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

Part 1: OpenMP

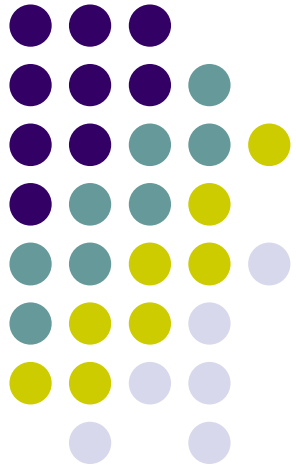
No 2: Diffusion Sample

Apr 20, 2023

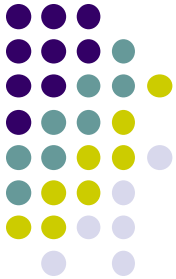
Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

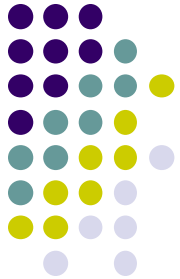


Overview of This Course



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

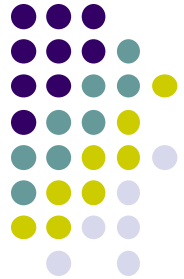
Summary of Previous Class



OpenMP is for shared-memory parallel programming

- #pragma omp parallel defines a parallel region, where multiple threads work simultaneously
- With #pragma omp for, loop-based programs can be parallelized easily
- Shared variables and private variables
- We have reviewed OpenMP version of mm sample

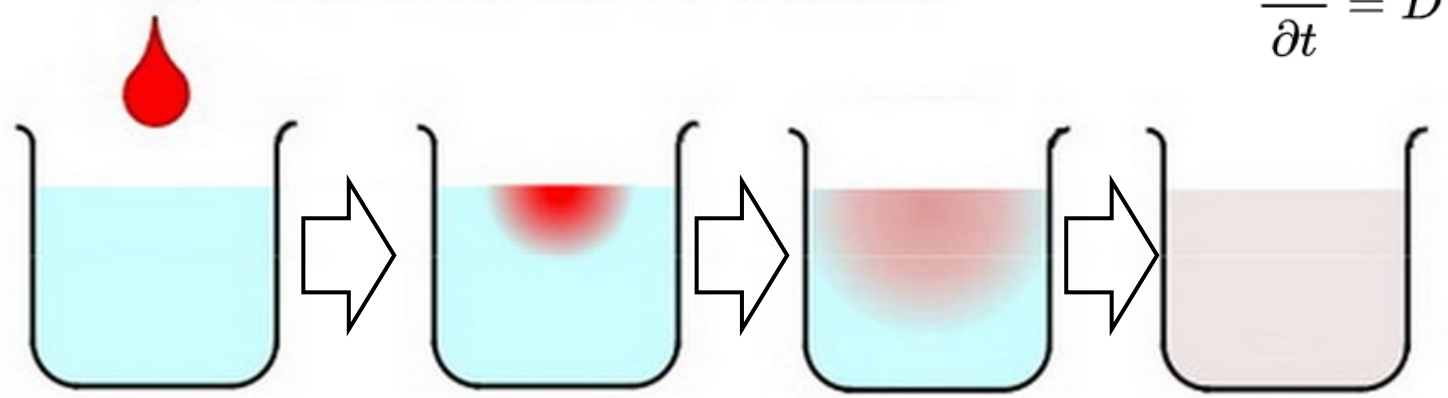
“diffusion” Sample Program



An example of diffusion phenomena:

- Pour a drop of ink into a water glass

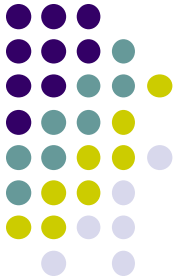
$$\frac{\partial \phi}{\partial t} = D\nabla^2 \phi(\vec{r}, t)$$



The ink spreads gradually, and finally the density becomes uniform (Figure by Prof. T. Aoki, GSIC)

- Density of ink in each point vary according to time → Simulated by computers
  - cf) Weather forecast compute wind speed, temperature, air pressure…

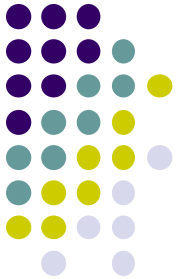
“diffusion” Sample on TSUBAME



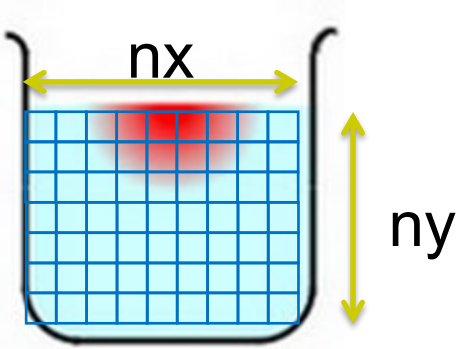
Available at /gs/hs1/tga-ppcomp/23/diffusion/

