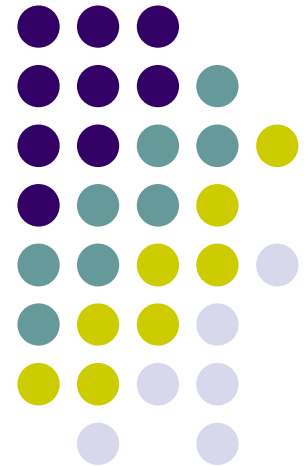# Practical Parallel Computing
(実践的並列コンピューティング)

2025 Class No.5
[OpenMP Part] (3)
Task Parallelism

Toshio Endo

endo@scrc.iir.isct.ac.jp

1

Overview of This Course

- Introduction Part
  - 2 classes
- OpenMP (OMP) Part
  - 4 classes   ← We are here (3/4)
  - Report (required)
- OpenACC (ACC) Part
  - 2 classes
  - Report (required)
- CUDA Part
  - 3 classes
  - Report (elective)
- MPI Part
  - 3 classes
  - Report (elective)

Slack channel
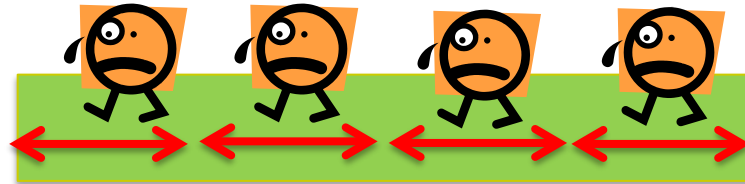in Science Tokyo Workspace
#dp-ppcomp-mcs-t418-2025

Today's Topic : Task Parallelism
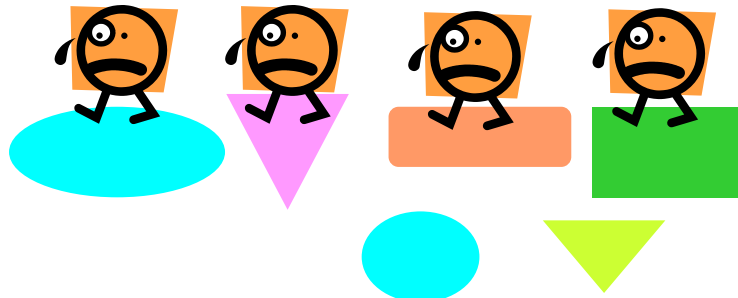# ~Comparison with Data Parallelism~

- Data Parallelism :
  - Every thread does uniform/similar tasks for different part of large data

cf) mm, diffusion samples

- Task Parallelism :
  - Each thread does different tasks
    - Sometimes the number of tasks is unknown beforehand
    - Sometimes tasks are generated recursively

cf) fib, sort samples today
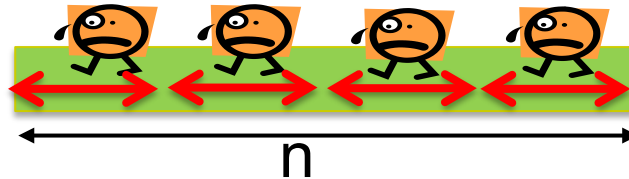
Related to assignment [O3]

