Summary

- Formalized parallel cost model for call-by-value λ-calculus.
- Related the model to standard parallel models --- PRAM, butterfly , hypercube.
- Analyzed sorting algorithms.

Conclusions:

- Parallel functional cost models are useful complexity models.
- Need to design parallel algorithms with appropriate data structures.

Future W ork

Models

- Random-access
- Speculative evaluation

Simulations

- Better simulations?
- Memory efficiency

Algorithms

Related W ork

Models

- Serial profiling semantics (W egbreit, LeM étayer , Sands, ...)
- Parallel ... (Roe, Blelloch, Skillicorn/Cai, Zimmermann)
- Pomset models of parallelism (Hudak)

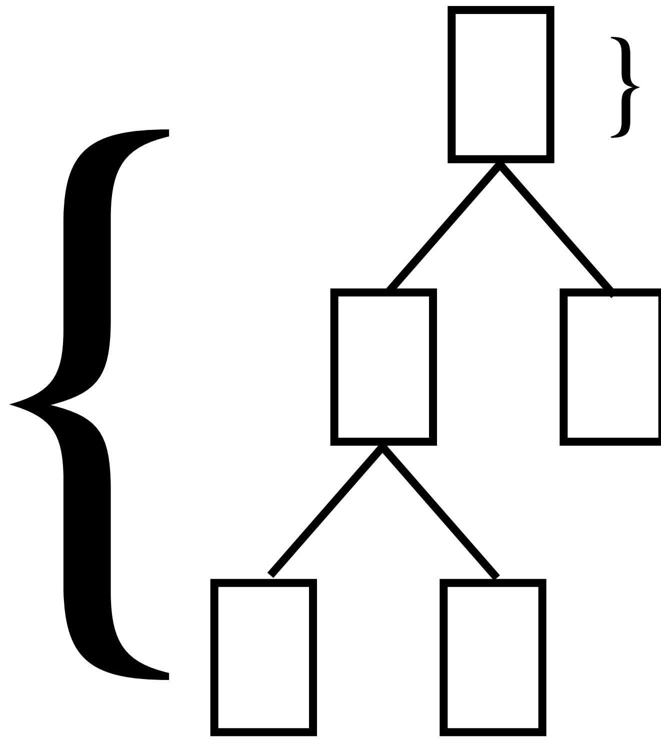
Mappings

- Mapping a language model to serial machine models (Jones)
- Efficient computation scheduling (Blelloch/Gibbons/Matias)

Algorithms

- Experimental analysis of parallelism (Sansom, W illem Bohm/Hiromoto,...)
- Functional data structures (Okasaki,...)