- Execution：./diffusion [nt]
- nt: Number of time steps
- nx, ny: Space grid size
  - nx=8192, ny=8192 (Fixed. See the code)
  - How can we make them variables? (See mm sample)
- Compute Complexity：O(nx × ny × nt)

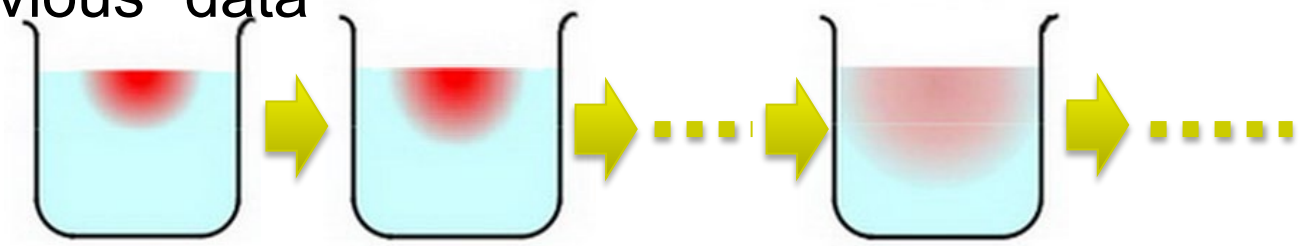
Expression of Space to be Simulated



- Space to be simulated are divided into grids, and expressed by arrays (2D in this sample)



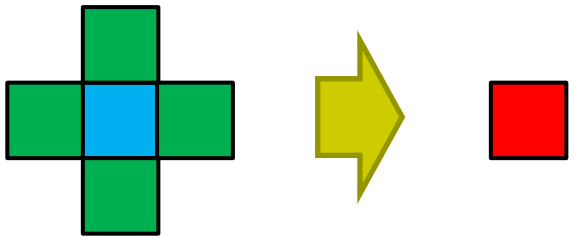
- Array elements are computed via timestep, by using “previous” data



Time step t=0 t=1 t=20

Stencil Computations

- A data point (*x,y*) at time *t+1* is computed using following data
  - point (*x,y*) at time t
  - "Neighbor" points of (*x,y*) at time t

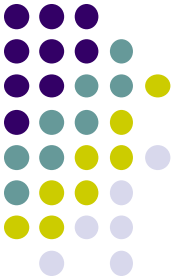


time t time t+1

  - In diffusion sample, the computation is simply "average of 5 points"
- Computations of similar type are called "stencil computations"
  - Frequently used in fluid simulations

Original meanings of "stencil"

Initial Conditions & Boundary Conditions



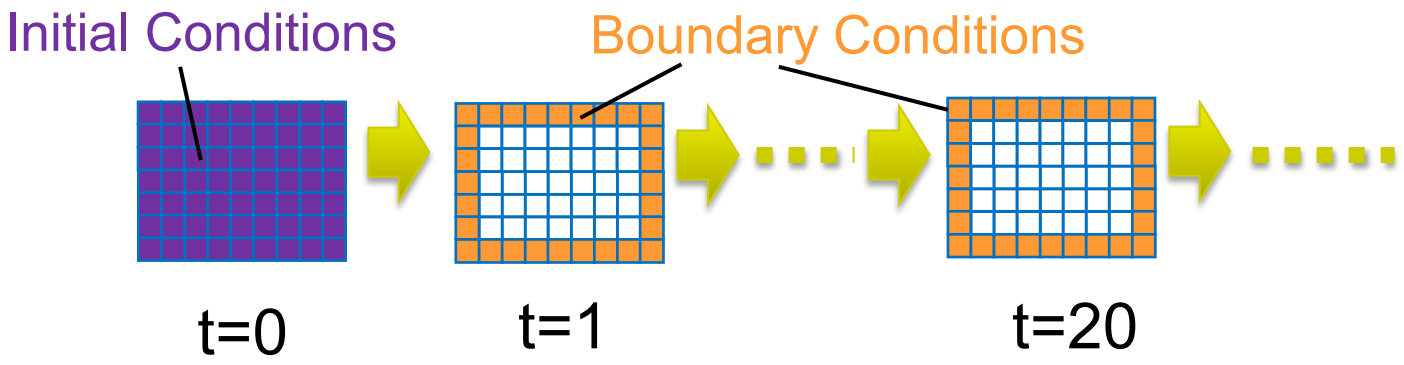
In stencil computations, following data points cannot be computed

Instead, we have to give them (for example, as input data)

- All points at t=0 (Initial conditions)
  - In diffusion sample, given in init()
- "Boundary" points for all t (Boundary conditions)
  - In diffusion sample, they are constant during simulation