Data Parallelism/Task Parallelism in OpenMP

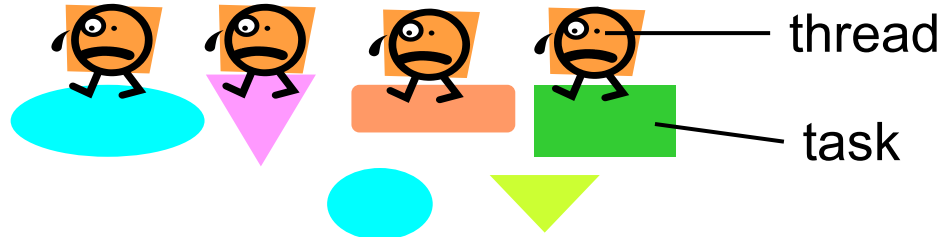- <u>#pragma omp for</u>
  - Used for <u>data</u> parallelism (basically)
  - Number of tasks is known before starting for-loop
    - for (i = 0; i < n; i++) … → *n* tasks are divided among threads
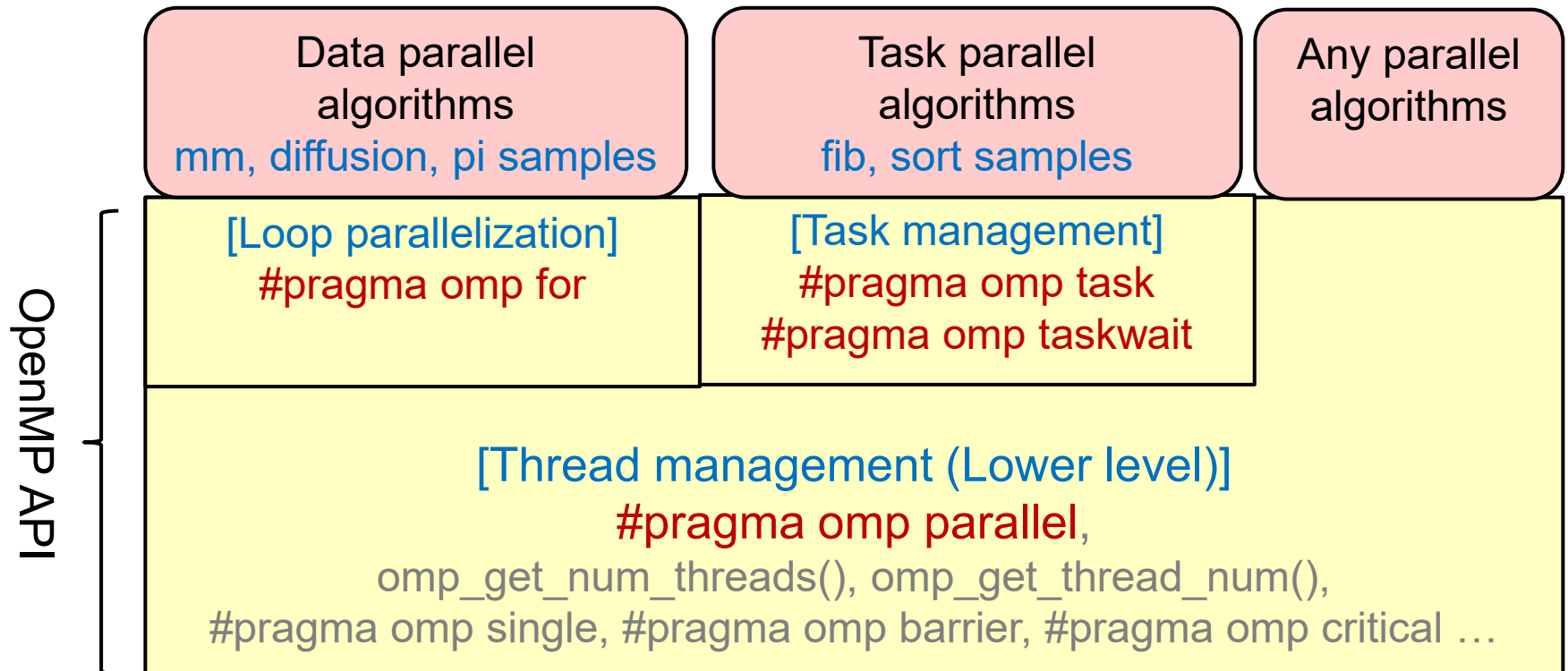

n

- <u>#pragma omp task</u>
  - Used for <u>task</u> parallelism (basically)
  - Number of tasks may change during execution

thread

task

Relationship of OpenMP Syntaxes

Data parallel algorithms mm, diffusion, pi samples	Task parallel algorithms fib, sort samples	Any parallel algorithms