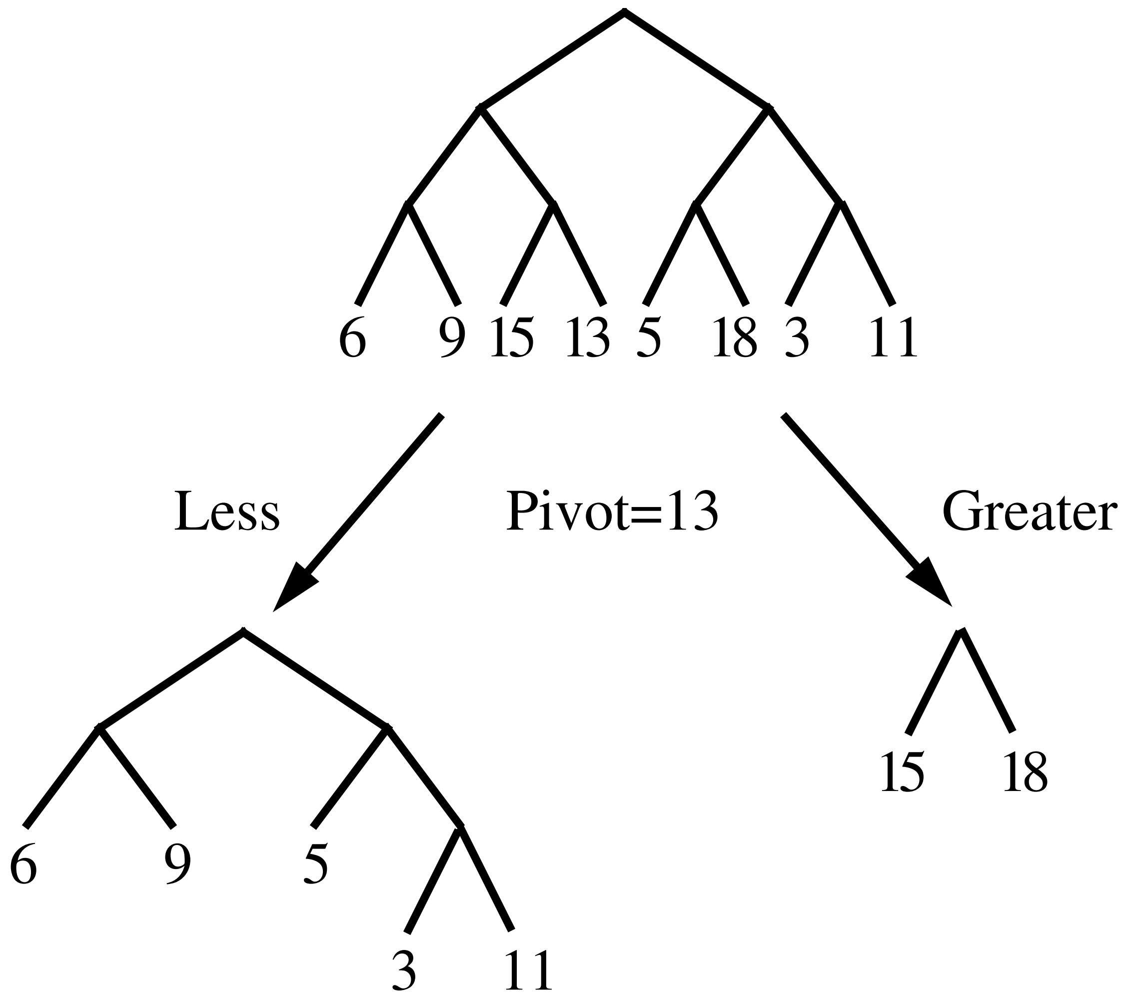
Quicksort



$w = O(n \log n)$ and $d = O(\log^2 n)$ (expected)

Mapped onto PRAM: $T = O\left(\frac{n \log n}{P} + \log^2 n \log P\right)$

Quicksort splitting



$w = O(n)$ and $d = O(\log n)$

Quicksort

```
fun qsort a =
  if (size a) < 2 then a
  else
    let val pivot = elt a ((size a)/2)
        val less  = select (fn x => x < pivot) a
        val eq    = select (fn x => x = pivot) a
        val grt   = select (fn x => x > pivot) a
    in
        append (qsort less
               (append (qsort eq)
                       (qsort grt)))
    end

fun sort a = rebalance (qsort a)
```

What About Algorithms?

Off-the-shelf algorithms have limited parallelism.

- Mergesort, quicksort recursive calls in parallel.
- Traversing list: $d = \Theta(n)$.
- Limits sort speedup to $O(\log n)$.

Need parallel algorithms

Results

- Quicksort: $w = O(n \log n)$, $d = O(\log^2 n)$ (expected).
- Mergesort: $w = O(n \log n)$, $d = O(\log^2 n)$.

