

**private**

# Shared Variables & Private Variables (2)



We let  $x$ ,  $y$  be shared, and  $z$  be private



*Single instance  
for each  $x$ ,  $y$*

*Each thread has  
its own instance for  $z$*

- When a thread updates a shared variable, other threads are affected
  - We should be careful and careful!



# Pitfall in Nested Loops (1)

- The following sample looks ok, but there is a bug
  - We do not see compile errors, but answers would be wrong ☹️

```
int i, j;  
#pragma omp parallel  
#pragma omp for  
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        ...  
    }  
}
```

Both i, j are declared  
outside parallel region  
→ Considered “shared”  
It is a problem to share **j**

cf)

Thread A is executing i=5 loop  
Thread B is executing i=8 loop

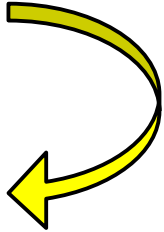
} The executions should be independent  
Each execution must include  
j=0, j=1...j=n-1 correctly  
**j must be private**



# Pitfall in Nested Loops (2)

Two modifications (Either is ok)

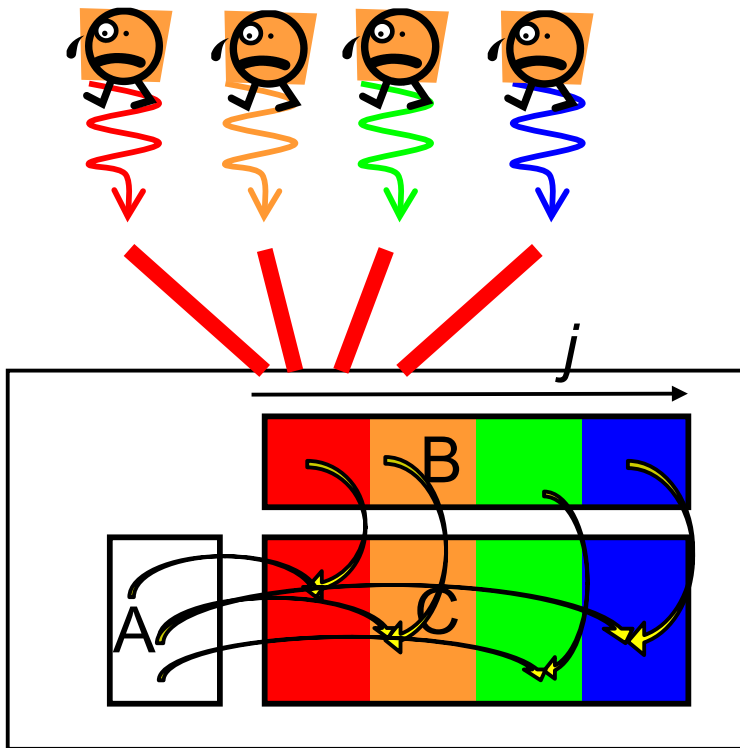
```
int i;  
#pragma omp parallel for  
for (i = 0; i < m; i++) {  
    int j;    // j is private  
    for (j = 0; j < n; j++) {  
        ...  
    } }  
}
```



```
int i, j;  
#pragma omp parallel for private(j)  
// j is forcibly private  
for (i = 0; i < m; i++) {  
    for (j = 0; j < n; j++) {  
        ...  
    } }  
}
```



# How Arrays are Accessed



- In mm sample, pointers A, B, C are global variables → shared variables
- Since all threads see same variables of A, B, C, contents of arrays are also shared
- It is programmers responsibility to make each thread does independent computation

# OpenMP Version of mm (Again)



- One of loops is parallelized

```
#pragma omp parallel private(i,l)
```

```
#pragma omp for
```

```
    for (j = 0; j < n; j++) {  
        for (l = 0; l < k; l++) {  
            for (i = 0; i < m; i++) {  
                C[i+j*ldc] += A[i+l*lda] * B[l+j*ldb];  
            } } }  
    }
```

*j* loop is parallelized

→ Each thread executes computations only for subset of [0, n)

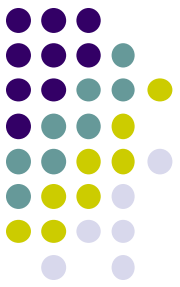
[Q] What if we parallelize other loops?

