2019
Practical Parallel Computing
(実践的並列コンピューティング)
No. 3

Shared Memory Parallel Programming with OpenMP (1)

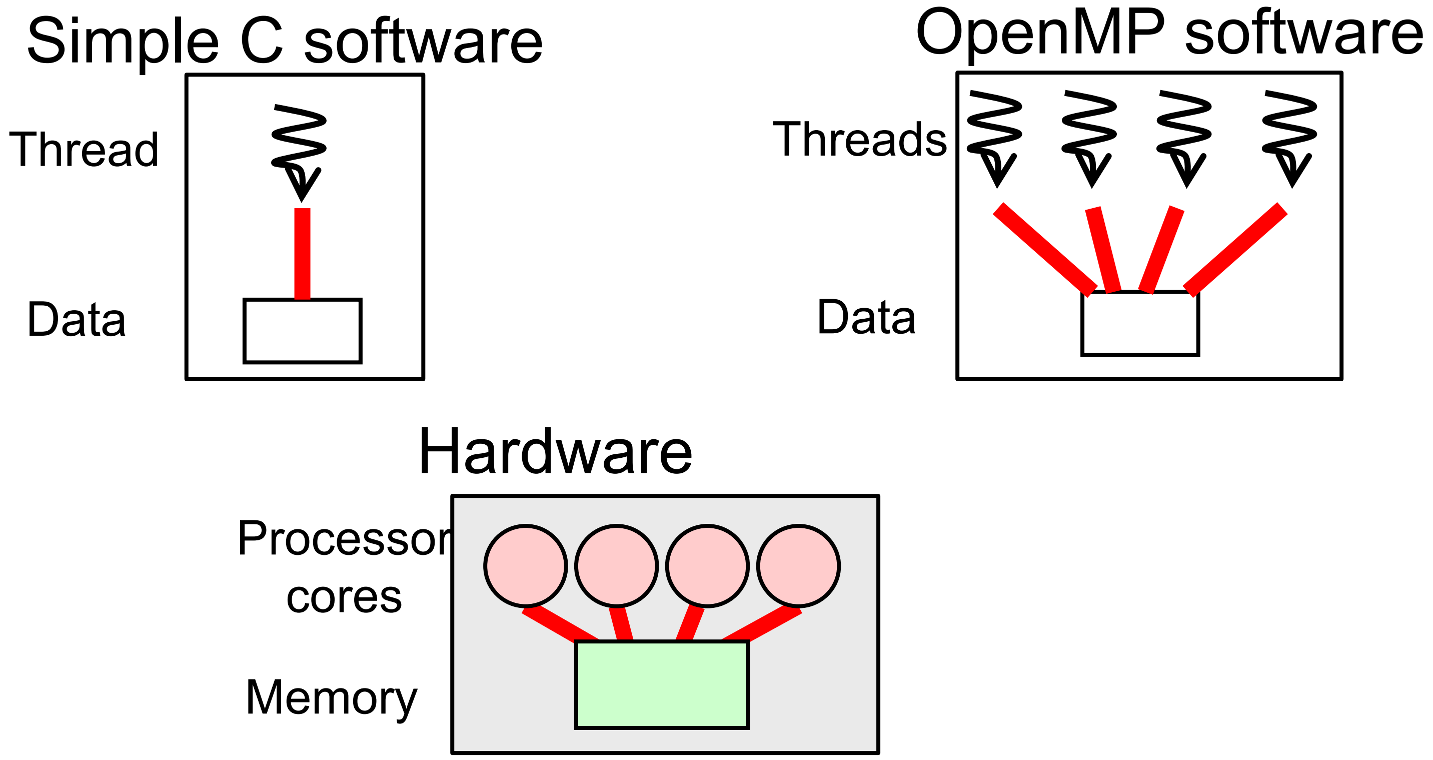
Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

What is OpenMP?

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



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

See ~endo-t-ac/ppcomp/19/ on TSUBAME

(1) There are several sub directories

- Pi (pi, pi-omp)
- Matrix multiply (mm, mm-omp)

(1) Copy them to (anywhere in) your own home directory

- Cf) cp –r ~endo-t-ac/ppcomp/19/pi-omp .

