Parallel Algorithms

Umut Acar
Carnegie Mellon University

July 11, 2018

1 Recap

Sequence Data Structure

- List, but parallel
- length, indexing into a sequence
- tabulate → map, append O(1) span, O(n) work
- filter O(n), O(lg n)
- aggregation operations
  - iterate O(n), O(n)
  - reduce O(n), O(lg n)
  - scan O(n), O(lg n)
- update O(n), O(1)
- input O(n+m), O(lg n)

To prove this need:
$f(f(x,y),z) = f(x, f(y,z))$

2 Maximum Contiguous Subsequence Sum Problem (MCSS)

2.1 Brute Force Algorithm

```
fun bf a =
    let b = (a[i, j] : 0 ≤ i ≤ j ≤|a|)
        c = map (λx.(reduce "+" x) b
    in
        reduce −∞ max c
    end
```

Creating b: O(n^2), O(1)
Creating c: O(n^3), O(lg n)
Computing result: O(n^2), O(lg n)

2.2 First chunk of better solution

O(n) work and O(lg n) span per step

```
fun mcss_b a i =
    let b = a[i, ..., |a|−1]
        (c, s) = scan "+" 0 b
    in
        max(reduce max −∞ c, s)
    end
```

2.3 Better solution

O(n^2), O(lg n)

```
fun mcss_reduction a =
    let b = ((mcss_b a i) : 0 ≤ i < |a|) ← tabulate
            [tabulate (λi.(mcss_b a i)) |a|]
    in
        reduce max −∞ b
    end
```

2.4 Computing from the end forward

O(n), O(lg n)

```
fun mcss_e a i =
    let b = a[0,...,i]
        (c, s) = scan "+" 0 b
        m = reduce min ∞ c
    in
        s − m
    end
```

2.5 Solve the Complete Problem

O(n) (four passes over the sequence), O(lg n)

```
fun mcss a =
    let (b, s) = scan "+" 0 a
        (c, _) = scan min ∞ b
        d = (b[i] − c[i] : 0 ≤ i < |a|) ← mcss_e[i]
    in
        reduce max −∞ d
    end
```