OpenMP API

[Loop parallelization]
#pragma omp for

[Task management]
#pragma omp task
#pragma omp taskwait

[Thread management (Lower level)]
#pragma omp parallel,
omp_get_num_threads(), omp_get_thread_num(),
#pragma omp single, #pragma omp barrier, #pragma omp critical …

※ This grouping is different from official one
https://openmp.org/specifications/

task/taskwait Syntaxes

See a sample at ppcomp-ex/omp/tasks/
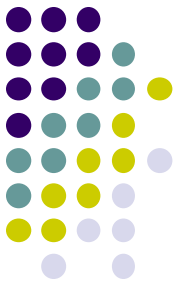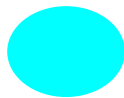
```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
  {
    A;
  }

#pragma omp task
    B;

    C;
#pragma omp taskwait
}
```

"task" syntax generates a task that executes the following block/sentence
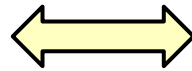
- A task is executed by one of threads who is idle (has nothing to do)
- New tasks and the original task may be executed in parallel
- Recursive task generation is ok
  - A parent task generates children tasks, and one of generates grandchildren…

"taskwait" syntax waits end of all children tasks

6

Relations between "Tasks" and "Threads"

Task A and task B are executed in parallel ⟷ Confusing ⟷ Thread A and thread B are executed in parallel
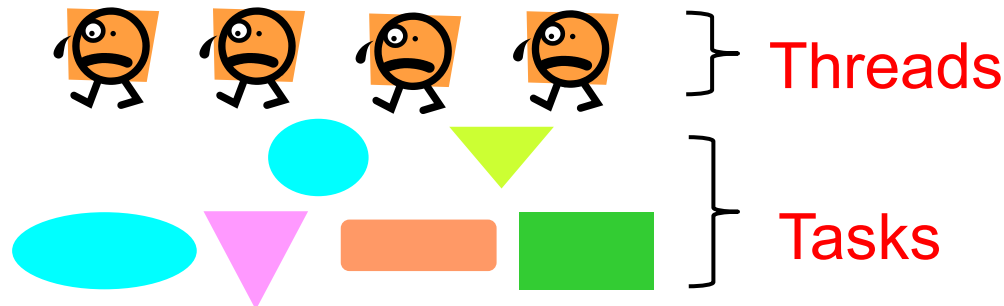
Each thread executes tasks one after another

- Number of threads is (basically) constant during a parallel region
  - OMP_NUM_THREADS, usually no more than number of processor cores
- Number of tasks may be changed frequently
  - may be >>number of processor cores
- When a thread becomes idle, it takes one of tasks and executes it

Threads

Tasks

Threads Executes Tasks
(see "omp/tasks" sample)


```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
    {
        A;
    }

#pragma omp task
        B;

        C;
#pragma omp taskwait
} (end of parallel region)
```

