CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 21: November 8, 2016

Parallel Algorithms

- In the last lecture we showed that $T \geq \frac{S}{P}$ where T is the parallel run time needed to solve any given problem using P processors, given that S is the best known sequential run time known to solve the same problem.
- In this lecture we will introduce parallel models of computing and study parallel algorithms for some fundamental problems.

Some Basic Definitions

- We say that a parallel algorithm for a given problem is optimal if $T = \frac{S}{P}$, where T is the parallel run time of the algorithm, P is the number of processors used, and S is the best known sequential run time for solving this problem.
- We say that a parallel algorithm is asymptotically optimal if $T = O\left(\frac{S}{P}\right)$, where T is the parallel run time, P is the number of processors, and S is the best known sequential run time.
- The work done by a parallel algorithm is defined to be PT, where P is the number of processors used and T is the parallel run time.
- The speedup obtained by a parallel algorithm is defined to be $\frac{S}{T}$, where S is the sequential run time and T is the parallel run time.

Parallel Models of Computing

- When it comes to sequential computing, the Random Access Machine (RAM) model has been widely accepted. In this model we assume that each basic operation takes one unit of time and calculate the run times of algorithms as the total number of basic operations performed in the algorithms.
- In contrast, numerous computing models have been introduced in the domain of parallel computing. These models differ in how inter-processor communications are enabled.
- Parallel models can be broadly categorized into two: fixed connection machines and shared memory machines (also known as Parallel Random Access Machines (PRAMs)).

Fixed Connection Machines

- A fixed connection machine is nothing but a directed graph $G(V, E)$ where each node corresponds to a processor and each edge corresponds to a communication link. Example fixed connection machines include linear array, mesh, hypercube, etc.
- In a fixed connection machine, if two nodes are connected by an edge they can communicate in one unit of time. If the two nodes that want to communicate are not connected by an edge, then the communication will happen along a path connecting these two nodes.

Parallel Random Access Machines

- A PRAM is a collection of identical processors, where each processor is a RAM. Communication takes place by writing into and reading from memory cells.
- For example, if processor i wants to communicate with processor j, then processor i can write a message in memory cell j; which then can be read by processor j in the next step.
- Note that processors i and j can communicate in two steps, independent of i and j. Thus one could perhaps think of a PRAM as a fixed connection machine where the underlying graph is complete.
- A PRAM is a synchronous machine with a global clock.
- Note that there could be conflicts in a PRAM for writing and/or reading. Depending on how these conflicts are resolved, we have several variants.
- In an Exclusive Read Exclusive Write (EREW) PRAM, in any time step, only one processor can either read from or write into any memory cell at the same time. Different procesors could read from or write into different distinct cells at the same time though.
- In a Concurrent Read Exclusive Write (CREW) PRAM, any number of processors can read from the same memory cell at the same time. However, at any time step any memory cell can be accessed by at most one processor for writing into.
- In a Concurrent Read Concurrent Write (CRCW) PRAM, both concurrent reads and concurrent writes are permitted.
- When we enable concurrent reads, processors that are conflicting will get to read the same message. If we enable concurrent writes, the conflicting processors might have different messages that they are trying to write. Thus we need a special maechanism to resolve write conflicts. Depending on how we do this, there could be variants.

- In a Common CRCW PRAM, the conflicting processors must have the same message that they are trying to write. In an Arbitrary PRAM, if there is a write conflict, an arbitrary one of the conflicting processors will succeed in writing. (Our algorithm should be correct independent of who gets to write). In a Priority CRCW PRAM, write conflicts are resolved based on priorities assigned to the processors. These priorities are typically static.

- In terms of computational power these models form a hierarchy. Here is a list in increasing order of computational power: EREW PRAM, CREW PRAM, Common CRCW PRAM, Arbitrary CRCW PRAM, and Priority CRCW PRAM.

- There are separation results among these models. To show that one model is more powerful than another, we just have to identify one problem that can be more efficiently solved on one model than the other.

