CSE 4502/5717 Notes - Lecture 2 - 1/24/18; Notes by Adomous Wright

Randomized Algorithms (Monte Carlo)

- Input: An array $A[1:n]$ of 0's and 1's
- A could be only of the following two types:
  - Type 1: All the elements are zero
  - Type 2: A has $\frac{n}{log(n)}$ 0's and $n - \frac{n}{log(n)}$ 1's
- Output: The type of the array
- Note that any deterministic algorithm will need $n/log\ n$ time.
- A faster Monte Carlo Algorithm:
  - $for\ i = 1\ to\ k$:
    - Flip an n-sided coin to get i;
    - $if\ A[i] = 1$:
      - return "Type 2"
      - break
  - return "Type 1"
- Analysis:
  - Note that if the array is of Type 1, the above algorithm will not output an incorrect answer. (Thus this algorithm is said to have "one-sided error".) Therefore, assume that the array is of "Type 2"
  - Probability $[a\ random\ element\ = 0] = \frac{1}{log(n)}$
  - Probability $[all\ the\ k\ elements\ are\ 0's] = (\frac{1}{log(n)})^k$
  - We want this to be $\leq n^{-\alpha}$
    - $(\frac{1}{log(n)})^k \leq n^{-\alpha}$
    - $-klog(log(n)) \leq -\alpha log(n)$
    - $k \geq \frac{\alpha log(n)}{log(log(n))}$
  - Runtime = $O(\frac{log(n)}{log(log(n))})$

Parallel Algorithms

- Let π be any problem
- Let P be the # of processors
- Let S be the best known sequential runtime to solve π
- Let T be the parallel runtime
- Fact: $T \geq \frac{S}{P}$
- Proof: (By Contradiction)
  - Assume that $T < \frac{S}{P}$, i.e., there is a parallel algorithm whose run time is <S/P.
  - We can simulate this parallel algorithm using one processor.

- One parallel step can be simulated in $\leq P$ steps sequentially,
- Thus the entire parallel algorithm can be simulated in $\leq PT < S$ steps, which is a contradiction.

Parallel Models

In sequential computing the Random Access Machine (RAM) model is widely used. In a RAM, we assume that each basic operation takes one unit of time. However, there are numerous parallel models of computing. These models differ in the way in which interprocessor communications are enabled. Two categories of parallel models can be found in the literature:

- Fixed Connection Machines
  - Any fixed connection machine can be thought of as a directed graph: $G(V, E)$
  - $V \rightarrow$ Processors
  - $E \rightarrow$ Communication Links
  - Examples:
    - Linear Array
      - $1 \rightarrow 2 \rightarrow 3 \rightarrow \ldots \rightarrow p$
    - Mesh:
      - Processors are on a 2-D grid
    - Hypercube: A hypercube of dimension n has 2^n processors.

- Shared Memory Machines (Parallel Random Access Machines, PRAMs)
- There are many versions of a PRAM depending on how read and write conflicts are handled.
  - EREW
    - Exclusive Read: At any time and at any memory cell, only one processor can access the cell for reading from;
    - Exclusive Write: At any time and at any memory cell, only one processor can access the cell for writing into;
  - CREW
    - Concurrent Read: At any time and at any memory cell, any number of processors can access the cell for reading from;
    - Exclusive Write: At any time and at any memory cell, only one processor can access the cell for writing into;
  - CRCW
    - Concurrent Read: At any time and at any memory cell, any number of processors can access the cell for reading from;

        - Concurrent Write: At any time and at any memory cell, any number of processors can access the cell for writing into; Note that we need a special mechanism to resolve write conflicts since the conflicting processors might want to write different numbers and we have to determine which among these will be written into the cell.
        - Resolving write Conflicts
            - Common: Concurrent write is allowed if the conflicting processors have the same message to write;
            - Arbitrary: When there is a write conflict, an arbitrary one of the conflicting processors will get to write. The algorithm should work correctly independent of who gets to write;
            - Priority: Write conflicts are resolved using priorities assigned to the processors (at the beginning of time).
- We say a parallel algorithm is optimal if $PT = S$
- The work done by a parallel algorithm = PT
- A parallel algorithm is asymptotically optimal if $PT = O(S)$
- Problem:
    - Input: $X = b_1, b_2, \ldots, b_n \in \{0, 1\}$
    - Output: The Boolean OR of $b_1, b_2, \ldots, b_n$
    - $S = n$
    - Claim: This problem can be solved in $O(1)$ time using *n* common CRCW PRAM processors.
    - Proof:
        - Processor 1 writes a 0 in RESULT:
        - $for\ i = 1\ to\ n$ in parallel:
            - $if\ b_i = 1\ then$ processor *i* tries to write a 1 in RESULT;
        - Runtime: $O(1)$
        - $PT = 2n \Longrightarrow$ The algorithm is not optimal
        - However, the algorithm is asymptotically optimal.
    - Note that we can use a similar algorithm to solve the Boolean AND problem as well.
- Another Problem:
    - Input: $X = k_1, k_2, \ldots, k_n \rightarrow arbitrary\ real\ \#'s$
    - Output: $Max\{k_1, k_2, k_3, \ldots, k_n\}$
    - Claim We can solve this problem in $O(1)$ time using n^2 common CRCW PRAM processors
    - Proof:

- Label the processors as: $P_{11}, P_{12}, \ldots, P_{nn}$; Group the processors such that there are *n* processors in each of *n* groups. Specifically, group G_i has the following processors: $P_{i1}, P_{i2}, \ldots, P_{in}, for\ 1 \le i \le n.$
- $for\ 1 \le i,j \le n$ in parallel:
  - P_{ij} computees b_{ij} = "$Is\ k_i \ge k_j$?"
- $for\ 1 \le i \le n$ in parallel
  - Processors in G_i compute $C_i = b_{i1} \wedge b_{i2} \wedge \ldots \wedge b_{in}$
- $for\ 1 \le i \le n$ in parallel :
  - *if* $C_i = 1$ then
    - P_{i1} tries to write k_i in RESULT
- Runtime = 4 = $O(1)$