Acar 2

yesterday : - clean semantics for parallelism
- clean cost model

λ-calc serves as a perfectly general model

work & span
↳ length of dependencies (longest)

$T_p \sim \frac{W}{P}$ ↖ best to hope for

$$T_p = \underbrace{\frac{W}{P}}_{O\left(\frac{n}{p}\right)} + \underbrace{S}_{\log^n n} \leq 2 \times OPT$$

close to linear speedup

~ 2-3x of handwritten C
model developed at CMU

$e_1 \underbrace{||}_{\text{"par"}} e_2$ emph. can run in parallel

cannot take any program and
expect ~~good~~ speedup

want $\frac{W}{P} + S \underset{\text{close to}}{\sim} \frac{W}{P}$

if S close to W, then it has bottlenecks

Can we design alg's that have low span?

requires tweak / new way of looking at things

ex. bricks - compression strength
cannot build heavy tower
steel - tension strength

likewise multi-core computing
we need to account for this change

Fundamental data structure / type

$\boxed{\text{Sequences}}$

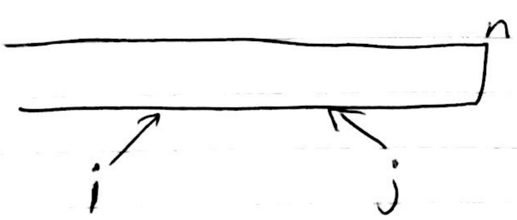
Sequence $(a_0, a_1, a_2, \ldots, a_n)$

$$\left.\begin{array}{l} a[0] = a_0 \\ a[i] = a_i \end{array}\right\} \begin{array}{l} O(1) \text{ work} \\ O(1) \text{ span} \end{array}$$

$$\text{length } a = |a| \quad \Big\} \begin{array}{l} O(1) \text{ w} \\ O(1) \text{ span} \end{array}$$

subsequence $(a, i, j) = a[i \ldots j]$ $O(1)$ wk, span

- different impl's - diff work / span bounds
- assume array-based implementation



<u>split mid</u> $a = (\text{subseq}\ (0 \cdots \lfloor \frac{n}{2} \rfloor - 1)$
$, \text{subseq}\ (\lfloor \frac{n}{2} \rfloor, \dots, n-1)$
$)$

- most updates are bulk — n updates
- immutability & parallel work well together

<u>tabulate</u> : $(\text{int} \to \alpha) \to \text{int} \to \alpha\ \text{seq}$

tabulate $(\lambda_i \to i)\ n = \{0, 1, \dots, n-1\}$

work: $O(n)$ but depends on function
(under assumption fn does const work)

span: $O(1)$

general case

<u>Work</u>
$$\sum_i W(f(i))$$

<u>Span</u>
$$\boxed{\max_i S(f(i))}$$

important

impl various operations using tabulate

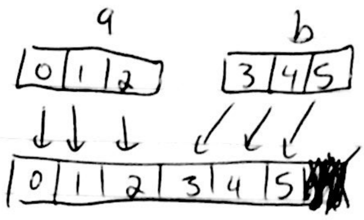
tabulate — empty $\equiv$ tabulate $(\lambda i.i)\ 0$
— singleton e $\equiv$ tabulate $(\lambda i.e)\ 1$
— map f a $\equiv$ tabulate $(\lambda i.\ f(a[i]))\ |a|$

$$(a_0, a_1, a_2, \dots, a_n)$$
$$\Downarrow f$$
$$(fa_0, fa_1, \dots, fa_n)$$

~~tabulate~~ — append a b = tabulate

$$\left(\lambda i.\ \text{if } i < |a| \text{ then } a[i] \text{ else } b[i - |a|]\right)$$

$$(|a| + |b|)$$



$O(|a| + |b|)$ work

$O(1)$ span

perfectly parallel computation

<u>iterate</u> : $\beta \to ((\alpha \times \beta) \to \beta) \to \alpha \text{ seq} \to \beta$

$$a = (a_0, a_1, \ldots, a_{n-1})$$
$$\quad\ 0 \quad 1 \quad \ldots \quad n-1$$

iterate b $(\lambda(x, \overset{y}{a_i}).\ x + \overset{y}{a_i})$ a

iterate 0 $(\lambda(x, a_i).\ x + a_i)$ a — sum of elements

$$((((0 + 1) + 2) + 3) + 4) \ldots (n-1))$$

ex. $(1, 0, 4, 5, 2, 0, 0, 3, 4)$

each mapped to ~~left~~ right-most non-zero element before the element

$(0, 1, 1, 4, 5, 2, 2, 2, 3)$

fun skipzero (x, y) = if $x > 0$ then x else y

iterate 0 skipzero

design insertion sort using iterate

iterate a = empty () insert a

~~[λ f~~

r is sorted

fun insert ($\cancel{x}$, a) = iterate ..