→ See ranges of for-loops in calc(); boundaries are skipped

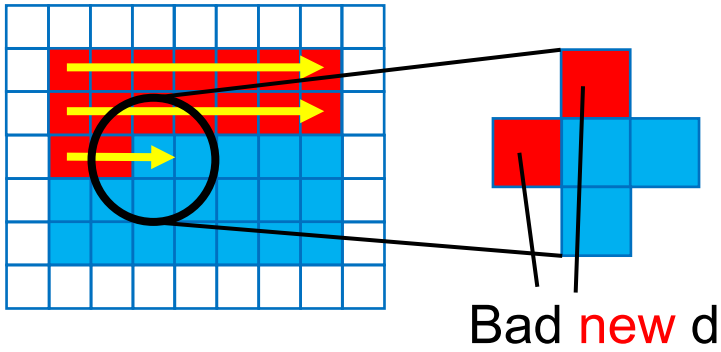
  - This is not good for simulation of a water glass ☹, but it's simple…



A Single Array Does not Work

Let us compute t ➔ t+1

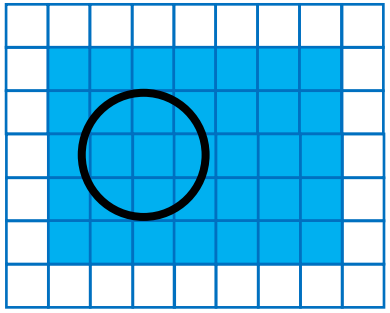
- With a single 2D array (Bug! ☹)



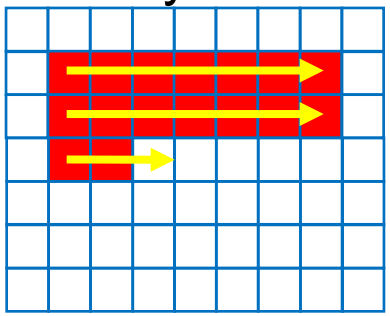
We need neighbor points at time t, but some have been already updated to t+1 ☹

- With separate 2D arrays (Good ☺)

An array for t



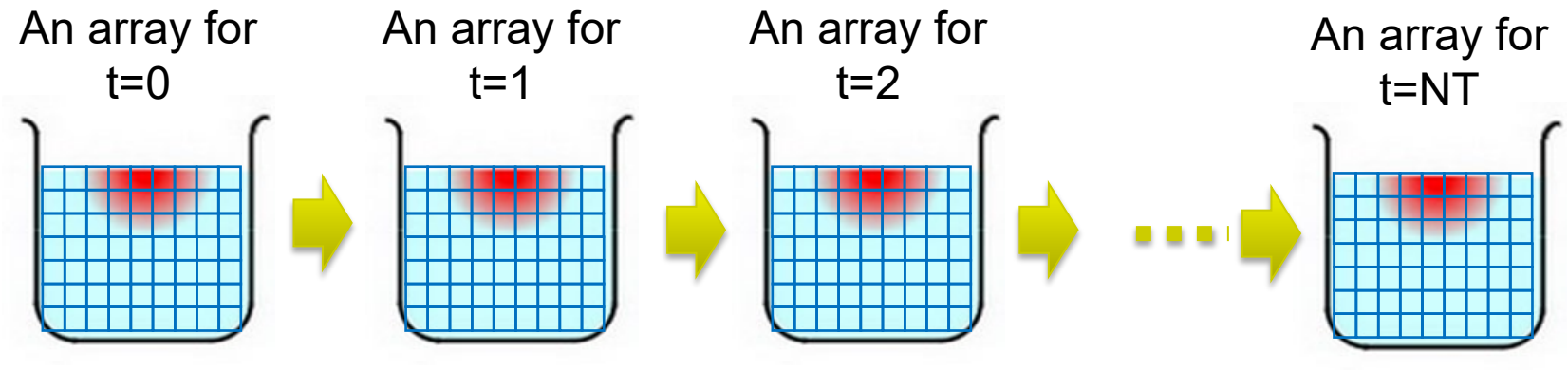
An array for t+1



We can access "old" neighbor points correctly ☺

Multiple Arrays?