Mapping the P-ECD Machine onto PRAM

- Step i takes time

$$T_i = O\left(\frac{|Q_i|}{P} + \log P\right)$$

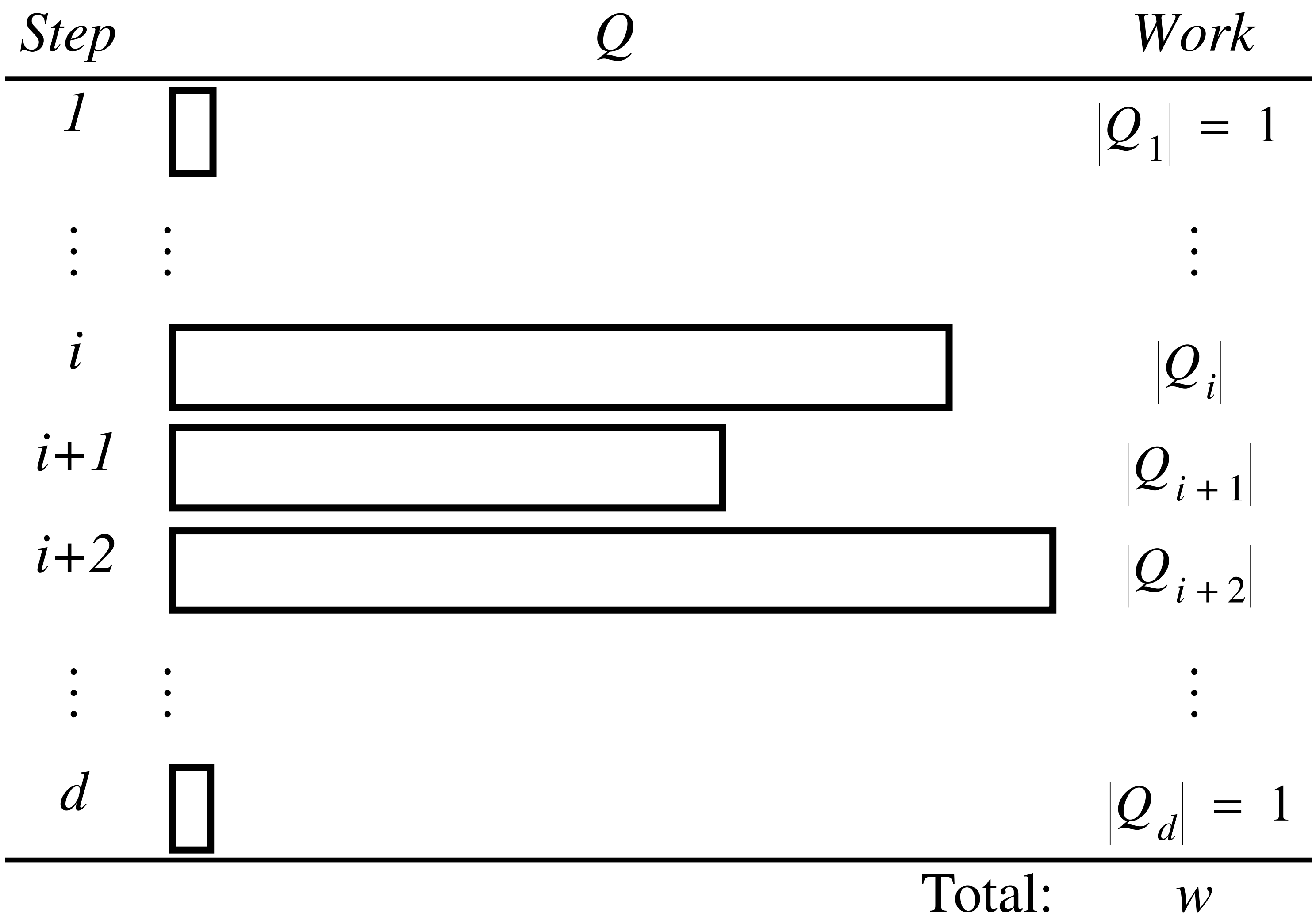
The $\log P$ is for load-balancing.

