Parallel Algorithm

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com


https://www.facebook.com/tutorialspointindia


https://twitter.com/tutorialspoint

About the Tutorial

A parallel algorithm can be executed simultaneously on many different processing devices and then combined together to get the correct result. Parallel algorithms are highly useful in processing huge volumes of data in quick time.

This tutorial provides an introduction to the design and analysis of parallel algorithms. In addition, it explains the models followed in parallel algorithms, their structures, and implementation.

Audience

This tutorial will help the undergraduate students of computer science learn the basic-to-advanced topics of parallel algorithm.

Prerequisites

In this tutorial, all the topics have been explained from elementary level. Therefore, a beginner can understand this tutorial very easily. However if you have a prior knowledge of writing sequential algorithms, it will be helpful in some chapters.

Copyright & Disclaimer

Table of Contents

1. PARALLEL ALGORITHM —INTRODUCTION

An **algorithm** is a sequence of steps that take inputs from the user and after some computation, produces an output. 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.

Concurrent Processing

The easy availability of computers along with the growth of Internet has changed the way we store and process data. We are living in a day and age where data is available in abundance. Every day we deal with huge volumes of data that require complex computing and that too, in quick time. Sometimes, we need to fetch data from similar or interrelated events that occur simultaneously. This is where we require **concurrent processing** that can divide a complex task and process it multiple systems to produce the output in quick time.

Concurrent processing is essential where the task involves processing a huge bulk of complex data. Examples include: accessing large databases, aircraft testing, astronomical calculations, atomic and nuclear physics, biomedical analysis, economic planning, image processing, robotics, weather forecasting, web-based services, etc.

What is Parallelism?

Parallelism is the process of processing several set of instructions simultaneously. It reduces the total computational time. Parallelism can be implemented by using **parallel computers**, i.e. a computer with many processors. Parallel computers require parallel algorithm, programming languages, compilers and operating system that support multitasking.

In this tutorial, we will discuss only about **parallel algorithms**. Before moving further, let us first discuss about algorithms and their types.

What is an Algorithm?

An **algorithm** is a sequence of instructions followed to solve a problem. While designing an algorithm, we should consider the architecture of computer on which the algorithm will be executed. As per the architecture, there are two types of computers:

- Sequential Computer
- Parallel Computer

Depending on the architecture of computers, we have two types of algorithms:

- **Sequential Algorithm** – An algorithm in which some consecutive steps of instructions are executed in a chronological order to solve a problem.

- **Parallel Algorithm –** The problem is divided into sub-problems and are executed in parallel to get individual outputs. Later on, these individual outputs are combined together to get the final desired output.

It is not easy to divide a large problem into **sub-problems**. Sub-problems may have data dependency among them. Therefore, the processors have to communicate with each other to solve the problem.

It has been found that the time needed by the processors in communicating with each other is more than the actual processing time. So, while designing a parallel algorithm, proper CPU utilization should be considered to get an efficient algorithm.

To design an algorithm properly, we must have a clear idea of the basic **model of computation** in a parallel computer.

Model of Computation

Both sequential and parallel computers operate on a set (stream) of instructions called algorithms. These set of instructions (algorithm) instruct the computer about what it has to do in each step.

Depending on the instruction stream and data stream, computers can be classified into four categories:

- Single Instruction stream, Single Data stream (SISD) computers
- Single Instruction stream, Multiple Data stream (SIMD) computers
- Multiple Instruction stream, Single Data stream (MISD) computers
- Multiple Instruction stream, Multiple Data stream (MIMD) computers

SISD Computers

SISD computers contain **one control unit**, **one processing unit**, and **one memory unit.**

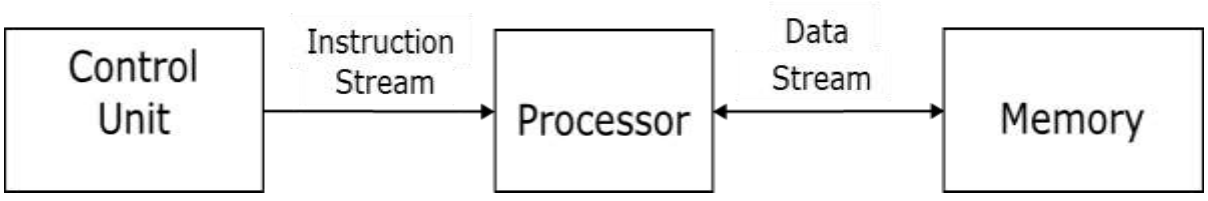

Figure : SISD Computers

In this type of computers, the processor receives a single stream of instructions from the control unit and operates on a single stream of data from the memory unit. During

computation, at each step, the processor receives one instruction from the control unit and operates on a single data received from the memory unit.