→ *i* loop is ok for correct answers, but may be slow

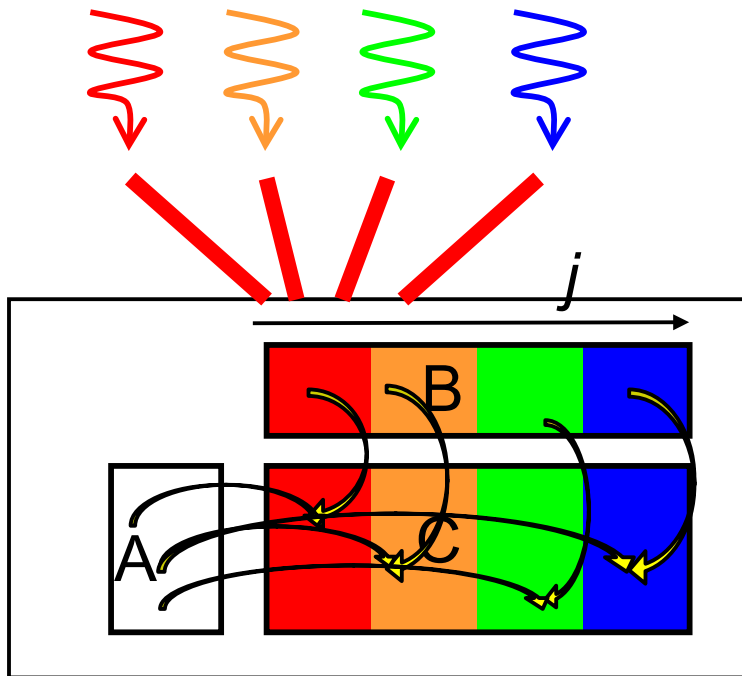
→ *l* loop causes wrong answers!



# Correct Parallelization and Bad Parallelization



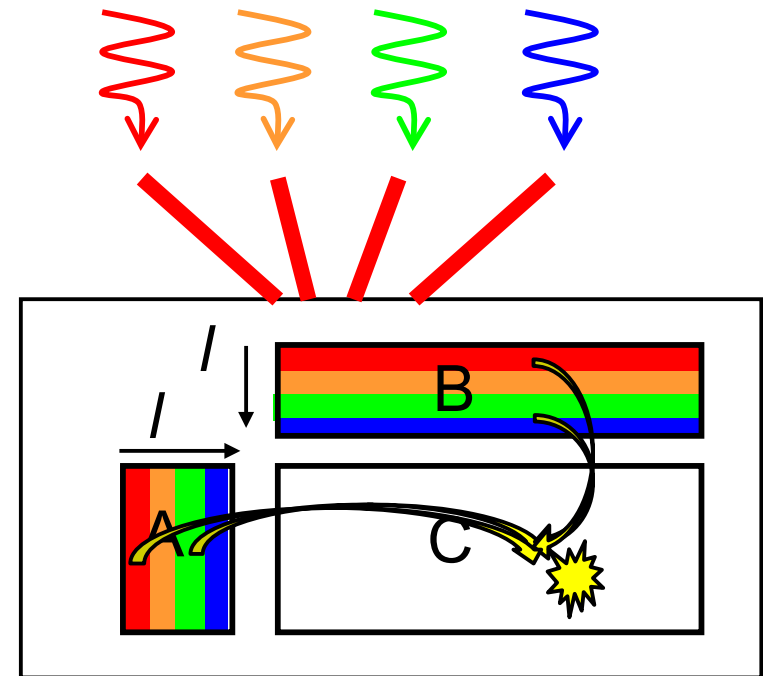
Parallelizing  $j$  loop



Simultaneous read from same data  
(in this case, A) is OK

Similarly, parallelizing  
 $i$  loop is ok

Parallelizing  $i$  loop (??)



Possible simultaneous write to  
same data

→ “Race condition” problem  
may occur.

Answers may be wrong !!

