humans are insufficiently parallel

- ~~abstract~~ (~~[illegible]~~)
- ~~truthful~~

my mind can keep in mind 7, 8 things.

machine models tell you how much food is when you finish eating it.

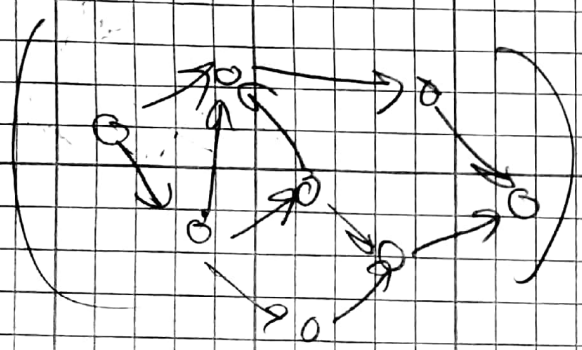
book restaurant.

make ~~restaurant~~

CS as advanced as restaurants.

two pipes is comma but parallel

$(a, b) \rightsquigarrow (a \parallel b)$



is not a fork join dag

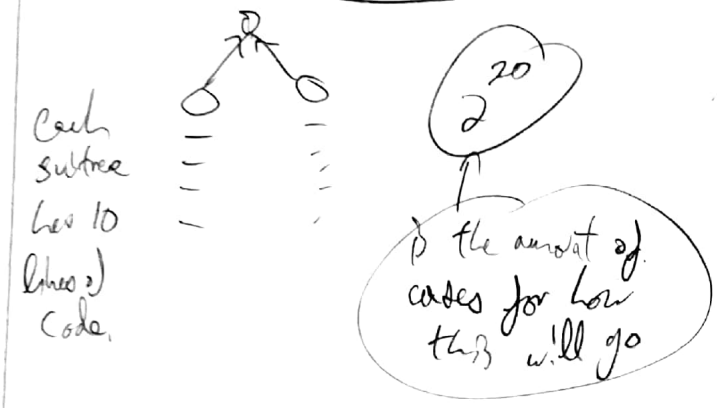
Acar

umutac@gmail.com
umut@cmu.edu

language based constructs.
- abstract
- truthful

R assume pure functional programs.
take shared state

no shared, mutable state!



① CBV λC is inherently parallel.

② in sequential you only care about work, which is sequential runtime. (additive)
in parallel, so have work + span/depth.
plus max

E

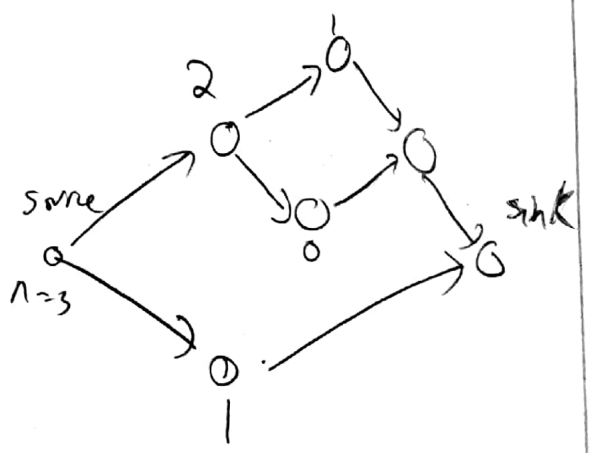
```
fn f x =
  if x ≤ 1 then x
  else let (a,b) = (f(x-1) || f(x-2))
    in a+b
  end.
```

nested, fork/join

this is a great fork, join

$$w(n) = \begin{cases} 1 & \text{if } n \le 1 \\ w(n-1)+w(n-2)+1 & \text{else} \end{cases}$$

$$d(n) = \begin{cases} 1 & \text{if } n \le 1 \\ \max(d(n-1), d(n-2)) + 1 & \text{else} \end{cases}$$



Mar

pg. 2

P Series-parallel Dags are fork-join dags

"efficient" in a "low span" setting is typically $\log^k$, where k is a multiplicative power and 2, 3, 4, etc.

Provably efficient implementations

Bounded implementations

someone gives me an algorithm w/ a W and S value.