Exec C

Exec B

Exec A
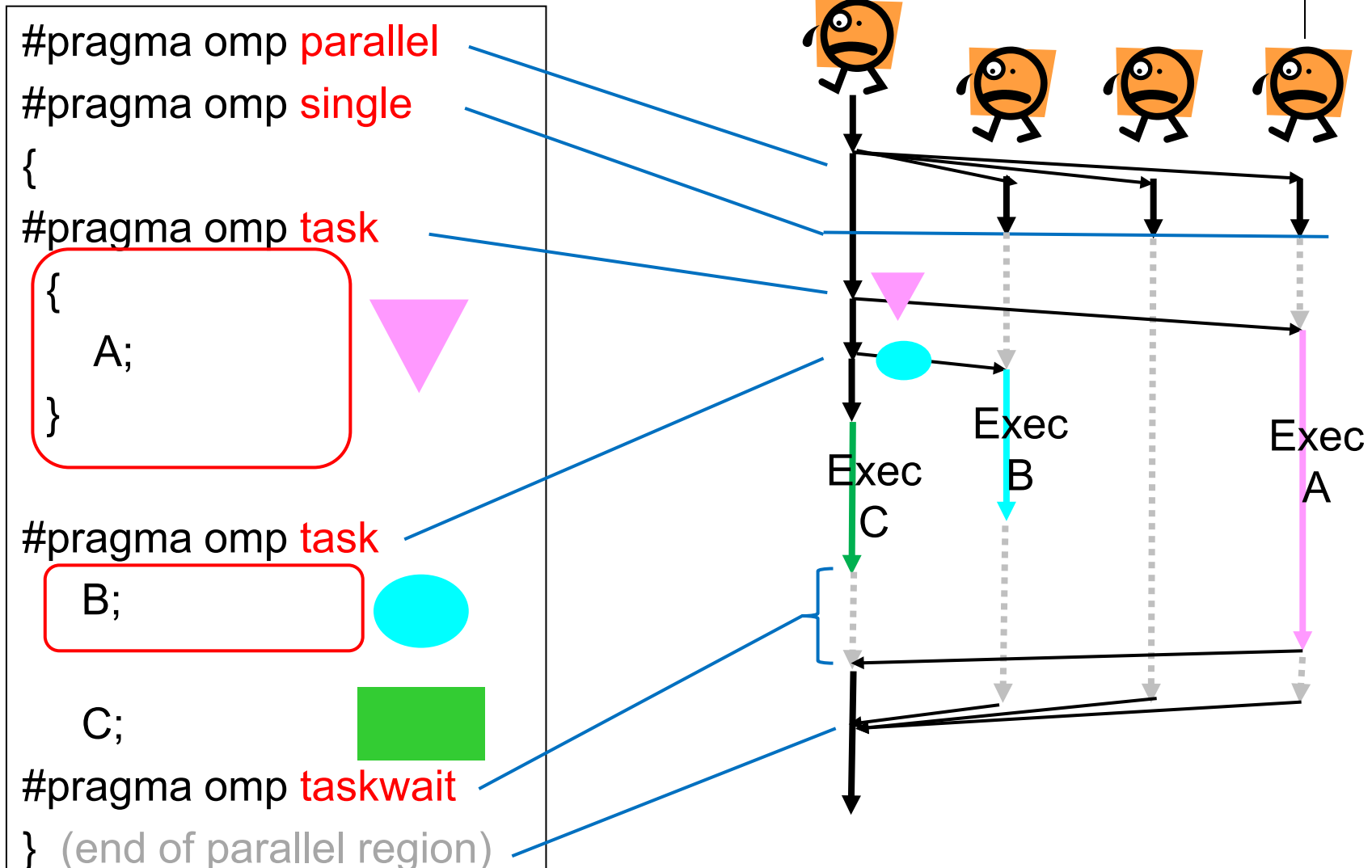
Note on Using "task" Syntax

- In OpenMP, tasks are taken and executed by idle threads

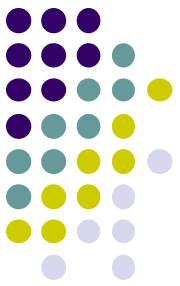→ We need to prepare idle threads before creating tasks

```
#pragma omp parallel        ←──Multiple threads start
#pragma omp single          ←──Only a single thread executes followings
 {                                (other threads become idle)


        :  (task generations)


 }                          ←──Parallel region finishes
```

[Q] What if we omit "omp parallel" & "omp single"?
→ There is 1 thread, which executes all tasks
→ No speed up! ☹

[Q] What if we omit "omp single"?
→ Every thread execute all tasks redundantly
→ No speed up! ☹

About Number of Tasks

- In the tasks-omp sample, there are 3 tasks in the world
  → No speed up with ≧ 4 threads

  "Too less tasks are bad ☹"

- To use threads (CPU cores) effectively, the number of tasks should be ≧ OMP_NUM_THREADS

  →Next, we see sample programs that generates plenty of tasks

  "Too much tasks are also bad ☹"

