PRAM (2) 
PRAM algorithm design techniques

- Reading for next class (Thu Aug 29): PRAM handout secns 3.6, 4.1
- Written assignment 1 is posted, due Tue Sep 10
  - work together with another student in the class and turn in joint solution
  - (or turn in solo solution if you prefer)
Topics

- PRAM Algorithm design techniques
  - pointer jumping
  - algorithm cascading
  - parallel divide and conquer
Design Technique: Pointer Jumping

- Fast parallel processing of linked data structures
  - linked lists
    - Membership, reduction and prefix sum of linked lists

- graphs (adjacency lists, edge lists)
  - connected components
  - minimum spanning trees
Example: Finding the roots of a forest

- **Input**
  \( G = (V,E) \) a forest of directed trees

- **Output**
  \( s[1:n] \) where for each vertex \( j \),
  \( s[j] \) is the root of the tree containing \( j \)

- **Representation of \( G \)**
  - in a directed tree
    - the root has no parent
    - every other vertex has a unique parent
  - \( V = \{1, \ldots, n\} \)
  - \( E \) is defined by \( s: V \rightarrow V \)
    - \( s(u) = v \) if \( v \) is parent of \( u \) in \( G \)
    - \( s(r) = r \) if \( r \) is a root in \( G \)
    - \( s \) is represented using an array \( s[1:n] \)
Following a list in parallel: Pointer jumping

- Let \((n, s[1..n])\) be the representation of directed forest \(G\)
- Pointer jumping operation
  - every vertex directs its edge to its grandparent in parallel
  - also called \textit{pointer doubling}

\[
\text{forall } i \text{ in } 1:n \text{ do } \\
s[i] := s[s[i]] \\
\text{endo}
\]
Analysis of pointer jumping

- pointer jumping halves distance to the root in $s$
  - let $d$ be the distance in $s$ from vertex $u$ to the root
  - after pointer jumping distance in $s$ from $u$ to root is $\lfloor d/2 \rfloor$

- $S(n) =$

- $W(n) =$

- PRAM model

```plaintext
forall i in 1:n do
    s[i] := s[s[i]]
endo
```
Pointer jumping in a forest

Initial Forest

after 1 doubling

after 2 doublings

All vertices point to the root of their tree
Finding roots of a forest

- pointer jumping reaches a fixed point when forest has max height $\leq 1$
  - vertex $i$ is distance 1 or less from root when $s[i] = s[s[i]]$

- forest height $\leq 1 \Rightarrow s[i] = \text{root of tree containing } i$

```plaintext
forall i in 1:n do
  while $s[i] \neq s[s[i]]$ do
    $s[i] := s[s[i]]$
  end do
enddo
```
Problem: find distance to root in directed forest

• Construct an algorithm for the following problem
  – Let \((n, s[1..n])\) be directed forest \(G\)
  – For each vertex \(1 \leq i \leq n\), set \(d[i]\) to be the distance from \(i\) to the root of its tree

• Invariant: let \(d[i]\) be the distance in \(G\) from \(i\) to \(s[i]\)
  – establish initially
  – maintain property with each pointer doubling
  – termination implies result

• Complexity
  \[ W(n) = \]
  \[ S(n) = \]

```plaintext
forall i in 1:n do
  d[i] := (s[i] == i) ? 0 : 1
end do
for i := 1 to (lg n) do
  forall i in 1:n do
    d[i] := d[i] + d[s[i]]
    s[i] := s[s[i]]
  end do
end do
```
Design Technique: Algorithm Cascading

• Technique for improving work efficiency of an algorithm
  – suppose we have
    • work-inefficient but fast parallel algorithm A
    • work-efficient but slow algorithm B (typically sequential)
  – combine ("cascade") A and B to get best of both

  “Speeding up by slowing down”
Example: histogram values in a sequence

- **Input**
  - Sequence \( L[1..n] \) with integer values in the range 1..k, where \( k = \lg n \)

- **Output**
  - \( R[1..k] \) with \( R[i] = \# \) occurrences of \( i \) in \( L[1..n] \)