(2) Executable binaries are generated by “make” command in each sub-directory

Compiling OpenMP Programs

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

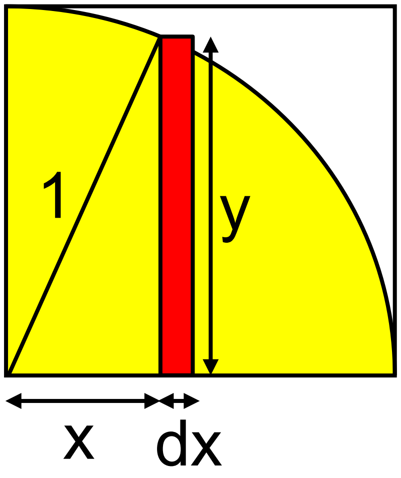
- gcc
  - -fopenmp option in compiling and linking
- PGI compiler
  - module load pgi, and then use pgcc
  - -mp option in compiling and linking
- Intel compiler
  - module load intel, and then use icc
  - -openmp option in compiling and linking

Also see outputs of “make” in OpenMP sample directory

“pi” sample

Estimate approximation of π (circumference/diameter) by approximation of integration

- Sequential version in “pi”, OpenMP version in “pi-omp”
- Method
  - Let SUM be approximation of the yellow area
  - 4 x PR → π
- Execution：./pi [n]
  - n: Number of division
  - Cf) ./pi 100000000
- Compute complexity： O(n)



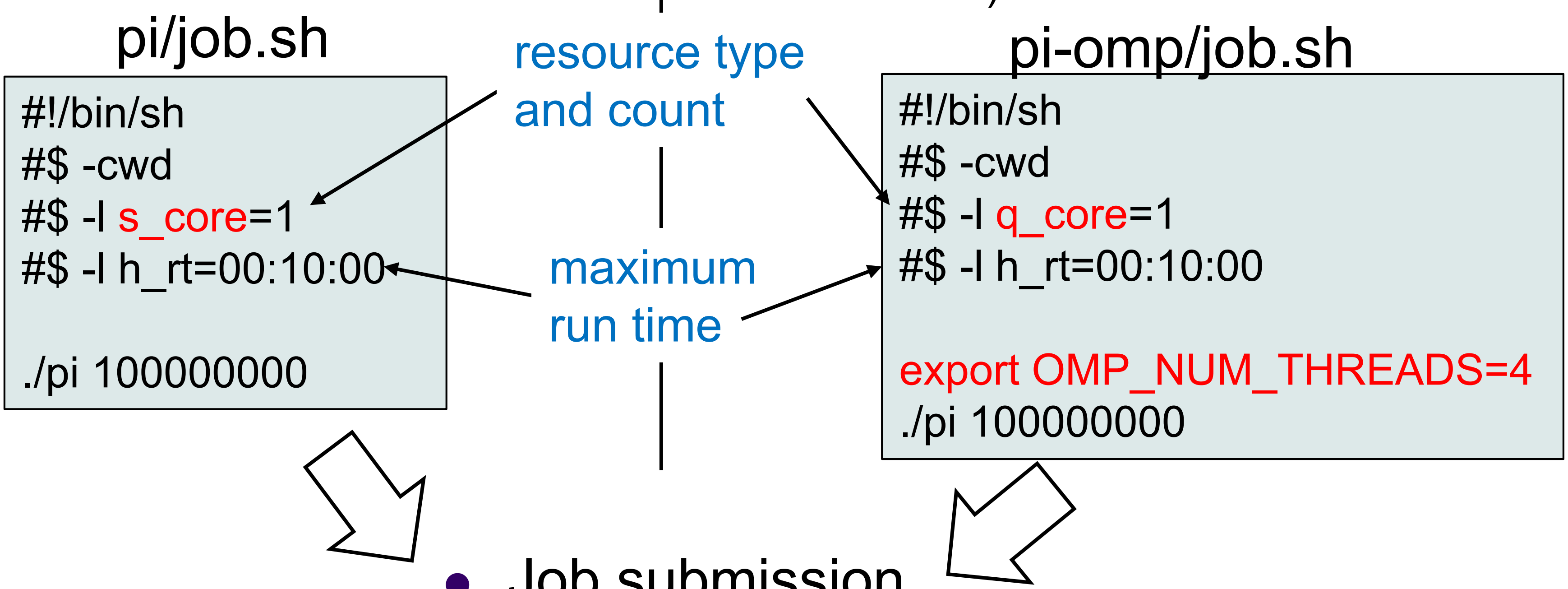
dx = 1/n
y = sqrt(1-x*x)

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

Submitting a Job to TSUBAME
~ in case of pi sample ~

- Sequential version
  - see pi directory

- OpenMP version
  - see pi-omp directory
  - in the case with 4 threads (4 processor cores)



pi/job.sh

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

./pi 100000000
```

resource type and count

maximum run time

pi-omp/job.sh

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

export OMP_NUM_THREADS=4
./pi 100000000
```

- Job submission
  - qsub job.sh

Notes on Job Submission (1)

