Parallel Algorithms

Umut Acar
Carnegie Mellon University
umutaco@gmail.com
umut@cmu.edu

July 9, 2018

1 Language Based Cost Models

- Abstract
- Truthful

Key Assumption Throughout: Pure Functional Programs
Shared state - Two programs that share the same data and each run ten lines of code (each) that rely on that data, then to prove correctness need to check 2^20 combinations or 10^100 computations. Pure functional programming does not have shared state, and therefore avoids this problem.

1.1 λ-Calculus

Cost Semantics is simple with a little twist.

1. Inherently parallel ($e_1 \parallel e_2$)
2. Twist for cost - Work (additive) and Span (max)

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

Work:

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

Span:

$$S(n) = 1 \text{ if } n \leq 1$$
$$= Max(S(n-1), S(n-2)) + 1$$

Good parallel algorithms have work as close to the sequential version as possible and have low span (O(log^d(n))).

1.2 Provably Efficient Implementations - Bounded Implementations

Someone provides an algorithm: $Alg(W, S)$, and our goal is to provide a programming language that proves the efficiency.

What we need:

- The result should be the same as sequential.
- $T_p \geq \frac{W}{p}$ and $T_p \geq S$ are lower bounds.
- Upper bound?

Note: T_p^{OPT} is NP-Complete

Schedule:

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

Brent's Theorem - 1974
W_1 is the number of work at level 1
$W = \sum_{i=1}^{S} W_i$
$T_p = \sum_{i=1}^{S} \left\lceil \frac{W_i}{p} \right\rceil = \sum_{i=1}^{S} \left\lfloor \frac{W_i}{p} \right\rfloor + 1 = \left(\sum_{i=1}^{S} \left\lfloor \frac{W_i}{p} \right\rfloor \right) + S \leq \left(\sum_{i=1}^{S} \frac{W_i}{p} \right) + S = \frac{W}{p} + S$
Implications:
$T_p \leq \frac{W}{p} + S \leq 2 * OPT$
$\left(T_p \geq \frac{W}{p}, T_p \geq S \right) \implies T_p \geq max(\frac{W}{p}, S) \implies OPT \geq max(\frac{W}{p}, S)$

Greedy:

If $\exists$ idle processor, and $\exists$ unprocessed vertex then assign
$T_p \leq \frac{W}{p} + S\frac{(p-1)}{p}$

Think of it as two buckets, a work bucket and an idle bucket. Every time a computation is made then place a coin into the work bucket. If there is an idle processor, then put a coin in the idle bucket. The total coins in the work bucket is W and the total number of coins placed int he idle bucket at each step is $(p-1)$ and there are a total of S steps, so the totla coins that can be in the idle bucket is $(p-1)S$ and $T_p = \frac{\text{total coins}}{p} \leq \frac{W+(p-1)S}{p}$

1.3 Load Balancing (Work Stealing)

But how can a work queue give out more than one task at a time? It can't, so need a queue for each processor (can think of this as a call stack).
But distributing from one processor queue to the other is a concurrent algorithm, not a parallel algorithm, and is extremely difficult to keep straight.
In a cost model for λ-calculus the run-time system does this for you.
$E[T_p] \leq \frac{W}{p} + S$ In expectation $\leftarrow$ TRUTHFUL!