CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 23: November 15, 2016

Parallel Algorithms: Prefix Computation

- **Input:** A sequence $X = k_1, k_2, \ldots, k_n$ of elements from a domain Σ. $\oplus$ is a binary, associative, and unit operation defined on Σ. Recall that an operation $\oplus$ on Σ is associative if for any three elements x, y, z in Σ, the following holds: $x \oplus (y \oplus z) = (x \oplus y) \oplus z = x \oplus y \oplus z$.

- **Output:** $k_1, k_1 \oplus k_2, k_1 \oplus k_2 \oplus k_3, \ldots, k_1 \oplus k_2 \oplus \cdots k_n$.

A Divide-and-conquer Algorithm for Prefix Computation

- We will first discuss a divide-and-conquer parallel algorithm that employs n CREW PRAM processors.

- Step 0) We partition the input into two equal halves: $X_1 = k_1, k_2, \ldots, k_{n/2}$ and $X_2 = k_{(n/2)+1}, k_{(n/2)+2}, \ldots, k_n$.

- Step 1) $\frac{n}{2}$ processors recursively perform a prefix computation on X_1. Let the output be $k'_1, k'_2, \ldots, k'_{n/2}$; At the same time the other $\frac{n}{2}$ processors recursively perform a prefix computation on X_2. Let the output be $k'_{(n/2)+1}, k'_{(n/2)+2}, \ldots, k'_n$.

- Note that $k'_1, k'_2, \ldots, k'_{n/2}$ is indeed the first half of the prefix outputs for X. Thus we can output these without any modifications.

- Step 3) We can modify $k'_{(n/2)+1}, k'_{(n/2)+2}, \ldots, k'_n$ by pre-adding $k'_{n/2}$ to every element. (Here the word 'adding' refers to the operator $\oplus$). The modified values will be the second half of the prefix outputs for X. This modification can be done in $O(1)$ time using $\frac{n}{2}$ CREW PRAM processors.

Run time analysis: Like for any recursive algorithm, we have to write a recurrence relation for the run time, and solve it. Let $T(n)$ be the run time of the above algorithm on any input of size n, where the number of processors used is n.

Then we get the following recurrence relation: $T(n) = T(n/2) + O(1)$ which solves to $T(n) = O(\log n)$. As a result, we get the following Lemma.

Lemma 1: *We can solve the prefix computation problem on any input of size n in $O(\log n)$ time using n CREW PRAM processors.* $\square$

An Optimal Prefix Computation Algorithm

- We can get an optimal prefix computation algorithm using a technique due to Richard Brent. The idea is to reduce the input size sufficiently, employ a nonoptimal algorithm to solve the problem on the reduced input, and use these results to obtain the results for the original input. Let $P = \frac{n}{\log n}$ and let $X = k_1, k_2, \ldots, k_n$ be the input. A detailed description is given below.

0) Assign $\log n$ elements per processor. Specifically, assign the elements
   $k_{(i-1)\log n+1}, k_{(i-1)\log n+2}, \ldots, k_{i \log n}$ to processor i, for $1 \leq i \leq P$;
1) **for** $i = 1$ **to** P **in parallel do**
2) Processor i performs a prefix computation on its $\log n$ elements
   $k_{(i-1)\log n+1}, k_{(i-1)\log n+2}, \ldots, k_{i \log n}$ to get
   $k'_{(i-1)\log n+1}, k'_{(i-1)\log n+2}, \ldots, k'_{i \log n}$;
3) P processors collective perform a prefix computation on
   $k'_{\log n}, k'_{2\log n}, \ldots, k'_n$ to get
   $k''_{\log n}, k''_{2\log n}, \ldots, k''_n$;
4) **for** $i = 2$ **to** n **in parallel do**
5) Processor i outputs $k''_{(i-1)\log n} \oplus k'_{(i-1)\log n+1}$,
   $k''_{(i-1)\log n} \oplus k'_{(i-1)\log n+2}, \ldots, k''_{(i-1)\log n} \oplus k'_{i \log n}$;

Run time analysis: Step 2 takes $O(\log n)$ time. In step 3 we have to perform a prefix computation on $\frac{n}{\log n}$ elements using $\frac{n}{\log n}$ processors. Using Lemma 1, we infer that Step 3 takes $O\left(\log\left(\frac{n}{\log n}\right)\right) = O(\log n)$ time. Step 5 takes $O(\log n)$ time. Thus the total run time of the algorithm is $O(\log n)$ resulting in the following Lemma.