- We repeat update of the array for NT times



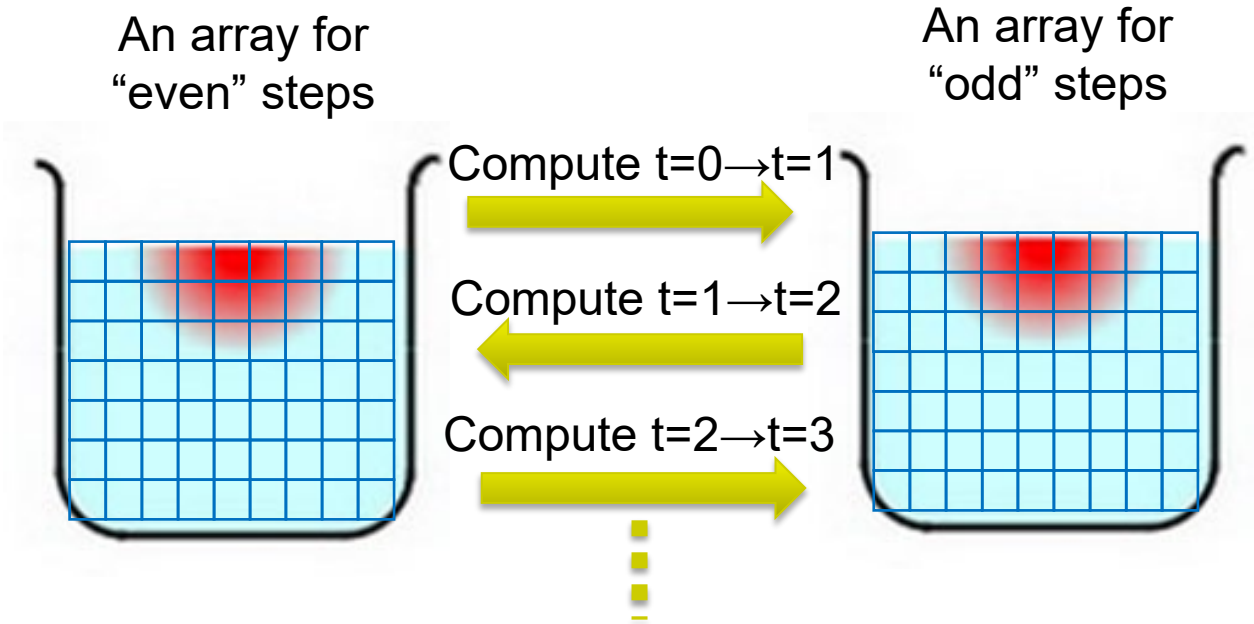
A simple way is to make arrays for all time steps

float data[NT+1][NY][NX]

- This uses too much memory
- Do we need all of (NT+1) arrays?

Double Buffering Technique

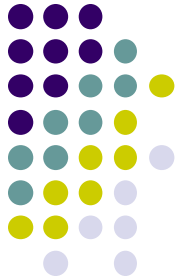
- It is sufficient to have “current” array and “next” array.
- It is better to use only “Double buffers”



The diffusion sample program uses

```
float data[2][NY][NX];
```

How We Parallelize “diffusion” sample (Related to Assignment [O1])



calc() takes long time, complexity is O(nx ny nt)

It mainly uses “for” loops

➔ #pragma omp parallel for is useful! But…

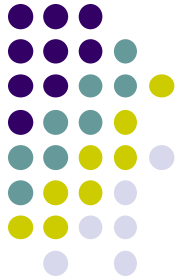
There are 3 (t, x, y) loops. Which should be parallelized?

[Hint1] Parallelizing either of spatial loop (x, y) would be good. Then spaces are divided into multiple threads

→ [Q] Parallelizing t loop is a not good idea. Why?

[Hint2] Take care of “pitfall in nested loops” (see slides in previous class)

Towards "Correct" Parallel Programming



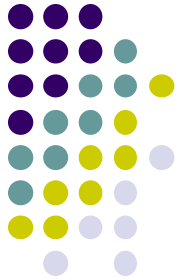
There are several types of bugs in parallel programming

- Bugs in compile time

- Bugs in run time
  - Bugs that abort execution (cf. segmentation fault)
  - Silent bugs → Hardest to find!

All bugs should be avoided!

When Can We 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.

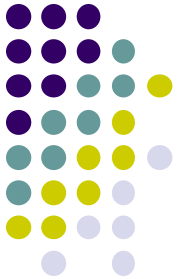
OK ☺: for (x = n; x >= 0; x-=4) …

ERROR ☹: for (i = 0; test(i); i++) …

ERROR ☹: for (p = head; p != NULL; p = p->next)

Errors in compile time

What are Differences between These Codes?



```
double D[100];
        :
```

Code A

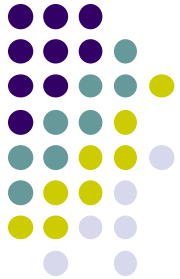
```
#pragma omp parallel for
  for (i = 0; i < 100; i++) {
     D[i] = D[i]+1.0;
  }
```

Code B

```
#pragma omp parallel for
  for (i = 0; i < 99; i++) {
     D[i+1] = D[i]+1.0;
  }
```

