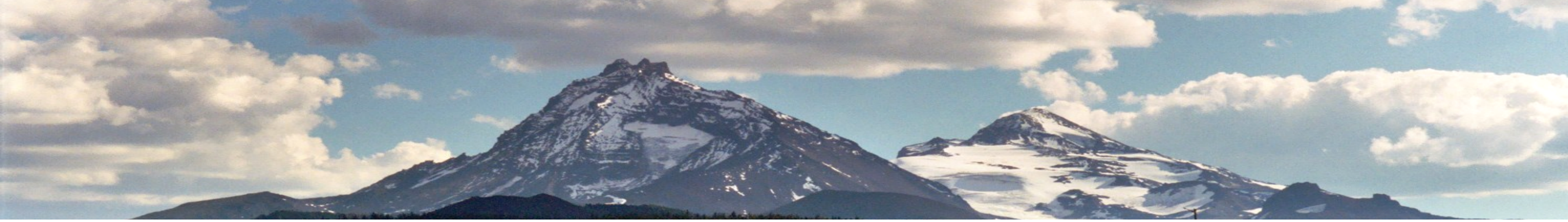


Fork-Join Pattern

Parallel Computing

CIS 410/510

Department of Computer and Information Science

UNIVERSITY OF OREGON

Fibonacci

- Recursive Fibonacci is simple and inefficient

```
long fib ( int n ) {
    if (n < 2) return 1;
    else {
        long x = fib (n-1);
        long y = fib(n-2);
        return x + y;
    }
}
```

