Parallel Processors and Computing

by

PARTHA ROY

Asso.Prof.

BIT, Durg

UNIT-1

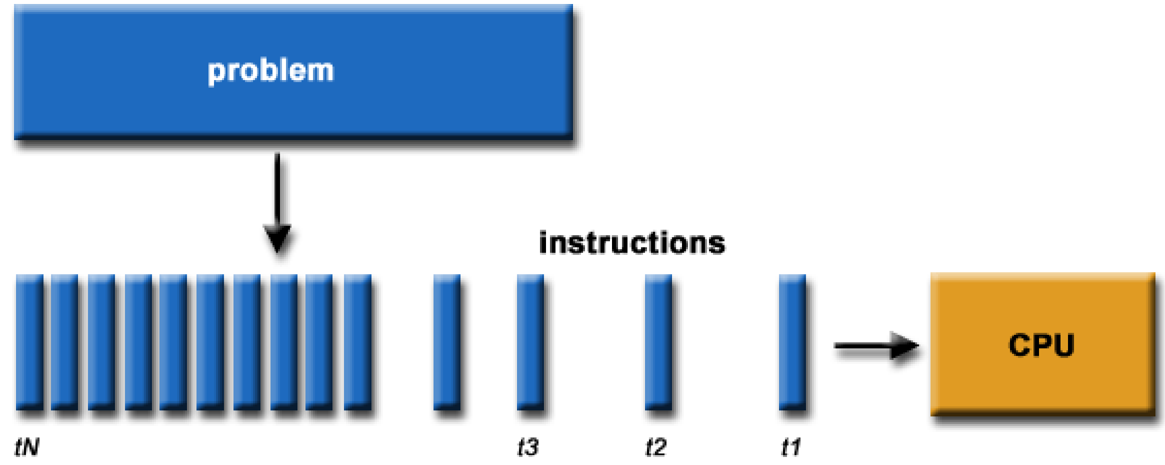
INTRODUCTION & TECHNIQUES OF PARALLELISM

WHAT IS PARALLEL COMPUTING?

Traditionally, software has been written for serial computation:

- To be run on a single computer having a single Central Processing Unit (CPU);
- A problem is broken into a discrete series of instructions.
- Instructions are executed one after another.
- Only one instruction may execute at any moment in time.

Image showing Serial Computation



In the simplest sense, parallel computing is the simultaneous use of multiple computing resources to solve a computational problem.

- To be run using multiple CPUs
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs

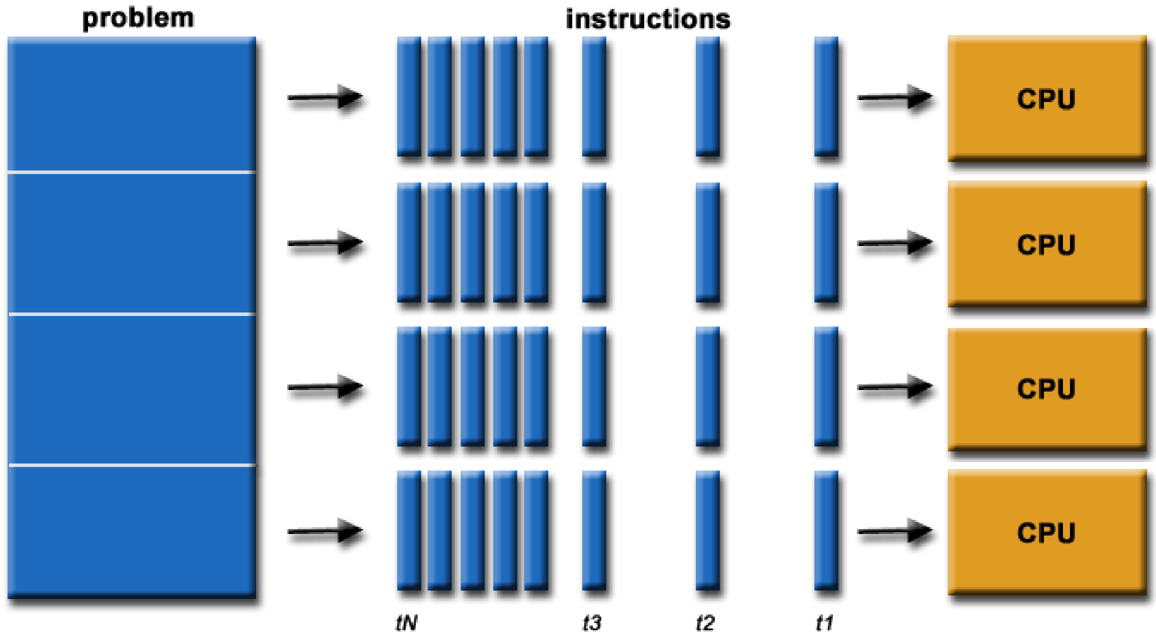
The computing resources can include:

- A single computer with multiple processors;
- An arbitrary number of computers connected by a network;
- A combination of both.

The computational problem usually demonstrates characteristics such as the ability to be:

- Broken apart into discrete pieces of work that can be solved simultaneously;
- Execute multiple program instructions at any moment in time;
- Solved in less time with multiple compute resources than with a single compute resource.

Image showing Serial Computation



Traditionally, parallel computing has been considered to be "the high end of computing" and has been motivated by numerical simulations of complex systems and "Grand Challenge Problems" such as:
weather and climate , chemical and nuclear reactions , biological, human genome , geological, seismic activity, electronic circuits , manufacturing processes, etc.

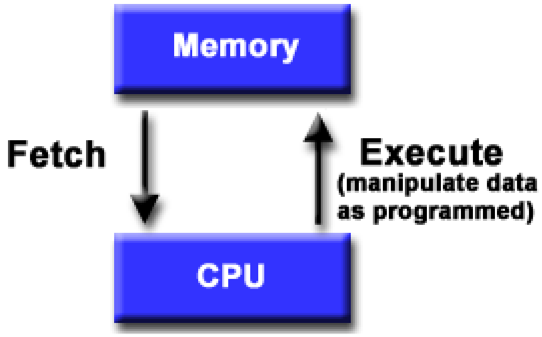
Today, commercial applications are providing an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. Example applications include:
parallel databases, data mining , oil exploration , web search engines, web based business services , computer-aided diagnosis in medicine , management of national and multi-national corporations , advanced graphics and virtual reality, particularly in the entertainment industry , networked video and multi-media technologies , etc.

Ultimately, parallel computing is an attempt to maximize the utilization of time.

VON NEUMANN ARCHITECTURE

For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer. A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.

Von Neumann Architecture (Stored Program Concept)



Basic design:

- Memory is used to store both program and data instructions
- Program instructions are coded data which tell the computer to do something
- Data is simply information to be used by the program
- A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then sequentially performs them.

ARCHITECTURAL CLASSIFICATION SCHEMES

Flynn's Taxonomy to classify Computers

One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy. Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of Instruction and Data. Each of these dimensions can have only one of two possible states: Single or Multiple.

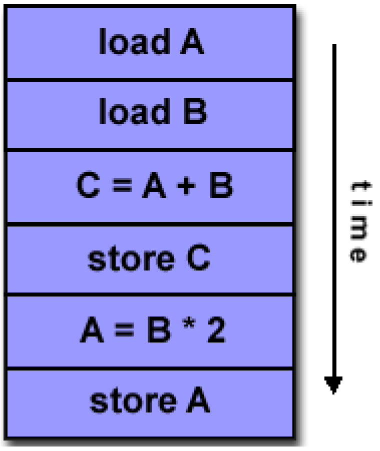
The matrix below defines the 4 possible classifications according to Flynn.