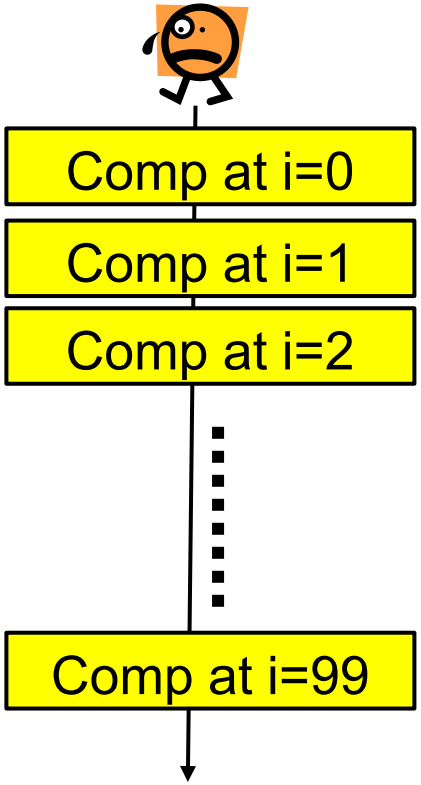
- Both codes are ok in compile time and can be executed
- But only code A is correct ☺ , code B has a bug ☹
  - Code B's results may be wrong

Sequential Execution and Parallel Execution of Loop



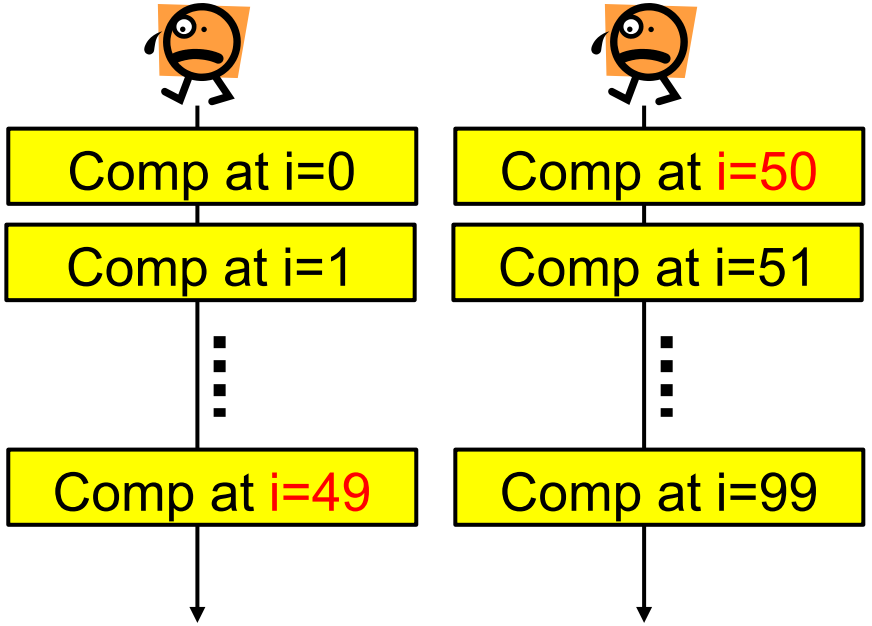
[Sequential]

```
for (i = 0; i < 100; i++) …
```



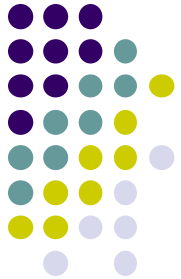
[Parallel]

```
#pragma omp parallel for
  for (i = 0; i < 100; i++) …
```



in case of 2 threads,
i=50 is computed before i=49

Difference between Two Codes



Code A

```
#pragma omp parallel for
  for (i = 0; i < 100; i++) {
     D[i] = D[i]+1.0;
  }
```

OK

It is ok to reorder 100 computations

Code B

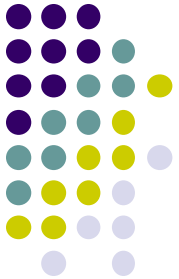
```
#pragma omp parallel for
  for (i = 0; i < 99; i++) {
     D[i+1] = D[i]+1.0;
  }
```

NG

Computations must be done in an order (i=0,1,2…)

➔ Parallelization breaks the order

Dependency between Computations



We define following sets for computation C

- Read set R(C): the set of variables read by C
- Write set W(C): the set of variables written by C
  - Ex) C: x = y+z ➔ R(C) = {y, z}, W(C) = {x}

We define dependency between C1 and C2

- If (W(C1) ∩ R(C2) ≠ ∅), C1 and C2 are dependent (write vs read)
- If (R(C1) ∩ W(C2) ≠ ∅), C1 and C2 are dependent (read vs write)
- If (W(C1) ∩ W(C2) ≠ ∅), C1 and C2 are dependent (write vs write)
- Otherwise, C1 and C2 are independent
  - ※ read vs read cases are independent

If C1 and C2 are independent, parallelization of C1 and C2 is safe ☺

Example of Dependency

Code A

```
#pragma omp parallel for
  for (i = 0; i < 100; i++) {
     D[i] = D[i]+1.0;    ← A_i
  }
```

$R(A_i) = \{D[i]\}$, $W(A_i) = \{D[i]\}$

All 100 computations are independent

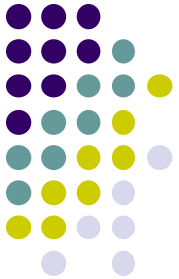
Code B

```
#pragma omp parallel for
  for (i = 0; i < 99; i++) {
     D[i+1] = D[i]+1.0;  ← B_i
  }
```

$R(B_i) = \{D[i]\}$, $W(B_i) = \{D[i+1]\}$

$R(B_{i+1}) \cap W(B_i) = \{D[i+1]\} \neq \emptyset$ → Dependent!

