# PSHW #4

## Chapter6

20191165

Sim Eunyeong

3/4.1~4.3/7

6.3 Many computer applications involve searching through a set of data and sorting the data. A number of efficient searching and sorting algorithms have been devised in order to reduce the runtime of these tedious tasks. In this problem we will consider how best to parallelize these tasks.

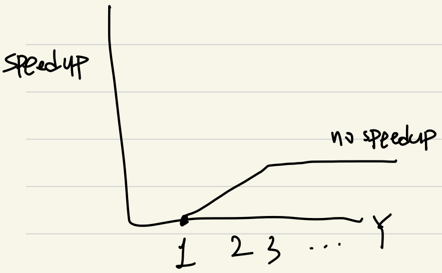
6.3.1 [10] <§6.2> Consider the following binary search algorithm (a classic divide and conquer algorithm) that searches for a value X in an sorted N-element array A and returns the index of matched entry:

```
BinarySearch(A[0..N−1], X) {
   low = 0
   high = N −1
   while (low <= high) {
        mid = (low + high) / 2
        if (A[mid] >X)
            high = mid −1
        else if (A[mid] <X)
            low = mid + 1
        else
            return mid // found
   }
   return −1 // not found
}
```

logN 이후 안변함.

Assume that you have Y cores on a multi-core processor to run BinarySearch. Assuming that Y is much smaller than N, express the speedup factor you might expect to obtain for values of Y and N. Plot these on a graph.

- the complexity of binary Search O(logN) already good?
- hard to parallelize because it is recursive.
- we can parallelize conditional statement.



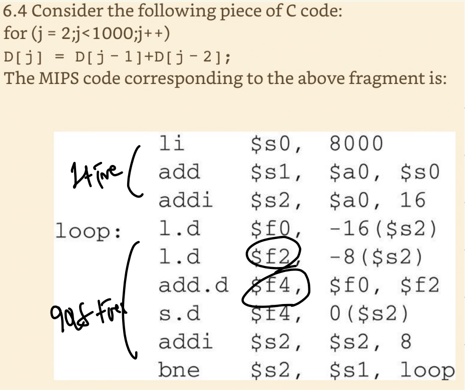
6.3.2 [5] <§6.2> Next, assume that Y is equal to N. How would this affect your conclusions in your previous answer? If you were tasked with obtaining the best speedup factor possible (i.e., strong scaling), explain how you might change this code to obtain it.

if Y==N, the complexity becomes O(1)

Y times speed up Concurrency degree ↑

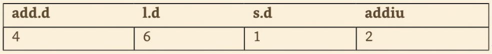
↓ size 나누고 Parallel

Just compare each element with X.



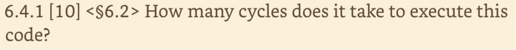
6.4 Consider the following piece of C code:
for (j = 2;j<1000;j++)
`D[j] = D[j - 1]+D[j - 2];`
The MIPS code corresponding to the above fragment is:

```
        li      $s0,  8000
1time ( add     $s1,  $a0, $s0
        addi    $s2,  $a0, 16
loop:   l.d     $f0,  -16($s2)
        l.d     $f2,  -8($s2)
        add.d   $f4,  $f0, $f2
998 time s.d    $f4,  0($s2)
        addi    $s2,  $s2, 8
        bne     $s2,  $s1, loop
```



| add.d | l.d | s.d | addiu |
|---|---|---|---|
| 4 | 6 | 1 | 2 |

cycle latency

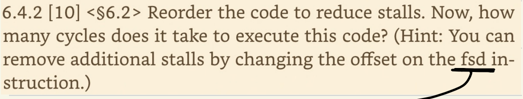


6.4.1 [10] <§6.2> How many cycles does it take to execute this code?

- # iterations = 1000 - 2 = 998

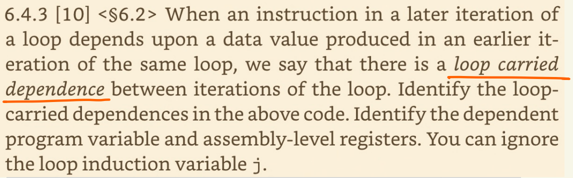
$\underbrace{2*6}_{12} + \underbrace{4*1}_{4} + \underbrace{1*1}_{1} + \underbrace{2*1}_{2} = 19$

$19 * 998 + 3 + 2 + 1$



6.4.2 [10] <§6.2> Reorder the code to reduce stalls. Now, how many cycles does it take to execute this code? (Hint: You can remove additional stalls by changing the offset on the fsd instruction.)

Impossible no this instruction ?

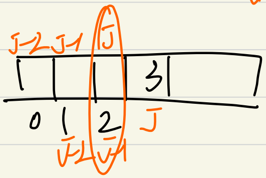


6.4.3 [10] <§6.2> When an instruction in a later iteration of a loop depends upon a data value produced in an earlier iteration of the same loop, we say that there is a *loop carried dependence* between iterations of the loop. Identify the loop-carried dependences in the above code. Identify the dependent program variable and assembly-level registers. You can ignore the loop induction variable j.

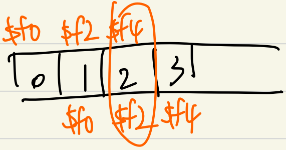
dependent program variable

:

D[j] and D[j-1] have loop carried dependence



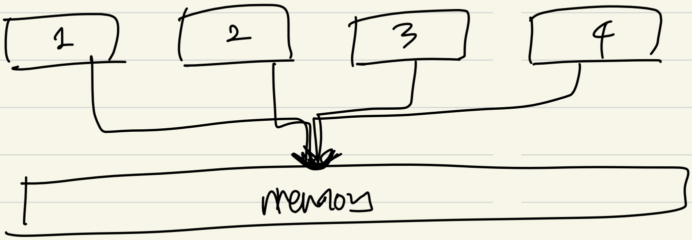
dependent assembly-level registers



$f4 and $f2 have loop-carried dependence

6.7 Consider the following portions of two different programs running at the same time on four processors in a symmetric multicore processor (SMP). Assume that before this code is run, both x and y are 0.

```
Core 1: x = 2;
Core 2: y = 2;
Core 3: w = x + y + 1;
Core 4: z = x + y;
```



6.7.1 [10] <§6.5> What are all the possible resulting values of w, x, y, and z? For each possible outcome, explain how we might arrive at those values. You will need to examine all possible interleavings of instructions.

```
Core 1: addi x, 2, 0
Core 2: addi y, 2, 0
Core 3: add t1, x, 0
        add t2, y, 0
        add w, t1, t2
        add w, w, 1
Core 4: add t1, x, 0
        add t2, y, 0
        add z, t1, t2
```

| x | y | w | z |
|---|---|---|---|
| 2 | 2 | 1 | 0 |
| 2 | 2 | 1 | 2 |
| 2 | 2 | 1 | 4 |
| 2 | 2 | 3 | 0 |
| 2 | 2 | 3 | 2 |
| 2 | 2 | 5 | 4 |
| 2 | 2 | 5 | 0 |
| 2 | 2 | 5 | 2 |
| 2 | 2 | 5 | 4 |

$x=y=2$

$\boxed{\text{Core3}}$

(1,2 ①

```
add t1, x, 0
add t2, y, 0
add w, t1, t2
add w, w, 1
```

① ⇒ w=5

6.7.2 [5] <§6.5> How could you make the execution more deterministic so that only one set of values is possible?

I can set policy like

read can happen only after all the writes done.

⋮

(synchronization)