2021 Practical Parallel Computing (実践的並列コンピューティング) No. 4

Part1: OpenMP (2)
Apr 22, 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 (2/4)
- Part 2: GPU programming
  - OpenACC and CUDA
  - 4 classes
- Part 3: MPI for distributed memory programming
  - 3 classes

Summary of Previous Class

OpenMP is for shared-memory parallel programming

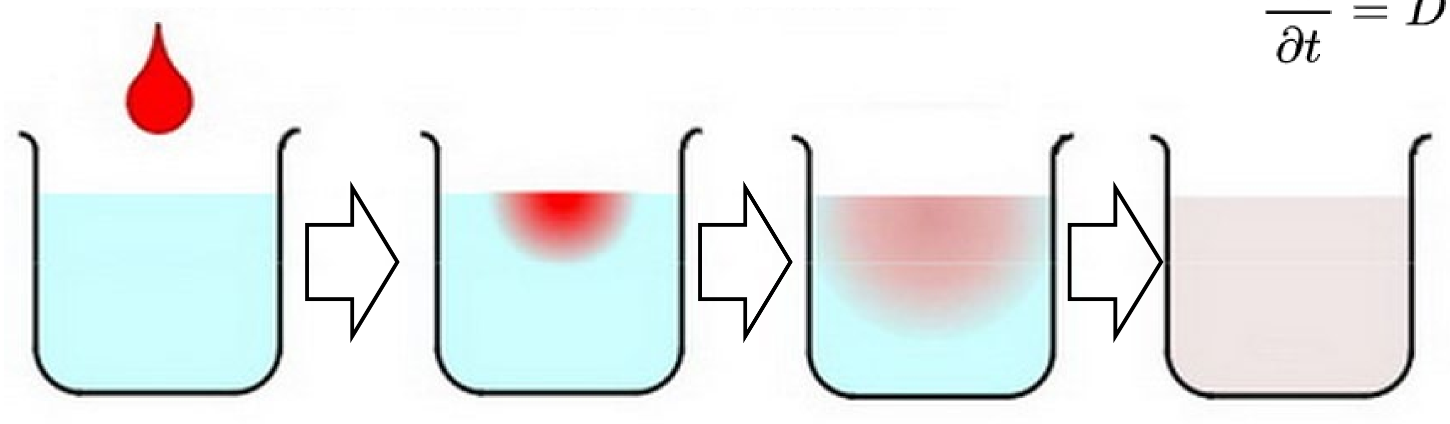
- #pragma omp parallel defines a parallel region, where multiple threads work simultaneously
- With #pragma omp for, loop-based programs can be parallelized easily
- Shared variables and private variables
- We have reviewed OpenMP version of mm sample