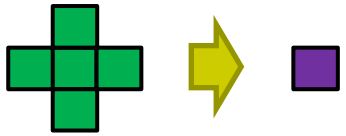
Dependency and Parallelism in Stencil Computations (1)



Consider 1D stencil computation:

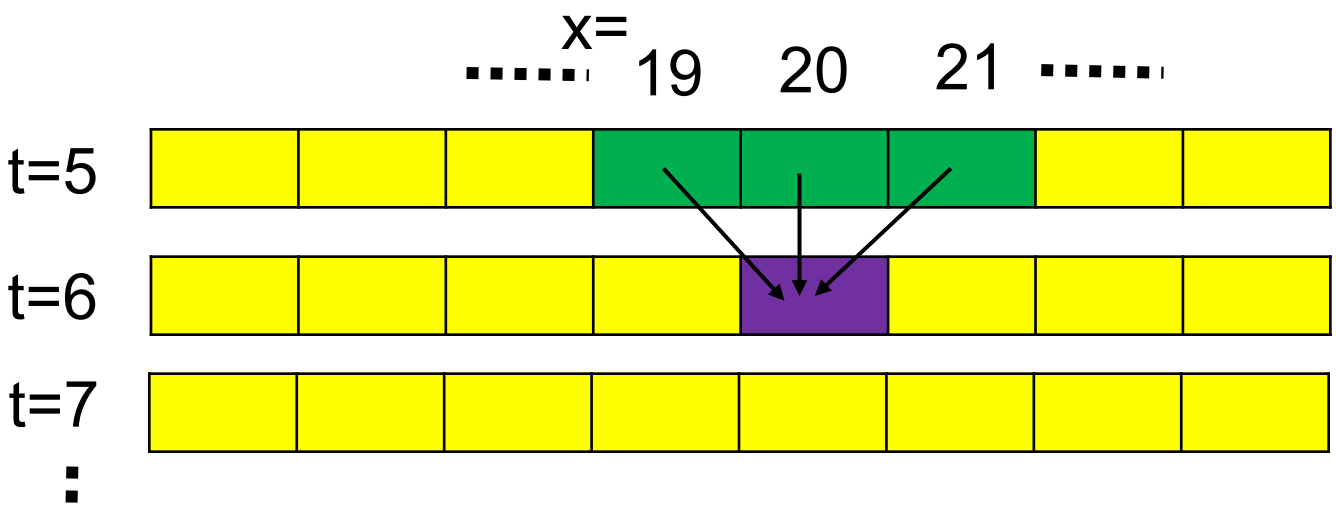
```
for (t = 0; t < NT; t++)
    for (x = 1; x < NX-1; x++)
        f_{t+1,x} = (f_{t,x-1} + f_{t,x} + f_{t,x+1}) / 3.0   /*  C_{t,x} */
```

※ This is simpler than "diffusion" (2D) sample



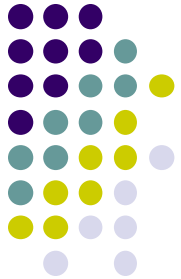
We let $C_{t,x}$ be computation of a single point $f_{t+1,x}$

$R(C_{t,x}) = \{f_{t,x-1}, f_{t,x}, f_{t,x+1}\}$, $W(C_{t,x}) = \{f_{t+1,x}\}$



※ This figure omits double buffering technique

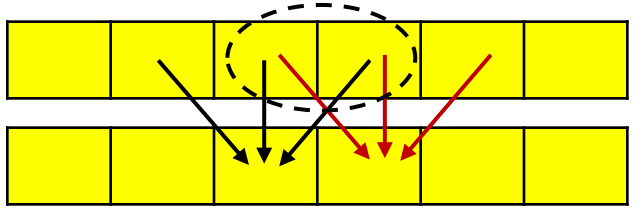
Dependency and Parallelism in Stencil Computations (2)



- Can we compute $C_{5,20}$ and $C_{5,21}$ in parallel? *(t is same, x is different)*
  - $R(C_{5,20})=\{f_{5,19}, f_{5,20}, f_{5,21}\}$, $W(C_{5,20})=\{f_{6,20}\}$
  - $R(C_{5,21})=\{f_{5,20}, f_{5,21}, f_{5,22}\}$, $W(C_{5,21})=\{f_{6,21}\}$

