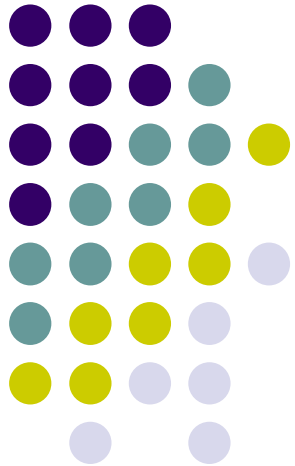
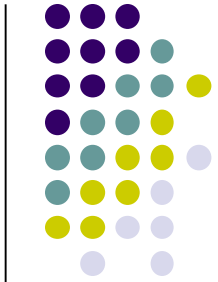
2021 Practical Parallel Computing (実践的並列コンピューティング) No. 6

Part1: OpenMP (4)
Apr 29, 2021



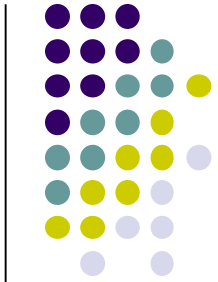
Toshio Endo
School of Computing & GSIC
endo@is.titech.ac.jp

Overview of This Course



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

Today's Topic

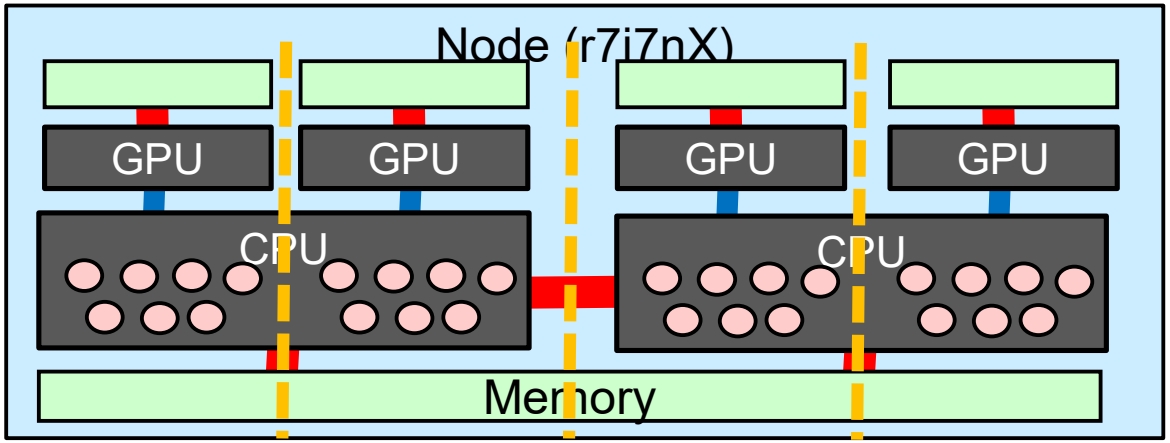


- TSUBAME Job submission

- Mutual exclusion, reduction, bottleneck in OpenMP
  - samples: pi, pi-bad-omp, pi-good-omp, pi-fast-omp, pi-omp

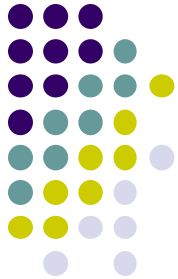
About TSUBAME Usage (1)

- In this lecture, “nodes on interactive queue” are mainly used
  - 7 cores (14 hyper threads)+ 1 GPU
  - may be shared by several users



It is ok to get credits in this lecture, but there is another way
「インタラクティブ利用」だけで単位取得はできるが、他の利用方法もある

About TSUBAME Usage (2)



Using the job scheduler is more general way to use a supercomputer

With job scheduler on TSUBAME3.0,

☺ We can use more and dedicated cores

- With OpenMP, we can use up to 28 cores (56 hyper threads)
- With MPI, we can use several nodes
- Cores are not shared with other users

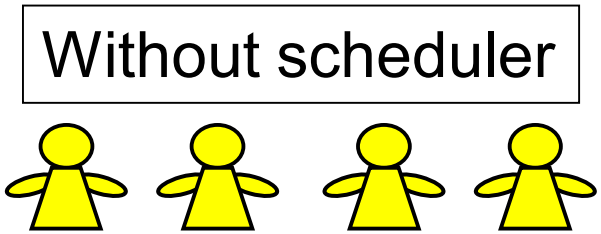
☹ It is not “real-time”

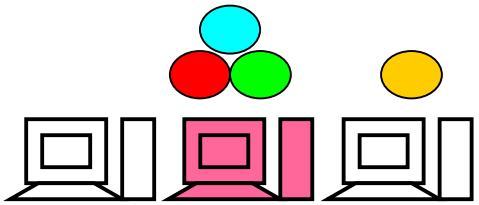
☹ Take care of charge! (TSUBAME point)

- In case of tga-ppcomp, Endo’s budget is used

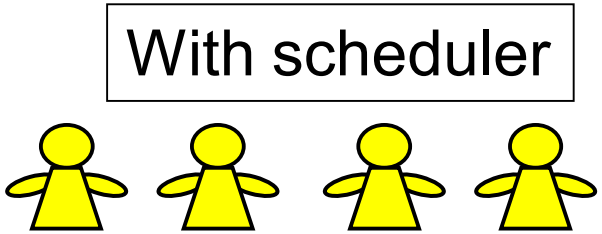
What is Job Scheduler?