"fib" Sample Program

sequential version

- Available at ppcomp-ex/base/fib/
  - Go to the directory and "make"
- It calculates the *Fibonacci number*
  - fib(n) = fib(n-1) + fib(n-2)
  - 1, 1, 2, 3, 5, 8, 13…
- Execution: ./fib [n]
  - ./fib 40 → outputs 40th Fibonacci number
- Recursive function call is used
  - It is an inefficient algorithm as a sample
- Computational complexity: O(fib(n)) $\fallingdotseq$ O(1.618^n)
  - It is unknown before execution
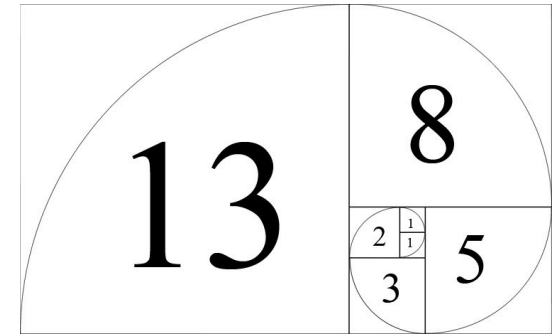
Using fib Sample

- This assumes you have downloaded samples from https://github.com/toshioendo/ppcomp-ex.git
  - (details are in ppcomp25-2 slides)

```
cd        [If you copied sample in sub-directory, go there]
cd ppcomp-ex/base/fib
ls
[you will see 3 files of fib.c, Makefile, job.sh]
make      [this creates an executable file]
./fib 40
```

We will use omp/fib-slow and omp/fib later

Algorithm of fib

```
long fib(int n)
{
  long f1, f2;
  if (n <= 1) return n;


  f1 = fib(n-1);



  f2 = fib(n-2);




  return f1+f2;
}
```

Recursive call is used

- fib(n) depends on fib(n-1) and fib(n-2)

recursive call is stopped

recursive call

OpenMP Version of fib (version 1)

```
long fib_r(int n)
{
  long f1, f2;
  if (n <= 1) return n;

#pragma omp task shared(f1)
  f1 = fib_r(n-1);

#pragma omp task shared(f2)
  f2 = fib_r(n-2);

#pragma omp taskwait

  return f1+f2;
}
```

Available at

ppcomp-ex/omp/fib-slow

● In this version,

     a task = recursive call

Tasks are generated
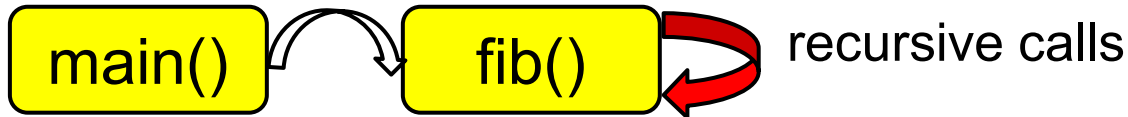
We wait for completion of the above 2 tasks

Don't forget "omp taskwait"

Note on
omp parallel → omp single

- We need "omp parallel & omp single" <u>only once</u>, but where?

(Sequential) fib

main() → fib() ↻ recursive calls

fib-slow-omp

main() → fib() → fib_r() ↻ recursive calls& task generation
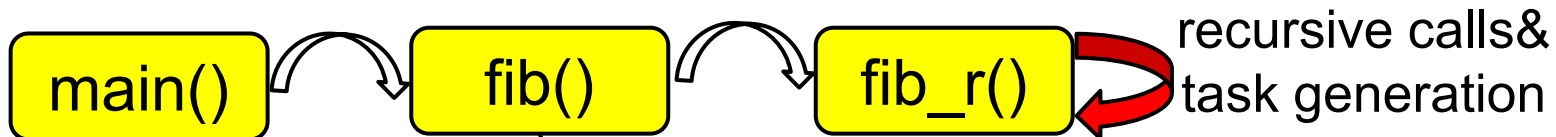
omp parallel & omp single here

Rules about Variables

In default, _copies_ of variables are created for each child task

- The value of "*n*" is brought from parent to a child task
  - → OK ☺

- But a child has a only copy → update to "f1" or "f2" is not visible to parent. NG! ☹

"shared(var)" option makes the variable "var" be shared between parent and the child

- Using it, update to "f1" or "f2" is visible to parent

The First Version is Too Slow

Execution time of ./fib 40
- On a TSUBAME4.0 interactive node (24cores)

fib

1	threads
0.60	seconds

fib-slow
-omp

1	2	4	threads
13	~140	~250	seconds

- OpenMP version is much slower than original fib ☹
  - Even with 1 thread, 20x slower
- Also it is much slower with multi-threads
- → How can we improve?