Flynn's Taxonomy of Multi-Processor systems	
SISD **Single Instruction, Single Data**	***SIMD*** **Single Instruction, Multiple Data**
MISD **Multiple Instruction, Single Data**	***MIMD*** **Multiple Instruction, Multiple Data**

Single Instruction, Single Data (SISD):

- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes

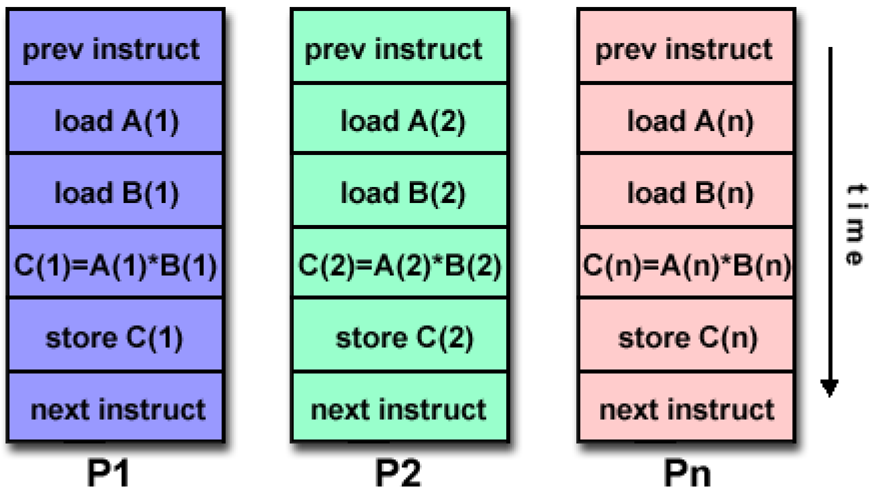
Single Instruction, Single Data (SISD)



Single Instruction, Multiple Data (SIMD):

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
  - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

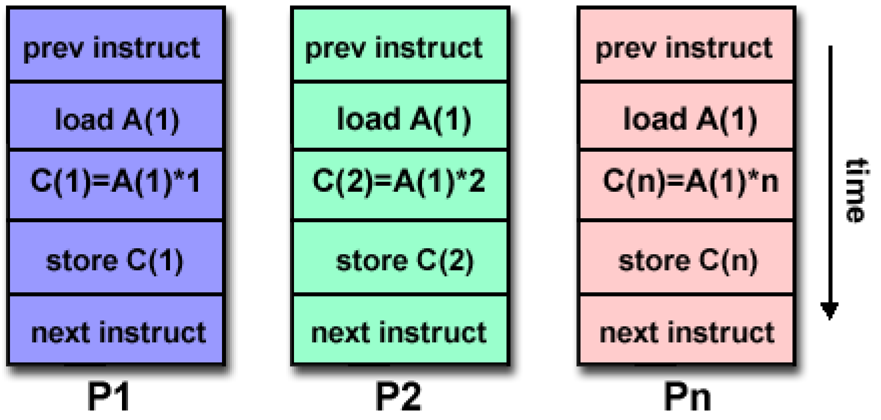
Single Instruction, Multiple Data (SIMD)



Multiple Instruction, Single Data (MISD):

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.

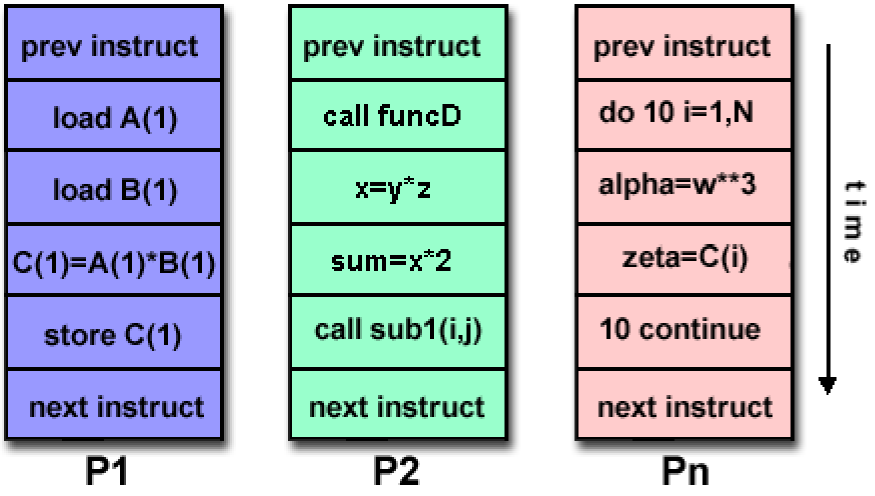
Multiple Instruction, Single Data (MISD)



Multiple Instruction, Multiple Data (MIMD):

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer "grids".