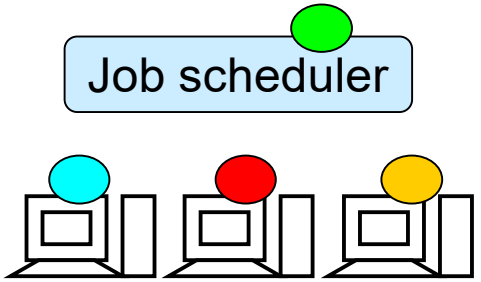
- The job scheduler does “traffic control” of many programs by many users
  - TSUBAME3.0 uses “Univa Grid Engine”





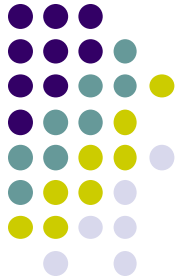
If users execute programs without control, there will be congestions





Scheduler determines nodes for each job.
Some program executions may be “queued”

Overview of Job Submission
(Section 5 in TSUBAME3.0 User's Guide at www.t3.gsic.titech.ac.jp)



(1) Prepare programs to be executed

(2) Prepare a text file called job script, which includes
- how the program is executed
- resource (nodes/CPUs) amounts required

(3) Submit the job to the job scheduler with qsub command (and wait patiently)

(4) Check the output of the job

Prepare a Job Script (Section 5.2.3)

- In the case of mm example
  - /gs/hs1/tga-ppcomp/21/mm
- job.sh is a sample job script
  - Different file name is ok, but with ".sh"

```
#!/bin/sh
#$ -cwd
#$ -l s_core=1
#$ -l h_rt=00:10:00

./mm 1000 1000 1000
```

Resource type and number: How many processor cores/ nodes are allocated

Maximum run time

What are done on the allocated node

Resource Types on TSUBAME3.0 (Section 5.1)

- Choose one of resource types (number of cores, mainly)
  - It is like “instance types” in cloud systems
- “mm” uses only a 1 core

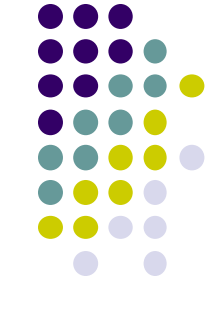
  → A TSUBAME node (28 cores + 4GPUs) is too large for mm
  - Please specify “proper” one

type	Resource type Name	Physical CPU cores	Memory (GB)	GPUs
F	f_node	28	240	4
H	h_node	14	120	2
Q	q_node	7	60	1
C1	s_core	1	7.5	0
C4	q_core	4	30	0
G1	s_gpu	2	15	1

#$ -l [resource_type] = [Number]

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

Job Submission (Section 5.2.4)



- Job submission command

  `qsub job.sh` ← File name of the job script

  - No charge (無料)
  - But this works only when h_rt <= 0:10:00 (10 minutes) and the number of resources must be <= 2

  `qsub –g [group-name] job.sh`

  - Charged! (有料)

- You will see output like: (Job ID)

  Your job 123456 ("job.sh") has been submitted

- If a job execution takes longer time, you have to specify a “TSUBAME group” name

Notes in This Lecture

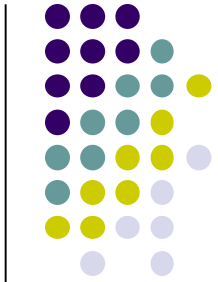
- Usually, avoid consumption of TSUBAME points
- 通常は無料利用の範囲にとどめてください
  - h_rt <= 0:10:00

- If necessary for reports, you can use up to 36,000 points in total per student
- 本講義のレポートの作成に必要な場合のみ、一人あたり合計で36,000ポイントまで利用を認めます
  - 36,000 points ≒ F_node x 10 hours
  - You can check point consumption on TSUBAME portal

- The TSUBAME group name is tga-ppcomp

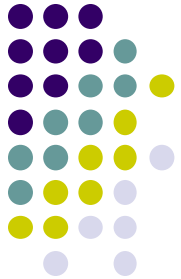
Users need to follow the rules at www.t3.gsic.titech.ac.jp
利用時には www.t3.gsic.titech.ac.jp に示される規則を守る必要があります

Check Job's Outputs



- Where "mm" s outputs go to?
- When the job is executed successfully, two files are generated automatically
  - File names look like
    - "job.sh.o123456" ← "stdout" outputs are stored
    - "job.sh.e123456" ← "stderr" outputs are stored

Other Commands for Job Management (Section 5.2.5, 5.2.6)