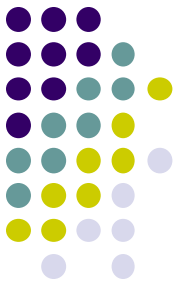


# Today's Summary

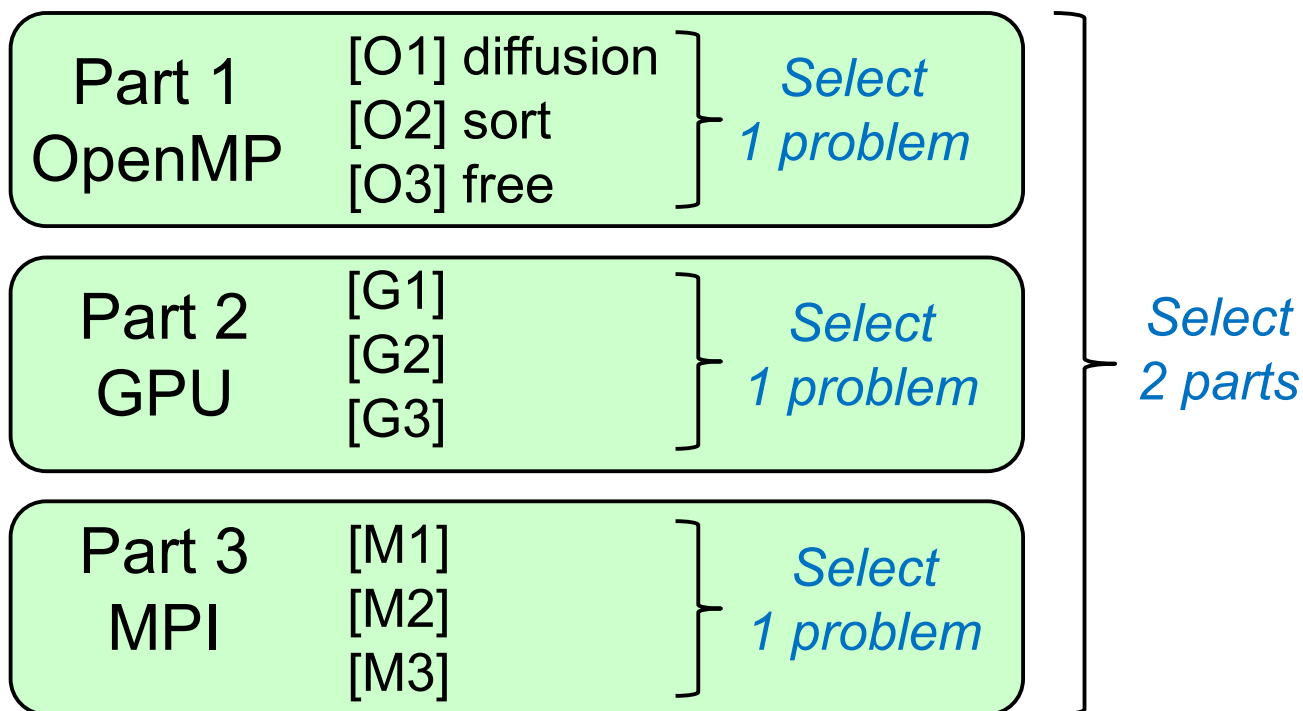
Introduction to OpenMP parallel programming

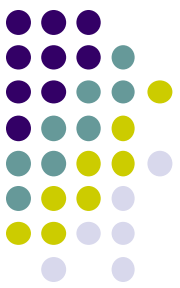
- Multiple threads work simultaneously with `#pragma omp parallel`
- With `#pragma omp for`, loop-based programs can be parallelized easily
- But it is programmer's responsibility to avoid bugs caused by race conditions

# 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 OpenMP Part (1)

Choose one of [O1]—[O3], and submit a report

Due date: May 13 (Thu)

Sorry, in the first version of slides, date was wrong!!

[O1] Parallelize “diffusion” sample program by OpenMP.

(</gs/hs1/tga-ppcomp/21/diffusion/> on TSUBAME)

Optional:

- To make array sizes variable parameters, which are specified by execution options. “malloc” will be needed.
- To parallelize it without “omp for”
  - `omp_get_thread_num()`, `omp_get_num_threads()` are needed

# Assignments in OpenMP Part (2)



[O2] Parallelize “sort” sample program by OpenMP.  
(</gs/hs1/tga-ppcomp/21/sort/> on TSUBAME)

Optional:

- Comparison with other algorithms than quick sort
  - Heap sort? Merge sort?

# Assignments in OpenMP Part (3)



[O3] (Freestyle) Parallelize *any* program by OpenMP.

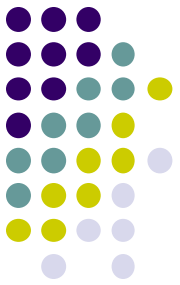
- cf) A problem related to your research
- More challenging one for parallelization is better
  - cf) Partial computations have dependency with each other
  - cf) Uniform task division is not good for load balancing

# 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

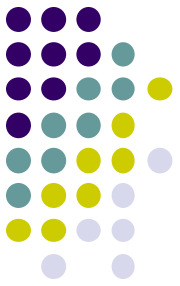
# Notes in Report Submission (2)



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 threads
    - On a interactive nodes,  $1 \leq \text{OMP\_NUM\_THREADS} \leq 14$
    - To use more CPU cores, you need to do “job submission”
  - With varying problem sizes
  - Discussion with your findings
  - Other machines than TSUBAME are ok, if available





# If You Have Not Done This Yet

Please do the followings as soon as possible

- Please make your account on TSUBAME
- Please send an e-mail to [ppcomp@el.gsic.titech.ac.jp](mailto:ppcomp@el.gsic.titech.ac.jp)

Subject: [TSUBAME3 ppcomp account](#)

To: [ppcomp@el.gsic.titech.ac.jp](mailto:ppcomp@el.gsic.titech.ac.jp)

Department name:

School year:

Name:

Your TSUBAME account name:

Then we will invite you to the TSUBAME group, [please click URL and accept the invitation](#)

その後、TSUBAMEグループへの招待を送ります。[メール中のURLをクリックして参加承諾してください](#)



# Next Class:

- Part1: OpenMP (2)
  - diffusion: simple simulation of diffusion phenomena
    - Related to assignment [O1]