Multiple Instruction, Multiple Data (MIMD)



AMDAHL'S LAW

Amdahl's law is a model for the relationship between the expected speedup of parallelized implementations of an algorithm relative to the serial algorithm, under the assumption that the problem size remains the same when parallelized.

For example, if for a given problem size a parallelized implementation of an algorithm can run 12% of the algorithm's operations arbitrarily quickly (while the remaining 88% of the operations are not parallelizable), Amdahl's law states that the maximum speedup of the parallelized version is 1/(1 – 0.12) = 1.136 times as fast as the non-parallelized implementation.

More technically, the law is concerned with the speedup achievable from an improvement to a computation that affects a proportion P of that computation where the improvement has a speedup of S. (For example, if an improvement can speed up 30% of the computation, P will be 0.3; if the improvement makes the portion affected twice as fast, S will be 2.) Amdahl's law states that the overall speedup of applying the improvement will be:

Amdahl's law : overall speedup

$$\frac{1}{(1-P)+\frac{P}{S}}.$$

Speedup:
Speedup is defined as the time taken by a program to execute in serial (with one processor) divided by the time taken to execute in parallel (with many processors). The formula for speedup is:

$$S = \frac{T_1}{T_j}$$

Where T_j is the time taken to execute the program using ‘j’ number of processors. Speedup also indicates the efficiency of multi processor systems as compared to uni-processor systems.

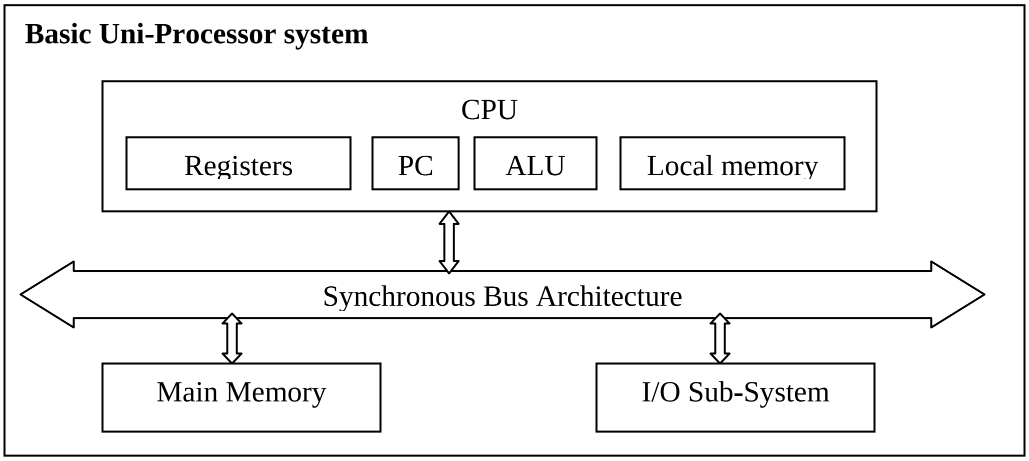
Amdahl's law is used to find the maximum expected improvement to an overall system when only a part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speedup using multiple processors.

The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program. For example, if a program needs 20 hours using a single processor core, and a particular portion of 1 hour cannot be parallelized, while the remaining portion of 19 hours (95%) can be parallelized, then regardless of how many processors we devote to a parallelized the execution of this program, the minimum execution time cannot be less than that critical 1 hour. Hence the speedup is limited up to 20×.

MOORE’S LAW :
According to Moore’s Law, the number of transistors on a chip roughly doubles every two years.

PARALLELISM IN UNI-PROCESSOR SYSTEMS
Basic Uni-Processor system consists of Four major components:

1. One Main Memory
2. Central Processing Unit (CPU).
    a. One set of General Purpose Registers along with Program Counter.
    b. One Set of Special CPU status registers for storing the current state of CPU and program under execution.
    c. One Arithmetic and Logic Unit (ALU).
    d. One Local Cache Memory.
3. One Input-Output Subsystem (I/O)
4. One Synchronous bus architecture for communication between I/O devices, Memory and CPU.