- qstat: To see the status of jobs under submission

```
qstat
```

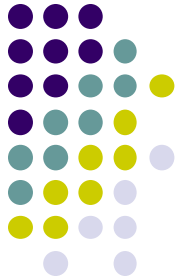
- qdel: To delete a job before its termination

```
qdel 123456
```

Job ID

For interactive sessions, you can use iqstat, iqdel commands

Prepare a Job Script for OpenMP Program (Section 5.2.3.2)



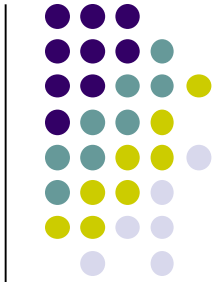
- In the case of mm-omp example
  - /gs/hs1/tga-ppcomp/21/mm-omp

job.sh

```
#!/bin/sh
#$ -cwd
#$ -l q_core=1
#$ -l h_rt=00:10:00

export OMP_NUM_THREADS=4
./mm 1000 1000 1000
```

Please choose a proper resource type
job-fnode.sh is an example with 28 cores

Today's Topic



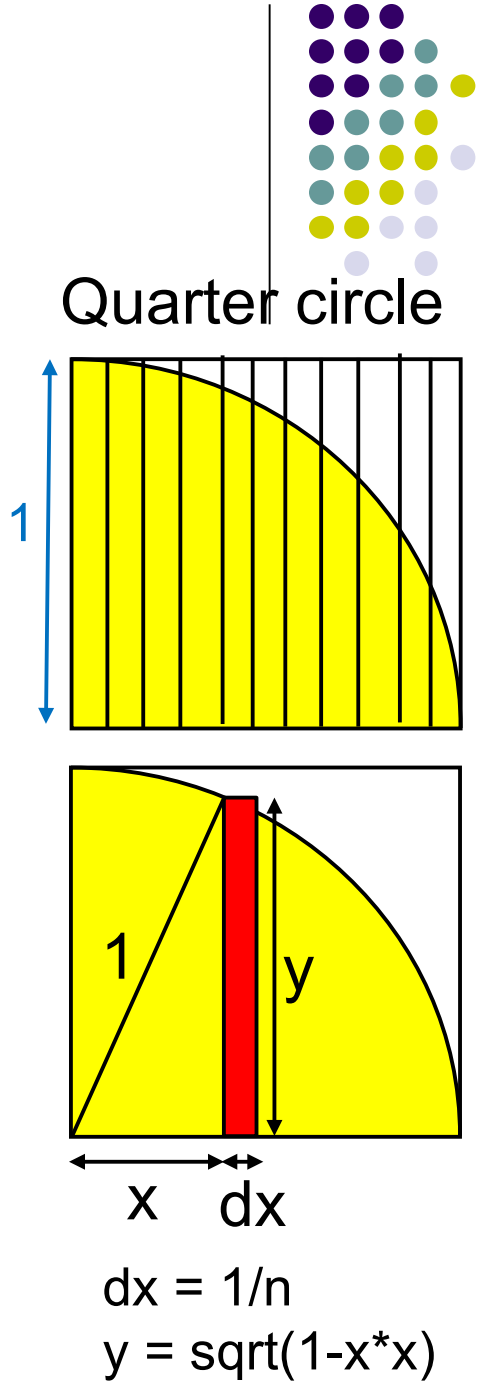
- TSUBAME Job submission
- Mutual exclusion, reduction, bottleneck in OpenMP

“pi” sample

Compute an approximation of π = 3.14159…
(circumference/diameter)

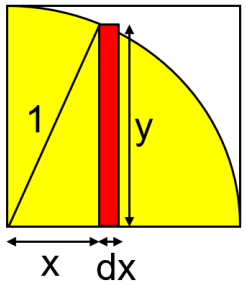
- Available at /gs/hs1/tga-ppcomp/21/pi/
- Method
  - SUM ← Approximation of the yellow area
  - π ← 4 x SUM
- Execution : ./pi [n]
  - n: Number of division
  - Cf) ./pi 100000000
- Compute complexity : O(n)

Note: This program is only for a simple sample.
π is usually computed by different algorithms.



Algorithm of “pi” (1)

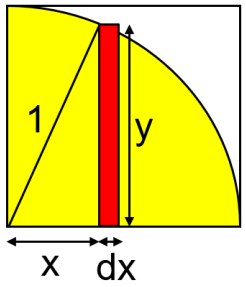
```
double pi(int n) {
    int i;
    double sum = 0.0;
    double dx = 1.0 / (double)n;



    for (i = 0; i < n; i++) {
        double x = (double)i * dx;
        double y = sqrt(1.0 - x*x);
        sum += dx*y;
    }

    return 4.0*sum; }
```



Algorithm of “pi” (2)

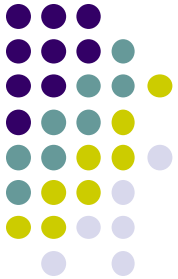
```
double pi(int n) {
    int i;
    double sum = 0.0;
    double dx = 1.0 / (double)n;

#pragma omp parallel
#pragma omp for
    for (i = 0; i < n; i++) {
        double x = (double)i * dx;
        double y = sqrt(1.0 - x*x);
        sum += dx*y;
    }

    return 4.0*sum; }
```

ok???



