Cost models based on the λ-Calculus
-or-
The Church Calculus the Other Turing Machine

Guy Blelloch

Parallelism (vs Concurrency)
- sequential semantics
- offers only performance

- identify what can be run in parallel
- when analyze program, what is new cost?
- add cost model to semantics
  - "bounded" implementation
  - can map costs onto real machine

CMU course
- algorithms from complete functional style
- all parallel
  (sequential algorithms fall out as special case)

λ-calc as a cost model
compare to other cost models (ram, unbounded memory)

Church / Turing Hypothesis
- same set of functions can be computed by these
- maybe two groups should merge together

Machine models and simulation

λ-calc - diff red order, etc.

today - call-by-value

→ complexity side of theory

Handbook of Theoretical Computer Science
Ch 1: Machine models & simulation

complexity measure
time & space consumed by algorithm

conventional — abstract machines
(not a physical machine)

virtual machine
machine models

↓

language-based cost models

Important → ability to [simulate]

relationships between them (machines)
which matter

book { 1/3 defn model
2/3 run simulations

unbounded register length w/ multiplication
cannot be efficiently made on a
phys. machine
(allows unbounded parallelism)

↳ in unit time

Machine Models

Turing machines
ex. 1 tape, 2 tape, etc

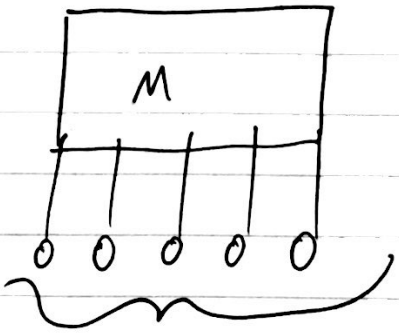
Random Access Machines
ex SRAM (succ, pred)
RAM (add, sub)
MRAM (add, sub, mult)
LRAM (log-len. words)
RAM-L (cost of instr. is word length)

Why log-n? – need to address anything (min to express address)
Pointer Coset – in practice, they grow as log n

Simulation results

Parallel Machine Models

circuit models
PRAM



p processors (constant-time mem access)
run in lock-step

2 Parts

1) Model itself
   - well-defn semantics
   - simple
   - close to programming paradigm
     (ex. C language)
     succeeds here

     no longer true in parallel world

2) Simulation
   - mapping of costs
   - good bounds on realistic machines

Work-based cost models

cost semantics

λ-calc – simple, clean

extend – constants (integers)
  – arrays

what costs?

reductions (not helpful)

Why? (lang-based over machine model)

- parallelism
  λ-calc is inherently parallel
  Turing - cannot do parallel
- more elegant (matter of taste)
- (in parallelism) closer to code & algorithms
- closer (in simulation) to "practical" machine models

Disadvantage
50 years of history

Call-by-value λ-calc BG 1995, FPGA
CBV λ-calc w/ arrays BG 1996 ICFP

$e = x \mid (e_1, e_2) \mid \lambda x.e$

$e \Downarrow v$ relation

$\lambda x.e \Downarrow \lambda x.e$ LAM

$$\frac{e_1 \Downarrow \lambda x.e \quad e_2 \Downarrow v \quad e[v/x] \Downarrow v'}{(e_1, e_2) \Downarrow v'} \text{ APP}$$

(don't need variable evaluation)
never see var at leaf
(w/o free vars)

λ-calc is Parallel

it is "safe" to evaluate e_1, e_2 in parallel
(call by name is inherently sequential)

What is cost model?