I need to give them a language that can express it.

in correctness terms, reasoning about sequential version is the same as reasoning about parallel

and result is the same

translation from person's pseudocode to my machine should preserve work

$\leq$ by convention, our assembly lang has $p = 6$

where T is runtime

lower bounds: $\left\{ T_p \geq \frac{W}{p},\ T_p \geq d \right.$

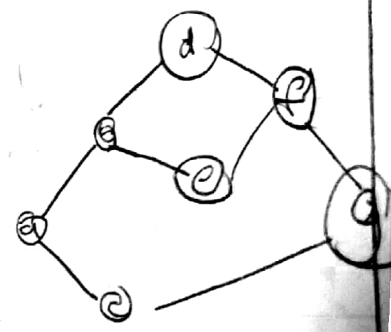
let $T_p^{optimal}$ be a scheduler algorithm that finds the most optimal runtime on p cores

$T_p^{optimal}$ is NP-complete.

but you can approximate

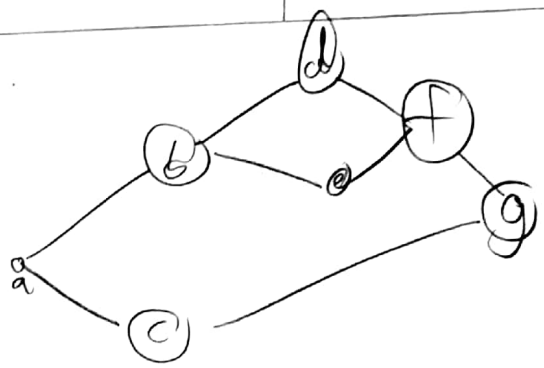
1.	a	
2.	b	c
3.	d	e
4.	f	
5.	g	

is a scheduler for the Series parallel DAG



pg 3

1. src a
2. b c
3. d e
4. f
5. g sink



w_i → # of vertices at level i

D level: level is horizontal distance from sink

$$W = \sum_{i=1}^{d} w_i$$

ceiling

$$T_p = \sum_{k=1}^{d} \left\lceil \frac{w_k}{p} \right\rceil$$

$$= \sum_{k=1}^{d} \left(\left\lfloor \frac{w_k}{p} \right\rfloor + 1 \right)$$

$$= \left(\frac{1}{p} \sum_{k=1}^{d} w_k \right) + d$$

$$= \frac{W}{p} + d$$

where $opt = length(T^{opt}$

upper bounds

Clash: $T_p \leq \frac{W}{p} + S \leq 2 * opt$

lower bounds

$T_p \geq \frac{W}{p}$, $T_p \geq S$ } $opt \geq \max(\frac{W}{p}, S)$

Cost of scheduler

vs.

length of schedule

<u>a greedy scheduler</u> if $\exists$ idle processor
and if $\exists$ ready vertex
then assign

$$T_p \leq \frac{W}{p} + S \cdot \left(\frac{p-1}{p}\right)$$

for every step, if you're either working or idling you toss a coin into the appropriate bucket, PER NODE

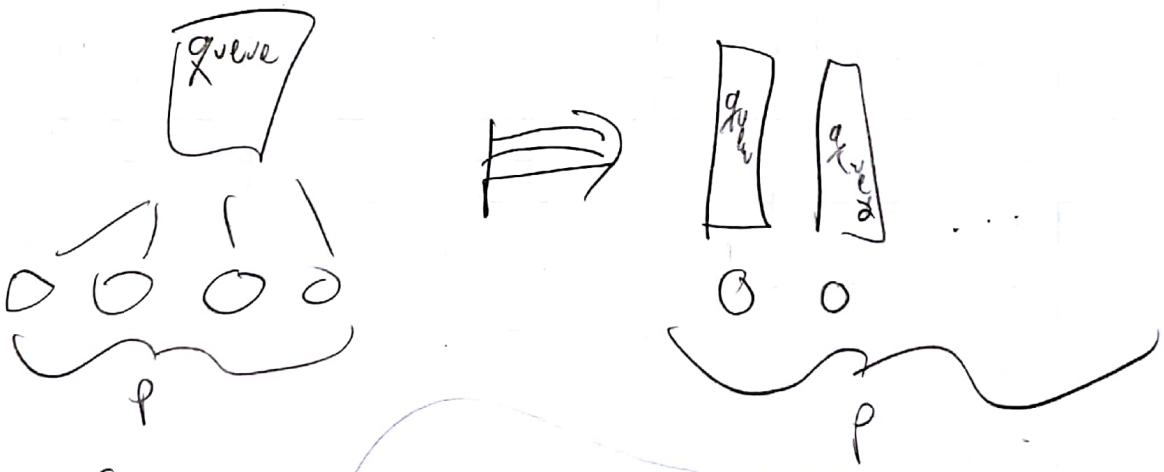
by end of process, you should have W coins