**Sequential algorithm**

\[
\begin{align*}
R[1:k] & := 0 \\
\text{for } i & := 1 \text{ to } n \text{ do} \\
& \quad R[L[i]] := R[L[i]] + 1 \\
\text{end do}
\end{align*}
\]

\[T_s(n) = \]
Parallel Algorithm: First try

\[ C_{i,j} = \begin{cases} 1, & \text{if } L_i = j \\ 0, & \text{otherwise} \end{cases} \]

\[ R_j = \sum_{i=1}^{n} C_{i,j} \]

\[
\begin{array}{cccc}
3 & 1 & 1 & 3 \\
2 & 3 & 1 & 3 \\
\end{array}
\]

\[
\text{integer } C[1:n,1:k] \\
\text{forall } i \text{ in } 1:n, j \text{ in } 1:k \text{ do} \\
\quad C[i,j] := (L[i]==j) ? 1 : 0 \\
\text{end do} \\
\text{forall } j \text{ in } 1:k \text{ do} \\
\quad R[k] := \text{REDUCE}(C[1:n,j], +) \\
\text{end do}
\]

PRAM
\[
W(n) = \\
S(n) = \\
\text{model}
\]
Cascading the histogram algorithm

- partition L into m “chunks” of size (lg n)
  - k = lg n (assume k divides n)
  - m = n / k = n / lg n

- compute mini-histogram sequentially within a chunk
  \[ S_{\text{chunk}} = \]
  \[ W_{\text{chunk}} = \]

- compute all m mini-histograms in parallel
  \[ S_{\text{all}} = S_{\text{chunk}} \]
  \[ W_{\text{all}} = m \cdot W_{\text{chunk}} \]

- combine histograms by summing
  \[ S_{\text{combine}} = \]
  \[ W_{\text{combine}} = \]

```plaintext
integer C[1:m, 1:k]
forall i in 1:m, j in 1:k do
    C[i, j] := 0
end do
forall i in 1:m do
    for j := 1 to k do
        C[i, L[(i-1)k+j]] += 1
    end do
end do
forall j in 1:k do
    R[k] := REDUCE(C[1:m, j], +)
end do
W(n) = S(n) =
PRAM model?
```
Parallel Divide and Conquer

• To solve problem instance P using parallel divide-and-conquer
  – divide P into subproblems (possibly in parallel)
  – apply D&C recursively to each subproblem in parallel
  – combine subsolutions to produce solution (possibly in parallel)

• Example: sorting
  – mergesort
    • combining
      – subproblems: left/right half of array
      – sort each subproblem
      – merge results
  – quicksort
    • partitioning
      – subproblems: values less than pivot, values greater than or equal to pivot
      – sort each subproblem
      – concatenate results
Parallel Mergesort (parallel divide and conquer)

- Assume parallel EREW merge \((A, B)\) for \(|A| = |B| = O(n)\) with
  \[
  W_{\text{merge}}(n) = O(n) \\
  S_{\text{merge}}(n) = O(\lg n)
  \]

```plaintext
mergesort(V[1:n]) = 
if n \leq 1 then S[1:n] := V[1:n] 
else
  m := n/2 
  
  { 
    R[1:m] = mergesort V[1:m] 
    \|
    R[m+1:n] = mergesort V[m+1:n] 
  } 
  S[1:n] := merge( R[1:m], R[m+1:n] ) 
endif
return S[1:n]
```
Parallel Mergesort (forall)

- Assume parallel EREW merge(A, B) for |A| = |B| = O(n) with
  
  \[ W_{\text{merge}}(n) = O(n) \]
  
  \[ S_{\text{merge}}(n) = O(\log n) \]

\[
\text{mergesort}(V[1:n]) = \\
\text{if} \quad n \leq 1 \quad \text{then} \quad S[1:n] := V[1:n] \\
\text{else} \quad m := n/2 \\
\hspace{1em} \text{forall} \ i \ \text{in} \ 0:1 \ \text{do} \\
\hspace{2em} R[i*m+1 : (i+1)*m] = \text{mergesort} \ V[i*m+1 : (i+1)*m] \\
\hspace{1em} \text{end do} \\
\hspace{1em} S[1:n] := \text{merge}( R[1:m], R[m+1:2*m] ) \\
\text{endif} \\
\text{return} \ S[1:n]
\]

\[ S_{\text{mergesort}}(n) = \]

\[ W_{\text{mergesort}}(n) = \]
Parallel Quicksort

- Assume parallel EREW $\text{partition}(A, p)$ for $|A| = O(n)$ with

  $W_{\text{partition}}(n) = O(n)$
  
  $S_{\text{partition}}(n) = O(\lg n)$

```plaintext
quicksort(V[1:n]) =
if n ≤ 1 then S[1:n] := V[1:n]
else
  p := V[ random(1:n) ]
  R[1:n], m := partition (V[1:n], p)
  h[0:2] := [0, m, n]
  forall i in 0:1 do
    S[h(i)+1 : h(i+1)] = quicksort R[h(i)+1 : h(i+1)]
  end do
end if
return S[1:n]
```