- Can we use #pragma omp for?
- We have to consider read&write access to sum, a shared variable

Can We Parallelize the loop in pi?



- Let us consider computations with different i

C1 (i=i1)

```
x = (double)i * dx;
y = sqrt(1.0 - x*x);
sum += dx*y;
```

these parts are independent

dependent

C2 (i=i2)

```
x = (double)i * dx;
y = sqrt(1.0 - x*x);
sum += dx*y;
```

R(C1) = {sum,dx}, W(C1) = {sum}

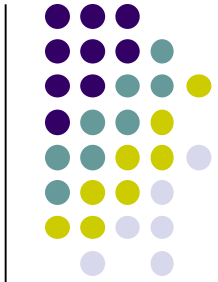
R(C2) = {sum,dx}, W(C2) = {sum}

※ private variables x, y and loop counter i are omitted

- W(C1) ∩ W(C2) ≠ ∅ ➔ Dependent!

➔ Do we have to abandon parallel execution?

Some Versions of pi Sample



- pi: sequential version

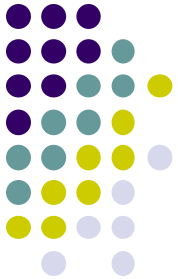
Followings use OpenMP

- pi-bad-omp:
  - *"#pragma omp parallel for"* is simply used

  → It has a bug that produces incorrect results
- pi-good-omp: results are correct, but slow
- pi-fast-omp: results are correct and faster
- pi-omp: same as pi-fast-omp but uses "reduce" option

All are at /gs/hs1/tga-ppcomp/21/

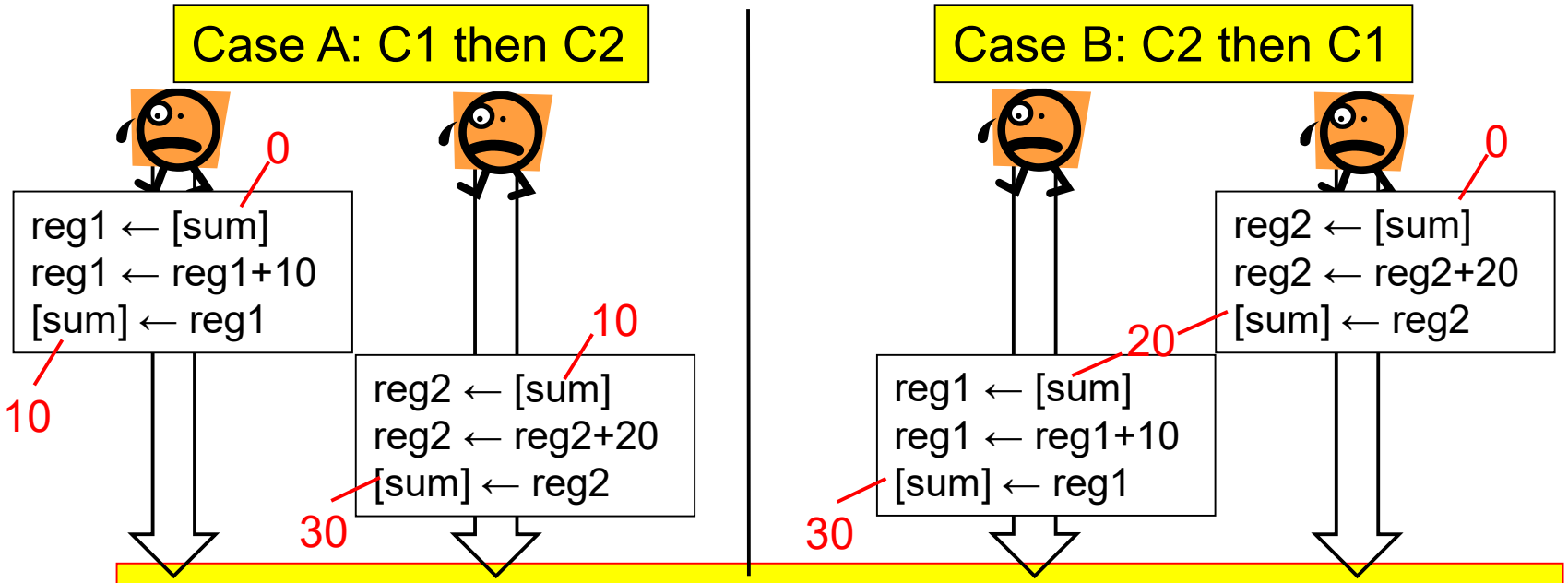
What's Wrong in pi-bad-omp? (1)



- Now we simply consider C1: sum += 10; & C2: sum += 20;
- We assume "sum = 0" initially
- [Q] Does execution order of C1 & C2 affect the results?
  - Note: "sum += 10" is compiled into machine codes like

```
reg1 ← [sum]
reg1 ← reg1+10
[sum] ← reg1
```