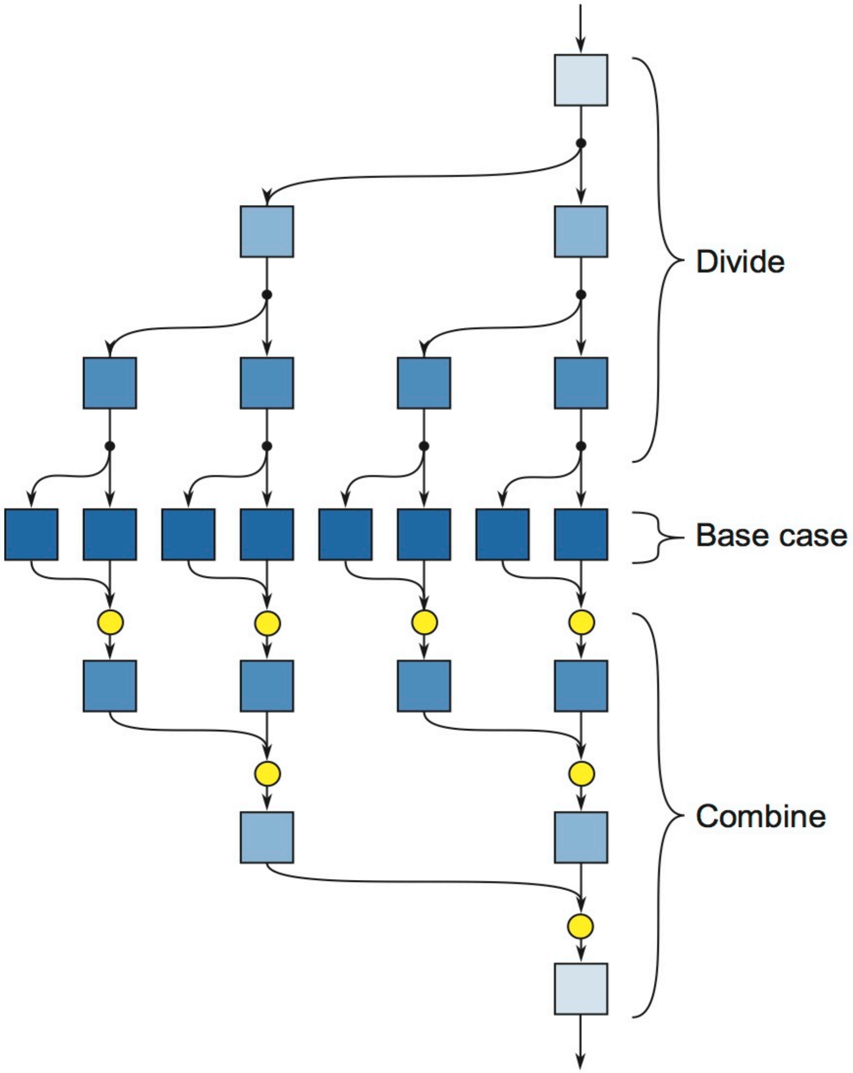
Fibonacci…

- Recursive Fibonacci is simple and inefficient
- But it does have the property that the sub-calls are independent
- Can we parallelize it?

Fibonacci…in Parallel?

```
long fib ( int n ) {
    if (n < 2) return 1;
    else {
        long x = fork fib (n-1);
        long y = fib(n-2);
        join;
        return x + y;
    }
}
```