Pitfall in "task" Syntax

- While OpenMP allows to generate many tasks, task generation cost is not negligible

Rough comparison：

| Function call cost | << | Task generation cost | << | Thread generation cost |

- In version 1, "./fib n" generates O(fib(n)) tasks

→ Creating too much tasks is bad!

- How can we reduce the number of tasks?

OpenMP Version of fib (version 2)

```
long fib_r(int n)
{
  long f1, f2;
  if (n <= 1) return n;

  if (n <= 30) {
    f1 = fib_r(n-1);
    f2 = fib_r(n-2);
  }
  else {
#pragma omp task shared(f1)
    f1 = fib_r(n-1);
#pragma omp task shared(f2)
    f2 = fib_r(n-2);
#pragma omp taskwait
  }
  return f1+f2;
}
```

if *n* is "sufficiently" small, we do not generate tasks

Available at
ppcomp-ex/omp/fib

- To avoid generating too many tasks, we compare n and a threshold (=30 here)
- If n is large, we generate tasks
- If n is small, we do not generate
- Changing threshold (=30) would affect performance

Performance of Version 2

Execution time of ./fib 40

fib

1	threads
0.50	seconds

fib-slow -omp

1	2	4	8	24	threads
13	~140	~250	-	-	seconds

fib-omp

| 1	2	4	8	24	threads
0.50	0.31	0.19	0.11	0.05	seconds

- Performance of Version 2 is largely improved and more stable

→ Restricting task generation is important for speed

"qsort" Sample Program Related to Assignment [O3]

Available at ppcomp-ex/base/qsort

- Execution: ./qsort [n]
- It sorts an array of length *n* by the quick sort algorithm
  - Array elements have double type
- Compute Complexity: O(n log n) on average
  - More efficient than $O(n^2)$ algorithm such as bubble sort

n

0.84	0.39	0.78	0.80	0.91	0.20	0.34	0.77

| 0.20	0.34	0.39	0.77	0.78	0.80	0.84	0.91
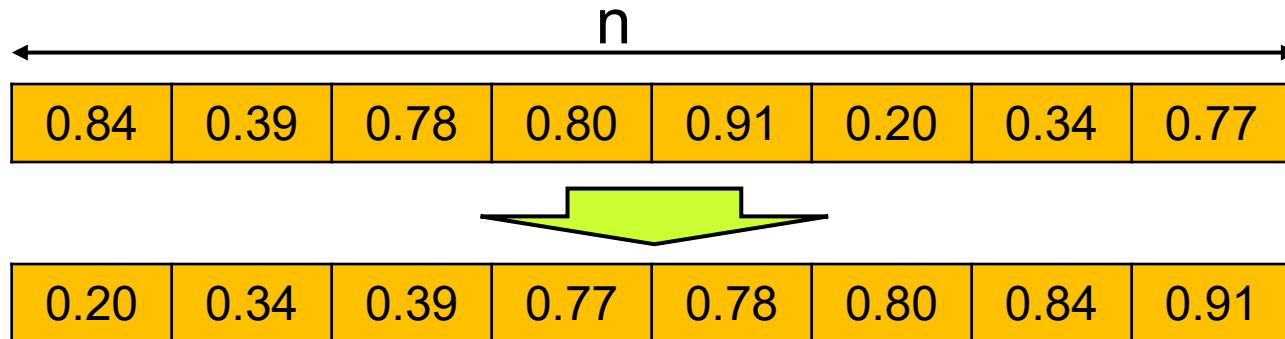
There is also ppcomp-ex/omp/qsort
- [NOTE] qsort.c is not parallelized
- You can use it as a start point

