CSE4001 - Parallel and Distributed Computing

B2 Slot

Theory DA 1

Submitted by: Alokam Nikhitha

Reg No:19BCE2555

Question 1.

Why is it difficult to construct a true shared-memory computer? What is the smallest number of switches required to connecting n processors to a shared memory with k words (each of which can be accessed independently)?

Answer:

A shared memory computer is not difficult to build. Building an efficient shared memory computer is difficult.
The link between the CPU or CPUs and the memory is today's most pressing issue in computer design. The data will be crunched extraordinarily quickly if you can get it into the CPU. Then you must get rid of it in order to make room for more. Memory is wrapped around the CPU to reduce data routes, and layers of cache are interposed to smooth down the bumps in that path.
However, if the memory is wrapped around one CPU with cache here, there, and everywhere, wrapping it around a second CPU with equivalent efficiency is extremely tough. Before caches were invented, it was thirty years ago.

For a true shared-memory computer such as EREW PRAM with p processors and a shared memory with b words, each of the p processors in the ensemble can access any of the memory words, provided that a word is not accessed by more than one processor simultaneously. To ensure such connectivity, the total number of switches must be $\theta(pb)$. For a reasonable memory size, constructing a switching network of this complexity is very expensive. Thus a true shared-memory computer is impossible to realize in practice.

Question 2.

Part a)
What is the purpose of parallel algorithms?

A parallel algorithm is an algorithm that can execute several instructions simultaneously on different processing devices and then combine all the individual outputs to produce the final result.

The parallelism in an algorithm can yield improved performance on many different kinds of computers. For example, on a parallel computer, the operations in a parallel algorithm can be performed simultaneously by different processors. Furthermore, even on a single-processor computer the parallelism in an algorithm can be exploited by using multiple functional units, pipelined functional units, or

pipelined memory systems. Thus, it is important to make a distinction between 1 the parallelism in an algorithm and the ability of any particular computer to perform multiple operations in parallel. Of course, in order for a parallel algorithm to run efficiently on any type of computer, the algorithm must contain at least as much parallelism as the computer, for otherwise resources would be left idle. Unfortunately, the converse does not always hold: some parallel computers cannot efficiently execute all algorithms, even if the algorithms contain a great deal of parallelism. Experience has shown that it is more difficult to build a general-purpose parallel machine than a general-purpose sequential machine.

Parallel algorithms are designed to improve the computation speed of a computer. For analyzing a Parallel Algorithm, we normally consider the following parameters −

- Time complexity (Execution Time),
- Total number of processors used, and
- Total cost.

Time Complexity

The main reason behind developing parallel algorithms was to reduce the computation time of an algorithm. Thus, evaluating the execution time of an algorithm is extremely important in analyzing its efficiency.

Execution time is measured on the basis of the time taken by the algorithm to solve a problem. The total execution time is calculated from the moment when the algorithm starts

executing to the moment it stops. If all the processors do not start or end execution at the same time, then the total execution time of the algorithm is the moment when the first processor started its execution to the moment when the last processor stops its execution.

Time complexity of an algorithm can be classified into three categories−

- Worst-case complexity − When the amount of time required by an algorithm for a given input is maximum.
- Average-case complexity − When the amount of time required by an algorithm for a given input is average.
- Best-case complexity − When the amount of time required by an algorithm for a given input is minimum.

Asymptotic Analysis

The complexity or efficiency of an algorithm is the number of steps executed by the algorithm to get the desired output. Asymptotic analysis is done to calculate the complexity of an algorithm in its theoretical analysis. In asymptotic analysis, a large length of input is used to calculate the complexity function of the algorithm.

Note − Asymptotic is a condition where a line tends to meet a curve, but they do not intersect. Here the line and the curve is asymptotic to each other.

Asymptotic notation is the easiest way to describe the fastest and slowest possible execution time for an algorithm using high bounds and low bounds on speed. For this, we use the following notations −

- **Big O notation**
- **Omega notation**
- **Theta notation**

Big O notation

In mathematics, Big O notation is used to represent the asymptotic characteristics of functions. It represents the behavior of a function for large inputs in a simple and accurate method. It is a method of representing the upper bound of an algorithm's execution time. It represents the longest amount of time that the algorithm could take to complete its execution. The function −

$$f(n) = O(g(n))$$

iff there exists positive constants c and n0 such that $f(n) \leq c * g(n)$ for all n where $n \geq n0$.

Omega notation

Omega notation is a method of representing the lower bound of an algorithm's execution time. The function −

$$f(n) = \Omega (g(n))$$

iff there exists positive constants c and n0 such that $f(n) \geq c * g(n)$ for all n where $n \geq n0$.

Theta Notation

Theta notation is a method of representing both the lower bound and the upper bound of an algorithm's execution time. The function −

$$f(n) = \theta(g(n))$$

iff there exists positive constants c1, c2, and n0 such that c1 * g(n) ≤ f(n) ≤ c2 * g(n) for all n where n ≥ n0.

Speedup of an Algorithm

The performance of a parallel algorithm is determined by calculating its speedup. Speedup is defined as the ratio of the worst-case execution time of the fastest known sequential algorithm for a particular problem to the worst-case execution time of the parallel algorithm.

$$\text{speedup} = \frac{\text{Worst case execution time of the fastest known sequential for a particular problem}}{\text{Worst case execution time of the parallel algorithm}}$$

Number of Processors Used

The number of processors used is an important factor in analyzing the efficiency of a parallel algorithm. The cost to buy, maintain, and run the computers are calculated. Larger the number of processors used by an algorithm to solve a problem, more costly becomes the obtained result.

Total Cost

Total cost of a parallel algorithm is the product of time complexity and the number of processors used in that particular algorithm.

Total Cost = Time complexity × Number of processors used

Therefore, the efficiency of a parallel algorithm is –

Efficiency =(Worst case execution time of sequential algorithm) / (Worst case execution time of the parallel algorithm)

Part b)

What are the key characteristics of a parallel algorithm?

Various characteristics of a parallel algorithm includes:

1. **Execution Time:**
   The time elapsed between the start and finish of a program's execution on a sequential computer is known as its serial runtime. The parallel runtime is the amount of time that passes between the start of a parallel computation and the completion of the last processing element. The serial runtime is denoted by Ts, whereas the parallel runtime is denoted by Tp.
   The execution time is divided into two categories:
   - Serial Execution Time.
   - Parallel Execution Time.
2. **Total Parallel Overhead:**
   The overheads incurred by a parallel program are encapsulated into a single expression referred to as the overhead function. We define overhead function or total overhead of a parallel system as the total time collectively spent by all the procession elements over and above that required by the fastest known sequential algorithm for solving the same problem on a single processing element.

3. **Speedup:**

   The measure that delivers this relative benefit of solving a problem in parallel is speedup, which is used to evaluate the performance gain realised by parallelizing a specific application over a sequential version.

   Speedup of a system, as discussed above, is defined as the ratio of the time taken to solve a problem on a single procession element to the time required to solve the same problem on a parallel computer with p identical procession elements.

4. **Efficiency:**

   Efficiency is the measure of the fraction of time for which a processing element is usefully employed. Efficiency is defined as the ratio of speedup to the number of processing elements. We denote efficiency by the symbol E and it is given as:

   E = S / p

5. **Scalability:**

   Scalability is a characteristic of a system or process to handle a growing amount of work or its potential under an increased or expanding workload. In a parallel system it can maintain constant efficiency E when increasing the number of processors and the size of the problem.