There are several notes since TSUBAME is a shared system

- Please specify the resource type properly, according to the number of threads (CPU cores)
  - s_core: 1 core
  - q_core: 4 cores
  - q_node: 7 cores (+ 1GPU)
  - h_node: 14 cores (+ 2GPUs)
  - f_node: 28 nores (+ 4GPUs)

  For detail, see TSUBAME3.0 User’s Guide (利用の手引き) Section 5.1

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” for accounting (charged/有料)

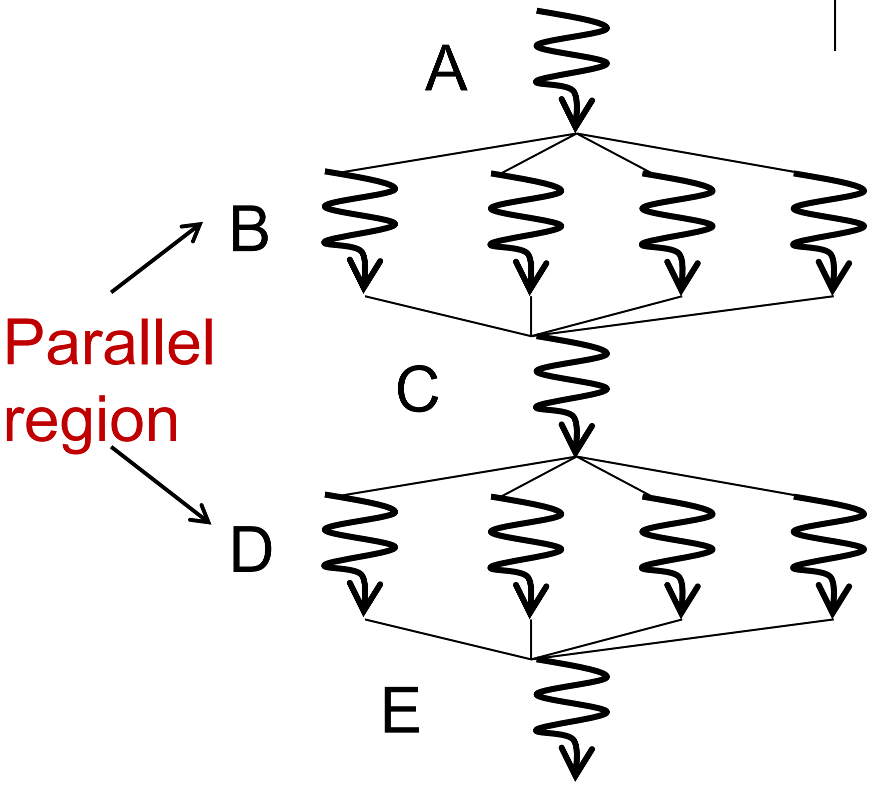
    ```
    qsub –g tga-ppcomp job.sh
    ```

  - 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
- Please do not execute CPU intensive programs on login nodes
  - It is OK to edit programs, compile programs, and submit jobs, and so on

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=4

    in command line

- Obtain number of threads
  - cf) n = omp_get_num_threads();

- Obtain "my ID" of calling thread
  - cf) id = omp_get_thread_num();
    - 0 ≦ id < n (total number)

#pragma omp for for Easy Parallel Programming

"for" loop with simple forms can parallelized easily

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

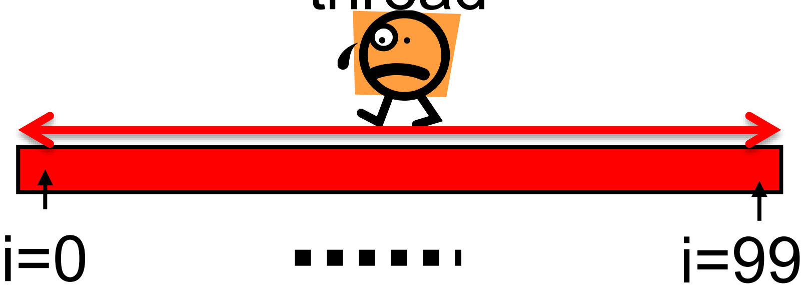
- "for" loop right after "omp for" is parallelized, with work distribution

- When this sample is executed with 4 threads, each thread take 100/4=25 iterations → speed up!!
  - Indivisible cases are ok, such as 7 threads

