Rose Whitt

322 HW 2 Written

1.1

1.1.1

```
acc ← accumulator(OR, boolean)
finish (acc) {
    for each Assignment f of a value in [k] to each node in V do
        colorable ← accumulator(AND, boolean)
        finish (colorable) {
            for each {u, v} ∈ E do
                async {
                    if f(u) = f(v) then
                        colorable.put(false)
                }
        }
        acc.put(colorable.get())
}
return acc.get()
```

1.1.2

$Work(n) = O(nkm)$.

If $n = |V|$, $m = |E|$, then there are nk assignments of f that satisfy the for loop on line 3.

1.1.3

Yes. The sequential algorithm may finish execution earlier than my parallel algorithm because the sequential algorithm can finish without checking all possibilities while mine only finishes when all possibilities are checked.

1.1.4

No, my parallel algorithm will always have Work that is greater than or equal to the sequential Work.

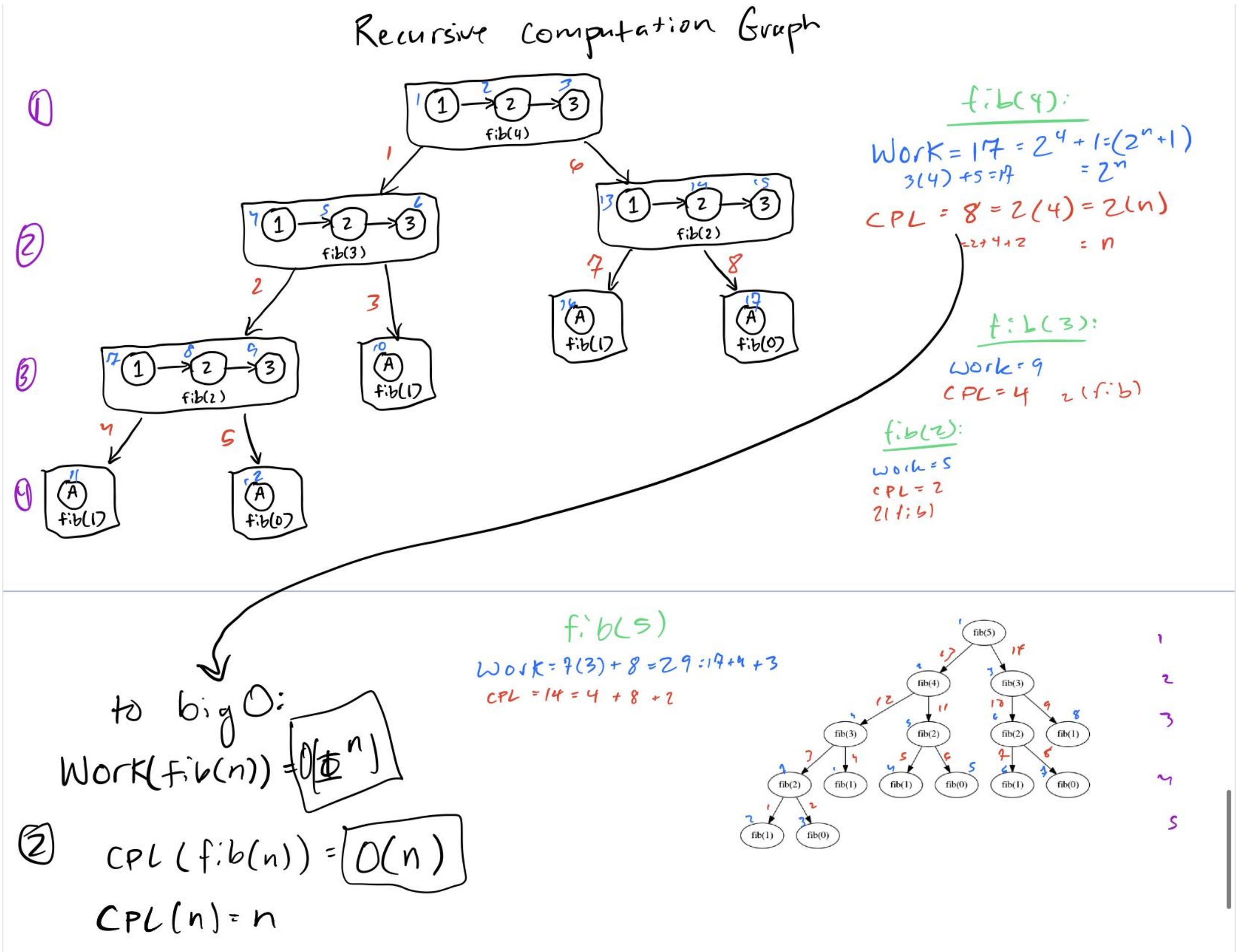
1.2

1.2.1

$Work(fib(n)) = O(\phi^n)$

1.2.2

Rose Whitt



Since every call to *fib(n)* relies on futures (*fib(n - 1)*, *fib(n - 2)*, . . . , *fib(1)*, *fib(0)*),

$Work(CPL(n)) \ = \ O(n).$

$CPL(n) \ = \ O(n).$

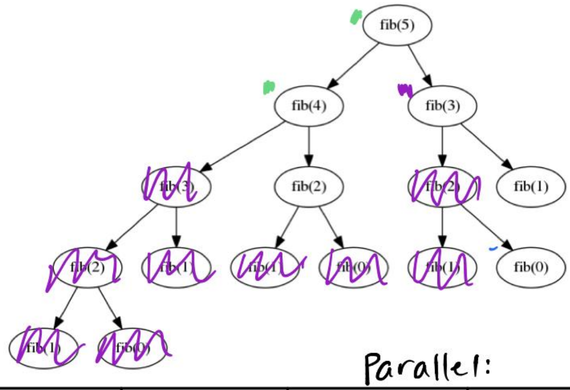
1.2.3

CPL(n) = n

$O(n)$

Bob's is faster than the sequential Memoized Fibonacci from HW1 because the futures allow this to be run on many processors.

Rose Whitt



m: fib has already been calculated

Parallel:

Start	1	2	3	4	. . . .
0	fib(5)				
1	fib(4)	fib(3)			
2	fib(2)	fib(1)	fib(0)		

Sequential:

Start	1
0	fib(5)
1	fib(4)
2	fib(3)
3	fib(2)
4	fib(1)
5	fib(0)