Parallel Cost Semantics and Bounded Implementations

Guy Blelloch
Carnegie Mellon University

July 10, 2018

1 Parallel Algorithms

1.1 `update`

update: α seq × (int × α) → α seq
fun update(A,(i,v)) = tabulate (λj.**if** (j=i) **then** v **else** A[j]) |A|

W = O(|A|)
S = O(1)

1.2 `inject`

inject: α seq × ((int × α) seq) → α seq
inject(A,V) = iterate A update U

W = O(|A|+|V|)
S = O(log(|U|))

1.3 `filter`

filter: (α → bool) → (α seq) → (α seq)
filter f s

W = O(|s|)
S = O(lg(n))

map f s

reduce () append

W = O(|s|(lg(|s|)))
S = O(lg(|s|)

1.4 Back to `inject`

inject (A,U)

(a,b,c,d,e,g)

(T,F,F,T,T,F)

Map to locations: (0,X,X,1,2,X)

1 for true, 0 for false: (1,0,0,1,1,0)

Prefix sum to: (0,1,1,1,2,3)

```
fun filter f A =
let
    val f1 = map (fn x => if f(x) then 1 else 0) A
    val (offset,total) = scan 0 op+ f1
    val U = tabulate (fn i => (offset[i],A[i])) |A|
in
    inject (tabulate (fn _ => a[0]) total) U
end
```

1.5 `flatten`

flatten: (α seq) seq → α seq

```
fun flatten A = reduce () append A
```

W = O(|R|log(|A|))

S = O(log|A|)