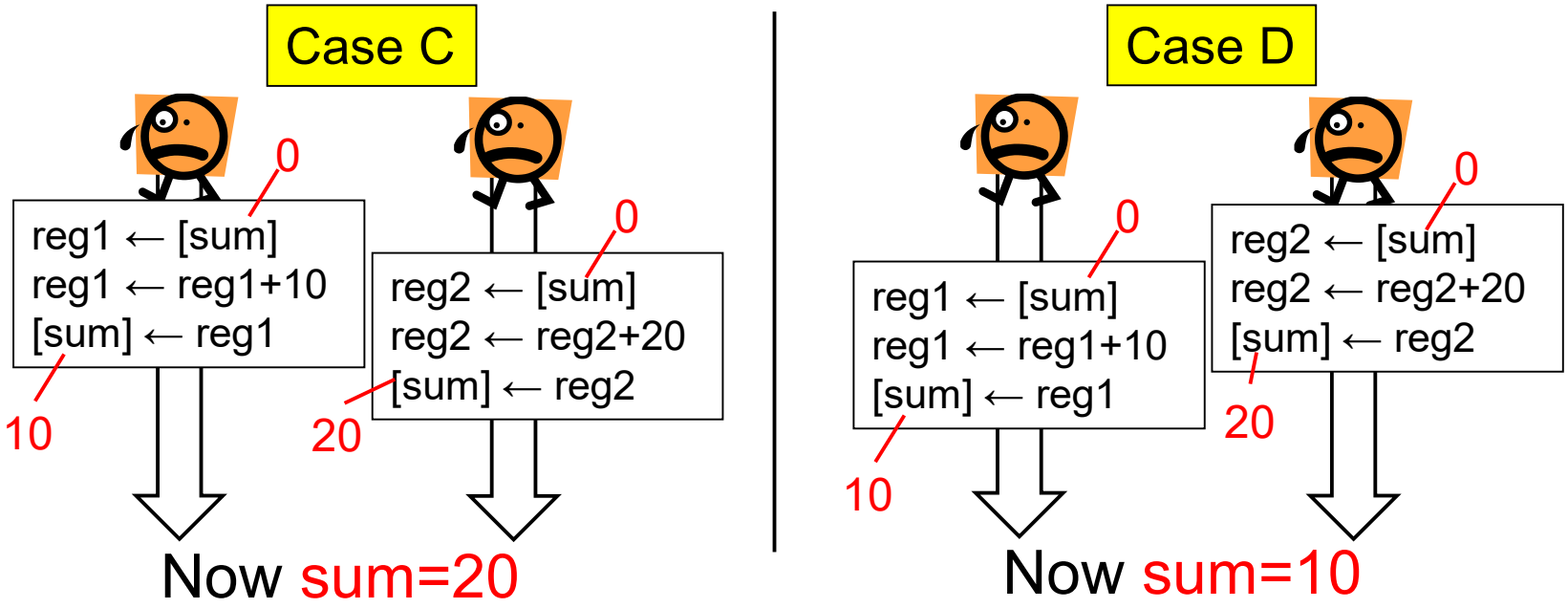
※ reg1, reg2… are registers, which are thread private



The results are same: sum=30. Ok to parallelize???

What's Wrong in pi-bad-omp? (2)

- No!!! The results can be different if C1 & C2 are executed (almost) simultaneously



The expected result is 30, but we may get bad results

Such a bad situation is called "Race Condition"

➔ Please try "pi-bad-omp"