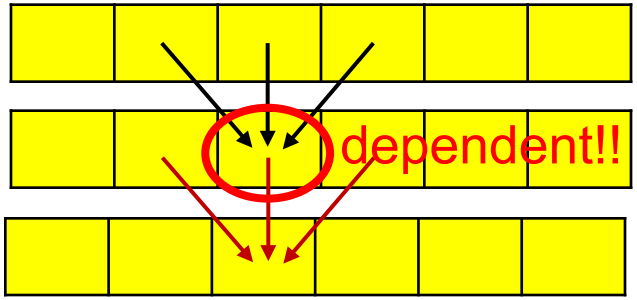
  → They are independent ☺ (for all pairs of x)

Read vs. Read is Ok

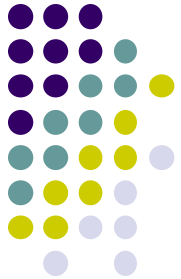


- Can we compute $C_{5,20}$ and $C_{6,20}$ in parallel? *(t is different)*
  - $R(C_{5,20})=\{f_{5,19}, f_{5,20}, f_{5,21}\}$, $W(C_{5,20})=\{f_{6,20}\}$
  - $R(C_{6,20})=\{f_{6,19}, f_{6,20}, f_{6,21}\}$, $W(C_{6,20})=\{f_{7,20}\}$

  → They are dependent ☹



In Assignment [O1]
- it is OK to parallelize x-loop or y-loop
- it is NG to parallelize t-loop



Job Submission on TSUBAME

This section describes larger usage of TSUBAME (>7 cores, >1node)

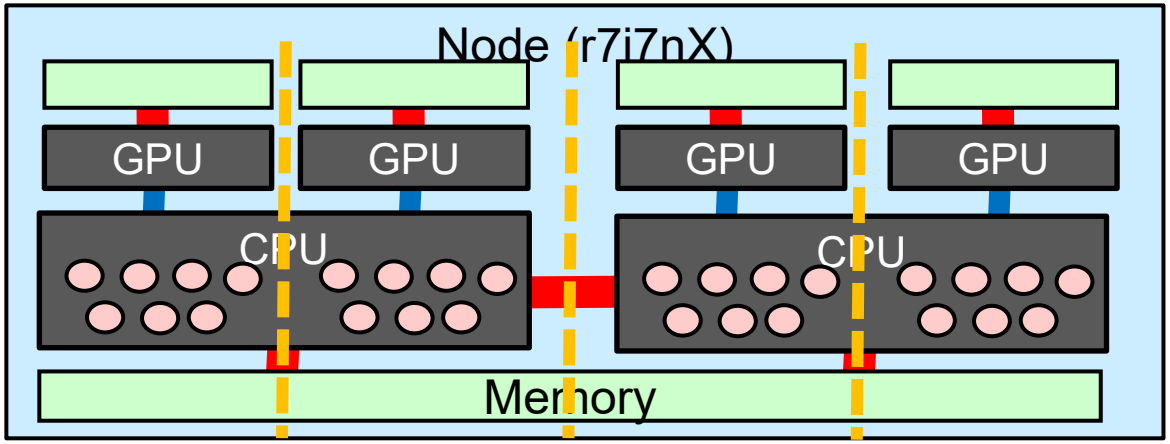
You can skip learning of this section, since you can get credits in this lecture with “interactive usage” as usual

この節ではTSUBAMEをより大規模に使う説明をする(>7コア、>1ノード)

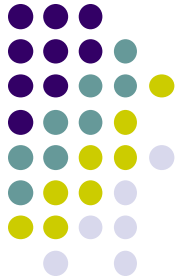
この授業の単位には普段の「インタラクティブ利用」で十分なので、この節を省いてもよい

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



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

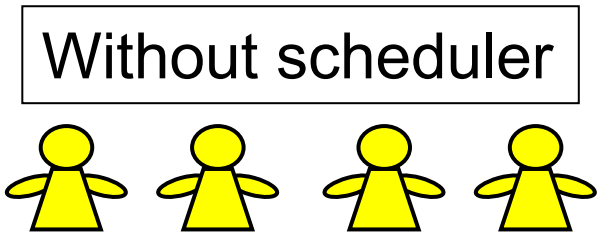
☹ It is not “real-time”

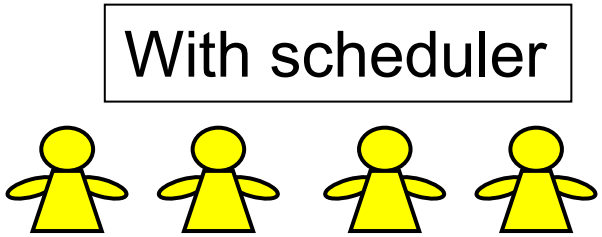
☹ Take care of charge! (TSUBAME point)

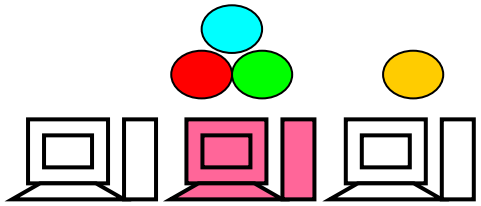
- In case of tga-ppcomp, Endo’s budget is used

What is Job Scheduler?

