CMSC 351 - Introduction to Algorithms
Spring 2012
Lecture 24

Instructor: MohammadTaghi Hajiaghayi
Scribe: Rajesh Chitnis

1 Introduction

In this lecture we give a brief introduction to Parallel Algorithms.

2 Parallel Algorithms

We want to compare Parallel Algorithms versus traditional Sequential (or serial) Algorithms. There are numerous types of parallel computers in operation and no longer can we adopt one "generic" model of computation and hope that it adopts to all parallel computers. In this short lecture we cannot cover most of parallel computing. We just give some examples. You can take other classes which will give more material on this topic.

As we have seen in non-deterministic computers having an unlimited number of machines can help us. Often the more processors we use, upto a certain limit, the faster the algorithm becomes. However since in real-world, the number of computers (processors) is limited, it is important to use the processors efficiently. Another important issue is **communication** among the processors. Generally, it takes longer to exchange data between the processors than it does to perform simple operations on the data. Does it automatically mean that the algorithm must minimize communications and arrange it in an effective way. For some computer systems, the answer is yes, but if processors can exchange data at a sufficiently high rate, the constraint on communication can become far less severe. Yet another important issue is **synchronization**, which can be a major problem for parallel algorithms that run on independent computers loosely connected by some communication network. The objective of this brief discussion is not to be self-contained, but rather justify the need for taking other classes if you wish to learn more on this topic.

2.1 Speed-Up

We denote the running time of an algorithm by $T(n, p)$ where n is the input size and p is the number of processors used. The ratio $S(p) = \frac{T(n,1)}{T(n,p)}$ is called the **speed-up** of the algorithm. If S(p)=p, then we call it a **perfect** speed-up. The value of $T(n, 1)$ is the best known sequential algorithm.

One can fix p and the minimize $T(n, p)$. However if p changes, we must have a new algorithm. It is more desirable to find an algorithm that works for any value of p. In general, we can modify an algorithm with $T(n, p) = X$ to an algorithm with $T(n, \frac{p}{k}) = kX$ by replacing each step of the original algorithm with k steps in which one processor simulates the execution of one step of the k processors. Generalizing this to case where k does not divide p is deferred to other classes.

3 Various Models of Parallel Computation and Communication

3.1 Shared-Memory Model

We assume that there is a random-access shared memory such that any processor can access any variable with unit cost. The assumption of unit cost regardless of the number of processors of the size of the memory is unrealistic but can become a good approximation if the rate at which processors can exchange data is high enough. Issue with concurrent access to shared data is a big topic in other courses, offering a diversity of problem definitions and solutions, including: allowing it in an abstract model, using programmer specified locking and or system handling. .

3.2 Interconnection Network

It can be represented by a graph such that the vertices correspond to the processors and two vertices are connected if the corresponding processors have a direct link between them. Each processor has a quick-access local memory, but the communication is done through messages (though maybe by traversing several links to arrive at their destination).

A subtype of parallel algorithms which is in use in our modern world and over the internet is a distributed algorithm which often uses interconnection networks. One of the major challenged in developing and implementing distributed algorithms is successfully coordinating the behavior of the independent parts of the algorithm in the face of processor failures and unreliable communication links.

4 Maximum-Finding Algorithm in Parallel

The problem is to find the maximum among n distinct numbers, given in an array. As we have seen $T(n, 1) = n - 1$. An efficient way to run a parallel

algorithm is to use a binary tree. The processors are divided into pairs for the first round (with possibly one processor sitting out, in case of an odd number of players), all the winners are again divided into pairs and so on until the finals. The total number of rounds is $\lceil \log_2 n \rceil$ and the number of processors is $\lfloor \frac{n}{2} \rfloor$. Thus $T(n, \lfloor \frac{n}{2} \rfloor) = \lceil \log_2 n \rceil$ and $S(\lfloor \frac{n}{2} \rfloor) = O(\frac{n}{\lceil \log_2 n \rceil})$ which is very good. Note that we used the shared-memory model here.

5 A Generalization: The Parallel-Prefix Problem

Let $\cdot$ be an arbitrary associative binary operation, namely it satisfies $x \cdot (y \cdot z) = (x \cdot y) \cdot z$, which we simply call product. For example $\cdot$ can represent addition, multiplication or maximum of two numbers.

The Problem: Given a sequence of real numbers $x_1, x_2, \ldots, x_n$, compute the product $x_1 \cdot x_2 \cdot \ldots \cdot x_k$ for all $1 \leq k \leq n$.

We denote by $PR(i, j)$ the product $x_i \cdot x_{i+1} \cdot \ldots \cdot x_j$. The goal is to compute $PR(1, k)$ for all $1 \leq k \leq n$. The sequential version of the problem can be solved trivially by simply computing the prefixes in order. The parallel case is harder and needs divide-and-conquer. We assume n is a power of 2.

Induction Hypothesis
We know how to solve the problem for $\frac{n}{2}$ elements.

The basis for one element is trivial. The algorithm proceeds by dividing the input in half, and solving each half by induction. Thus we obtain the value of $PR(1, k)$ and $PR(\frac{n}{2}) + 1, \frac{n}{2} + k$ for all $1 \leq k \leq \frac{n}{2}$. The values of $PR(1, m)$ for $1 \leq m \leq \frac{n}{2}$ can be used directly. The values $PR(1, m)$ for $\frac{n}{2} < m \leq n$ can be attained by computing $PR(1, \frac{n}{2}) \cdot PR(\frac{n}{2} + 1, m)$. Both terms are known by induction. Note that we use the associativity of the $\cdot$ operation. This step can be done in one step if we have n processors. Thus overall we have $T(n, n) = O(\log n)$ and $S(n) = O(\frac{n}{\log n})$ which is very good.

References

[1] Udi Manber, *Introduction to Algorithms - A Creative Approach*