L18: Parallelism and Concurrency

Rachata Ausavarungnirun

(rachata.a@tggs.kmutnb.ac.th)

March 11th, 2020

Architecture Research Group

Software System Engineering

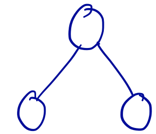
Thai-German Graduate School, KMUTNB

Before We Begin

- Please go to this link and try if Hangout Meet works ok
  - meet.google.com/bgu-zhup-zby

- I want to make sure everything works fine in case we need to move the class to an online format

Parallelism



- This means you do multiple things at once

- It is a simultaneous execution of computations
  - Does not have to be related
  - Can be related

- Warning: Some examples here are in c++
  - Rayon and parallel constructs in Rust naturally enable these

Concurrency

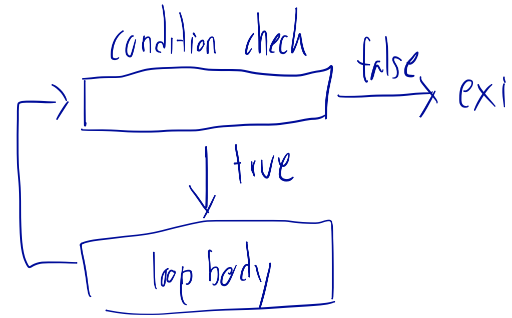
- This is handling multiple things at once - Things done on CPU1 cannot delay CPU2

- The thing computer do concurrently should be independent from each other

- Think of it this way (and, of course, I will tie HW in this)
  - Concurrency is a process of truly running things independently on the hardware level
  - Think Amdahl's law with critical path that we discussed

- https://blog.golang.org/concurrency-is-not-parallelism

Parallel For-loop



- In C, you can use Cilk - execute in parallel
  - cilk_for(int i=0; i<n; i++)
    B[i] = A[i] + 1

(each iteration is independent)

- You can use OpenMP annotation - Make many threads on each iterations
  - #pragma omp for
    for(int i=0; i<n; i++)
    B[i] = A[i] + 1

B[i] = B[i-1]

Cannot Parallel - Have to wait for B[i-1] before start new one

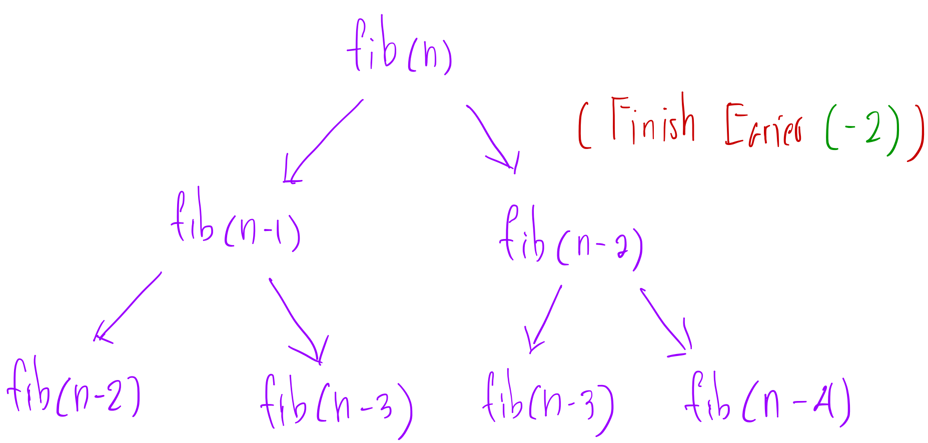
- Notice how there are no dependency?