SIMD Computers

SIMD computers contain **one control unit**, **multiple processing units**, and **shared memory or interconnection network**.

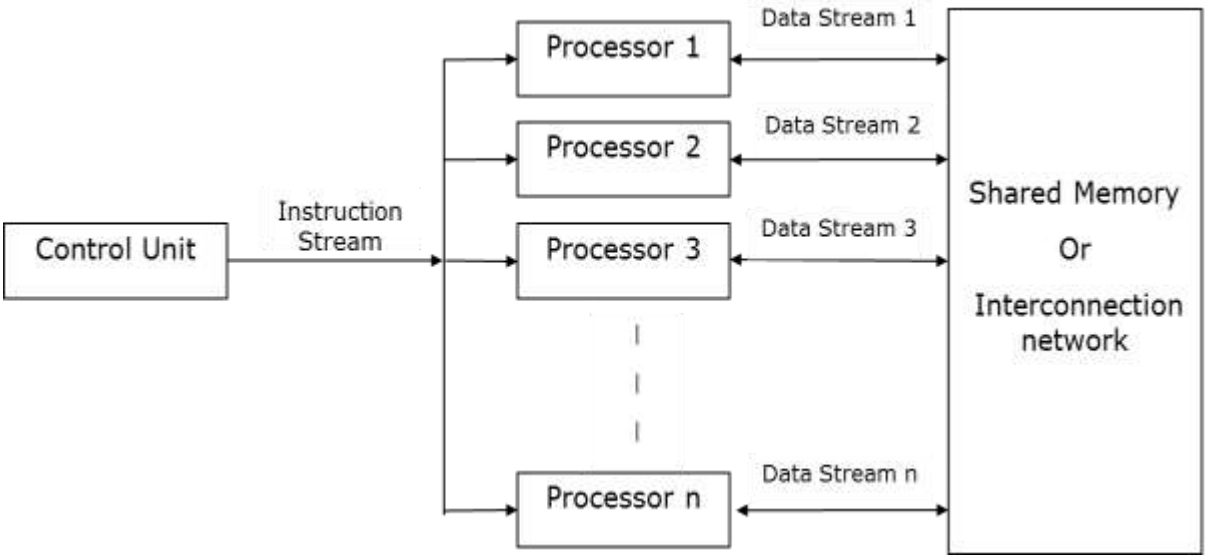

Figure : SIMD Computers

Here, one single control unit sends instructions to all processing units. During computation, at each step, all the processors receive a single set of instructions from the control unit and operate on different set of data from the memory unit.

Each of the processing units has its own local memory unit to store both data and instructions. In SIMD computers, processors need to communicate among themselves. This is done by **shared memory** or by **interconnection network**.

While some of the processors execute a set of instructions, the remaining processors wait for their next set of instructions. Instructions from the control unit decides which processor will be **active** (execute instructions) or **inactive** (wait for next instruction).

MISD Computers

As the name suggests, MISD computers contain **multiple control units**, **multiple processing units**, and **one common memory unit**.

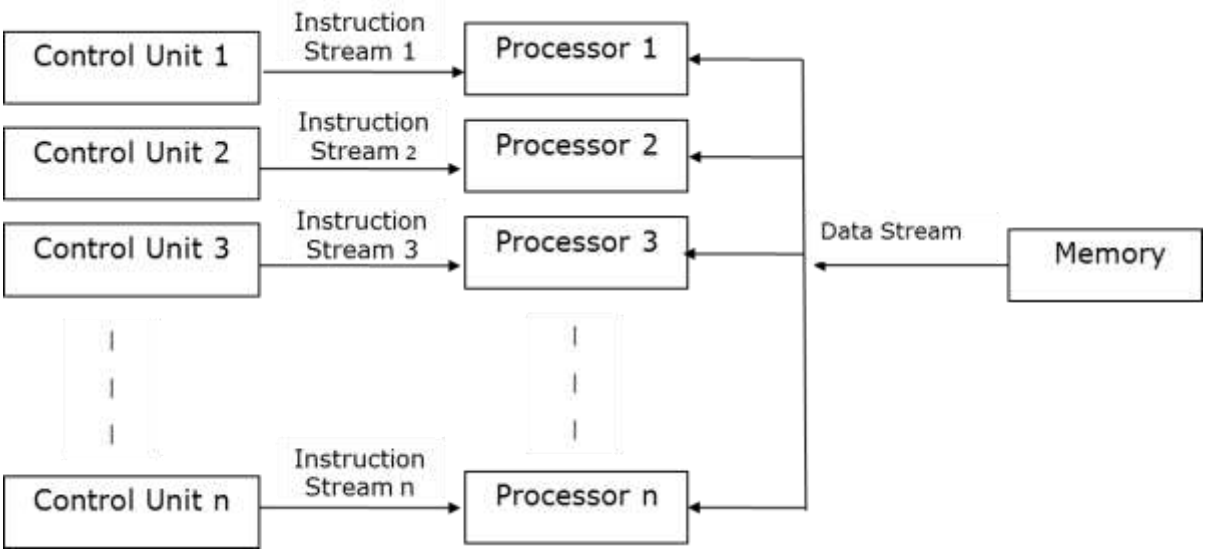

Figure : MISD Computers

Here, each processor has its own control unit and they share a common memory unit. All the processors get instructions individually from their own control unit and they operate on a single stream of data as per the instructions they have received from their respective control units. This processor operates simultaneously.