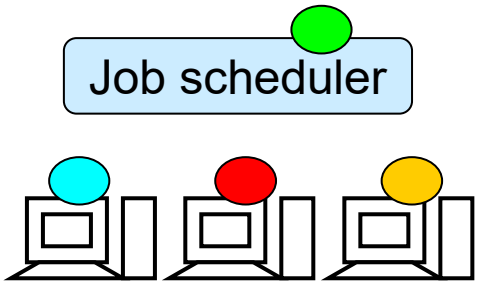
- The job scheduler does “traffic control” of many programs by many users
  - TSUBAME3.0 uses “Univa Grid Engine”





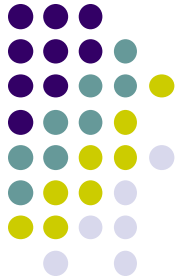


If users execute programs without control, there will be congestions



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

Standard route

qsub does not work in iqrsh. Please do this on log-in node

Prepare a Job Script (Section 5.2.3)

- In the case of mm-omp example
  - /gs/hs1/tga-ppcomp/23/mm-omp
- job.sh is a sample job script
  - Different file name is ok, but with ".sh"

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

export OMP_NUM_THREADS=4
./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
  - Please specify “proper” one
  - For example, if you use 1 core, f_node (28 cores) is too large (and expensive)

Resource type	Physical CPU cores	Memory (GB)	GPUs
f_node	28	240	4
h_node	14	12	2
q_node	7	60	1
q_core	4	30	0
s_core	1	7.5	0
s_gpu	2	15	1

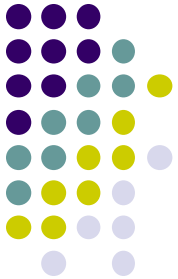
In Part1&2, use “1”

#$ -l [resource_type] = [Number]

#$ -l s_core=1
↑ Allocates only 1 core

#$ -l f_node=1
↑ Allocates 28 cores

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`

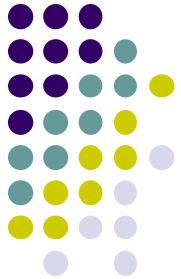
  - Charged! (有料)

- 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



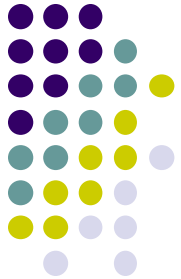
- First, please consider usage of interactive node (web usage/iqrsh)
- まずはインタラクティブノードの利用を検討してください (web usage/iqrsh)

- If necessary for reports, you can use up to 18,000 points in total per student. For more, please ask Endo
- 本講義のレポートの作成の目的で、一人あたり合計で18,000ポイントまで利用を認めます。より必要な場合は遠藤へ相談を
  - 18,000 points ≒ F_node x 5 hours
  - You can check point consumption on TSUBAME portal

- The TSUBAME group name is tga-ppcomp

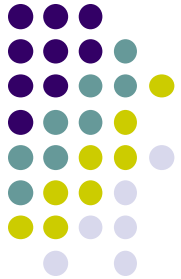
Users need to follow the rules at www.t3.gsic.titech.ac.jp
利用時には www.t3.gsic.titech.ac.jp に示される規則を守る必要があります

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
```

```
job-ID     prior   name       user         state submit/start at  …
-----------------------------------------------------------------
 11019490 0.00000 job.sh     endo-t-ac    qw    04/21/2022 09:19:30
```

Job ID

qw: not started yet
r: running now

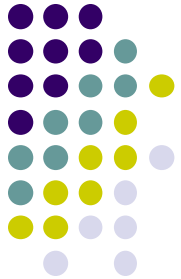
- qdel: To delete a job before its termination

```
qdel 123456
```

← Job ID

For interactive sessions, you can use iqstat, iqdel commands

Assignments in OpenMP Part (Abstract)



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

Due date: May 12 (Thu)

[O1] Parallelize “diffusion” sample program by OpenMP.
(/gs/hs1/tga-ppcomp/23/diffusion/ on TSUBAME)

[O2] Parallelize “sort” sample program by OpenMP.
(/gs/hs1/tga-ppcomp/23/sort/ on TSUBAME)

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

For more detail, please see OpenMP (1) slides

Next Class:

- OpenMP(3)
  - “task parallelism” for programs with irregular structures
  - sort: Quick sort sample
    - Related to assignment [O2]