So far have seen:
- Binary-tree expressions (max, etc.)
- Prefix sum
- Vector compression and expansion (uses prefix sum)
- Max-finding in $O(1)$ (using CRCW-arb)

Today:
- Parallel merging
- Pointer techniques
- Tree-traversal
- Quicksort (1-version)

Parallel Merging
- How can we merge two ordered arrays in parallel on a PRAM?
- One means is to compute the indices of where the elements of one array are inserted into the other array.
- We can then use something similar to array compression to do the actual insertion.

Computing indices for Parallel Merging
- A single index can be computed with binary search: Find the position in the other array at which the element would be inserted.
- Add the element's current index to it to get the net final position.
- Do all binary searches in parallel.
- Each array computes the final indexes of its elements in the merged array.
- Each processor stores its elements simultaneously.

Parallel Merging
- Cost: $n$ binary searches in parallel
- $O(\log n)$ time (on CREW PRAM)

Exercise
- What is an upper bound for sorting using parallel merging?
Using Pointers in a PRAM

- “Pointer Jumping” technique
- As with prefix sum, this has many uses.
- Basic idea: in a chain of pointers stored in the common memory, the extremities of a chain can be determined in a way that doubles the length of the chain at each step:
  - If a location points “N hops away” now, it will point “2N hops away” on the next step.
  - This is because concatenating two N-hop chains gives a 2N-hop chain.

List-Ranking Problem

- A pointer-jumping application
- Given a chain of pointers, determine the rank of each element in the chain

List-Ranking Step-by-Step

Step 0

List-Ranking Step-by-Step

Step 1

List-Ranking Step-by-Step

Step 2
List-Ranking Step-by-Step

Step 3

Summary

- A list can be ranked in $O(\log n)$ time on a PRAM.

Exercises

- Construct the PRAM code for list ranking.
- Show that a list can be prefixed-summed in $O(\log n)$.

Pre-Ordering a Tree

- Recall: Pre-Order:
  - Visit the root
  - Visit recursively the left sub-tree in pre-order
  - Visit recursively the right sub-tree in pre-order

Pre-Order Example

Pre-Order Example
How to Construct a Pre-Order on a PRAM in O(log n)?

- Create a list of nodes, two per arc of the original graph:
  - One node for the arc in the normal (downward) direction
  - A second for the arc in the other direction
  - Add a parity-indicator
  - Connect the nodes to represent the original arcs

Pre-Order Example

We now have an alternate representation from which we can recover the original tree.

Try it
What now?

List ranking variation
Count downward nodes only

Pre-order of original is reverse of this order using first component of the root and the targets of the ↓nodes only

Check with Original

Exercise

- We just showed that a pre-order traversal can be done in time $O(\log n)$. Do the same for an in-order traversal.

Application of In-Order Traversal

- A PRAM version of Quicksort
- Construct a tree indicating how the nodes partition (without actually moving any data)
- An in-order traversal of the tree gives the nodes in sorted order.
Assuming that the tree splits fairly evenly across each level, what is the time taken to do this version of Quick sort (assuming the asserted bound for in-order traversal).
### “Parallel sorting in O(1)”
(from http://www.cs.uku.fi/~penttonen/parallel/sort.html, has demo)

```plaintext
procedure Sort(modifies A: array 1..n of integer)
for i in 1..n pardo K[i]:=0
for i in 1..n pardo
  for j in 1..n pardo
  for i in 1..n pardo A[K[i]]:=A[i]
```

How many processors?
What kind of conflict resolution?
How much effort?

### Misc. Notes on PRAM

- Cole’s sorting method based on merging:

- Parallel recognition of a context-free language: \( P(\log^2 n) \) time using \( n^6 \) processors.

- Other problems/algorithms are known.

### Using PRAM results in real life

- What are some problems of simulating PRAM’s on real multiprocessors, say a on a Distributed-Memory Machine (DMM)?

### PRAM -> DMM problems

- Memory conflict resolution at word-level
- Memory conflict resolution at memory-module level
- Communication delays

### PRAM -> DMM possible resolutions

- Memory conflict resolution at word-level: Use only EREW model
- Memory conflict resolution at memory-module level
  Split memory into multiple modules;
  Use multiple copies of contended data (must provide for reconciling)
- Communication delays
  Use 2-phase, random routing

### Efficient Simulation of PRAM

- Karp, et al. STOC 1991
- An \( n \log \log(n) \log^*(n) \) processor CRCW-arb PRAM is simulated on an \( n \)-processor DMM (Distributed memory machine).

- Average slowdown \( O(\log \log(n) \log^*(n)) \).