“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…

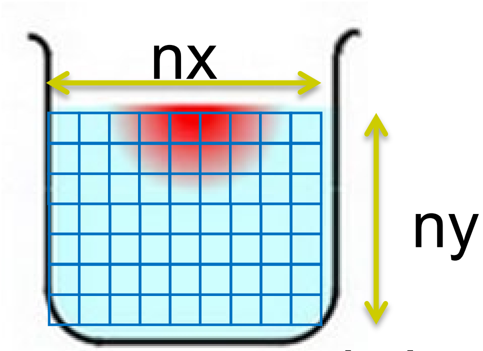
“diffusion” Sample on TSUBAME

Available at /gs/hs1/tga-ppcomp/21/diffusion/

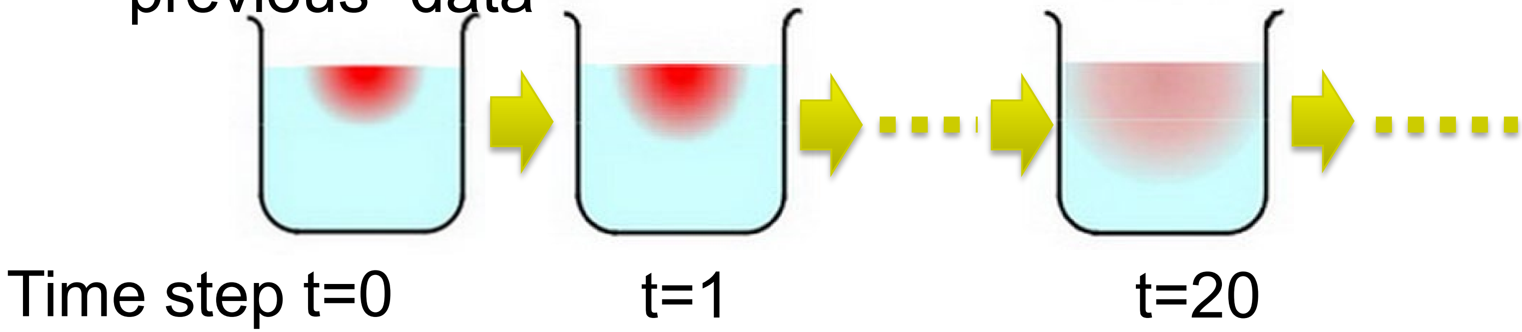
- Execution：./diffusion [nt]
- nt: Number of time steps
- nx, ny: Space grid size
  - nx=8192, ny=8192 (Fixed. See the code)
  - How can we make them variables? (See mm sample)
- Compute Complexity：O(nx × ny × nt)

Expression of Space to be Simulated

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

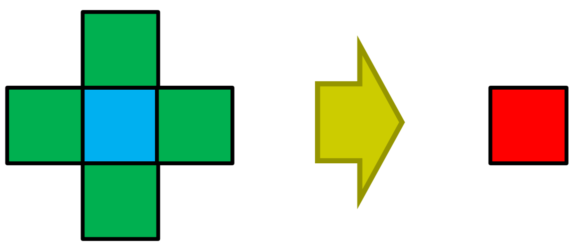


- Array elements are computed via timestep, by using "previous" data



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"

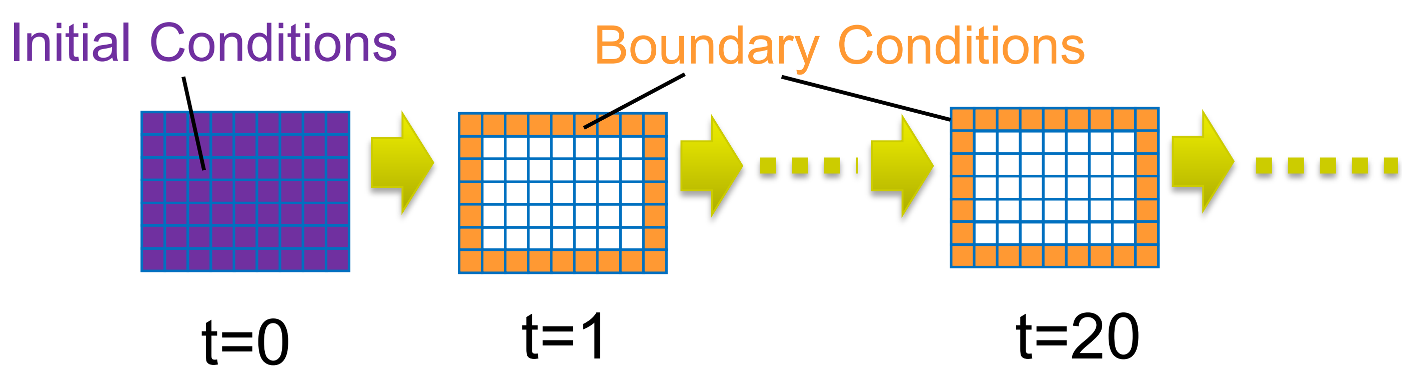
Initial Conditions & Boundary Conditions

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

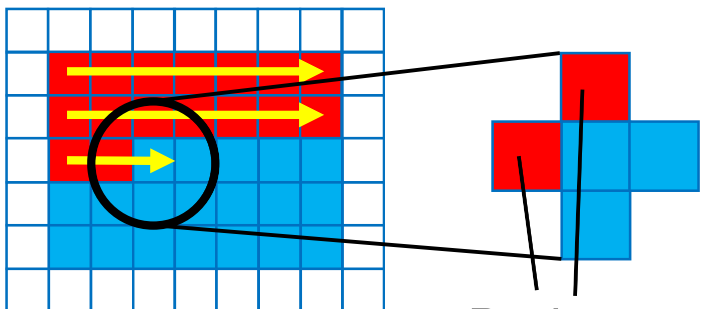
  → See ranges of for-loops in calc(); boundaries are skipped
  - This is not good for simulation of a water glass ☹, but it's simple…