in any step, you can put p-1 coins in

$\times S$ because you have s layers.

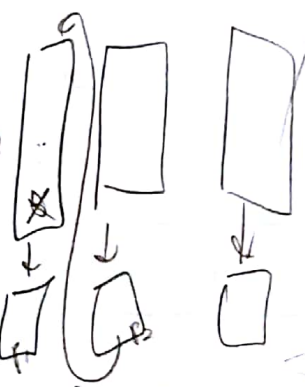
$$T_p = \frac{\text{total coins}}{p}$$
$$\leq \frac{W + (p-1)S}{p}$$

T_p^{opt} is <u>unscalable</u> because there is a <u>CENTRALIZED QUEUE</u>



We need some <u>load-balancing</u>

locally, this can be managed like a stack



steal

P_2 steals from the end of P_1's queue

this is concurrency not parallel

Unit 2.1

Clean semantics for parallelism

Clean cost model

λ-Clc (work, span)

$T_p \sim \frac{W}{p}$

$\mapsto \frac{W}{p} + d$

$\leq 2 \times (OPT)$

work buildings have high compression strength, low tension strength

steel buildings are for tension strength

multi core computers are like steel buildings

- We want low-span algs,

- Fundamental data structure: Sequences

  - assume immutability

$a = (a_0, a_1, \ldots, a_n)$

$a[k] = a_k$; $\leftarrow$ $O(1)$ work, $O(1)$ span

$\text{length}(a) = n = |a|$; $O(1)$ work, $O(1)$ span

$\text{Subseq}(a, i, j) = a[i:j]$; $O(1)$ work, $O(1)$ span

$\text{splitmid } a = \text{subseq}(a, [0, \ldots \underset{\lfloor \frac{n}{2} \rfloor - 1}{\frac{n}{2}}])$, $\text{subseq}(a, [\underset{\lfloor \frac{n}{2} \rfloor}{\frac{n}{2}} + 1 \ldots n-1])$

tabulate :: $(\text{int} \to \alpha) \to \text{int} \to \alpha \text{ seq}$

tabulate $(\lambda i \to i)$ $n = \langle 0, 1, \ldots, n-1 \rangle$

$\text{wk}(\text{tabulate}) = O(n) \rightsquigarrow = \sum_{k=0}^{n-1} (\text{wk}(f(k)))$

$\text{Spn}(\text{tabulate}) = O(1) \rightsquigarrow = \max_{k=0}^{n-1} (f(k))$

generalize

where f is some $\lambda x.e$

empty = tabulate $(\lambda k.k)$ 0

singleton e = tabulate $(\lambda k, e)$ 1

map f a = tabulate $\lambda k.f(a[k])$ $|a|$

append a b = tabulate $\left(\begin{array}{l}\lambda i. \text{ if } i < |a| \\ \text{then } a[i] \\ \text{else } b[i - |a|]\end{array}\right)$ $(|a| + |b|)$

$\text{wk}(\text{append } a\ b) = O(|a| + |b|)$

$\text{Spn}(\text{append } a\ b) = O(1)$

iterate : $\beta \to ((\alpha \times \beta) \to \beta) \to \alpha \text{ seq} \to \beta$

iterate b $(\lambda(x, y).x + y)$ a

next running

Unit 2.2

some funct w $(1,0,4,5,2,0,0,3,4) \mapsto (0,1,1,4,5,2,2,2,3)$

(helper) fn skipZero (x,y) = if $x>0$ then x else y

insertionSort a = iterate () insert a

fn insert (x,a) = iterate, or tabulate.

