Umut Acar

umut.acar@gmail
umut@cmu.edu

Parallelism is HARD! ?
is EASY!

Thinking in parallel
design a good parallel alg.
efficiency < theory / practice
correctness

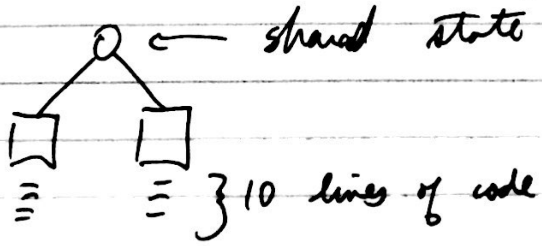
learning parallel about as difficult as sequential algorithms

language-based cost models
gives
reason abt complex math. object
w/o deceiving us

reflect accurate cost of things reasonably

Key assumption: pure functional programs

shared state, multiple actors R/W state
(semantics difficult)
exponential interleavings of code



how many interleavings? 2^{20} $10+10$ layout of code
ABAA ....

w/ pure functional (interleaving complexity goes away)

important that it is an abstraction
- runtime system can use it
- programmer can't

λ calculus - simple cost semantics
little twist from what you're used to

1. inherently parallel
   (e_1, e_2) - can eval in parallel
   also write $(e_1 \| e_2)$

2. twist : cost ?
   seq - only care about work = seq. run time (additive)
   introduces : SPAN $\rightsquigarrow$ max (not add)

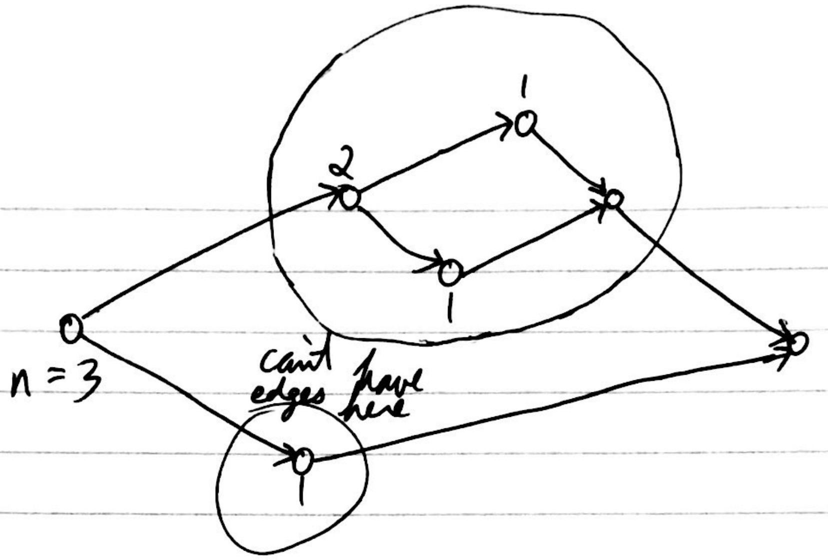
```
fn f x =
    if x ≤ 1 then
        x
    else
      let (a, b) = (f(x-1) || f(x-2))
      in
          a + b
      end
```

comma but parallel

$$W(n) = \begin{cases} 1 & \text{if } x \leq 1 \\ W(n-1) + W(n-2) + 1 & \text{otherwise} \end{cases}$$

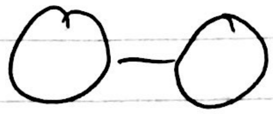
$$S(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ \max\left(S(n-1), S(n-2)\right) + 1 & \text{otherwise} \end{cases}$$

Scanned with CamScanner

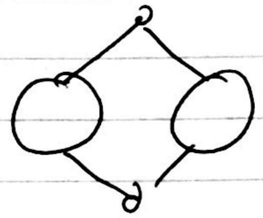


span — longest path

Series-parallel DAGs

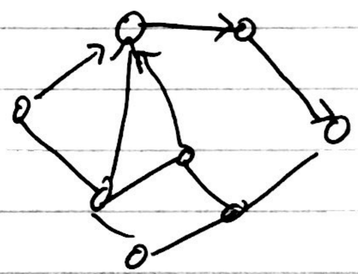


—or—



nested parallelism
fork/join parallelism

not DAG for in fork/join →



(you can w/ futures)

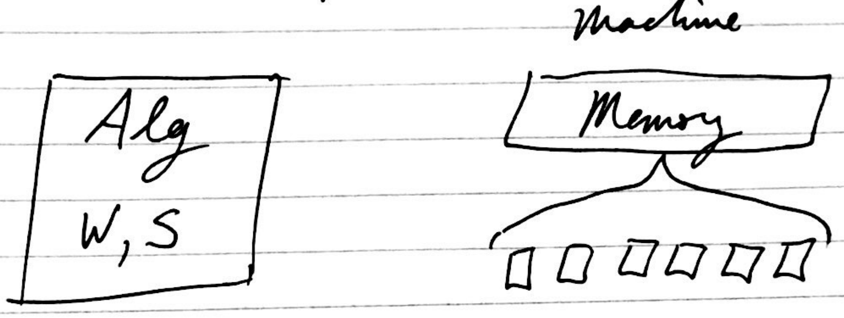
Good parallel alg:

