# Parallel Computing:
# Problem Set 4 Document

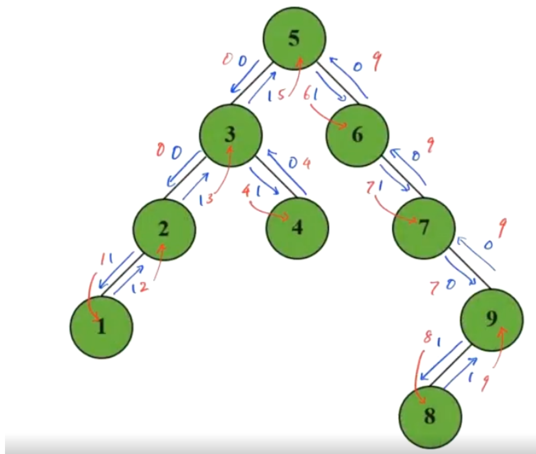
Due on Dec 18th at 23:59

Mailbox: yuzhh1@shanghaitech.edu.cn
Student ID: 2020533156
Student Name: Zhenghong Yu

# Contents

# 1 Problem 1

Design a PRAM algorithm to compute an in-order traversal of a binary tree with $n$ nodes in $O(\log n)$ parallel time and $O(n)$ work.
**Solution:**




Let $r$ be the root of the tree. First compute an Euler tour of the tree starting at $r$. Next, place values on each edge $(u, v)$ as follows:

- If $(u, v)$ is a downwares edge(i.e. $u$ is the parent of $v$), then if $v$ does not have a left child, set $(u, v)$'s value to 1 and otherwise set its value to 0.
- If $(u, v)$ is an upwards edge(i.e. $v$ is the parent of $u$), then if $u$ is $v$'s left child, set $(u, v)$'s value to 1, and otherwise set it to 0.

After setting the values, do a prefix sum on the linked list formed by the Euler tour. Finally, the in-order number of each node $v$ is defined as:

- If $v$ has a left child $u$, then $v$'s value is the prefix sum value of edge $(u, v)$.
- If $v$ do not have a left child, then let $u$ be $v$'s parent. $v$'s value is the prefix sum value of $(u, v)$.

Since we can compute the Euler tour, set the edge weights and assign edge prefix sum values to nodes in $O(1)$ time, and also prefix sum the edge weights in $O(\log n)$ time and $O(n)$ work, then the time complexity is $O(\log n)$ and the work is $O(n)$.

# 2 Problem 2

Design a PRAM algorithm to compute a histogram of a size $n$ array in $O(\log n)$ parallel time and $O(n)$ work. Assume all the values in the array are integers in the range 1 to $k = O(\log n)$. The output of the algorithm should be an array of size $k$, where the $i$'th entry is the number of occurrences of value $i$.
**Solution:**
Let $k = O(\log n)$ be the number of different values in the array A. It's easy to solve the problem using $kn$ processors in $O(\log n)$ time and $O(nk)$ work as follows:

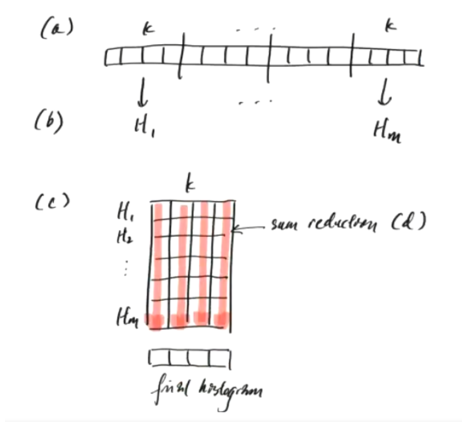
- Assign one processor to each pair $(i, v)$, where $1 \le i \le n$ is an index in A and $1 \le v \le k$ is one of the values.

- In parallel, each processor $(i, v)$ sets a value $sum[i, v]$ to 1 if $A[i] = v$, otherwise set it to 0.
- In parallel, for each $1 \leq v \leq k$, do a parallel sum reduction on $sum[1, v], sum[2, v], \cdots, sum[n, v]$. Set the histogram value for $v$ equal to the sum.

To make this algorithm more efficient (i.e. do $O(n)$ work), we use accelerate cascading. Specifically, we do the following,

- Partiton A into $m = n/k$ chunks each of size $k$. Assign one processor to each pair $(i, v)$, where $1 \leq i \leq m$ is the index of a chunk, and $1 \leq v \leq k$ is one of the values. Notice that we use $O(n)$ processors.
- In parallel, each processor $(i, v)$ sequentially computes a histogram $H_i$ for all the values in its chunk of A in $O(k)$ time.
- Let C be an $m * k$ matrix where $i$th row is equal to $H_i$, for $1 \leq i \leq m$.
- Using $n$ processors, do a parallel sum reduction on each column of C in parallel, to produce a size $k$ array that's the final histogram.

Step(b) takes $O(k) = O(\log n)$ parallel time and $O(n)$ work, Step(d) also takes $O(\log n)$ parallel time and $O(n)$ work.




# 3 Problem 3

Design a PRAM algorithm to implement Quicksort. What is the (expected) time and work complexity of your algorithm?

**Solution:**

The main step in Quicksort is to take an array and split it using a pivot value $v$ into a left part containing all values $\leq v$, and a right part containing all the values $> v$. This can be done in a way similar to sort on a single digit in radix sort in$O(\log n)$ time and $O(n)$ work. If we pick the pivots randomly, then Quicksort's recursion tree has $O(\log n)$ depth with high probability, and each level of the tree does $O(n)$ work in total. Thus, the total expected parallel time is $O(log^2 n)$ and the total work is $O(n \log n)$.