insertionSort $\langle 3,2 \rangle$

$\mapsto$ $\langle\rangle$
$\langle 3 \rangle$
$\langle 2,3 \rangle$
$\langle 1,2,3 \rangle$

where $f = \lambda(x,y).(y \times y)$

$$wk(\text{iterate } b\ f\ a) = \sum_{k=0}^{|a|-1} \left(wk(f(x, a[k])) \right)$$

$$Spn(\text{iterate } b\ f\ a) = \sum_{k=0}^{|a|-1} \left(Spn(f(x, a[k])) \right)$$

---

we're not gonna parallelize w/o explicitly assoc of input binary function (to iterate)

w assoc, we can divide & conquer

iterate Prefixes : $\beta \to (\beta \times \alpha \to \beta) \to \alpha\ seq \to \beta\ seq$

reduce id f a $\mapsto f(\ldots f(f(id, a[0]), a[1]), \ldots, a[n-1])$ elif a = 1 then a[0] parallel

fn reduce id f a = if $|a| = 0$ then id else let (b,c) = splitmid a

(rb, rc) = reduce id b || reduce id c

in f (rb, rc)

end...

$$wk(\text{reduce id } f\ a) = 2\, wk\left(\frac{|a|}{2}\right) + O(1)$$
$$= O(|a|)$$

$$Spn(\text{reduce id } f\ a) = \max\left(Spn\left(\frac{|a|}{2}\right) + 0, 1 \right)$$
$$= \lg(|a|)$$

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

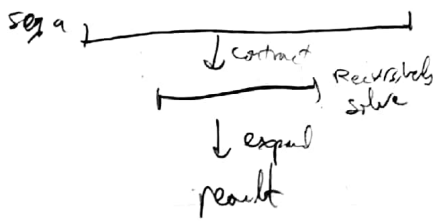
MSort = reduce () merge (map singleton a)

$wk: O(n \lg n)$ $Spn: O(\lg^2 n)$

Unit 23

Contraction

```
fn reduce id f a =
  if |a|=0 then id else if |a|=1 then f(id, a[0])
  else b=tabulate (λi.(f(a[2i], a[2i+1])) |a|/2
    reduce id f b
```



$$W(n) = W\left(\frac{n}{2}\right) + O(n) = O(n)$$

$$Span(n) = Span\left(\frac{n}{2}\right) + 1 = O(\lg n)$$

---

iterate : sequential

reduce : parallel

you need it on a per-prefix basis.
You can parallelize by reducing on each,
but this is $W(n) = O(n^2)$

helper fnc,

Scan id f a =

(reduce id f (),
(a[0]),
(a[0], a[1]),
⋮

15216 CMU

Sequences
Contraction
Divide & conquer

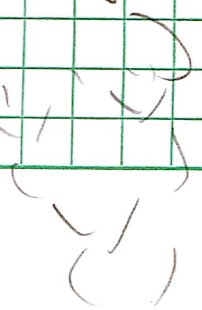
table lite is "interpret a set as a function from int, as array"

iterate is fold(l, x)

reccomb, distrib

fold is a contraction, take one step w a binary operation

its like pairwise fold



4b

flatten A : (α seq) seq → α seq

flatten A = reduce ⟨⟩ append A : $W_K(\text{flatten}) = O(|A| \log |A|)$

result

$S_{pn}(\text{flatten}) = O(\log |A|)$

Gyy 3.1

~~Scan, reduce, plus~~

$W(n) = 2W(\frac{n}{2}) + O(n)$
$= O(n \log n)$

$(1,0,2,7,0,5) \mapsto (0,1,1,3,10,10,15)$
$\downarrow$
$(1,9,5)$

$Spn(n) = O(1) + Spn(\frac{n}{2}) + O(1)$
$= O(\log n)$

update : $(\alpha\ seq) \times int \times \alpha \to \alpha\ seq$

update A i v = tabulate $\lambda k.$ if $(i=k)$ then v else A[k] $|A|$

$wk(update) = O(|A|)$ $\quad Spn(update) = O(1)$

multiupdate

inject : $\alpha\ seq \times (int \times \alpha)\ seq \to \alpha\ seq$

inject A V = iterate A update V

inject $\overset{A}{\langle a,b,c,d\rangle}$ $\overset{V}{\langle (0,x),(2,y),(0,z)\rangle}$
$\mapsto \langle z,b,y,d\rangle$