- work close to seq alg.
  (<u>work-efficient</u>)
- low span e.g. ($\log^2 n$)

How do we make it truthful
push into runtime system
w/ provably efficient implementation

Provably efficient implementation

Bounded implementation



Want: – result should be the same as sequential
(parallel algs accept seq. semantics)

– take adv of all cores as much as possible

lower bounds
$$\begin{cases} T_p \geq \frac{W}{P} \\ T_p \geq S \end{cases}$$

upper bound caveat: T_p^{OPT} (optimal execution schedule) is NP-complete!

2 cores

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

schedule

level-by-level scheduler

level (longest distance from source)

not breadth-first (could go to 'g' before 'f' is done (after 'c'))

scheduling difficult
make sure all deps are completed before proceeding on edge in DAG

w_i is total # vertices at level i

$$w = \sum_{i=1}^{S} w_i \qquad (S - \text{span})$$

$$T_p = \sum_{i=1}^{S} \underbrace{\left\lceil \frac{w_i}{p} \right\rceil}_{\text{time each level takes}} = \sum_{i=1}^{S} \left\lfloor \frac{w_i}{p} \right\rfloor + 1$$

$$= \left(\sum_{i=1}^{S} \left\lfloor \frac{w_i}{p} \right\rfloor \right) + S \leq \sum_{i=1}^{S} \frac{w_i}{p} + S = \boxed{\frac{W}{p} + S}$$

Brent's Theorem 1972

$T_p \le \frac{W}{P} + S \quad \le 2 * OPT$

Why? $T_p \ge \frac{W}{P}$, $T_p \ge S$ $\Big\}$ $T_p \ge \max\left(\frac{W}{P}, S\right)$

$OPT \ge \max\left(\frac{W}{P}, S\right)$

Justifies truthfulness of cost model
(W/in factor of two)

The cost of scheduler
vs
The length of schedule (T_p)

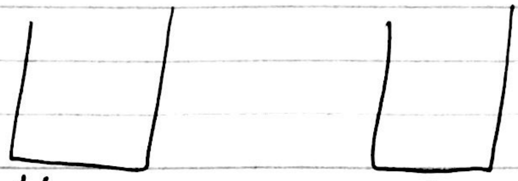
wasteful (better: greedy) $10 \cdot P + 1$
one level only uses one processor

variant: greedy

if ~~one~~ exists idle processor
and exists ready vertex
then assign
guarantees some bound:

$T_p \le \frac{W}{P} + S\,\frac{(P-1)}{P}$

Pf: arora Blumfe Plato
1997/8



wk bucket

idle bucket

toss coin into work bucket
idle toss coin into idle bucket

each step collect p coins

$$T_p = \frac{\text{total coins}}{p}$$

(why? collect p at a time)

at end: work bucket has # coins = work

idle bucket:
at any step: $\boxed{\text{max is } p-1}$
(computation not done yet)

for any step w/ idle processor:

span of remaining DAG?
↓
it reduces by 1
(making progress on span)
must have finished a layer of DAG

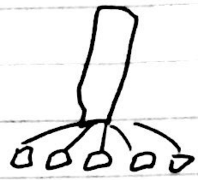
total idle coins $(P-1) \times S \Rightarrow T_p = \frac{w + (P-1)S}{P}$

Scanned with CamScanner

Current exposition does not acct for <u>cost of Scheduler</u>

Natural way to implement:

— Queue of work (centralized)



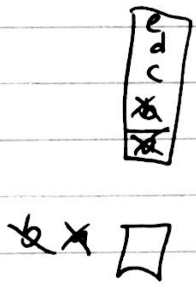
can only serve work one-at-a-time

— Distributed Queues.

← queues

← processors

vertex executes — spawns 2 vertices



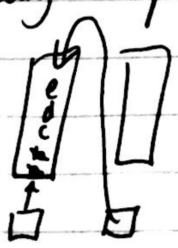
← manage as a stack

if creates new vertices, pushes onto its stack

load balancing ⇒ moving vertices to other processors

processor tries to steal work from another processor queue

stealing operates at top end



they don't contend

—concurrent algorithm (mutating, effectful, shared state difficult to implement efficiently, correctly

1995 - 2005

reasonable

w/o language model, you effectively implement your own scheduler

this alg gives $E[T_p] = \frac{W}{P} + S$ in expectation

steals happening uniformly

actually includes cost of scheduler

high-level cost model can be truthful

implemented many times (greedy scheduler)

work-stealing

Burton - Sleep

Leiserson

Blelloch

Blumofe

has good cache locality

simulates seq. execution except at steal points

Java, Parallel ML, Cilk, Haskell

Scanned with CamScanner