Lemma 2: *We can solve the prefix computation problem on any input of size n in $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors.* □

Observation: The above algorithm is asymptotically optimal since $S = n$ and the work done by the parallel algorithm is $O(n)$.

Some Applications

- Numerous problems can be solved efficiently by reducing them to prefix computations. We'll see two examples.

- *Computing the rank of an element*: Here we are given a sequence $X = k_1, k_2, \ldots, k_n$ of arbitrary real numbers and a number $k \in X$. The goal is to compute the rank of k in X. (Recall that $rank(k, X) = |\{q \in X : q < k\}| + 1$). This problem can be reduced to prefix sums computation as follows.

1) Create a bit array $a[1:n]$ such that $a[i] = 1$ if $k_i < k$ and
   $a[i] = 0$ otherwise, for $1 \leq i \leq n$.
2) Compute the prefix sums of $a[1], a[2], \ldots, a[n]$ to get $b[1], b[2], \ldots, b[n]$;
3) Output $b[n] + 1$;

- Observe that step 1 can be done in $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors (by assigning $\log n$ input elements per processor). Step 2 takes $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors (c.f. Lemma 2). Step 3 takes 1 unit of time. Thus the entire algorithm runs in $O(\log n)$ time using $\frac{n}{\log n}$ CREW PRAM processors.

- As a corollary to the above algorithm, we can sort n given elements in $O(\log n)$ time using $\frac{n^2}{\log n}$ CREW PRAM processors. The idea is to assign $\frac{n}{\log n}$ processors per key and compute its rank in parallel. Followed by this, the keys are output based on their ranks. Let $X = k_1, k_2, \ldots, k_n$ be the input.

1) **for** $i = 1$ **to** n **in parallel do**
2) Using $\frac{n}{\log n}$ processors compute the rank r_i of k_i;
3) **for** $i = 1$ **to** n **in parallel do**
4) Processor i outputs k_i in memory cell r_i;

- *Splitting an input sequence*: Here the input is a sequence $X = k_1, k_2, \ldots, k_n$ of arbitrary real numbers and another real number k. The goal is to permute X such that all the elements of X that are smaller than k appear before the rest of the elements. We can also reduce this problem to prefix computations as follows.

1) Create a bit array $a[1:n]$ such that $a[i] = 1$ if $k_i < k$ and
   $a[i] = 0$ otherwise, for $1 \leq i \leq n$.
2) Compute the prefix sums of $a[1], a[2], \ldots, a[n]$ to get $b[1], b[2], \ldots, b[n]$;
3) **for** $i = 1$ **to** n **in parallel do**
4) **if** $k_i < k$ **then** write k_i in cell $b[i]$;
/* Note that all the elements less than k have now
   been placed in successive cells */
5) Place the elements of X that are $> k$ in a similar manner;

Run time analysis: Step 1 can be done in $O(\log n)$ time using $\frac{n}{\log n}$ processors (by assigning $\log n$ elements per processor). Step 2 takes $O(\log n)$ time (c.f. Lemma 2). Step 3 also can be done in $O(\log n)$ time (if we assign $\log n$ elements per processor). Steps 1 through 4 thus take a total of $O(\log n)$ time. As a result, Step 5 will also take the same amount of time. Therefore, the total run time of the algorithm is $O(\log n)$.

Parallel Sorting

- A number of parallel 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). In the next Lecture we will study Preparata's algorithm.