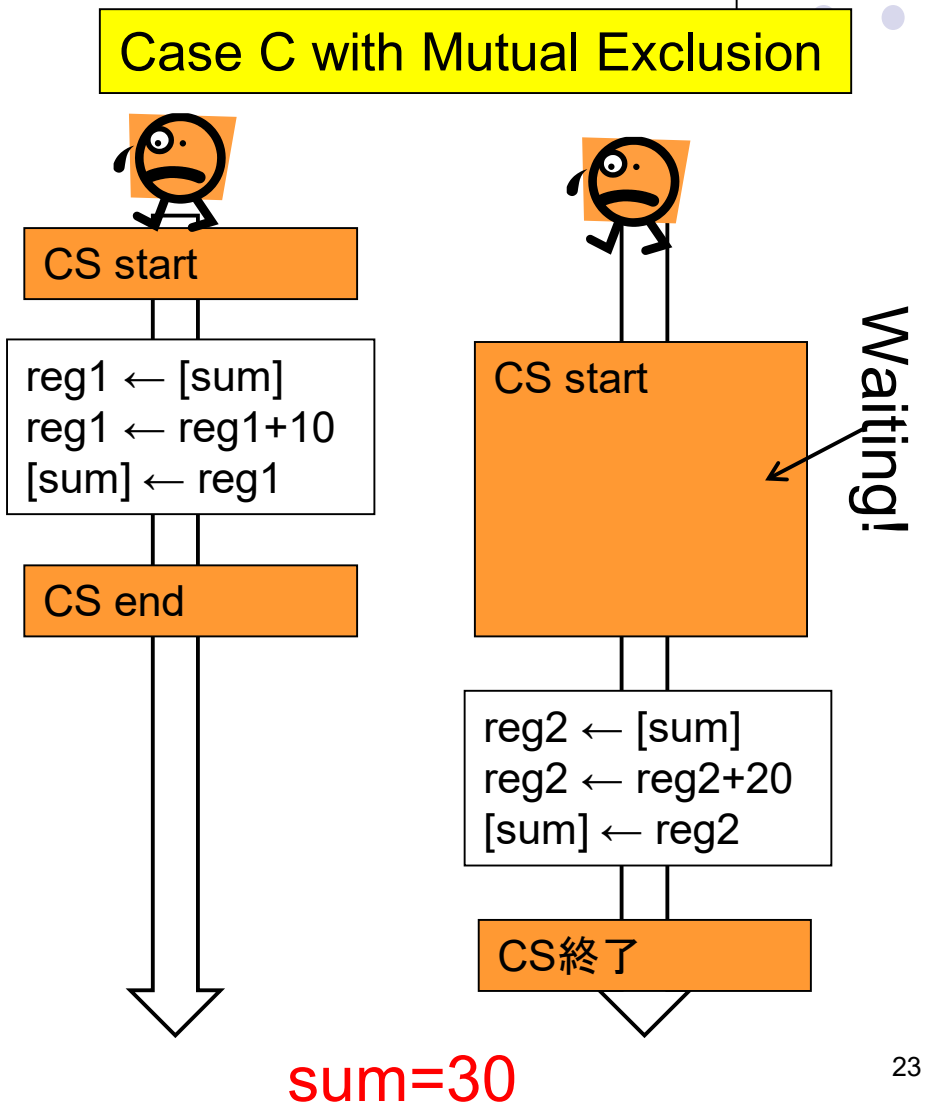
Mutual Exclusion to Avoid Race Condition

Mutual exclusion (mutex):

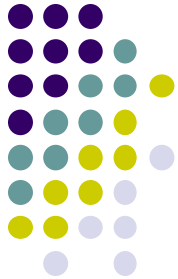
Mechanism to control threads so that only a single thread can enter a “specific region”

- The region is called critical section

⇒ With mutual exclusion, race condition is avoided



Mutual Exclusion in OpenMP



#pragma omp critical makes the following block/sentence be critical section

```
   double sum = 0;
#pragma omp parallel
   {
      [ do something ]
#pragma omp critical
      sum += myans;
   }
```

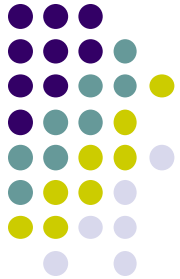
Please try "pi-good-omp"

cf) ./pi 100000000

- Computes integral by multiple threads
- The algorithm uses "*sum* += …"
- The answer is 3.1415…

But we see pi-good-omp is very slow ☹

Towards "Fast" Parallel Software



- If the entire algorithm is divided into independent computations (such as mm example), the story is easy
- But generally, most algorithms include both
  - Computations that can be parallelized
  - Computations that cannot (or hardly) be parallelized