Quick Sort

- A recursive algorithm
  - Take a value, called "pivot" from the array
  - Partition array into two parts, "small" and "large"
  - "small" part and "large" part are sorted recursively

Smaller values than "pivot"   Larger values than "pivot"

O(log n)
depth
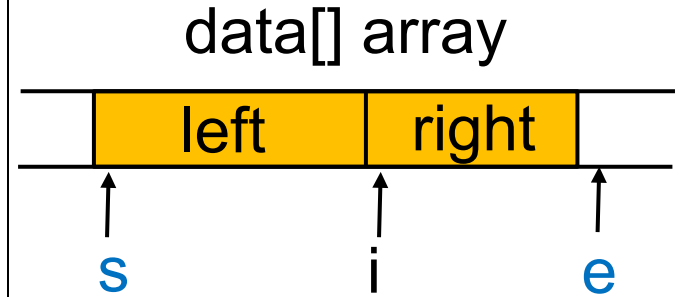on average
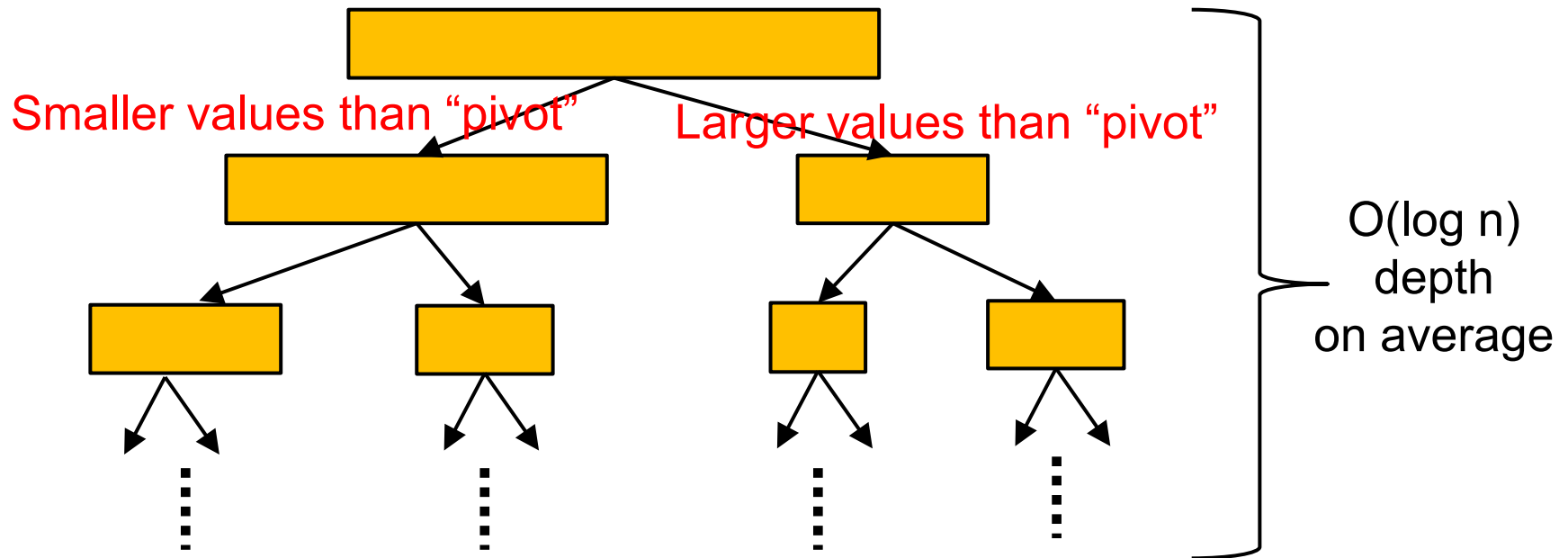
Structure of sort Sample

```
int sort(double *data, int s, int e)
{
  int i, j;
  double pivot;
  if (e-s <= 1)  return 0;

  /* pivot selection */
      :

  /* partition data[ ] into 2 parts */
      :
  /* Here "i" is boundary of 2 parts */

  sort(data, s, i);   /* Sort left part recursively*/
  sort(data, i, e);  /* Sort right part recursively */
}
```

