Luis Serrano
CSE-5095-005

January 23, 2014
Prof. Sanguthevar Rajasekaran

Class Notes

Research Topics in Big Data Analytics

Last Class Recap:

Last class the professor talked about two performance measurements used to describe the time and memory required by an algorithm. He also mentioned how the notion of asymptotic function can be employed to provide a more flexible analysis on the measurements. We then jumped to a specific class of algorithms called randomized algorithms. These algorithms use the outcome of coin flips to make some of their decisions. We defined two types of randomized algorithms: Monte Carlo and Las Vegas. We also saw an example of how a Las Vegas algorithm can have a smaller asymptotic runtime than a deterministic algorithm.

Randomized Algorithms

We present an example of a Monte Carlo algorithm to show how it can also have an asymptotic runtime smaller than a deterministic algorithm.

Example

Input: An array $A[1:n]$.

Note: A can only be of two types: 1) Type 1: It has all zeroes. 2) Type 2: It has $n/2$ zeroes and $n/2$ ones.

Output: The type of A.

Deterministic Algorithms:

Any deterministic algorithm needs at least $(n/2) + 1$ steps.

A Randomized Algorithm (Monte Carlo):

```
for i = 1 to k do:
  Pick a random element x of A;
  if x == 1 then:
    return 2;

return 1;
```

Analysis:

Note that if A is of type 1 then the algorithm doesn't give an incorrect answer. Thus, consider what happens when A is of type 2.

Probability of a correct answer in one basic step is	$\geq (1/2)$
Probability of a failure in one basic step is	$\leq (1/2)$
Probability of failure in k basic steps is	$\leq (1/2)^k$

We want the last probability to be $\leq n^{-\alpha}$

$$(1/2)^k \leq n^{-\alpha} \quad \Rightarrow \quad -k \leq -\alpha \log n \quad \Rightarrow \quad k \geq \alpha \log n.$$

As a result, the runtime of the algorithm is $O(\log n)$.

Parallel Algorithms:

Let Π be any problem. Let S be the best sequential run time. Let T be the parallel runtime for solving Π using P processors.

Fact: $T \geq S/P$

Proof: Assume that $T < S/P$. Let M be a sequential machine. Then, each parallel step can be sequentially simulated in $\leq P$ steps. The entire parallel algorithm can be simulated sequentially in $\leq PT$ steps. But, $PT < S$, which is a contradiction □.

Definition: Speedup = S/T. Work done = PT.

Definition: We say a parallel algorithm is optimal when $T = S/P$. A parallel algorithm is asymptotically optimal if $T = O(S/P)$.

Lemma: (Slow-down Lemma) If a parallel algorithm takes T time steps using P processors then the same algorithm runs in time $O(PT/P')$ on a P'-processor machine as long as $P' \leq P$.

Due to the nature of parallel algorithms it is necessary to use new models of computing that provide guidelines of how communications are handled.

Models of Computing:

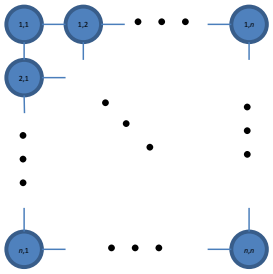
- Fixed Connection Machines: A fixed connection machine is a directed graph $G(V, E)$ where V are the processors and E are the communication links.

Examples

Linear Array

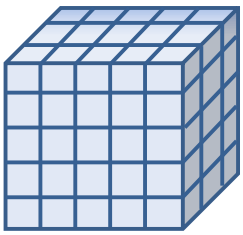
Dimension : 1
Diameter : n-1
Degree : 2

Mesh



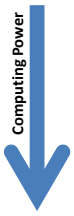
Dimension : 2
Diameter : 2(n-1)
Degree : 4

Hypercube (an example)



Dimension	:	3
Diameter	:	$3(n\text{-}1)$
Degree	:	6

- Shared Memory or Parallel Random Access Machines (PRAMs): The communication happens by writing into and reading from memory. There are different types of models that differ in how conflicts should be handled when accessing the memory. Some of them are:



1- EREW (Exclusive Read, Exclusive Write)
2- CREW (Concurrent Read, Exclusive Write)
3- CRCW (Concurrent Read, Concurrent Write)
    1. Common: Only allow it if they all are writing the same message.
    2. Arbitrary: One of them is chosen arbitrarily.
    3. Priority: Conflicts are resolved by a fixed priority.

Example

Input: Binary values $b_1, \ldots, b_n$
Output: $b_1 \wedge \ldots \wedge b_n$

Fact: We can solve this problem in constant time using n common CRCW PRAM processors.

Algorithm:

```
Processor 1 sets result = 1;
for 1 ≤ i ≤ n in parallel do:
  if b_i == 0 then:
  Processor i tries to set result = 0;
```

Analysis:

$$P = n, T = 2 \quad \Longrightarrow \quad \text{Work done} = 2n = O(n)$$

Because any sequential algorithm has to spend at least n-1 operations, this algorithm is asymptotically optimal.

$$\text{Speedup} = n/2 = \Theta(n)$$

Example

Input: A collection of arbitrary real numbers $X = k_1, \ldots, k_n$
Output: Maximum element of X.

Fact: We can solve this problem in O(1) time using n^2 common CRCW PRAM processors.

Algorithm:

Divide the n^2 processor in the following way:

$P_{1,1}, \ldots, P_{1,n}$ $\quad$ $P_{2,1}, \ldots, P_{2,n}$ $\quad$ $\ldots$, $\quad$ $P_{n,1}, \ldots, P_{n,n}$

G_1 $\quad$ G_2 $\quad$ G_n

```
Assign k_i to G_i, 1 ≤ i ≤ n;
for 1 ≤ i, j ≤ n in parallel do:
  Processor P_{i,j} computes b_{i,j} = (k_i ≥ k_j);
for 1 ≤ i ≤ n in parallel do:
  Processors in G_i compute C_i = b_{i,1} ^ … ^ b_{i,n};
for 1 ≤ i ≤ n in parallel do:
  if C_i == 1 then:
    P_{i,1} writes k_i to result;
```

Example (Prefix Computation)

Input: $k_1, \ldots, k_n \in \Sigma$. An operator $\bigoplus$ which is binary, unit time computable and associative.
Output: $k_1, k_1 \bigoplus k_2, \ldots, k_1 \bigoplus \ldots \bigoplus k_n$

Algorithm:

1- Using $n/2$ CREW processors do a prefix computation on $k_1, \ldots, k_{n/2}$. Using $n/2$ CREW processors do a prefix on $k_{(n/2)+1}, \ldots, k_n$. Let the results be $k'_1, \ldots, k'_n$.
2- Output $k'_1, \ldots, k'_{n/2}$. Pre-$\bigoplus$ $k'_{n/2}$ to each one of $k'_{(n/2)+1}, \ldots, k'_n$.

Analysis:

Let $T(n)$ be the runtime of this algorithm on any input of size n using n CREW PRAM processors. Then, $T(n) = T(n/2) + 1 = O(\log n)$.

$$P = n, T = O(\log n). \quad \Rightarrow \quad PT = O(n \log n)$$

Because $S = n$, this algorithm isn't asymptotically optimal.

Lemma: We can solve this problem in $O(\log n)$ time using $n/\log n$ CREW PRAM processors.

Proof: First, we assign a group of log n elements to each processor; each one then does a prefix computation. Then they calculate the prefix values for each group's last member. Finally, each processor i pre-$\bigoplus$ the prefix value of the last member in the i-1 group to each of their group elements.