- In a PRAM, we assume that the input as well as the output will be given in the common memory.

Boolean And Problem

- Consider the following problem: Input is a sequence $b_1, b_2, \ldots, b_n$ of bits. The problem is to compute the Boolean And of these n bits.

- Clearly, this problem can be solved sequentially in $n-1$ time steps. (Ignoring -1) $S = n$.

- **Lemma:** This problem can be solved in $O(1)$ time using n common CRCW PRAM processors.

- **Proof:** Here is an algorithm:

  0) Processor i is assigned b_i, for $1 \leq i \leq n$;
  1) Processor 1 writes a 1 in *Result* (*Result* being a memory cell);
  3) **for** $i = 1$ **to** n **in parallel do**
  4) **if** $b_i = 0$ **then** processor i tries to write zero in *Result*;

- The correctness of the above algorithm is clear. Even if there is a single zero bit in the input, in step 4 *Result* will be changed to zero. Also, steps 1 and 4 take $O(1)$ time each implying that the total run time is $O(1)$. □

- The above algorithm is asymptotically optimal since $T = O\left(\frac{S}{P}\right)$.

Finding the Minimum Element

- For this problem, the input is a sequence $X = k_1, k_2, \ldots, k_n$ of arbitrary real numbers. The problem is to output the smallest element of X.
- For this problem also, $S = n$.
- **Lemma:** We can find the minimum of n given elements in $O(1)$ time using n^2 common CRCW PRAM processors.
- **Proof:** We make use of the Boolean And algorithm as a subroutine. Details follow.

0) Let the processors be labelled $P_{1,1}, P_{1,2}, \ldots, P_{1,n}, P_{2,1}, P_{2,2}, \ldots, P_{2,n}, \cdots, P_{n,1}, P_{n,2}, \ldots, P_{n,n}$;
   The processors are grouped into n groups, where $G_i = \{P_{i,1}, P_{i,2}, \ldots, P_{i,n}\}$,
   for $1 \leq i \leq n$. Group G_i is assigned k_i and G_i is required to check if k_i is the smallest element of X;
1) **for** $1 \leq i, j \leq n$ **in parallel do**
2) $\quad$ $P_{i,j}$ computes a bit $b_{i,j}$ = "Is $k_i \leq k_j$?";
3) **for** $i = 1$ **to** n **in parallel do**
4) $\quad$ Processors in G_i collectively compute $c_i = b_{i,1} \wedge b_{i,2} \wedge \cdots \wedge b_{i,n}$;
5) $\quad$ **if** $c_i = 1$ **then** $P_{i,1}$ tries to write k_i in *Result*;

- The correctness of the above algorithm is clear. Step 2 takes $O(1)$ time. Step 4 takes $O(1)$ time as well if we use the Boolean And algorithm described above. Step 5 also takes $O(1)$ time yielding a total run time of $O(1)$. $\square$
- Note that the above algorithm is not asymptotically optimal.

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:** $k_1, k_1 \oplus k_2, k_1 \oplus k_2 \oplus k_3, \ldots, k_1 \oplus k_2 \oplus \cdots k_n$.
- Prefix computation is a fundmental problem in parallel computing. This problem involves both local computations and interprocessor communications. A number of problems can be solved in parallel by reducing them to prefix computation.
- Some examples: 1) $\Sigma = \mathbb{R}; \oplus = +$; 2) $\Sigma = \mathbb{R}; \oplus = \min$; and 3) $\Sigma = 2 \times 2$ matrices; $\oplus = *$

A Divide-and-conquer Algorithm for Prefix Computation

- We will first discuss a divide-and-conquer parallel algorithm that employs n CREW PRAM processors.
- We partition the input into two equal halves: $X_1 = k_1, k_2, \ldots, k_{n/2}$ and $X_2 = k_{(n/2)+1}, k_{(n/2)+2}, \ldots, k_n$.
- $\frac{n}{2}$ processors 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 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.
- 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.