data[] array

| left | right |

s   i   e

Harder to parallelize

Generating 2 tasks
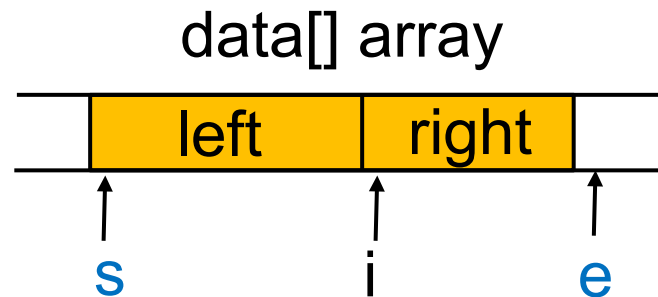would be a good idea

[Q] How can we restrict too much task generation?

Is it Correct to Parallelize Recursive Calls in sort?

C1
C2

```
                .
sort(data, s, i);   /* Sort left part recursively*/
sort(data, i, e);   /* Sort right part recursively */
```

- Let us discuss why computations C1 and C2 can be parallelized
  - Analyze read-set R and write-set W of each

data[] array

| left | right |

s   i   e

- $R(C1) = W(C1) = \{data[s], data[s+1], \ldots data[i-1]\}$ ⎱ Disjoint
- $R(C2) = W(C2) = \{data[i], data[i+1], \ldots data[e-1]\}$ ⎰ → independent!

Even with recursive task generations, this discussion can be applied

[Revisited] When We Can Use "omp for"

- Loops with some (complex) forms cannot be supported, unfortunately ☹

- The target loop must be in the following form

```
#pragma omp for
  for (i = value; i op value; incr-part)
    body
```

```
"op" : <, >, <=, >=, etc.
"incr-part" : i++, i--, i+=c, i-=c, etc.
```

```
OK ☺:   for (x = n; x >= 0; x-=4)
NG ☹:   for (i = 0; test(i); i++)
NG ☹:   for (p = head; p != NULL; p = p->next)
```

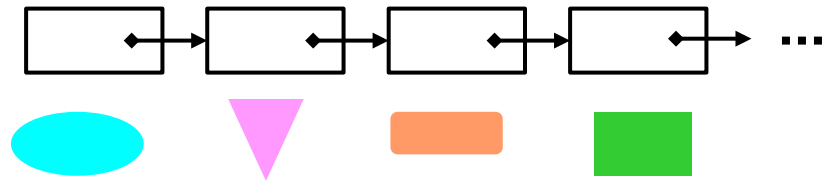⟹ *Instead, we can parallelize it with "task" syntax*

Parallelize Irregular Loops with "task" Syntax

- In list search, number of iterations cannot be known before execution → we can use "task"

```
#pragma omp parallel
#pragma omp single
  {
      for (p = head; p != NULL;
          p = p->next) {
#pragma omp task
          [Do something with p]
      }
#pragma omp taskwait
  }
```

- A task for one list node = one OpenMP task

Note：
- The number of generated tasks = List length.
→ Consider total task generation costs

- Using TSUBAME job scheduler:
  - ppcomp-sup slides

Assignments in OpenMP Part (Abstract)

Choose one of [O1]—[O4], and submit a report

Due date: May 1 (Thu)

[O1] Parallelize "diffusion" sample program by OpenMP.

[O2] Parallelize "bsort" sample program by OpenMP.

[O3] Parallelize "qsort" sample program by OpenMP.

[O4] (Freestyle) Parallelize *any* program by OpenMP.

For more detail, please see ppcomp25-3 slides

Plan of OMP Part

- Class #3
  - Introduction to OpenMP
- Class #4
  - Data parallelism with for loops
  - diffusion sample [O1], bsort sample [O2]
- Class #5 (Today)
  - Task parallelism
  - qsort sample [O3]
- Class #6
  - What are bottlenecks, race conditions?