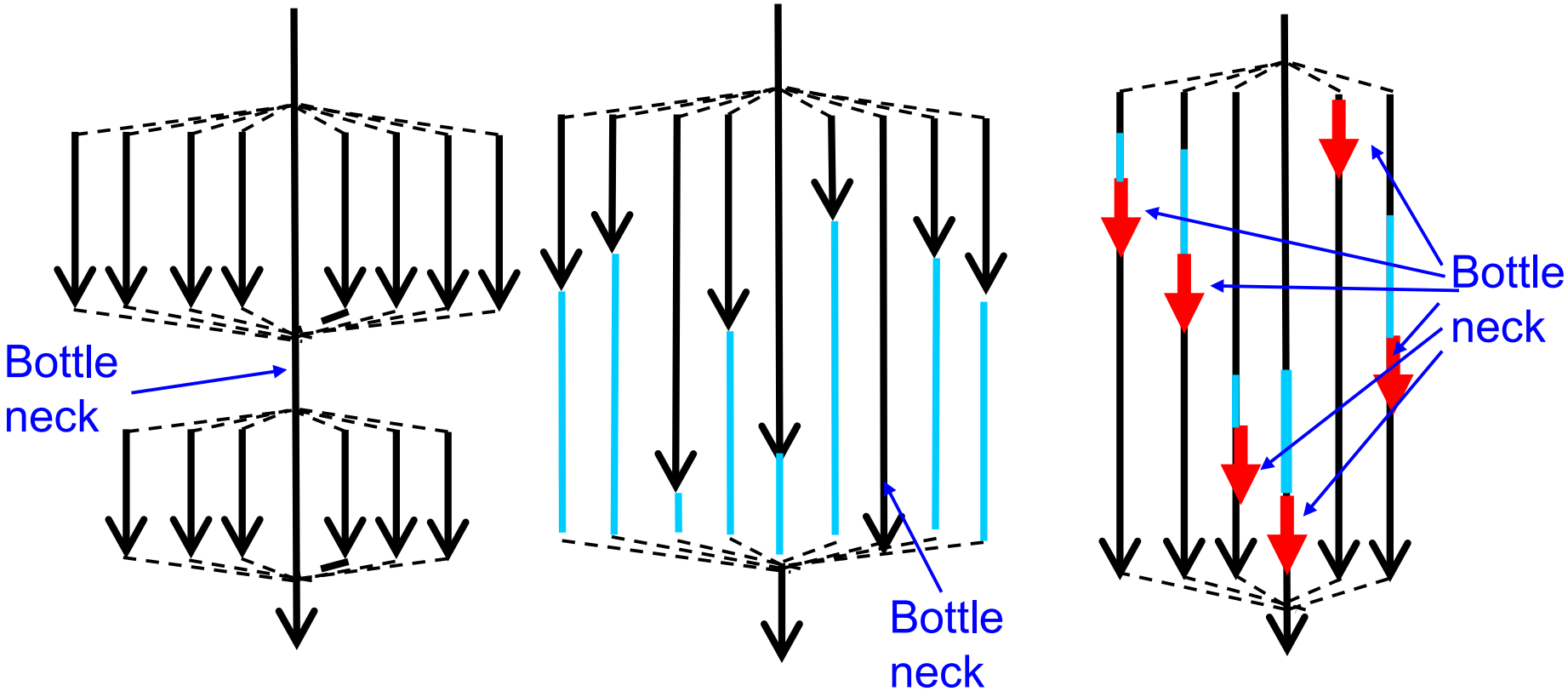
⇒ The later part raises problems called "bottleneck"

Various Bottlenecks

Bottleneck by sequential part

Bottleneck by load imbalance

Bottleneck by critical sections



Moreover, There are architectural bottlenecks

Amdahl's Law

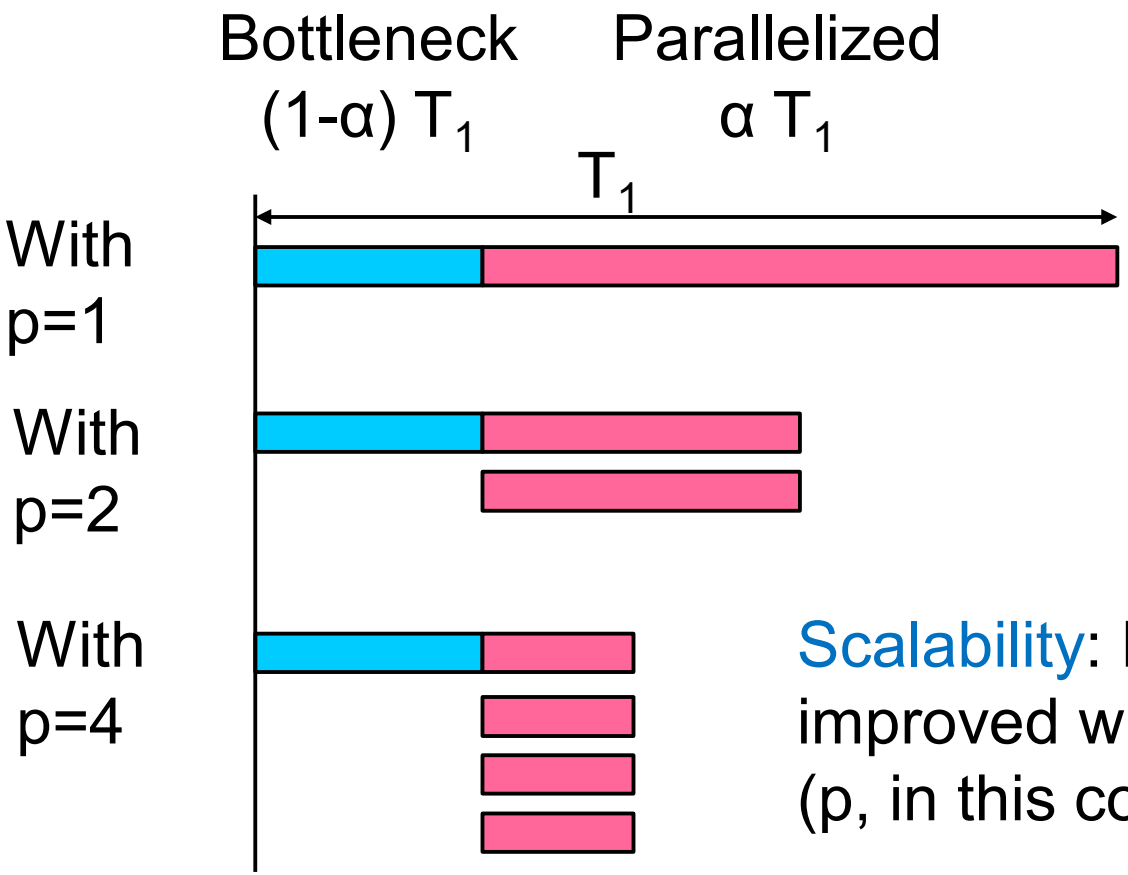
- We consider an algorithm. Then we let
  - T_1 : execution time with 1 processor core
  - α: ratio of computation that can be parallelized
  - 1-α : ratio that CANNOT be parallelized (bottleneck)