Part I Cost model $e \Downarrow v ; w, d$

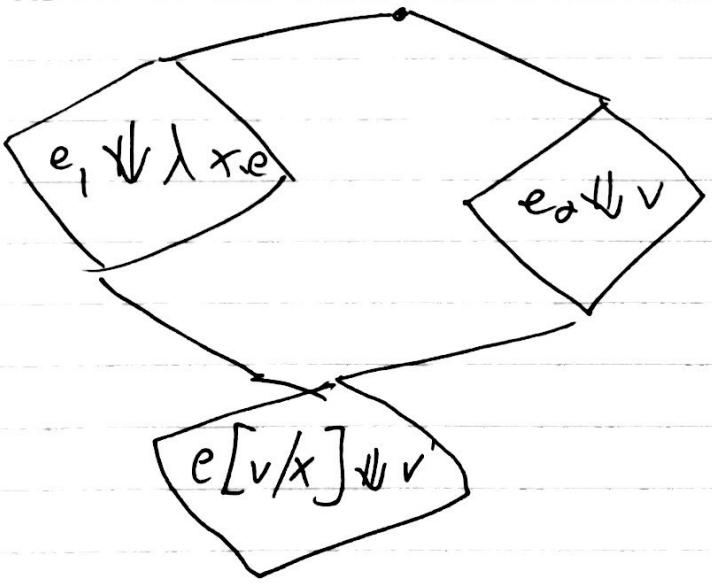
1) work (W) (sequential work) not time

2) span (D) (parallel depth)
captures dependence depth

$$\lambda x.e \Downarrow \lambda x.e \; ; \; 1, 1 \qquad \boxed{LAM}$$

$$\frac{e_1 \Downarrow \lambda x.e ; w_1, d_1 \qquad e_2 \Downarrow v ; w_2, d_2 \qquad e[v/x] \Downarrow v ; w_3, d_3}{(e_1\ e_2) \Downarrow v' ; 1 + w_1 + w_2 + w_3 , 1 + \max(d_1, d_2) + d_3} \boxed{APP}$$

(purpose is asymptotic analysis)
constants are probably higher



work matters more than span

let, datatypes, case can all be implemented in constant overhead

if e_1 then e_2 else e_3

work

(no parallelism here)

Defining basic constructs

Integers (log overhead)
- list of bits (T/F)
- church numerals do not work in CBV

Part 2 - Simulation

simulate on RAM
PRAM
- what about cost of subst, var lookup?
- " finding a redex?

Sequential CEK machine

$$(C, E, K) \Rightarrow (C', E', K')$$

control $C ::= e_1\ e_2 \mid \lambda x.e \mid x$
environment $E ::= x \to v$
continuation $K ::= done \mid arg(e, E, K) \mid fun(e, E, K)$

rules:

$$e_1\ e_2, E, K \Rightarrow e_1, E, arg(e_2, E, K)$$

$$x, E, K \Rightarrow E(x), E, K$$

$$v, E, arg(e, E', K) \Rightarrow e, E', fun(v, E, K)$$

$$v, E, fun(\lambda x.e, E', K) \Rightarrow e, E' + (x \to v), K$$

lookup, insertion non constant time

host to be persistent dictionary
(log in size of env. for tree)

Parallel Simulation
involves scheduling

PCEK sequence of triples
can eval any subset in parallel

scheduler - decide which to eval

Simulation Bounds

assumes # vars is constant

Thm. FPCA 95
If $e \Downarrow v; w, d$ then v can be calc
on a CREW PRAM w/ p processors
in $O(\frac{w}{p} + d \log p)$ time.

Can't really do better than

$$\max\left(\frac{W}{P}, D\right)$$

if $W/P > D \log P$
then "work dominates"

we refer to W/D as the parallelism
$\frac{W}{D}$ = parallelism ⇝ approx # proc you can make use of

for an inherently sequential prog, 'D' will be large, perh. as large as 'W'.

cannot do better than
linear speed up (W/P)
D -

Parallel constants — 1-calc

$c \Downarrow c, 1, 1$ CONST

APPC

Qsort - span — linear $O(n)$
parallism $O(\log n)$
work $= O(n \log n)$

not a very good parall. alg.

Tree quicksort

balanced tree input

can do filter in parallel

work step same $O(n \log n)$

filter takes $O(\log n)$

span $O(\log^2 n)$

parallelism $= O(n / \log n)$

combination of work & depth tells about alg

Cost composition

	work	span
seq	add	add
par	add	max

(divide by 2)

$$W(n) = 2\,W\left(\frac{n}{2}\right) + W_{split}(n) + W_{join}(n)$$

$$S(n) = S\left(\frac{n}{2}\right) + S_{split}(n) + S_{join}(n)$$