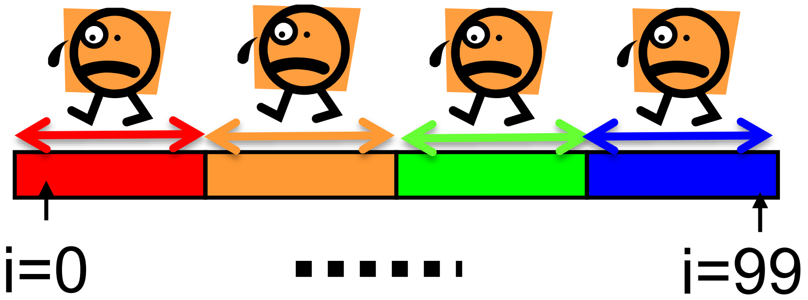
- Abbreviation: omp parallel + omp for = omp parallel for

Why “omp for” Reduces Execution Time

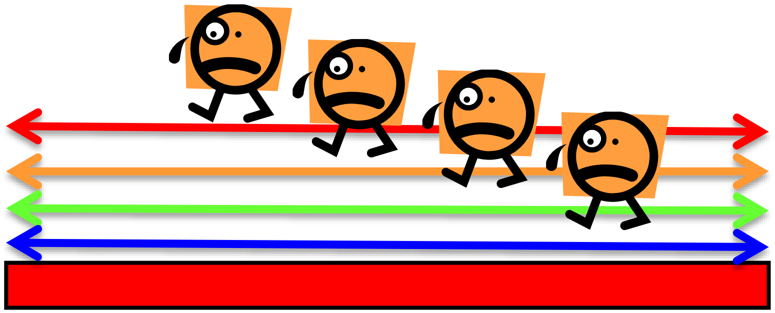
Without OpenMP thread



With “omp parallel” & “omp for”



- 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 ☹

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)
```

Advanced Topic on “omp for” (1): reduction

- Typical code pattern in for loop: Aggregate result of each iteration into a single variable, called reduction variable
  - cf) We add +1 to “count” variable in pi-omp sample
  - For such cases, “reduction” option is required