Fork-join

- (A || B) means A and B can be run in parallel
- Use rayon for this

```
int fib(int n) {
  if n < 2 { return n; }
  let (x, y) = rayon::join(|| fib(n-1), || fib(n-2))
  return x + y;
  }
```

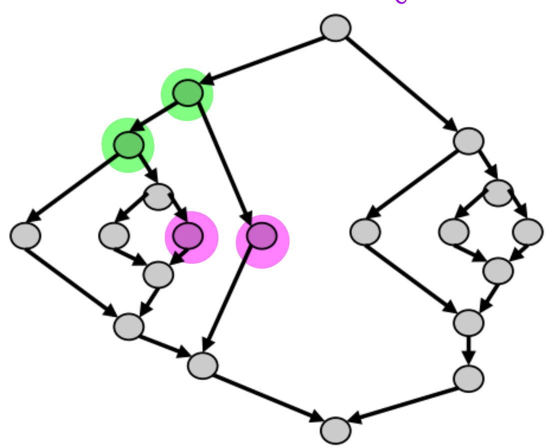
- How does the execution looks like in dataflow?



Nested Parallelism

- Basically combines parallel loop and fork join
- What is the maximum concurrent computation we get out of this parallel task?

(Both left and right are equal)



(Have to use as many CPU as possible to make it faster)

Ex

1 CPU → 1 ns for 1 add

8 CPU → 1 ns for 8 add

- With this dependency graph, two tasks are parallel if there is not path between each other

Yes

No

Cost Analysis - How to compute the cost

- W = work
  - The total number of operations (How many instructions)
- D = depth (or span)
  - The longest chain of dependency (Data Flow Graph)
- Parallelism = work/depth
  - This determine the number of processors that can be effectively used
- Remember Amdahl's Law?

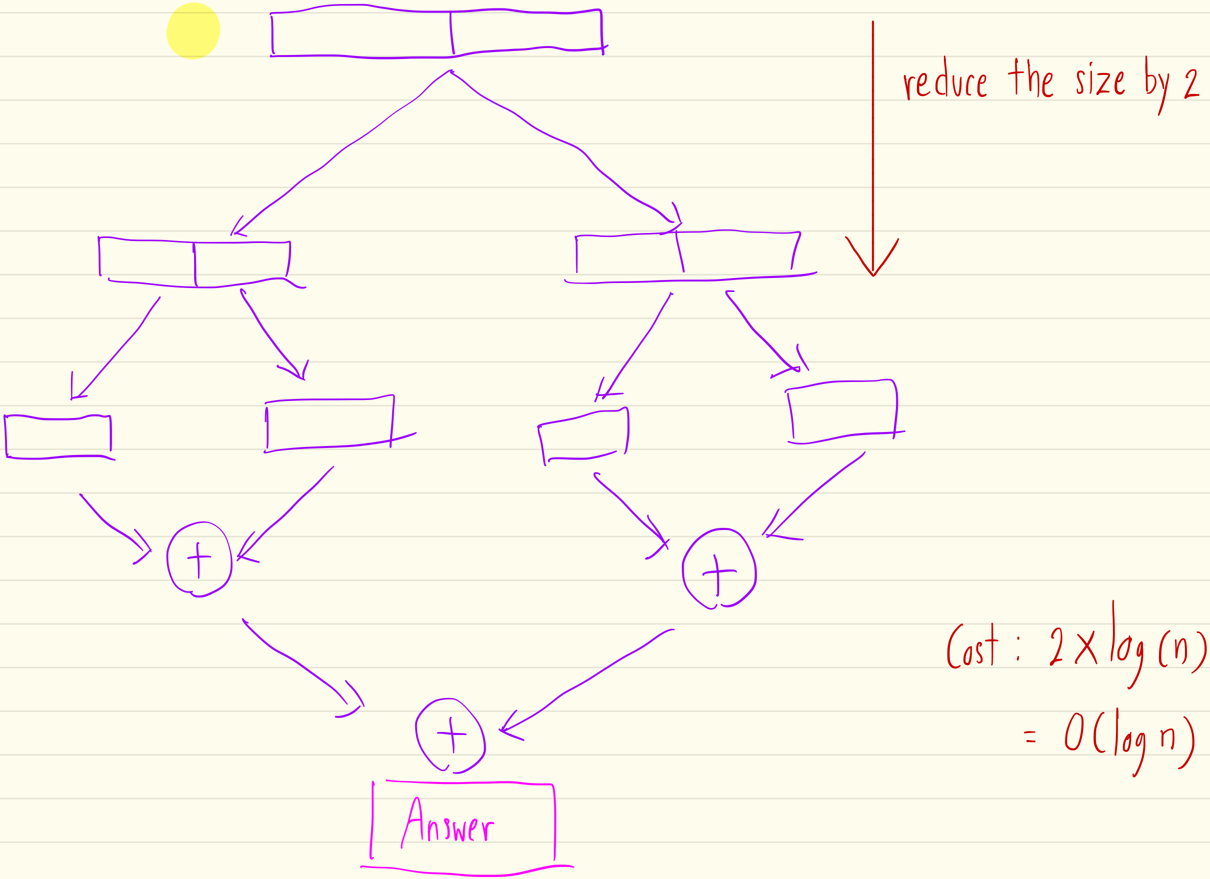
- Lingering question from a HW/system person (me):
  - Well, what do we do with the CPUs that are doing nothing?