- $T = \sum_{i=1}^{d} T_i$

$$< k v_e \sum_{i=1}^{d} \frac{|Q_i|}{P} + \log P$$

$$= k v_e \left(\frac{w}{P} + d \log P\right)$$

Steps of the P-ECD machine



The P-ECD machine

State transitions on a state (Q,M) where:

$$Q = [(E_1, C_1, D_1), (E_2, C_2, D_2), \ldots, (E_q, C_q, D_q)]$$

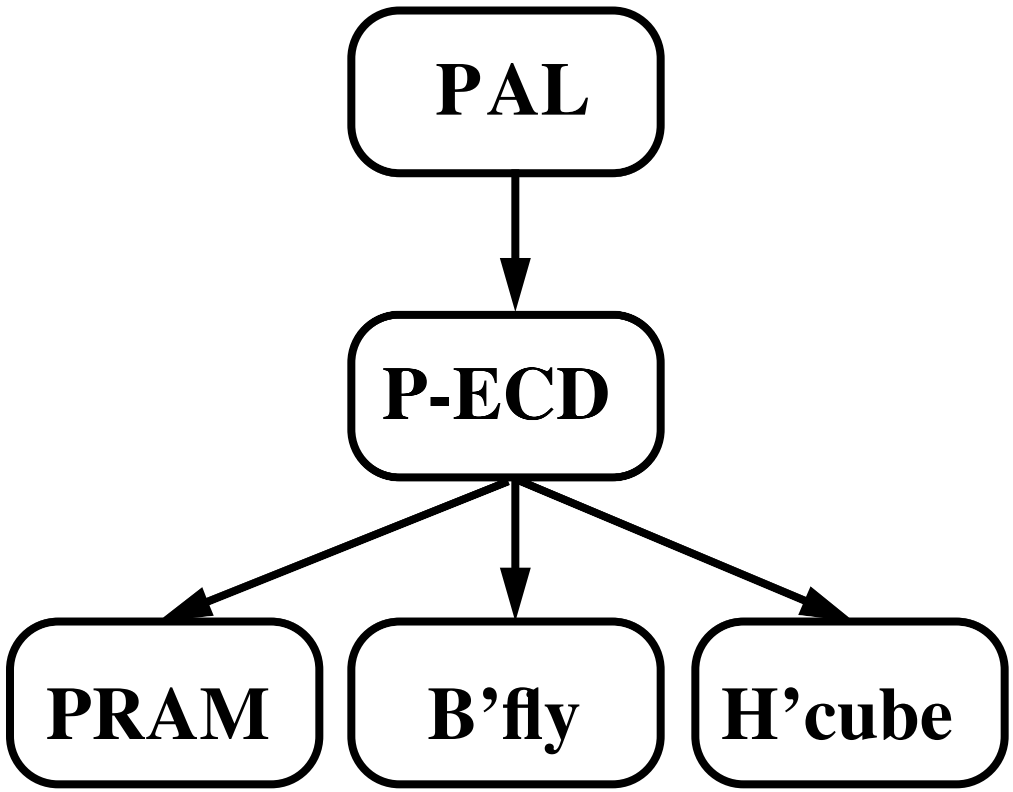
and

$$M = [V_1, V_2, \ldots, V_m]$$

Q: Substates evaluated in parallel. Size varies as states fork or terminate.

M: Partial results of one branch while waiting for other .

PAL Simulation



PAL Simulation Result

Theorem : If $[\,]\vdash e \rightarrow v; w, d$, then CREW PRAM with P processors evaluates e to v within

$$k v_e \left(\frac{w}{P} + d \log P\right)$$

time for some constant k. $\square$

Work-efficient simulation if $w \geq d P \log P$.

Within $\log P$ of optimal.

Extending the model

Basic model uses core λ-calculus

Many higher-level constructs simulated with only constant overheads

- let, letrec (Y -combinator), datatypes, conditionals

Include arithmetic to avoid non-constant overheads.