A Single Array Does not Work

Let us compute t ➔ t+1

- With a single 2D array (Bug! ☹)

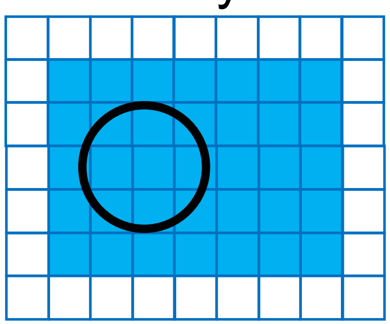


Bad new data

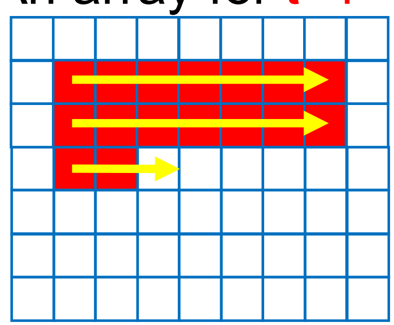
We need neighbor points at time t, but some have been already updated to t+1 ☹

- With separate 2D arrays (Good ☺)

An array for t



An array for t+1



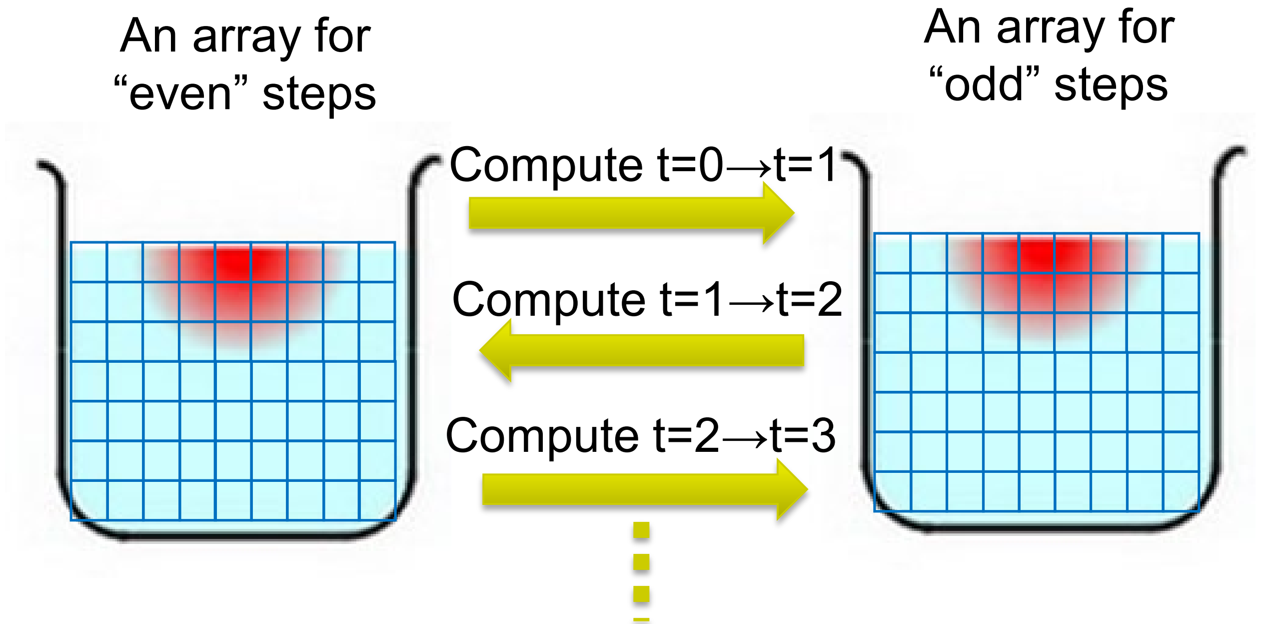
We can access “old” neighbor points correctly ☺

Double Buffering Technique

- A simple way is to make arrays for all time steps, but it consumes too much memory! (nx × ny × nt?)