$wk(inject) = O(|A| + |V|)$

$Spn(inject) = O(\log |V|)$

suppose:
map f A = $(a,b,c,d,e,f) \mapsto (t,f,f,t,t,f)$
$\xrightarrow{scan} (\langle a\rangle,\langle\rangle,\langle\rangle,\langle d\rangle,\langle e\rangle,\langle\rangle)$ $wk(|A|)$ $Spn(1)$

filter : $(\alpha \to bool) \to (\alpha\ seq) \to (\alpha\ seq)$

reduce $\langle\rangle$ append $(\langle a\rangle,\langle\rangle,\langle\rangle,\langle d\rangle,\langle e\rangle,\langle\rangle)$: $wk = O(\lg n)$, $Spn = O(n \lg n)$

$wk(filter) = O(|A|)$

$Spn(filter) = O(\lg n)$

```
fun filter f A = let val f1 = map (fn x => if f(x) then 1 else 0) A
                     val (off, total) = scan 0 (λxy. x ⊕ y) A
                     val U = tabulate (fn i => (off[i], A[i])) |A|
                 in  inject (tabulate (fn _ => a[0]) total) U
                 end.
```

$wk(filter) = O(n)$

$Spn(filter) = O(\lg n)$

UNIT 3.1

Maximum Contiguous Subsequence (MCSS)

Kadane solved

$wk(MCSS) = O(n)$

Brute force

```
fn bf a =
    let
        b = (a[i..j] : 0 ≤ i,j ≤ |a|)   (i.e. tabulate)
        c = map (λx. reduce "+" x) b
    in reduce -∞ max c
end
```

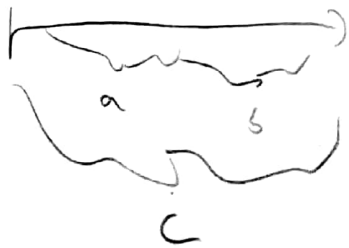
$wk(bf) = O(n^3)$ "tabulate then reduce" map reduce tabulate is $(O(n))(O(n^2)) = O(n^3)$

$Span(bf) = O(\lg n)$

MCSS_B consider all subsequences that begin at a given index and find the max

```
MCSS_B a i =
    let
        b = a[i..(|a|-1)]
        (c,s) = scan "+" 0 b
    in map (reduce max -∞ c a)

fn MCSS-Reduction a =
    let
        b = ((MCSS_B a i) : 0 ≤ i < |a|)
    in reduce map -∞ b
end
```



UNIT 3.2

```
fun MCSS_C a i =
  let
    b = a[0..i]
    (c,s) = scan + 0 b
    m = reduce min ∞ c
  in
    s-m
  end.
```

$$\left(\begin{array}{l} wk(MCSS\_C) = O(n) \\ Sp(MCSS\_C) = O(\lg n) \end{array}\right)$$

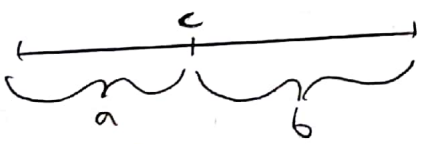
```
fun MCSS a =
  let
    b = scan + 0 a
    c = scan min ∞ b
    d = (b[i] - c[i] : 0 ≤ i ≤ |a|)
  in reduce max -∞ d
  end
```

$wk(MCSS) = O(n)$

$wk(MCSS) = O(\lg n)$

<u>Divide & Conquer MCSS</u>

$wk = O(n)$



$M_a = MCSS$

$M_b = MCSS$

M_{ab} = best across (a,b)

ret Max (M_a, M_b, M_{ab})

already solved because we have all the seq, that end $\overline{ac}$ and those that sta c, $(\overline{cb})$, so we can add them.

$(M_a, P_a, S_a, T_a) = MCSS\ a$

$(M_b, P_b, S_b, T_b) = MCSS\ b$

You can also return the max of these and the max in a and b.

return max $(M_a, M_b, S_a + P_b, P_a, T_a + P_b, S_b, S_a + T_b)$