Formalizing the model

How do we formalize work and depth?

- "Profiling" operational semantics
- Parallel Applicative-order λ-calculus (P AL)
- Judgment: $E \vdash e \to v; w, d$

$$\frac{\begin{array}{l} E \vdash e_1 \to cl(E', x, e'); w_1, d_1 \\ E \vdash e_2 \to v_2; w_2, d_2 \\ E'[x \;\; v_2] \vdash e' \to v; w_3, d_3 \end{array}}{E \vdash e_1\ e_2 \to v; w_1 + w_2 + w_3 + 2, max(d_1, d_2) + d_3 + 2} \quad \text{APP}$$

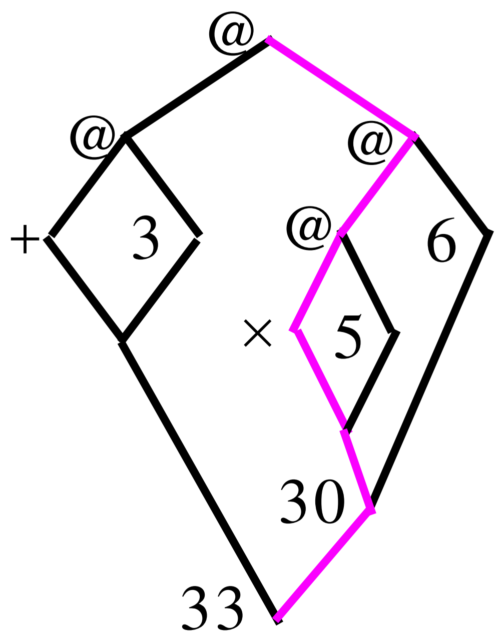
- Convenient for proving simulation bounds

Formal Model of “time”

Work : total number of operations

Depth : parallel depth of the computation

$3 + 5 \times 6$



Work = 1 1 (nodes)

Depth = 6

(longest path)

Work efficient if work within constant factor of best serial running time.

Approach

1) Define functional language complexity model

- Profiling semantics

2) Formally relate to other models (simulation bounds)

- Traditional machine models
- Other functional language models

3) Design parallel algorithms

- Serial algorithms don’t necessarily parallelize.
- No program transformations to increase parallelism.
- Represent data as trees

Goals

Overall: How parallel are functional languages?

- How fast can we solve various problems? “Big O” E.g., sorting.
- What are good algorithms and data structures?
- How to formally relate to other models? E.g., PRAM.
- What language features are important? E.g., eager vs. lazy ,
arrays vs. no arrays.

This paper

- Pure call-by-value, no language extensions (sequential)
- Function and arguments evaluate in parallel

Parallel Implementations

Parallelism via language extensions

- Data-parallel arrays, parallel-or , futures, pcall, ...

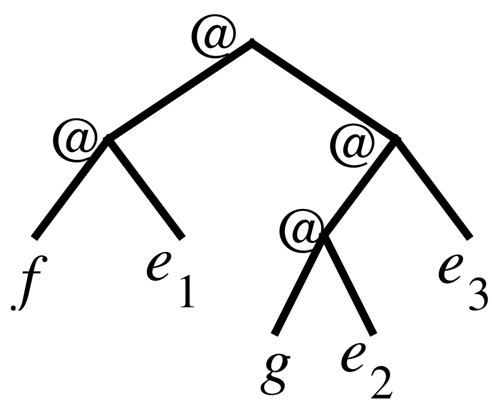
Parallelism without language extensions

- Parallel graph reduction
- Dataflow
- Compiler techniques

Inherent Parallelism in Functional Languages

Referential transparency allows parallel execution of functional languages.

$$f\ e_1\ (g\ e_2\ e_3)$$



Parallelism
in
Sequential Functional Languages

Guy Blelloch and John Greiner

Carnegie Mellon University