→ It is sufficient to have "current" array and "next" array.

→ It is better to use only "Double buffers"



※ Sample program uses a global variables
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

→ How about using #pragma omp parallel for ?

→ Good! 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)
```

Bugs 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

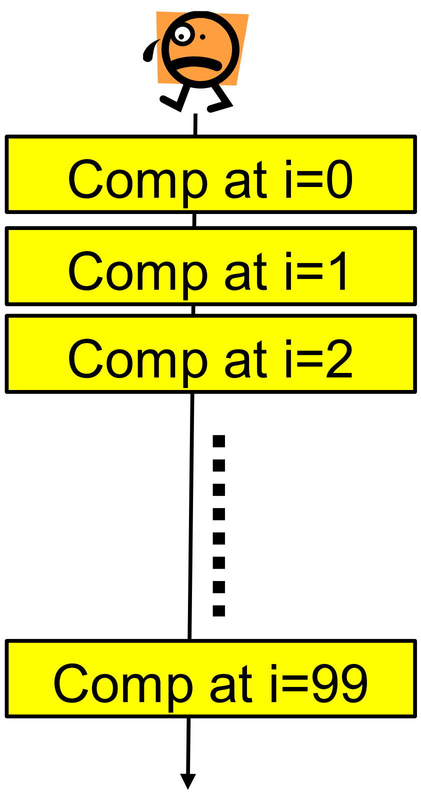
```
#pragma omp parallel for
  for (i = 0; i < 99; i++) {
     D[i+1] = D[i]+1.0;
  }
```

- 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]

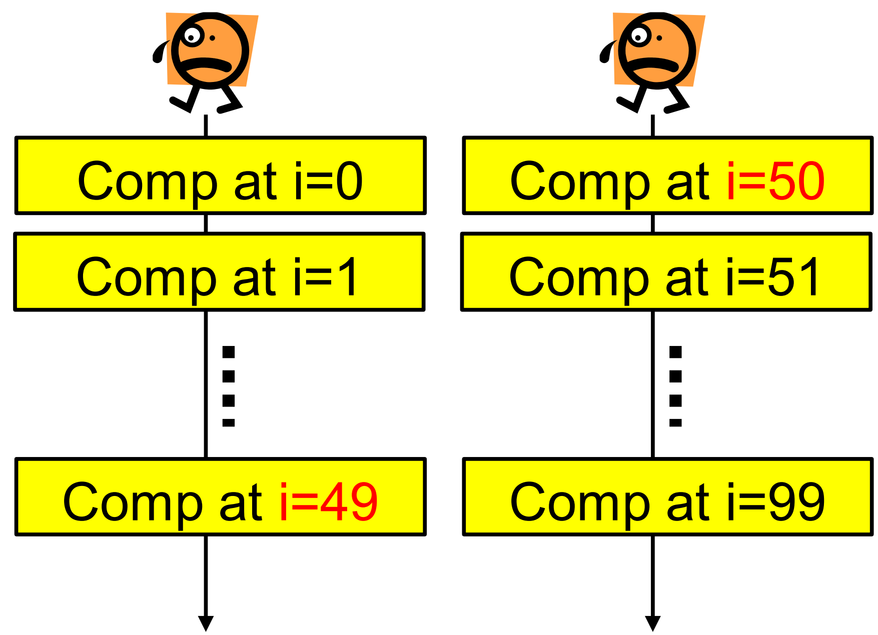
for (i = 0; i < 100; i++) …



[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

<u>If C1 and C2 are independent, parallelization of C1 and C2 is safe ☺</u>

Example of Dependency

Code A

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

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

All 100 computations are independent

Code B

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

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

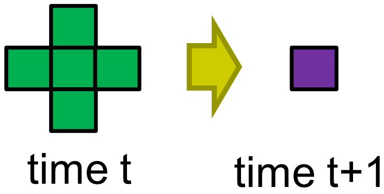
