 SAMFYB texify

723a9cf 14 days ago

1 contributor

194 lines (136 sloc) 6.9 KB

Lecture 19 Sequences (Cost Graphs)

April 3, 2018

dag: Cars don't run into each other!

Cost Graphs

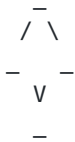
Cost Graph: Parallel Series, DAG with single source & sink.

Example:

(1 + 2) + 3



Base Case	Serial Computation	Parallel Computation
Single Node (sink = source)	o-o	See Below



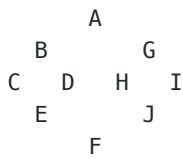
Work is Number of Nodes **Span** is Length of the Longest Path (Count Edges)

For example, in the above (1 + 2) + 3 instance, $W = 7$ and $S = 4$.

Brent's Theorem: Time to perform a computation is $O(\max(\frac{W}{p}, S))$ where p is the number of processors.

Example:

(1 + 2) + (3 + 4)



(Constraint: Computations below need results from Nodes above.)

Time / Processors (#1, #2)

-----+-----		
1		A
2		B G
3		C D
4		H I
5		E J
6		F

$$\max(\frac{W}{p}, S) = \max(\frac{10}{2}, 4) = 5$$

Note: The particular structure of this graph does not allow the actual optimal to be achieved.

Sequences

Question: What is the cost of finding the length of a list? How can we do better? If we define our own type?

- Sequences are like `list` as we can walk through it sequentially.
- Sequences are like `tree` as it has `log` properties.

Notation: $\langle x_0, \dots, x_{n-1} \rangle$

Implementation

```
signature SEQ =
sig
  type 'a seq (* abstract *)

  val empty : unit -> 'a seq (* notation: <> *)
  (* We need this so that in the same code we can have 'a seq, 'b seq, so on at the same time. *)
  (* In addition, the initialization function allows us to do stuff behind the scene. *)

  (* Side Note
   * We cannot pattern match directly on sequences.
   * We can however pattern match with 'list view' & 'tree view' of sequences.
   *)

  exception Range (* like 'index out of range' *)

  val tabulate : (int -> 'a) -> int -> 'a seq
  val length : 'a seq -> int
  val nth : 'a seq -> int -> 'a
  val map : ('a -> 'b) -> 'a seq -> 'b seq (* like 'map' for List *)
```

end

$$S = O(\log(n))$$

- $\text{mapreduce } f \ z \ g \ \langle x_0, \dots, x_{n-1} \rangle = f(x_0) * f(x_1) * \dots * f(x_{n-1}) * z$
- Work & Span is same as `reduce`
- $\text{filter } p \ \langle x_0, \dots, x_{n-1} \rangle = \langle \text{all } x_i \text{ s.t. } p \ x_i = \text{true} \rangle$
- $W = O(n)$ and $S = O(\log(n))$ because there is some additional work (for span)

```
structure Seq := SEQ = struct (* etc. *) end
```

```
fun sum (s : int Seq.seq) : int = Seq.reduce (op +) 0 s
```

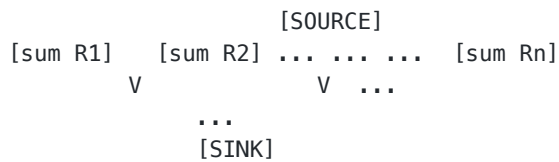
```
type row = int Seq.seq
```

```
type room = row Seq.seq (* 2D sequence *)
```

```
fun count (class : room) : int = sum (Seq.map sum class)
```

```
fun count' (class : room) : int = Seq.mapreduce sum 0 (op +) class (* alternative implementation *)
```

For `count'`



$$S = O(\log(n) + \log(m))$$