Example

- ```
  fn seq_sum(v: &[i32]) -> i32 {
    let mut total = 0i32;
    for num in v {
     total += num
    }
   total
  }
  ```
- What is W? $- O(n)$
- What is D? $- O(n)$

$\text{Parallelism} = \frac{O(n)}{O(n)} = 1$

↳ No parallel
  ```

Example #2

- ```
  fn par_sum(v: &[i32]) -> i32 {
    if v.len() <= 1 {
    return seq_sum(v);
    }
    let (left, right) =
  v.split_at(v.len()/2);
    use rayon::join;
    let (left_sum, right_sum) =
     join(|| par_sum(left),||par_sum(right));
    left_sum + right_sum
  }
  ```

(Split left sum and right sum)

- What is W? - O(n) - Add n items
- What is D? - log(n)

Parallelism = $\frac{O(n)}{O(\log n)}$ = better than 1
  ```



Cost : $2 \times \log(n)$

$= O(\log n)$

Why This Cost Model?

- Simple (just draw the dependency graph)
- Brent's Theorem:
  - Can schedule in O(W/P + D) time on P processors

    number of processors (How many CPU needed)
  - See how this follow exactly as Amdahl's law?
- Lower bound: How much time do we need if we have P processors?
- When you design a parallel algorithm
  - Work efficient: Work should be ~ the same as sequential running time
  - A polynomial parallelism of $O(n^{1/2})$ is generally good

Example: Not Very Good Quicksort

```
def qs(xs: Seq[Int]): Seq[Int] = {
  if (xs.length <= 1) xs
  else {
     p = RNG.choice(xs) - Random Pick number in the list
     s0 = [ e for e in xs if e < p ]
     s1 = [ e for e in xs if e == p]
     s2 = [ e for e in xs if e > p ]
     (r0, r2) = par(qs(s0) || qs(s2))

     r0 ++ s1 ++ r2
  }
}
```

