CSE 6512 Randomization in Computing
Lecture Notes 23

Prepared by Yuan Song[*]

1 Review - Assignment Problem

1.1 Problem Description

Input is a sequence of keys $X = k_1, k_2, \cdots, k_n$. Each key k_i belongs to a group g_i, $1 \le i \le n$, $1 \le g_i \le Q$. Let n_i be the number of keys that belong to group i, $1 \le i \le Q$. Upper bounds on group sizes are given: N_1, N_2, $\ldots$, and N_Q. In particular, $N_i \ge n_i$, for $1 \le i \le Q$ and $\sum_{i=1}^{Q} N_i = O(n)$. The problem is to permute X such that all the keys in group 1 appear first, followed by all the keys in group 2, ..., followed by all the keys in group Q.

Theorem 1. *We can solve the assignment problem in $\widetilde{O}(\frac{\log n}{\log\log n})$ time using $\frac{n}{\log n}(\log\log n)^2$ arbitrary CRCW PRAM processors.*

1.2 Algorithm Summary

Let the number of processors be $P = \frac{n}{\log n}(\log\log n)^2$.

1. Do a prefix computation on $2N_1$, $2N_2$, ..., $2N_Q$ to identify the boundaries of the buckets.
   $\Longrightarrow$ This takes $O(\frac{\log n}{\log\log n})$ time.
2. For every key do:
   Make $\log\log n$ attempts to place it in the right bucket.
   $\Longrightarrow$ This takes $O(\frac{\log n}{\log\log n})$ time using $\frac{n}{\log n}(\log\log n)^2$ processors.
3. Do a prefix computation to compute the number Z of not yet placed keys.
   $\Longrightarrow$ This takes $O(\frac{\log n}{\log\log n})$ time.
4. Each unplaced key gets $\left\lfloor \frac{P}{Z} \right\rfloor$ processors. For each unplaced key do:
   all of its processors attempt to place the key in the right bucket. Do this until success.
   $\Longrightarrow$ We can show that this takes $\widetilde{O}(\frac{\log n}{\log\log n})$ time.
5. Do a prefix computation to compress the array.
   $\Longrightarrow$ This takes $O(\frac{\log n}{\log\log n})$ time.

[*]Email:yuan.song@engr.uconn.edu

1.2.1 Algorithm Analysis

In step 2, the probability of failure in any attempt is $\leq \dfrac{1}{2}$

$\Longrightarrow$ The probability of failure in $\log\log n$ attempts is $\leq \left(\dfrac{1}{2}\right)^{\log\log n} = \dfrac{1}{\log n}$

$\Longrightarrow$ The expected number of unplaced keys Z at the end of step 2 is $\leq \dfrac{n}{\log n}$

Using Chernoff bounds, this number Z is $\widetilde{O}(\dfrac{n}{\log n})$

$\Longrightarrow$ The number of processors per key in step 4 is $\widetilde{\Omega}((\log\log n)^2)$

$\Longrightarrow$ The probability of failure in one attempt of step 4 is $\leq \left(\dfrac{1}{2}\right)^{(\log\log n)^2}$

$\Longrightarrow$ The probability of failure P_4 in $\dfrac{\log n}{\log\log n}$ attempts in step 4 satisfies:

$$\begin{aligned} P_4 &\leq \left(\frac{1}{2}\right)^{(\log\log n)^2 \dfrac{\log n}{\log\log n}} \\ &= \left(\frac{1}{2}\right)^{\log n \log\log n} \end{aligned}$$

$\Longrightarrow$ For any constant α, we have $P_4 < n^{-\alpha}$. $\square$

2 General Sorting

Theorem 2. *We can sort n general keys in $\widetilde{O}(\dfrac{\log n}{\log\log n})$ time using $n(\log n)^\epsilon$ arbitrary CRCW PRAM processors, for any constant $\epsilon > 0$.*

2.1 An Algorithm

1. Pick a sample of size $\sqrt{n}$;
2. Sort the sample and use the sample keys to divide the input sequence into $\sqrt{n}+1$ parts;
3. Assign $(\log n)^\epsilon$ processors per input key and perform a $(\log n)^\epsilon$-ary search[1] in the sorted sample to figure out the part numbers of the input keys. Notice that the sample keys are boundaries of the $\sqrt{n}+1$ parts;
4. Compute estimates on the part sizes using the sampling method that we have seen for the parallel integer sorting algorithm;
5. Use the assignment algorithm to rearrange the input based on the part numbesr of the input keys. (Notice that we have estimates of part sizes computed in step 4);
6. Recursively sort each part. If the parts are $X_1, X_2, \ldots, X_{\sqrt{n}+1}$, assign $|X_i|$ processors to part i (for $1 \leq i \leq (\sqrt{n}+1)$.

[1]In k-ary search, we equally divide the current search space into k parts, find which part the key is in and recursively search that part. If we use k processors, then searching for any key takes $\dfrac{\log n}{\log k}$ time.