```
   int count = 0;
#pragma omp parallel
   {
#pragma omp for reduction (+:count)
     for (i = 0; i < 100; i++) {
        count += f(i);
     }
   }
```

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

Name of reduction variable

If we forget to write “reduction” option → The answer would be wrong

Advanced Topic on "omp for" (2): schedule

- Usually, each thread takes iterations uniformly
  - cf) 1000 iterations / 4 threads = 250 iteration per thread
- For some computations (execution times per iteration are varying), the default schedule may degrade performance

  #pragma omp for schedule(…) may improve

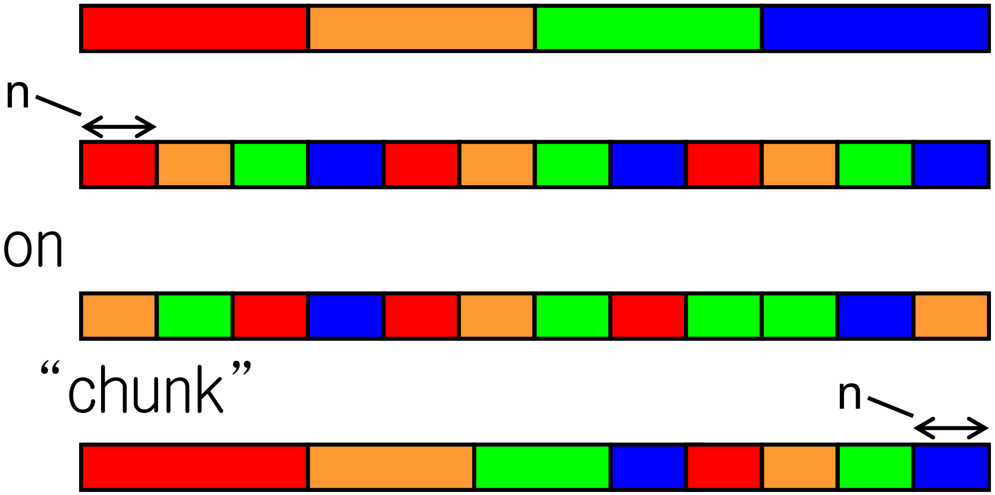
- schedule(static)

  uniform (default)
- schedule(static, n)

  block cyclic distribution
- schedule(dynamic, n)

  idle thread takes next "chunk"
- schedule(guided, n)

  "chunk" size gets smaller as the advance



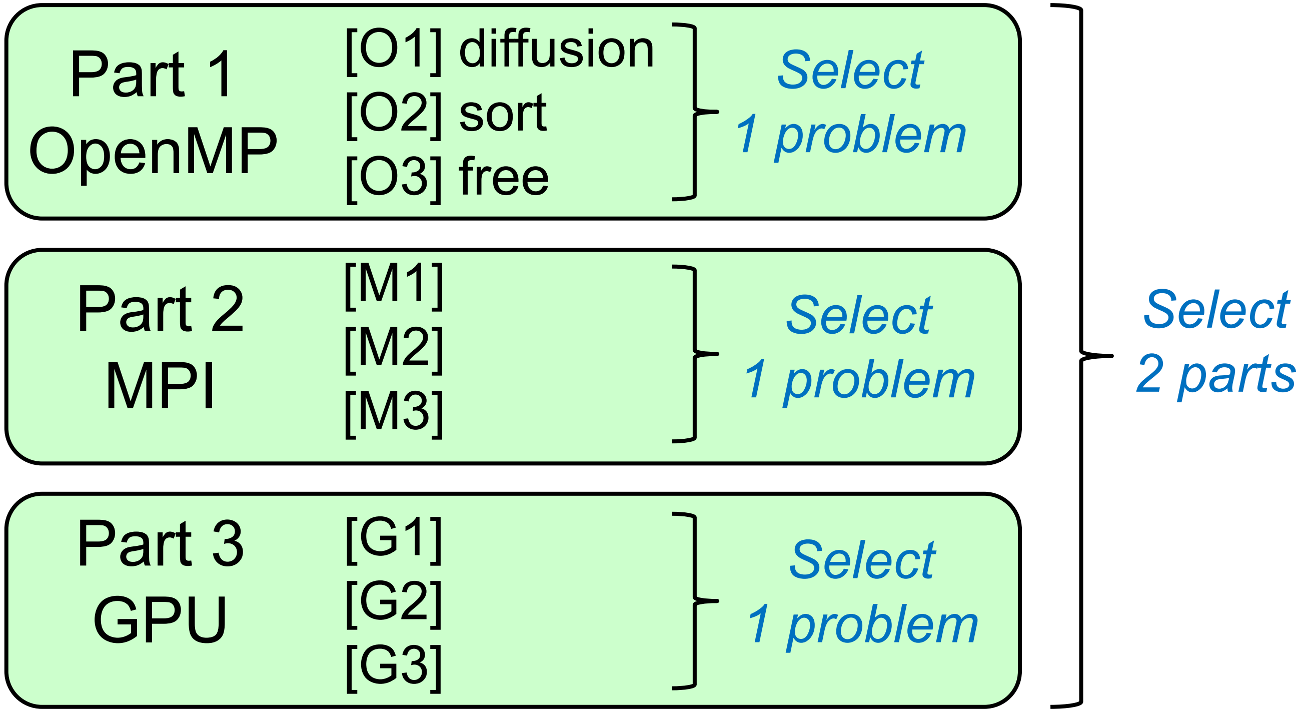
Time Measurement in Samples

- gettimeofday() function is used
  - It provides wall-clock time, not CPU time
  - Time resolution is better than clock()

```
#include <stdio.h>
#include <sys/time.h>
     :
{
   struct timeval st, et;
   long us;
   gettimeofday(&st, NULL); /* Starting time */
   ・・・Part for measurement ・・・
   gettimeofday(&et, NULL); /* Finishing time */
   us = (et.tv_sec-st.tv_sec)*1000000+
        (et.tv_usec-st.tv_usec);
   /* us is difference between st & et in microseconds */
}
```

Assignments in this Course

- There is homework for each part. Submissions of reports for 2 parts are required
- Also attendances will be considered



Part 1 OpenMP

[O1] diffusion
[O2] sort
[O3] free

Select 1 problem

Part 2 MPI

[M1]
[M2]
[M3]

Select 1 problem

Part 3 GPU

[G1]
[G2]
[G3]

Select 1 problem

Select 2 parts

Assignments in OpenMP Part (1)

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

Due date: May 9 (Thursday)

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

(~endo-t-ac/ppcomp/19/diffusion/ on TSUBAME)

Optional：

- Make array sizes variable parameters, which are specified by execution options. "malloc" will be needed.
- Improve performance further. Blocking, SIMD instructions, etc, may help.

Assignments in OpenMP Part (2)

[O2] Parallelize “sort” sample program by OpenMP.

(~endo-t-ac/ppcomp/19/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 Submission

- Submit the followings via OCW-i

  (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

- 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 processor cores
    - With varying problem sizes
    - Discussion with your findings
    - Other machines than TSUBAME are ok, if available

Next Class:

- OpenMP(2)
  - mm: matrix multiply sample
  - diffusion : heat diffusion sample using stencil computation
    - Related to assignment [O1]

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