Basic Uni-Processor system

Parallelism can be achieved in Uni-Processor systems using two main methods:

1. Hardware method.
2. Software method.

Hardware method of parallelism:

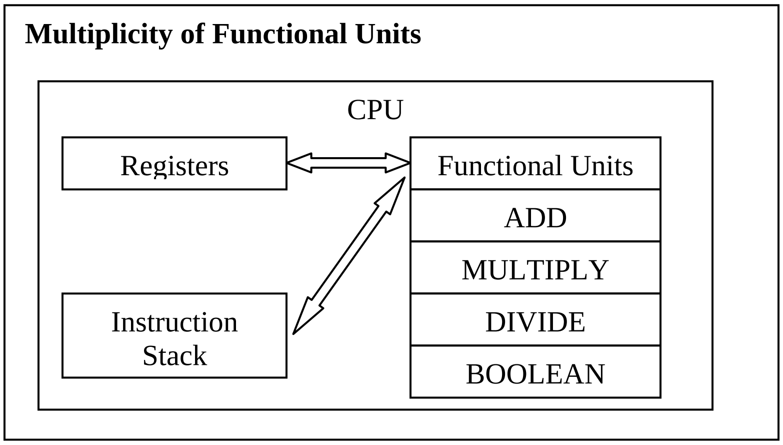
1. Multiplicity of Functional Units.
2. Pipelining within the CPU.
3. Overlapped CPU and I/O operations.

Software method of parallelism:

1. Multiprogramming
2. Time sharing.

Hardware method of parallelism:

1. **Multiplicity of Functional Units:**
    a. Many operations of CPU can be distributed among specialized Functional Units which can operate in parallel.



2. **Pipelining within the CPU:**
    a. In addition with multiple Functional Units various phases of Instruction Execution such as Instruction Fetch, Instruction Decode, Instruction Execution and Result Storage can be Pipelined.
    b. The instructions are prefetched and related data is buffered so that the instructions can be overlapped through pipes (memory queues).
3. **Overlapped CPU and I/O operations:**
    a. The I/O operations are assigned to special I/O controllers, channels and I/O processors.
    b. So the processor can work parallel when I/O operations are happening.

Software method of parallelism:

The main idea is to perform resource sharing between many user programs so that every program can execute quicker. The programs are not allowed to finish at a stretch but are executed in chunks. This gives an impression that all programs are executing parallel. The Operating System is the only way using which Software level parallelism can be achieved.

1. **Multiprogramming:**
    a. Usually every process or program consists of either CPU-bound (computation intensive) instructions or I/O bound (Input Output intensive) instructions or a combination of both.
    b. When a system has many processes then, while CPU is busy with some CPU-bound process then at the same time a waiting I/O bound process can be allocated I/O resources for its execution. This is called Multiprogramming.
    c. Here processes do not have to wait for each other to complete and hence can execute simultaneously (parallel).
2. **Time sharing**
    a. In Multiprogramming systems a process which takes a very long time in the CPU or I/O processing can drastically reduce the system performance, as other processes have to wait in queue.
    b. This problem gets solved in Time-sharing systems where we assign Time-Slices to every process and a preemptive (pausing) strategy is used to automatically pause a process when its allocated time span is over.
    c. Here every process is assigned a specific time slice to utilize the CPU resources. The preempted processes go into waiting state and when given a chance by the Scheduler are again assigned the CPU resources. This happens till all the processes are finished or the system is shutdown.

Mathematically the analysis of parallel processing systems can be represented as:
Let there be

k – stages in the system
n - number of major tasks to be processed
t - time needed to execute every subtask in every major task

then,

Total time for non-pipelined system

$$\mathbf{T_{np} = n * k * t}$$

Total time for pipelined system

$$\mathbf{T_p = (n + (k – 1)) * t}$$

In actual case there is unequal time t delay between stages, so

$$\mathbf{t_{act} = t_{max} + t_{latch}}$$

where,

t_{max} is the maximum time required for the slowest task in the system.
t_{latch} is the time delay introduced due to the latches in the system.

frequency at which tasks that can be pushed into the system without the possibility of any collision is,

$$\mathbf{f = 1 / t_{act}}$$