CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 24: November 17, 2016

Parallel Sorting

- A number of parallel sorting algorithms have been proposed in the literature. The first deterministic optimal parallel algorithm proposed was by AKS in 1981. This was for a sorting network. Reischuk paralellized Frazer and McKellar's algorithm on the CRCW PRAM to get an asymptotically optimal randomized sorting algorithm (1981). A summary of some of these algorithms is shown in the following Table. In this table, ϵ is any constant greater than zero.

Authors	**Year**	**Model**	P	T	**R/D**
Batcher	1961	Hypercube	n	$\frac{1}{2}\log^2 n$	D
Preparata	1971	CREW PRAM	$n \log n$	$O(\log n)$	D
Ajtai, Komlos, and Szemeredi	1981	Sorting n/w	n	$O(\log n)$	D
Reischük	1981	CRCW PRAM	n	$\widetilde{O}(\log n)$	R
Reif and Valiant	1985	CCC	n	$\widetilde{O}(\log n)$	R
Rajasekaran and Reif	1987	CRCW PRAM	$n(\log n)^\epsilon$	$\widetilde{O}\left(\frac{\log n}{\log\log n}\right)$	R
Cole	1988	CRCW PRAM	$n(\log n)^\epsilon$	$O\left(\frac{\log n}{\log\log\log n}\right)$	D

- We will discuss Preparata's algorithm that uses the following Lemma proven by Valiant:

  Lemma 1: *We can merge two sorted sequences of length n each in $O(\log\log n)$ time using n CREW PRAM processors.*

- Before presenting Preparata's algorithm we will state and prove the slow-down lemma.

- **The slow-down lemma:** If a parallel algorithm runs in time T on a P-processor machine M, then the same algorithm can be run on a P'-processor machine M' in $O\left(\frac{PT}{P'}\right)$ time, for any $P' \leq P$.

- **Proof:** We will simulate the machine M on the machine M'. Specifically, processor 1 of M' will be in-charge of simulating the first $\left\lceil \frac{P}{P'} \right\rceil$ processors of M; processor 2 of M' will be in-charge of simulating the next $\left\lceil \frac{P}{P'} \right\rceil$ processors of M; and so on.

  If we do so, then each parallel step of M can be simulated in $\leq \left\lceil \frac{P}{P'} \right\rceil$ steps on M'. Thus the entire algorithm under concern can be simulated on M' in no more than $T\left\lceil \frac{P}{P'} \right\rceil \leq T\left(\frac{P}{P'}+1\right) = O\left(\frac{PT}{P'}\right)$ time. $\square$

- Now we are ready to see Preparata's algorithm. The basic idea behind Preparata's algorithm is to compute the rank of each element and output the elements in the order of their ranks. Let $X = k_1, k_2, \ldots, k_n$ be the input and let $P = n \log n$. Detailed steps follow.

    0) Partition X into $\log n$ parts $X_1, X_2, \ldots, X_{\log n}$ where each part has $\frac{n}{\log n}$ keys;
    1) **for** $i = 1$ **to** $\log n$ **in parallel do**
    2) Sort X_i recursively using n processors to get S_i;
    3) **for** $1 \leq i, j \leq \log n$ **in parallel do**
    4) Merge S_i with S_j using $\frac{n}{\log n}$ processors;
    5) **for** $i = 1$ **to** n **in parallel do**
    6) Using $\log n$ processors compute the global rank r_i of k_i by
    7) adding the $\log n$ partial ranks computed in Step 4;
    8) **for** $1 \leq i \leq n$ **in parallel do**
    9) Processor i writes k_i in cell r_i;

- **Run time analysis:** Let $T(n)$ be the run time of this algorithm on any input of size n when the number of processors used is $n \log n$. Step 1 takes no more than $T\left(\frac{n}{\log n}\right)$ time. In Step 3, for every i and j, we are merging S_i with S_j. S_i and S_j are of size $\frac{n}{\log n}$ each. Thus we can merge them in $O\left(\log\log\left(\frac{n}{\log n}\right)\right) = O(\log\log n)$ time (c.f. Lemma 1). Let S be one of the sorted sequences obtained in Step 1 and let k be any key in S. By looking at the position of k in S, we get to know how many keys of S are less than k. When S is merged with S_i (for any i), we get to know how many keys of S_i are less than k. As a result, we get to know $\log n$ partial ranks for each input key at the end of Step 3. If we add up these partial ranks and add 1, we will get the global rank of k. This is what we do in Step 5. This addition can be done using the prefix computation algorithm. Note that in Step 3, for each key, we have to add $\log n$ integers using $\log n$ processors and hence Step 5 will take $O(\log\log n)$ time. Step 8 takes one unit of time. In summary, we get the following recurrence relation for $T(n)$:

$$T(n) \leq T\left(\frac{n}{\log n}\right) + O(\log\log n).$$

  We can use repeated substitutions to infer that $T(n) = O(\log n)$. Thus we get the following Theorem.

- **Theorem.** We can sort n arbitrary elements in $O(\log n)$ time using $n \log n$ CREW PRAM processors. □

Intractable Problems

- There are many problems for which the best known algorithms take a very long time (e.g., exponential in some of the input parameters). No one has been able to prove that these problems are difficult and need this much time. Thus there is a gap in our understanding of these problems. We refer to these problems as *intractable problems*. Two examples are: Satisfiability (SAT) and Clique.

- **SAT:** For this problem the input is a Boolean formula in Conjunctive Normal Form (CNF) on n variables. The problem is to check if there is an assignment to the variables under which the value of the formula is true (T). A Boolean formula is in CNF if it is a conjunction of disjunctions.

- $F = (\bar{x_1} \vee x_2 \vee x_4) \wedge (x_2 \vee \bar{x_5}) \wedge (x_4 \vee x_1)$ is a formula in CNF.

- We can solve the above problem in $O(2^n|F|)$ time. This is because we can compute the value of the formula under each possible assignment. There are 2^n possible assignments and on each such assignment we can compute the value of F in $O(|F|)$ time.

- **Clique:** The input for this problem are an undirected graph $G(V, E)$ and an integer $k, 1 \leq k \leq |V|$. The task is to check if G has a clique of size k. A clique is nothing but a subgraph that is complete. We want to check if there exists a subset of k nodes in G such that each pair of nodes in this subset is connected by an edge.

- We can solve the clique problem in $O\left(\binom{n}{k}k^2\right)$ time, where $n = |V|$. This is beacuse there are $\binom{n}{k}$ subsets of k nodes. For each such subset, checking if the nodes form a clique can be done in $O(k^2)$ time (assuming that the graph is in adjacency matrix form).

- In the next class, we will describe some commonalities across these seemingly difficult problems.