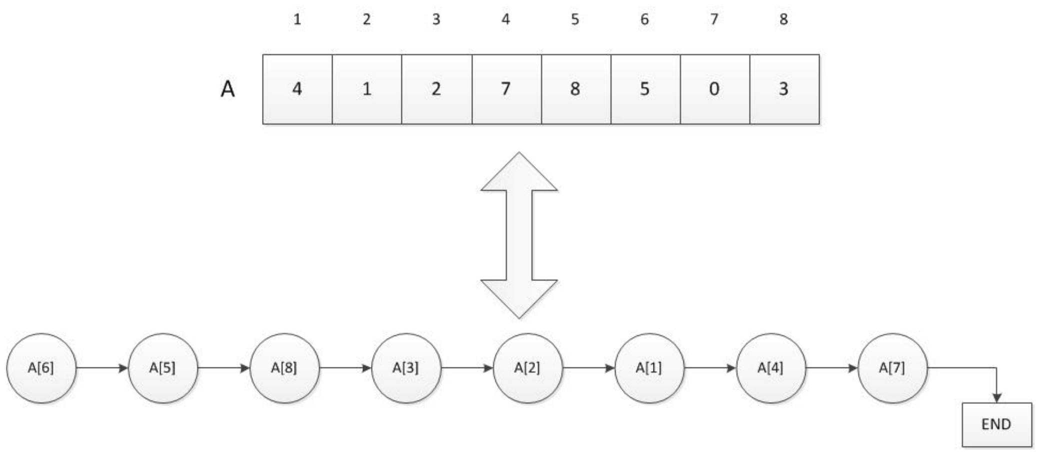

Figure 1: List ranking problem: An example input

Theorem 3. *We can sort n integers in the range $[1, n(\log n)^c]$ in $\widetilde{O}(\frac{\log n}{\log\log n})$ time using $\frac{n}{\log n}\log\log n$ arbitrary CRCW PRAM processors.*

3 List Ranking

3.1 Problem Description

Input: A linked list represented as an array. The value of any element in the array is the index of its right neighbor in the list, i.e., $A[i] = j \implies j$ is i's right neighbor in the list. An example of the input is shown in the figure 1.
Output: The rank of every node in the list. The rank of any node is the number of nodes to its right in the list.

Remark 3.1. *A sequential algorithm starts from the head of the list and traverses the nodes from left to right and this takes $O(n)$ time.*

3.2 Algorithms

3.2.1 A Simple Parallel Algorithm

Theorem 4. *We can solve list ranking in $O(\log n)$ time using n CREW PRAM processors.*

Proof. The theorem can be proved using the following algorithm. The algorithm runs in iterations. We assign one processor per node in the list. Initially each node is assigned a rank of 1 except for the rightmost node that is assigned a rank of 0. Each node has a neighbor pointer that points to its right neighbor in the list. In each iteration, we first compute the sum of the rank of the current node and that of its current neighbor. The result is saved at the current node. Second, we use "pointer jumping", i.e., the neighbor pointer is replaced with the pointer that points to the neighbor of the neighbor.

The above process is repeated until the neighbor pointers of all the nodes point to the end of the list. An example of the algorithm with 7 nodes is shown in the figure 2. This algorithm takes $O(\log n)$ time. □

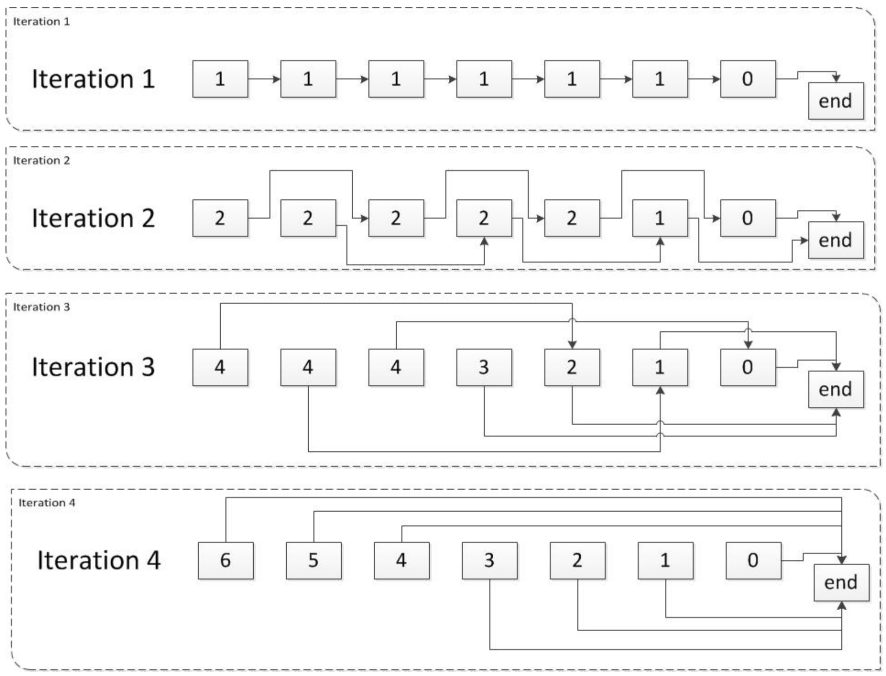

Figure 2: Pointer jumping algorithm: An example

3.2.2 An Optimal Randomized Parallel Algorithm

We have a randomized optimal algorithm using $\frac{n}{\log n}$ processors. The basic idea of the algorithm is as follows:

1. Assign $\log n$ nodes per processor.
2. Splice nodes out in stages.
3. Solve list ranking problem on the resultant shorter list.
4. Splice nodes back in in reverse order.

The details and the analysis of the algorithm will be discussed in next lecture.