Chapter -1

(1) Implicit Parallelism: Trends in Microprocessor Architectures

- The traditional logical view of a sequential computer consists of a memory connected to a processor via a datapath. All three components – processor, memory, and datapath – present bottlenecks to the overall processing rate of a computer system. A number of architectural innovations over the years have addressed these bottlenecks. One of the most important innovations is multiplicity – in processing units, datapaths, and memory units. This multiplicity is either entirely hidden from the programmer, as in the case of implicit parallelism, or exposed to the programmer in different forms. In this chapter, we present an overview of important architectural concepts as they relate to parallel processing. The objective is to provide sufficient detail for programmers to be able to write efficient code on a variety of platforms. We develop cost models and abstractions for quantifying the performance of various parallel algorithms, and identify bottlenecks resulting from various programming constructs.
- While microprocessor technology has delivered significant improvements in clock speeds over the past decade, it has also exposed a variety of other performance bottlenecks. To alleviate these bottlenecks, microprocessor designers have explored alternate routes to cost-effective performance gains. In this section, we will outline some of these trends with a view to understanding their limitations and how they impact algorithm and code development. The objective here is not to provide a comprehensive description of processor architectures.
- Clock speeds of microprocessors have posted impressive gains - two to three orders of magnitude over the past 20 years. However, these increments in clock speed are severely diluted by the limitations of memory technology. At the same time, higher levels of device integration have also resulted in a very large transistor count, raising the obvious issue of how best to utilize them.

In sort Parallesim include

Multiplicity of Functional units
Pipelining within the CPU
Hierarchical Memory Systems
Multiprogramming and Timesharing

(Refer :Sasikumar Book for Above Topic)

Prepared By : Mr.D.S.Vyas
(Asst.Prof,CCMCA)

(2) Limitations of Memory System Performance

- The effective performance of a program on a computer relies not just on the speed of the processor but also on the ability of the memory system to feed data to the processor. At the logical level, a memory system, possibly consisting of multiple levels of caches, takes in a request for a memory word and returns a block of data of size b containing the requested word after l nanoseconds. Here, l is referred to as the latency of the memory. The rate at which data can be pumped from the memory to the processor determines the bandwidth of the memory system.

Example
It is very important to understand the difference between latency and bandwidth since different, often competing, techniques are required for addressing these. As an analogy, if water comes out of the end of a fire hose 2 seconds after a hydrant is turned on, then the latency of the system is 2 seconds. Once the flow starts, if the hose pumps water at 1 gallon/second then the 'bandwidth' of the hose is 1 gallon/second. If we need to put out a fire immediately, we might desire a lower latency. This would typically require higher water pressure from the hydrant. On the other hand, if we wish to fight bigger fires, we might desire a higher flow rate, necessitating a wider hose and hydrant. As we shall see here, this analogy works well for memory systems as well. Latency and bandwidth both play critical roles in determining memory system performance.

(3) Dichotomy of Parallel Computing Platforms
In the preceding sections, we pointed out various factors that impact the performance of a serial or implicitly parallel program. The increasing gap in peak and sustainable performance of current microprocessors, the impact of memory system performance, and the distributed nature of many problems present overarching motivations for parallelism. We now introduce, at a high level, the elements of parallel computing platforms that are critical for performance oriented and portable parallel programming. To facilitate our discussion of parallel platforms, we first explore a dichotomy based on the logical and physical organization of parallel platforms. The logical organization refers to a programmer's view of the platform while the physical organization refers to the actual hardware organization of the platform. The two critical components of parallel computing from a programmer's perspective are ways of expressing parallel tasks and mechanisms for specifying interaction between these tasks. The former is sometimes also referred to as the control structure and the latter as the communication model.

(4) Physical Organization of Parallel Platforms

A natural extension of the serial model of computation (the Random Access Machine, or RAM) consists of p processors and a global memory of unbounded size that is uniformly accessible to all processors. All processors access the same address space. Processors share a common clock but may execute different instructions in each cycle. This ideal model is also referred to as a parallel random access machine (PRAM). Since PRAMs allow concurrent access to various memory locations,

depending on how simultaneous memory accesses are handled, PRAMs can be divided into four subclasses.

One of the major overheads in the execution of parallel programs arises from communication of information between processing elements. The cost of communication is dependent on a variety of features including the programming model semantics, the network topology, data handling and routing, and associated software protocols.

(5) Message Passing Costs in Parallel Computers

The time taken to communicate a message between two nodes in a network is the sum of the time to prepare a message for transmission and the time taken by the message to traverse the network to its destination. The principal parameters that determine the communication latency are as follows:

(a) Efficient algorithms for routing a message to its destination are critical to the performance of parallel computers. A routing mechanism determines the path a message takes through the network to get from the source to the destination node. It takes as input a message's source and destination nodes. It may also use information about the state of the network. It returns one or more paths through the network from the source to the destination node.