(or tabulate ..)

ex: insert sort $\langle 3, 2, 1 \rangle$

$\langle \rangle$

$\langle 3 \rangle$

$\langle 2, 3 \rangle$

$\langle 1, 2, 3 \rangle$

$$\text{Work} = \sum_i W(f(x, a[i]))$$

$$\text{Span} = \sum_i S(f(x, a[i]))$$

work, span are basically the same

Ex: Summing up a sequence a

iterate 0 $(\lambda(x,y) = x+y)$ a

$$(((0 + a[0]) + a[1]) \; \ldots \; a[n-1])$$

don't need to do in this order
bec. addition is associative

assoc – can reorder operations

skip zero $(1, 0, 2, 3, 2)$

$(0, 1, 1, 2, 3) \Rightarrow$ sequence

<u>iterate Prefixes</u> (gives seq of intermediate results)

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

(<> ,
< a[0] > ,
< a[0], a[1] > ,

⋮

)

$\boxed{\text{reduce}}$ id f a

where id is identity for f.

$f(f(f(id, a[0]), a[1]), a[2])$

ex. reduce 0 $(\lambda(x,y).x+y)$ a

$id + a[0] + a[1] + \dots a[n-1]$

impl. using divide & conquer to get parallelism

```
fun reduce id f a =
    if |a| = 0 then id
    else if |a| = 1 then a[0]
    else let(b,c) = splitmid a
          (rb, rc) = reduce id b || reduce id c
      in
          f(rb, rc)
      end
```

$$W(n) = 2W\left(\frac{n}{2}\right) + O(1)$$
$$= O(n)$$
$$S(n) = \max\left(\frac{n}{2} + 0, 1\right)$$
$$= \lg n$$

mergesort = reduce <> $\underbrace{\text{merge}}_{\substack{O(\lg n) \\ \text{span}}}$ (map singleton a)

$O(n)$ W
$O(1)$ S

$O(n \lg n)$ work

$O(\lg^2 n)$ span

another technique (besides divide & conquer)

$\boxed{\text{contraction}}$

: 1

| contract (reduce size by constant fraction)

] recursively solve

↓ expand (to account for whole thing)

result

fn reduce id f a =

if $|a| = 0$ then id

else if $|a| = 1$ then $f(id, a[0])$

else

tabulate $(\lambda i . f(a[2i], a[2i+1]))$ $\frac{|a|}{2}$

reduce id f b

[Can use non-powe of 2]

(1, 2, 3, 4)

(3, 7)

(10)

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

$S(n) = S(\frac{n}{2}) + 1 = O(\lg n)$

iterate sequential
reduce parallel (if f is assoc, gives same result as iterate)

sometimes you want to compute properties of prefix of sequence

scan id f a

```
(reduce id f <>
,reduce id f (a[0])
,   :        (a[0], a[1])
,             :
,            (a[0], ..., a[n-1])
,reduce id f a
)
```

reduce reduce reduce reduce reduce reduce

(1 , 0 , 2 , 7 , 0 , 5)

```
reduce 0 skipzero ()
         ___      (1)
         ___      (1,0)
         ___      (1,0,2)
         ___      (1,0,2,7)
         ___      (1,0,2,7,0)
         ___      (1,0,2,7,0,5)
```

= ((0,1,1,2,7,7), 5)

simpler example

scan plus

$(1, 0, 2, 7, 0, 5)$

$= ((0, 1, 1, 3, 10, 10), 15)$

goal: linear work
log n span

$(\underline{1, 0}, \underline{2, 7}, \underline{0, 5})$

pairwise

$(1, 9, 5)$

scan

$= ((0, 1, 10), 15)$

elements at even positions match

$\underline{0}, 1, \underline{1}, 3, \underline{10}, 10$

left: squeeze these values in

lecture notes $\underline{\text{15210 CMU}}$

chs Sequences, Contraction