CSE6512 Lecture 24 Notes

Manal Alharbi

November 17, 2011

Optimal List Ranking:

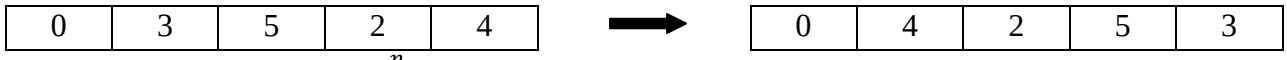
$P = \frac{n}{\log n}$. Each processor gets $\log n$ nodes.

Phase1: Each node is spliced out. This is done in stages.

Phase 2: The nodes are spliced back in in the reverse order.

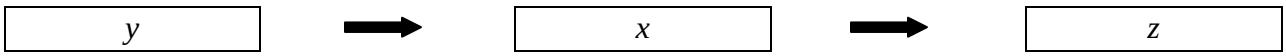
Note: At beginning, doubly link the list.

For example:



This takes $\log n$ *time using* $\frac{n}{\log n}$ *processors.*

Consider the process of splicing out nodes. When a node is spliced out, we end up with a shorter list. We splice out nodes in stages and the length of the list decreases with every stage. When the length of the list is two, we trivially solve the list ranking problem. From there on we splice the nodes back in in the reverse order. When a node is spliced back in, its correct rank will be computed. Consider some stage in the process of splicing out. Let *y*, *x*, and *z* be successive nodes of the list in this stage.



- Note that rank (*y*) at this stage is the number of nodes in the global list between *y* and *x*.
- If *x* is spliced out, we'll keep the following information for *x*:
    - Stage number, the left neighbor of *x* (it is *y* in this stage), and the rank of the left neighbor of *x*.
- When *x* is spliced back in, its global rank is computed as the current rank of *y* - the rank of *y* that was stored before (when *x* was spliced out).

Note: In the following algorithm, a *stage* refers to one execution of the **Repeat** loop.

Phase 1:

Repeat

For $1 \leq i \leq P$ in $\|^l$ do

- Processor i picks the next node to be spliced out.
- It flips a 2- sided coin.
- If tail, then it waits for the next stage;
- If head, it checks if the right neighbor of this node is under consideration by the corresponding processor and this processor also has a head. If so, it waits for the next stage. If not, it splices the node out.

Until the number of nodes is ≤ 2;

Phase 0: solve the list ranking problem on the reduced list.

Phase 2: splice the nodes back in in the reverse order.

Analysis:

The time needed for Phase 2 is the same as the time spent in Phase 1. Thus, we consider only Phase 1.

- Probability of success for any processor in any stage is $\geq \frac{1}{4}$. Since the probability of this processor getting an H and the right neighbor processor getting a T is = $\frac{1}{4}$.
- If we have c α logn stages, the number of successes for any processor is B (c α logn, $\frac{1}{4}$)

Using Chernoff Bounds,

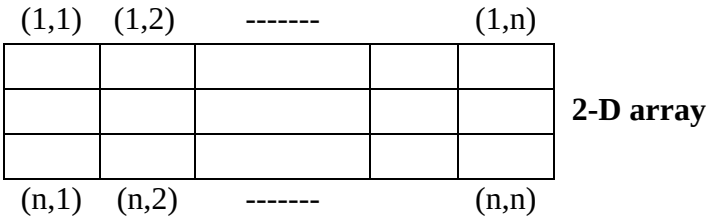
Probability [this number $\leq \ 1-\in \ \frac{c\ \alpha \log n}{4}$] is $\leq$ exp $\left(\frac{-\in^2 c\ \alpha \log n}{4.2}\right)$

Let $\in = \frac{1}{2}$

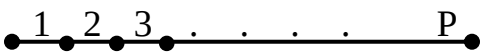
The above probability is $\leq$ exp $\left(\frac{-c\ \alpha \log n}{32}\right)$. This probability is $\leq\ n^{-\alpha}$ if c ≥ 32.

➔ The run time of the algorithm is $O(\log n)$.

Mesh algorithms:



2-D array



1-D array

Packet Routing

Partial permutation Routing

- Each node in the network is the origin of at most one packet and each node is the destination of at most one packet.
- Send the packets to their correct destinations as quickly as possible.
- **Run time of a packet routing algorithm** is defined to be the time taken by the last packet to reach its destination.

Note: only one packet can be sent along any edge at any time.

Queue size is defined as the maximum number of packets that any node will have to store at any point in the entire algorithm.

To resolve edge conflicts:

Queue disciplines:

1) FIFO
2) LIFO
3) Farthest origin first.
4) Farthest destination first.

Problem one:

In a linear array, each node is the origin of ≤ 1 packet. Rout the packets.

Assumption: The links are bidirectional.

1-D array

Corollary: Packets moving from L to R don't affect packets that move from R to L.

Fact: Problem one can be solved in $\leq (P-1)$ steps.

Algorithm: Every packet uses the shortest path from its origin to its destination. No two packets will ever conflict.

➔ If a packet's origin and destination are *i* and *j*, respectively, and if the packet is moving from left to right, then the time it takes to reach its destination is (*j*-*i*). There is no delay for the packets due to edge conflicts!

Problem two:
In a linear array at most one packet is destined for any node. Route the packets.

Fact: The above problem can be solved in $\leq (P-1)$ steps.

Proof: in the next lecture.