Sorted left

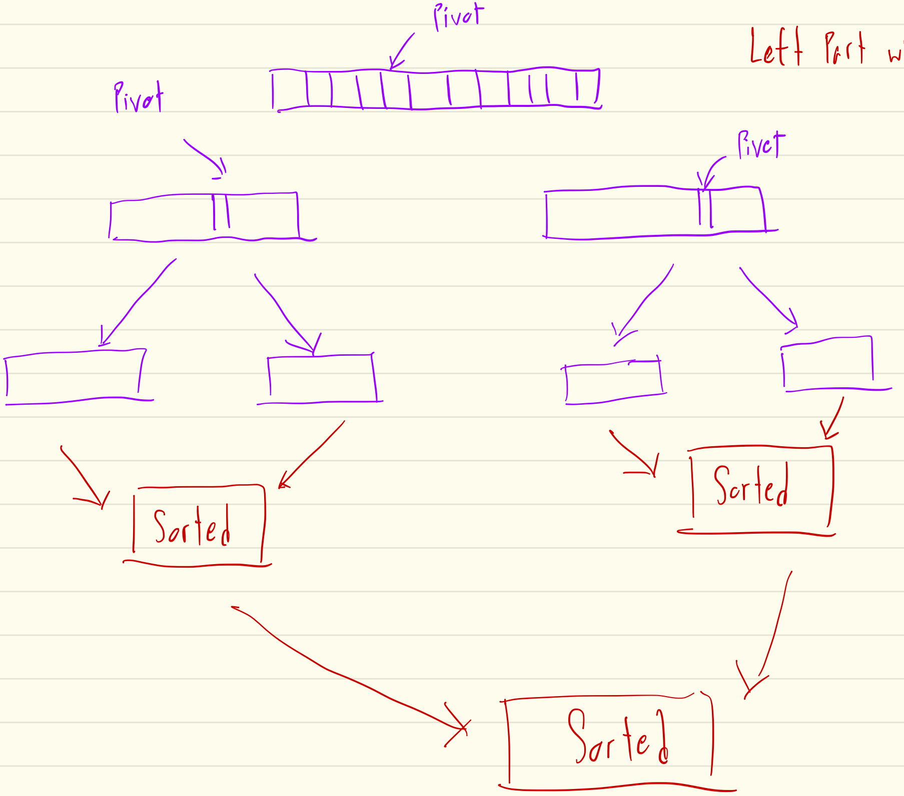
Sorted Right

$W = O(n)$

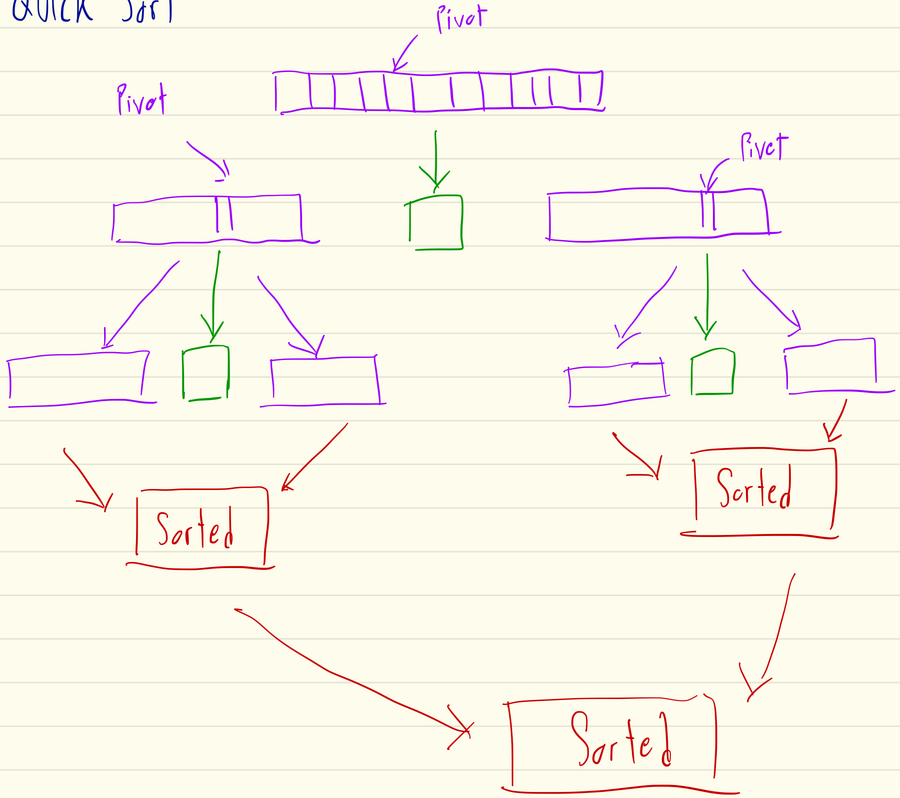
$D = O(\log n)$

Normal Quick Sort

Left Part will smaller than Right Part



Better Quick Sort



A Bad Parallel Program

- Consider this example: what if
  - Partitioning and concatenation is done in O(n) work and O(logn) depth
  - But you perform serial recursive calls = Finish one before another
    - What is my complexity?