$S_{\text{quicksort}}(n) = \ldots$

$W_{\text{quicksort}}(n) = \ldots$
Planar Convex Hull Problem

• **Input**
  – \( S = \{(x_i, y_i)\} \) set of \( n \) points in the plane
  – assume \( x_i \) distinct, \( y_i \) distinct, and no three points co-linear

• **Output**
  – tour of smallest convex polygon containing all points of \( S \)

• **Complexity**
  – \( T_S^*(n) = \Theta(n \log n) \)
Two Parallel Algorithms for Planar Convex Hull

• two divide and conquer algorithms
  – combining approach
  – partitioning approach

• combining algorithm (like mergesort)
  – assume input points presented in order of increasing x coordinate
    • can be obtained using $O(n \lg n)$ work, $O(\lg^2 n)$ step sorting algorithm
  – optimal worst case performance

• partitioning algorithm (like quicksort)
  – no assumptions about order of input points
  – suboptimal worst case performance
  – very good expected case performance
D&C algorithm via combining

1. Divide $S$ into US, LS by line $P_1 – P_n$
2. Compute Upper Convex Path and Lower Convex Path using D&C algorithm
3. Combine UCP, LCP to construct convex hull
Construction of upper convex path

Divide

Combine (1): find upper common tangent

Combine (2): create upper convex path

Recur
Analysis (Combining algorithm)

• Upper/Lower Convex path
  – Find common tangent (UCT/LCT)
    • binary search of convex paths to find tangent points [Overmars & van Leeuwen]
    • Sequential: \( S(n) = W(n) = O(\lg n) \)

  – Connect paths
    • CREW: \( S(n) = O(1) \), \( W(n) = O(n) \)
    • EREW: \( S(n) = O(\lg n) \), \( W(n) = O(n) \)

• Convex Hull
  • \( S(n) = S(n/2) + O(\lg n) \)
    – \( S(n) = O(\lg^2 n) \)
  • \( W(n) = 2 W(n/2) + O(n) \)
    – \( W(n) = O(n \lg n) \)

  – Work-efficient, since \( T_S(n) = \Theta(n \lg n) \)
D&C algorithm via partitioning

1. Divide S into US, LS by line \( P_i-P_j \) where \( P_i, P_j \) have extremal x coordinates
2. Compute Upper Convex Path and Lower Convex Path using D&C algorithm
3. Combine UCP, LCP to construct convex hull
Construction of upper convex path

Locate point at max distance from $P_i - P_j$

Discard interior points and partition remaining points

Recur: find upper convex paths

Combine upper convex paths
Analysis (Partitioning algorithm)

• Upper/Lower Convex path for n points above baseline
  – Find point at maximum distance from baseline
    • $S(n) = O(lg n)$, $W(n) = O(n)$
  – Partition
    • $S(n) = O(lg n)$, $W(n) = O(n)$
  – Combine
    • $S(n) = O(lg n)$, $W(n) = O(n)$

• Convex Hull
  – Find extremal points for initial baseline
    • $S(n) = O(lg n)$, $W(n) = O(n)$
  – Construct UCP, LCP
    • $S(n) = \max( S(n_1), S(n_2) ) + O(lg n)$
    • $W(n) = W(n_1) + W(n_2) + O(n)$
      – $n_1 + n_2 \leq n$
  – Combine paths
    • $S(n)=O(1)$, $W(n) = O(n)$
Analysis of parallel partitioning algorithm

• Analysis
  – **Expected** partition, no points eliminated
    • \( S(n) = S(n/2) + O(\lg n) \)
      - \( S(n) = O(\lg^2 n) \)
    • \( W(n) = 2 \, W(n/2) + O(n) \)
      - \( W(n) = O(n \lg n) \)
  – **Worst-case** partition, no points eliminated
    • \( S(n) = S(n - 1) + O(\lg n) \)
      - \( S(n) = O(n \lg n) \)
    • \( W(n) = W(1) + W(n - 1) + O(n) \)
      - \( W(n) = O(n^2) \)
  – **Expected** partition, random points in the unit square
    - \( S(n) = O(\lg n (\lg \lg n)) \)
    - \( W(n) = O(n \lg \lg n) \)
Reminder: Master theorem for recurrence relations

• Recurrence form

\[ H(n) = aH\left(\frac{n}{b}\right) + f(n) \quad \text{where} \quad a \geq 1, b > 1 \]

\[ H(1) = O(1) \]

• Solution

\[ H(n) = \Theta\left(a^k\right) + \Theta\left(\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right)\right) \]

where \( k = \log_b n \)