Recursive Fork Join

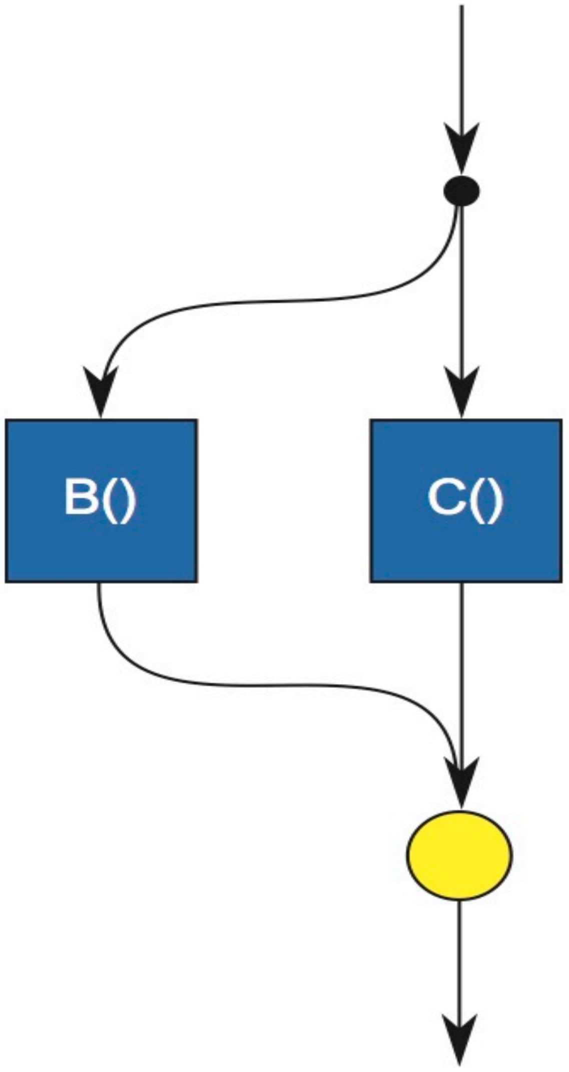


Fork Join

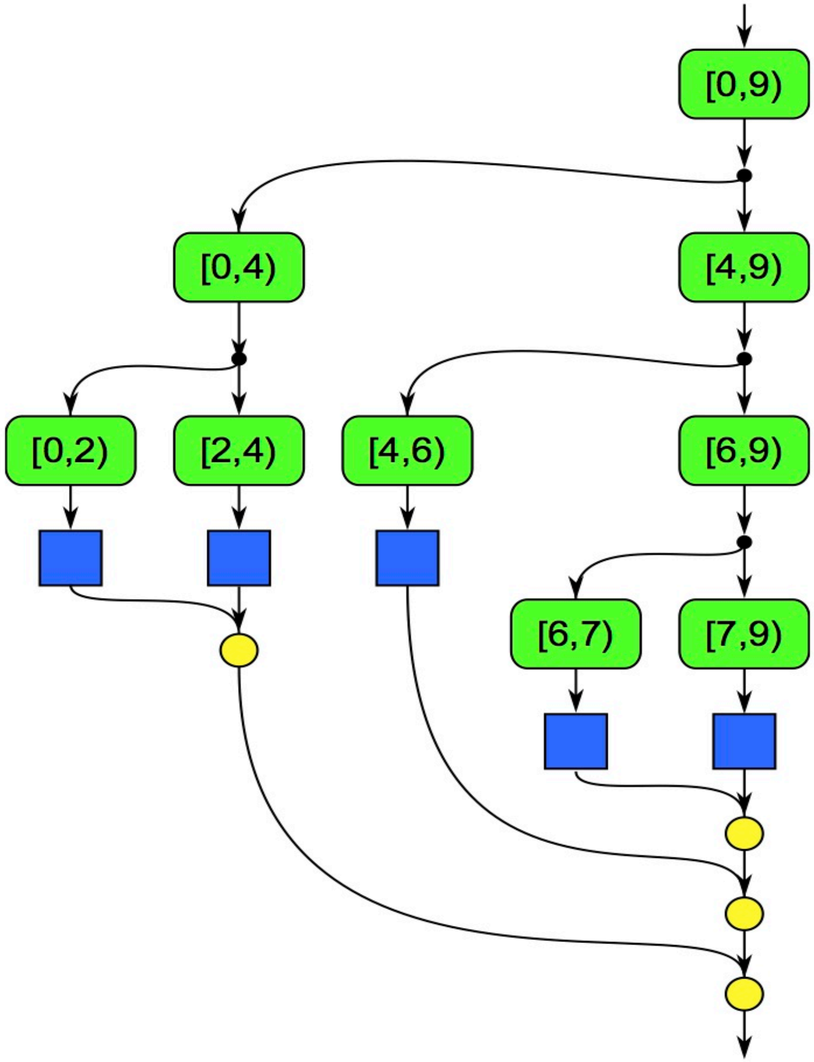
- Simple Idea for Concurrency
  - “fork” new tasks
  - “join to delay execution until forked tasks have finished

Fork Join Control Flow

```
fork B();
C();
join
```



Executing Map as Fork/Join



Work Stealing

- The runtimes for TBB/CilkPlus do something known as work stealing
- Each **worker thread** has a queue of tasks
- When a call to fork is executed, the thread puts the task on its queue
- This provides good locality…but can cause starvation
- So, if a thread runs out of work, it "steals" some tasks from a different queue

Steal Continuation vs. Steal Child

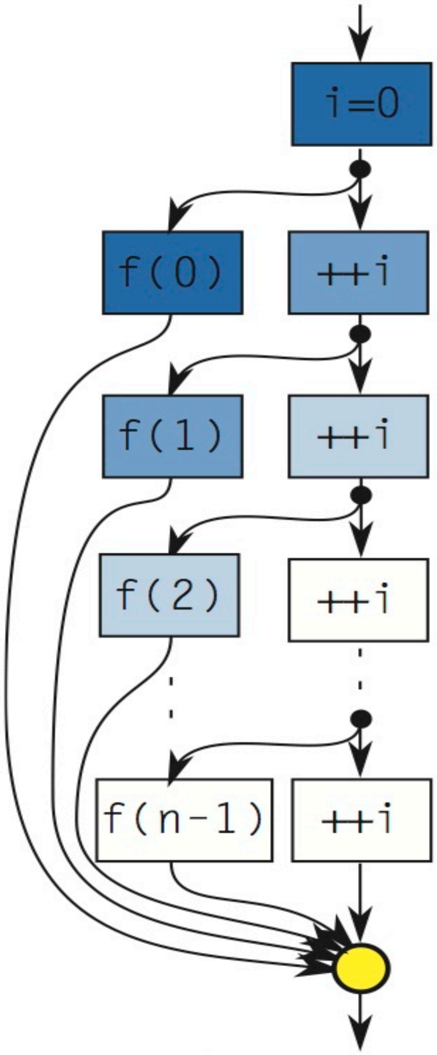
- TBB and CilkPlus handle fork in different ways
- Given code of the form:

```
    fork f();
    g();
    join;
```

