CS7960 L15 : Parallel | Selection + Max

PRAM

1 disk
P processors
n input items

Each time step a processor can:
read, write, operate (+,-,*,<<,...)

shared memory: CRCW (although CREW more realistic)

Key technique : Accelerating Cascades
 Use fast, large work algorithms until threshold
 Switch to slower, less work algorithms.

2 examples: Selection, Max

Selection (n):
INPUT A = [a_1, a_2, ..., a_n]
  (unsorted)

Select select(k,A) item a_i s.t.
  |{a_j in A | a_j < a_i}| <= k-1
  |{a_i in A | a_j > a_i}| <= n-k

Sequential?   O(n)

PRAM: O(log n * log log n) time, O(n) work

```
Algorithm 1.

 Sort A->B O(n log n) Work, O(log n) time.
 Return B(k).

------------------------

Algorithm 2.

 Reduces problem of size m -> (3/4)m
  *  O(m) work, O(log m) PTime.
  *  requires O(log n) rounds
  *  Total: O(log^2 m) time, O(m) work

---

Input A (size m)

1. A into m/log m blocks A1, A2, ... A{m/log m} of size
log m
2. PARDO (h = 1 to log m)  x_h = sequential-median(A_h)
3. X = {x_1,...,x{m/logm}}
   Use x = median(X) (via Alg1(X))  O(m) work, O(log m)
time
4. Partition A to L,M,R s.t.
    l in L has a < x
    m in M has m = x
    r in R has r > x
5.  If (k <= |L|) reçur on select(k,L)
    If (k > |L|, k< |L|+|M|) return x
    else recur on select(k - |L| - |M|, R)

Fact:  min{|L|, |R|} > m/4
  -> recursive call has size at most (3/4)m

1.  (free)
2.  O(log m) time, O(m) work
```

3. O(log m) time, O(m) work
4. O(1) time, O(m) work
5. recur
T(m) = O(log m) + T((3/4) m) = O(log m) for O(log m) rounds = O(log^2 m)
W(m) = O(m) + W((3/4) m) = O(m) [geometrically decreasing]

Accelerating Cascades:
 1. Run Alg 2 until size m = n / log n | log_{4/3} log n = O(log log n) rounds
    O(log n log log n) time, O(n) Work   [dominates]
 2. Run Alg 1 O(log n) time, (n / log n * log(n/log n)) = O(n) Work

 Key technique!

Max (n):
INPUT A = [a_1, a_2, ..., a_n]
  (unsorted)
Return largest element.

Sequential?   O(n)

PRAM: O(log log n) time, O(n) work

Algorithm 1.

O(1) time, O(n^2) work. ?

Compare all O(n^2) pairs. element which never loses is max.

Algorithm 2.

O(log log n) time, O(n log log n) work ?

```
Subdivide A into sqrt{n} equal sized sub-arrays
 A1 = {a_1, ..., a_{sqrt{n}}
 A2 = {a_{1+sqrt{n}}, ..., a_{2sqrt{n}}
 ...
 A{sqrt{n}} = {a_{n-sqrt{n}}, ..., a_n}

PARDO h = 1 to sqrt{n}
  x_h = Alg2-Max(A_h)
X = {x_1, ..., x_{sqrt{n}}}
return x = Alg1-Max(X)

T(n) = T(sqrt{n}) + O(1)  = O(log log n)
W(n) = sqrt{n} W(sqrt{n}) + O(n) = O(n log log n)
```

Note n = 2^2^t (for some t)
  then sqrt{n} = sqrt{2^2^t} = 2^2^{t-1} <- doubly geometrically decreasing

----

Accelerating Cascades:

```
1. Divide A into n/loglog n blocks A1,A2,...A{n/loglog n}
each of size log log n.
   ParDo (h = 1 to log log n)
     x_h = Linear-Max(Ai)
2. X = {x_1, ..., x_{n/loglog n} }
   return x = Alg2-Max(X)
```

Step 1 takes O(log log n) time, and O(n) Work
Step 2 takes O(log log n) time, and (n / log log n)*loglog n = O(n) Work