Yesterday: - Clean semantics & parallelism
- Clean cost model
Lambda calculus
Work & Span $T_p \sim \frac{W}{P}$

We want $T_p = \frac{W}{P} + S \leq 2 \cdot OPT$

$\frac{W}{P} + S \approx \frac{W}{P}$ $\alpha\left(\frac{n}{p}\right) + \alpha(\lg^3 n)$
$n \to \infty$

So S must be small, so we start with the following question:

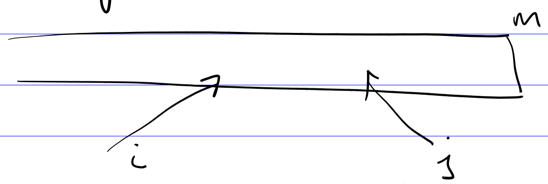
* How to design algorithms with low span?

- Fundamental Data Structure: Sequence

Sequence: $a = (a_0, a_1, \ldots, a_{n-1})$ $\left.\begin{array}{l} a[0] = a_0 \\ a[i] = a_i \end{array}\right\} O(1)$

length $a = |a|$ $O(1)$ work/span

subseq $(a, i, j) = a[i \ldots j] = O(1)$ work/span



splitmid $a = \left(\text{subseq}\left(0 \ldots \lfloor \frac{n}{2} \rfloor - 1\right), \text{subseq}\left(\lfloor \frac{n}{2} \rfloor \ldots n\right)\right)$

Key Operation: Tabulate: $(\text{int} \to \alpha) \to \text{int} \to \alpha\ \text{seq}$

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

work: $O(n)$ span: $O(1)$

$\sum_i W(f(i))$ $\max_i S(f(i))$

Implementing some std ops on seq w/ tab.

tabulate

- empty seq = tabulate $(\lambda i.i)$ 0
- singleton e = tabulate $(\lambda i.e)$ 1
- map f a = tabulate $(\lambda i. f\, a(i))$ $|a|$
- append a b = tabulate {
  $(\lambda i.$ if $i < |a|$ then $a[i]$
  else $b[i-|a|])$ $|a|+|b|$ }

$O(|a|+|b|) \longrightarrow$

iterate $\beta \rightarrow ((\alpha \times \beta) \rightarrow \beta) \longrightarrow \alpha\, seq \longrightarrow \beta$

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

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

iterate 0 $(\lambda(x, a_i)\, x+a_i)$ tabulate $(\lambda i.i)$ 5

$= ((((0 + 0) + 1) + 2) + 3) + 4$

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

EX: (1 0 4 5 2 0 0 3 4) ?
(0 1 1 4 5 2 2 2 3 4)

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

EX: Insertion Sort

inssort a = iterate ( ) insert a

fn insert (x, r) = iterate ... ?

can you use tabulate?

What's the work of iterate?

$$Work = \sum_i W(f(x, a[i])) \qquad Span = \sum_i S(-)$$

Summing up a sequence

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

$= (\cdot((0 + a(0)) + a(1)) + \cdots + a(n-1)$

But $+$ is associative, so we could split the sum into multiple parallel threads.

Iterate Prefixes

e.g. sum, $(1\ 0\ 2\ 3\ 2) \rightarrow (0, 1, 1, 3, 6, 8)$

iterate_prefixes : $\beta \rightarrow (\beta \times \alpha \rightarrow \beta) \rightarrow \alpha\ seq \rightarrow \beta\ seq$

Reduce id f a where id is identity for f.

$\cdots f(f(f(id; a(0)); a(1)), a(2)) \cdots$

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

$id + a(0) + a(1) + \cdots + a(n-1)$

fn reduce id f a = if $|a| = 0$ then id

~~else if a=1 then a(0)~~

else let $(b, c) = \{a(0, \ldots \frac{|a|}{2} - 1), a(\frac{|a|}{2}, \ldots |a|-1)$

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

in f(rb, rc) end

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

$S(n) = \max(S(\frac{n}{2}), 1) = \lg n$

Mergesort = reduce ⟨⟩ $\underbrace{\text{merge}}_{O(\lg n)\ \text{span}}$ $\underbrace{\text{map singleton a}}_{O(n)\ O(1)}$

$O(n \log n)$ work

$O(\lg^2 n)$ span

Contraction

fun 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))) $\frac{|a|}{2}$

reduce id f b }

- iterate ⇝ sequential
- reduce ⇝ parallel

Scan id f a

gives for each prefix of a

(reduce id f ⟨⟩,
reduce id f (a(0))
" " (a(0), a(1))
⋮
reduce id f (a(0) -- a(n-1)))

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

reduce 0 skipzer ⟨⟩
" " (1)
" (1 0)
" (1 0 2) = ((0 1 1 2 7 7), 15)
" (1 0 2 7)
(1 0 2 7 0)
red 0 skipzer (1 0 2 7 0 5)

Nature of this computation suggests we need iteration but in fact we can do it in parallel if we're clever

(1 0 2 7 0 5)
↓
((0 1 1 3 10 10), 15)

Want: O(n) work
lg(n) span

(1 0 2 7 0 5)
(1 9 5)
↓
((0, 1, 10), 15)

↪ appear in original comp result.

Reference: notes for CMU 15210
Lect. notes for sequences, contraction.