- Hint 1: Try drawing the dependency graph

- Hint 2: This is not very good

↳ Cannot run in Parallel

A Better Parallel Program

- Consider this example: what if
  - Partitioning and concatenation is done in O(n) work and O(logn) depth
  - The recursive calls are not made in parallel

- What is the work?
- What is the depth/span?

- Which one gets lower? Why?

- What is the parallelism?

Designing a Good Parallel Program

- This in terms of operations on collections
  - map, reduce, filter …
  - Why are these good?

Do it with every elements on the same time

- Map: applies a function f to every elt of the collection
- **Reduce: pairwise combines elements in a tree until you have 1 using a (n assoctive) binary operator**
- Filter: retains an element if pred(e) returns true

Map's Parallelism

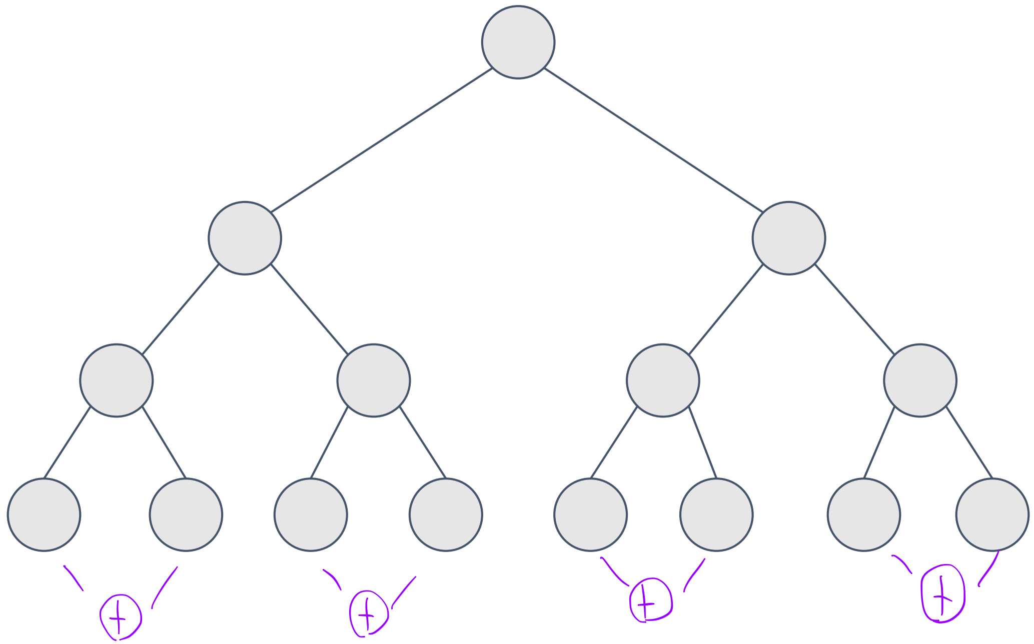
- Apply a function on all elements

- What is W and what is D?

$O(n)$ $O(1)$

Reduce's Parallelism

Combine until get 1 element



- What is W and what is D? Why?

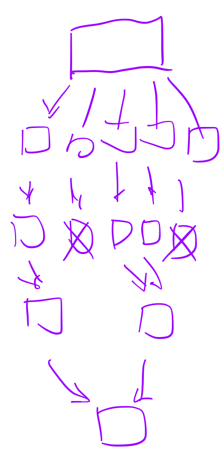
O(n) O(log n)

Filter's Parallelism - Apply function, if true keep it, false throw it away

- Major challenges
  - How many are selected?
  - How to retain and put into the output?

  Don't know

- Filter can be done in parallel if designed properly
  - Complexity depends on the elements being filtered
  - Convert this into the dataflow graph is a good way to tell its parallelism



In-class Exercise 17

- What is the dependency graph looks like and what is the complexity of our parallel quicksort (both W and D)?