MIMD Computers

MIMD computers have **multiple control units**, **multiple processing units**, and a **shared memory** or **interconnection network**.

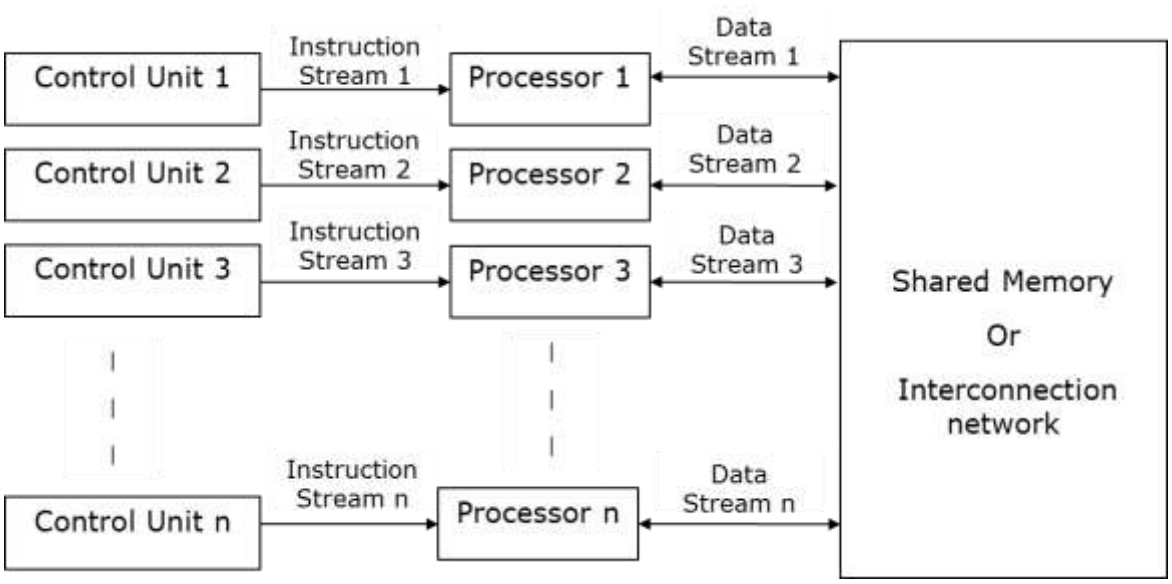

Figure : MIMD Computers

Here, each processor has its own control unit, local memory unit, and arithmetic and logic unit. They receive different sets of instructions from their respective control units and operate on different sets of data.

Note:

- An MIMD computer that shares a common memory is known as **multiprocessors**, while those that uses an interconnection network is known as **multicomputers**.
- Based on the physical distance of the processors, multicomputers are of two types:
    - **Multicomputer –** When all the processors are very close to one another (e.g., in the same room).
    - **Distributed system –** When all the processors are far away from one another (e.g.- in the different cities)

2. PARALLEL ALGORITHM – ANALYSIS

Analysis of an algorithm helps us determine whether the algorithm is useful or not. Generally, an algorithm is analyzed based on its execution time (**Time Complexity**) and the amount of space (**Space Complexity**) it requires.

Since we have sophisticated memory devices available at reasonable cost, storage space is no longer an issue. Hence, space complexity is not given so much of importance.

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 $\mathbf{n_0}$ such that $\mathbf{f(n) \leq c * g(n)}$ for all **n** where $\mathbf{n \geq n_0}$.

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 $\mathbf{n_0}$ such that $\mathbf{f(n) \geq c * g(n)}$ for all **n** where $\mathbf{n \geq n_0}$.

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 $\mathbf{c_1}$, $\mathbf{c_2}$, and $\mathbf{n_0}$ such that $c1*g(n) \leq f(n) \leq c2 * g(n)$ for all **n** where $\mathbf{n \geq n_0}$.

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.

$$\text{Total Cost} = \text{Time complexity} \times \text{Number of processors used}$$

Therefore, the **efficiency** of a parallel algorithm is:

$$\text{Efficiency} = \frac{\text{Worst case execution time of sequential algorithm}}{\text{Worst case execution time of the parallel algorithm}}$$