$R(B_{i+1}) \cap W(B_i) = \{D[i+1]\} \neq \emptyset$ ➔ Dependent!

Dependency and Parallelism in Stencil Computations (1)

Consider 1D stencil computation:

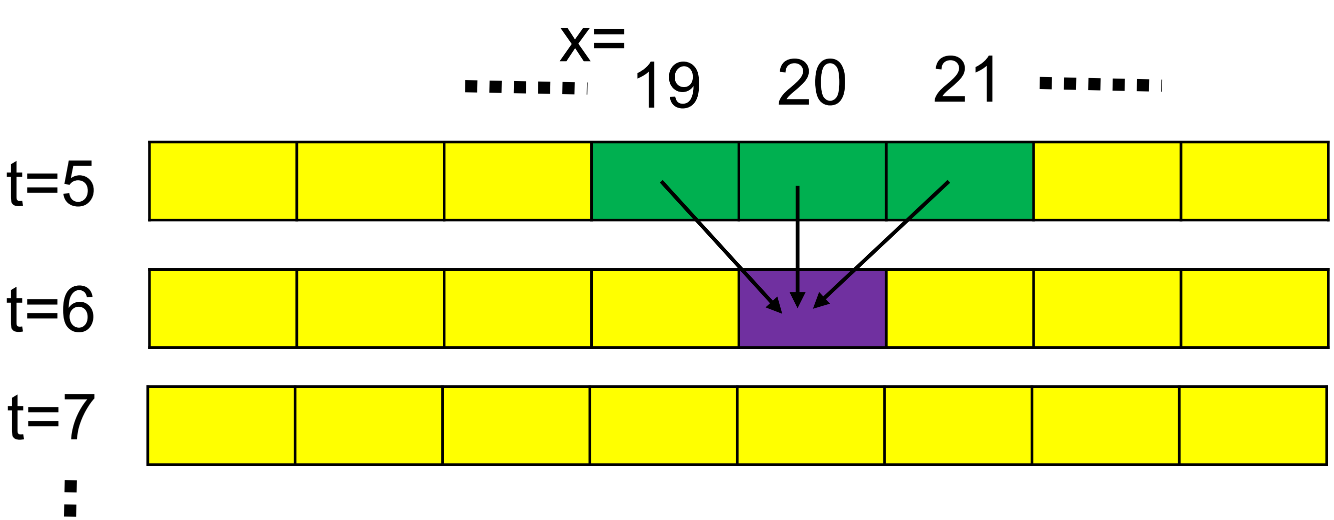
for (t = 0; t < NT; t++)
    for (x = 1; x < NX-1; x++)
        $f_{t+1,x} = (f_{t,x-1} + f_{t,x} + f_{t,x+1}) / 3.0$ /* $C_{t,x}$ */

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



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

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



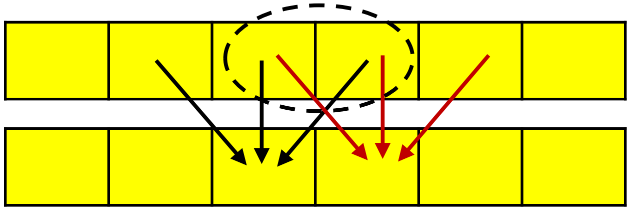
※ This figure omits double buffering technique

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)*
  - $R(C_{5,20})=\{f_{5,19}, f_{5,20}, f_{5,21}\}$, $W(C_{5,20})=\{f_{6,20}\}$
  - $R(C_{5,21})=\{f_{5,20}, f_{5,21}, f_{5,22}\}$, $W(C_{5,21})=\{f_{6,21}\}$

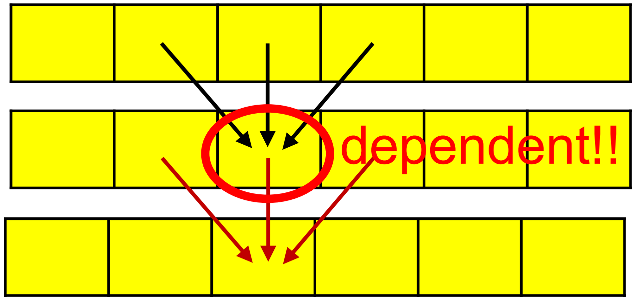
  → They are independent ☺ (for all pairs of x)

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

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 on Apr 19

Next Class:

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