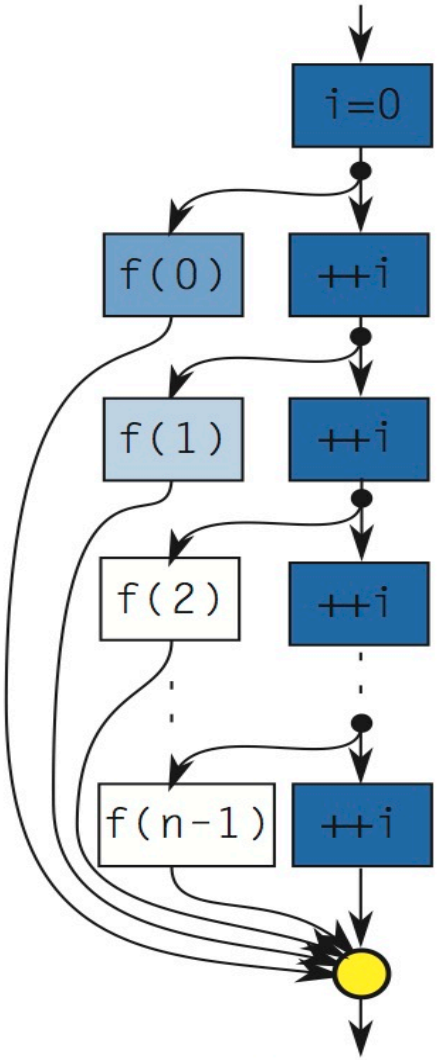
- A TBB thread would put f() on its queue and then execute g(). It would only start working on the queue when it got to the join.
- A Cilk thread would put **both** f() and the remainder of the program (g(); join; etc) on its queue. It is probable that it will execute f() before g().

Steal Continuation vs. Steal Child

Steal Continuation

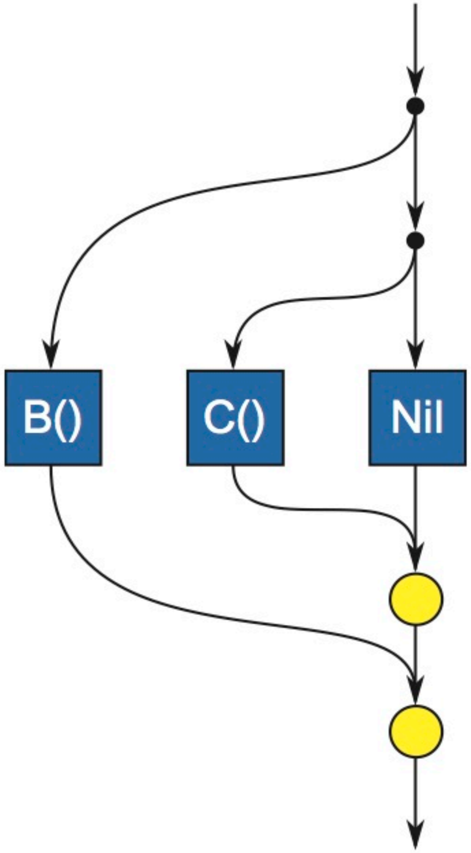


Steal Child



Extra Forking

- Simple Idea for Concurrency
  - “fork” new tasks
  - “join” to delay execution until forked tasks have finished



Don’t do this!

Performance of Fork/Join

Let A||B be interpreted as “fork A, do B, and join”

Work: $T(A||B)_1 = T(A)_1 + T(B)_1$

Span: $T(A||B)_\infty = \max(T(A)_\infty, T(B)_\infty)$

From these you can figure out the Work/Span of algorithms using the asymptotic analysis technique you learned in CIS 315/621