Lec 19

* Different parallelism

  * Deterministic parallelism — well-defined, deterministic answer, want functional programming

  * Non-deterministic parallelism — answer may vary as threads interfere, time happen affect things
    └ usually called concurrency

* Cost graphs

  Helps understand work/span of parallel programmes.

  It's a <u>directed acyclic graph</u> (DAG)!

  - Source node : no edge goes in
  - Sink node : no edge goes out
  - Edges : dependencies

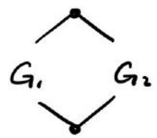
  Base Case: sink = node

  •

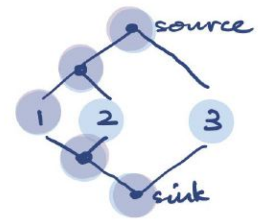
  Sequential composition : do one graph then next graph
  viz. series

$$\begin{array}{c} G_1 \\ \downarrow \\ G_2 \end{array}$$

Parallel Composition viz. fork-join
viz. parallel



(1+2) × 3



Analyzing such graph:

- Work : } num of nodes in { graph
- Span : } num of nodes in { longest path

work: 7
span: 5

Note this won't be same number as old approach. But asymptotically same

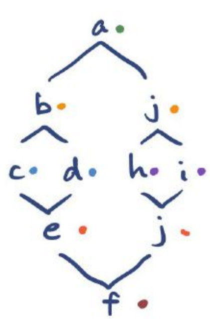
Brent's Theorem: an expression with work W and S can be evaled on p processors in time

$$\Omega\left(\max\left(\frac{W}{p}, S\right)\right)$$

Just an estimate. Hard to do that well, assumes full utilisation all the time

Scheduling - pebbling

Consider:



	processors	
time	p1	p2
1	a	
2	b	j
3	c	d
4	h	i
5	e	j
6		f

Brent's theorem: $W = 10$, $S = 5$, $p = 5$
→ predicts 5

Actual : 6

Sequences

Abstract datastruct to do parallelism

Notation: $\langle x_0, \dots, x_{n-1} \rangle$ ← list is sequential, but seq gives parallel access

Eeq $\langle x_0, \dots, x_{n-1} \rangle \cong \langle y_0, \dots, y_{m-1} \rangle$ iff $m=n$ and $x_i \cong y_i \ \forall i \in 1 \dots n-1$

Note empty () gives value restriction.
it's not value and ML doesn't allow poly exp

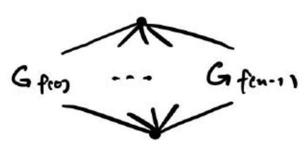
```
Signature SEQUENCE
  sig
    type 'a seq
    exception Range of string
    val empty : unit -> 'a seq
    val tabulate : (int -> 'a) -> int -> 'a seq
    val length : 'a seq -> int
    val nth : 'a seq -> int -> 'a
    val map : ('a -> 'b) -> 'a seq -> 'b seq
    val reduce : ('a * 'a -> 'a) -> 'a -> 'a seq -> 'a      <- like fold
    val mapreduce : ('a -> 'b) -> 'b -> ('b*'b -> 'b) -> 'a seq -> 'b
    val filter : ('a -> bool) -> 'a seq -> 'a seq
     :
  end
```

Cost Graphs (assumes pure functional)

empty () $\cong \langle \ \rangle$

O(1) work & span

tabulate f n $\cong \langle f(0), \dots, f(n-1) \rangle$



Work: if all the G constant time then O(n)
Span: if . . . O(1)

length $\langle x_0, \dots, x_{n-1} \rangle \cong n$

← promise $O(1)$

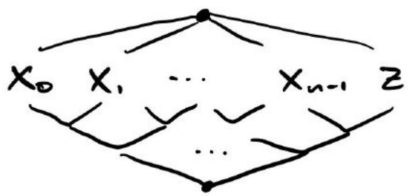
nth $\langle x_0, \dots, x_{n-1} \rangle \; i \cong x_i$ if i in range else exception

map $f \; \langle x_0, \dots, x_{n-1} \rangle \cong \langle f(x_0), \dots, f(x_{n-1}) \rangle$

W/S same as tabulate

requires g associative in order to group things. Let $x \odot y = g(x, y)$

reduce $g \; z \; \langle x_0, \dots, x_{n-1} \rangle \cong x_0 \odot \cdots \odot x_{n-1} \odot z$



If g constant, work $O(n)$
span $O(\log n)$

Aside: in 210 we require z to be identity for g.
viz. $g(x, z) = x$.

(sloppy notation)

filter $p \; s \cong \langle x \in s \mid p \, s \rangle$ in same order

If $p \in O(n)$, work $O(n)$
span $O(\log n)$

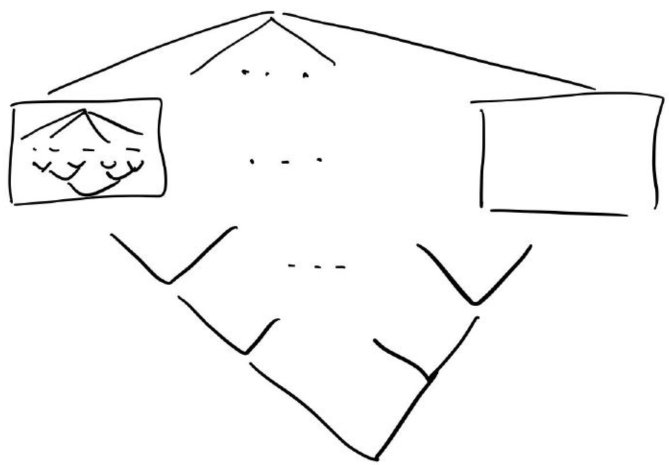
why? maybe look at some $\log n$ span impl

(note not $O(n)$ work)

```
fun filter p s =
  let
    val nothing = empty ()
    fun keep x = if p x then singleton x else nothing
  in
    mapreduce keep nothing append s
  end
```

Ex: count num of students in room

```
fun sum (s: int Seq.seq) : int =
    Seq.reduce (op+) 0 s
type row = int Seq.seq
type room = row Seq.seq
fun count (class : room) : int =
    sum (Seq.map sum class)
```



Work $O(mn)$
Span $O(\log m + \log n)$