(b) Routing mechanisms can be classified as minimal or non-minimal. A minimal routing mechanism always selects one of the shortest paths between the source and the destination. In a minimal routing scheme, each link brings a message closer to its destination, but the scheme can lead to congestion in parts of the network. A non-minimal routing scheme, in contrast, may route the message along a longer path to avoid network congestion.

(6) Impact of process mapping

a programmer often does not have control over how logical processes are mapped to physical nodes in a network. For this reason, even communication patterns that are not inherently congesting may congest the network. corresponds to a situation in which processes have been mapped randomly to processing nodes. In this case, it is easy to see that each link in the machine carries up to six channels of data between processes. This may potentially result in considerably larger communication times if the required data rates on communication channels between processes are high. ▪

ReferFrom Sashikumar book

(7) Classification of Computer (instruction,Data Stream)
The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available in the architecture:

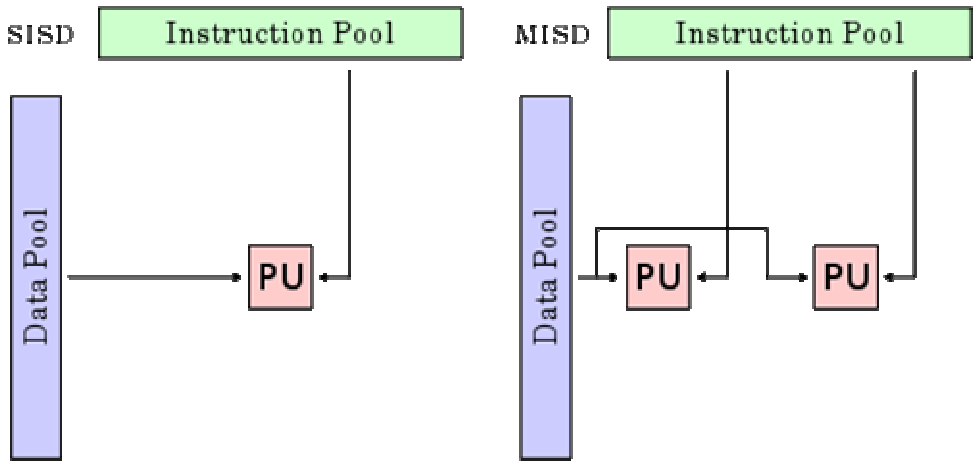
Single Instruction, Single Data stream (SISD)
A sequential computer which exploits no parallelism in either the instruction or data streams. Single control unit (CU) fetches single Instruction Stream (IS) from memory. The CU then generates appropriate control signals to direct single processing element (PE) to operate on single Data Stream (DS) i.e. one operation at a time
Examples of SISD architecture are the traditional uniprocessor machines like a PC (currently manufactured PCs have multiple processors) or old mainframes.

Single Instruction, Multiple Data streams (SIMD)
A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array processor or GPU.

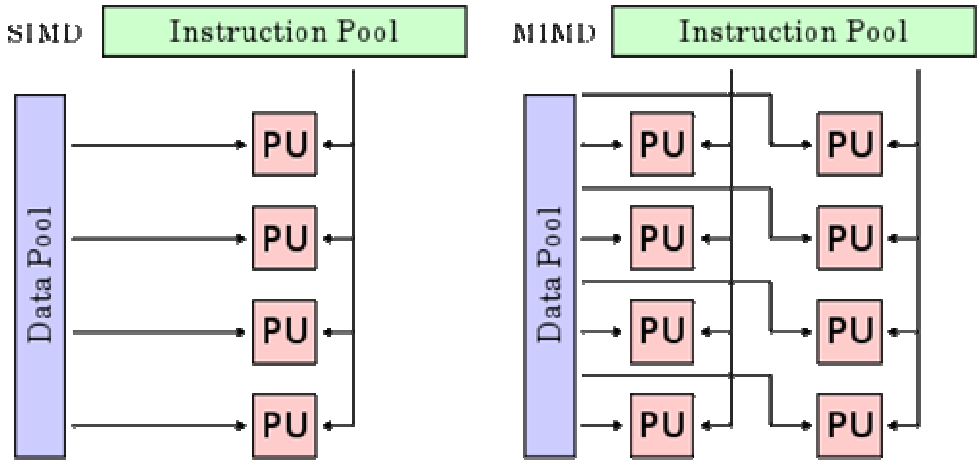
Multiple Instruction, Single Data stream (MISD)
Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.

Multiple Instruction, Multiple Data streams (MIMD)
Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space.
Diagram comparing classifications
Visually, these four architectures are shown below where each "PU" is a processing unit:

SISD MISD



SIMD MIMD



Prepared By : Mr.D.S.Vyas
(Asst.Prof,CCMCA)