⇒ Estimated execution time with p processor cores is $T_p = ((1 - \alpha) + \alpha / p) T_1$

Due to bottleneck, there is limitation in speed-up no matter how many cores are used

$T_\infty = (1-\alpha) T_1$

An Illustration of Amdahl's Law

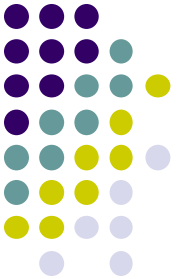


Scalability: How performance is improved with larger resources (p, in this context)

Amdahl's law tells us

- if we want scalability with p∼10, α should be >0.9
- if we want scalability with p∼100, α should be >0.99

The Fact is Harder Than Theory



- According to Amdahl's law, T_p is monotonically decreasing

→ Is large p always harmless ??

Performance comparison of pi-omp and pi-good-omp

export OMP_NUM_THREADS= [p]

./pi 100000000

p	pi-omp pi-fast-omp	pi-good-omp
1	0.80 (sec)	1.8 (sec)
2	0.40 (sec)	9.4 (sec)
5	0.16 (sec)	10.9~13.0 (sec)
10	0.08 (sec)	13~16 (sec)

Slower! ☹

Reducing bottleneck is even more important (than Amdahl's law tells)

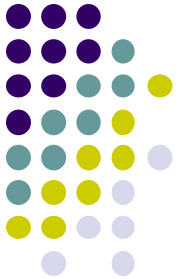
Reducing Bottlenecks

- Approaches for reducing bottlenecks depend on algorithms!
  - We need to consider, consider
  - Some algorithms are essentially difficult to be parallelized

- Some directions
  - Reducing access to shared variables
  - Reducing length of dependency chains
    - called “critical path”
  - Reducing parallelization costs
    - entering/exiting “omp parallel”, “omp critical”… is not free

:

Cases of “pi” Sample



- “pi-good-omp” is slow, since each thread enters a critical section too frequently

→ To improve this, another pi-fast-omp version introduces private variables

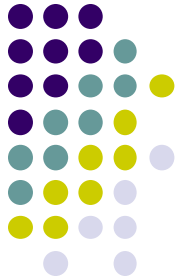
Step 1: Each thread accumulates values into private “local_sum”

Step 2: Then each thread does “sum += local_sum” in a critical section once per thread

→ pi-fast-omp is fast and scalable ☺

Why is pi-omp (the first omp version) also fast?
“omp for reduction(…)” is internally compiled to a similar code as above

Reduction Computations in "omp for"



- *"Summation in a for-loop"* is one of typical computations
  → called reduction computations
- In OpenMP, they can be integrated to "omp for"

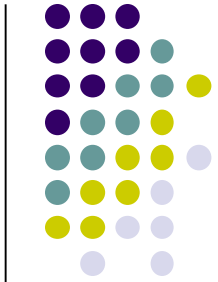
```
    double sum = 0.0;

#pragma omp parallel
#pragma omp for reduction (+:sum)
    for (i = 0; i < n; i++) {
        double x = (double)i * dx;
        double y = sqrt(1.0 - x*x);
        sum += dx*y;
    }
```

Operator is one of
+, -, *, &&, ||,
max, min, etc

Name of reduction variable

→ pi-omp is fast, like pi-fast-omp ☺
→ Also, programming is easier than pi-fast-omp ☺

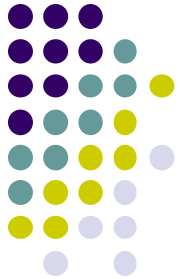
What We Have Learned in OpenMP Part



- OpenMP: A programming tool for parallel computation by using multiple processor cores
  - Shared memory parallel model
  - #pragma omp parallel → Parallel region
  - #pragma omp for → Parallelize for-loops
  - #pragma omp task → Task parallelism

- We can use multiple processor cores, but only in a single node node

Assignments in OpenMP Part (Abstract)



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

Due date: May 13 (Thu)

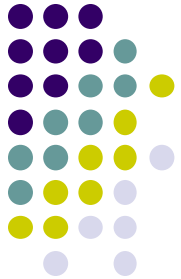
[O1] Parallelize “diffusion” sample program by OpenMP.
(/gs/hs1/tga-ppcomp/21/diffusion/ on TSUBAME)

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

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

For more detail, please see OpenMP (1) slides

Next Class:



- Part 2: GPU Programming (1) on May 6
  - What GPU programming is
  - Introduction to OpenACC