HIGH PERFORMANCE COMPUTING

A single processor computer consists of:

- Input Unit: which accepts the list of instructions to solve a problem (a program) and the relevant data
- Memory/Storage Unit: where program, data and intermediate results are stored
- Processing Unit: which interprets and executes the instructions
- Output Unit: which displays or prints the results

This structure of a computer is known as Von Neumann Architecture.
In this,

- A program is first stored in the memory
- The PE retrieves one instruction at a time and executes it.

Hence the operation is sequential.

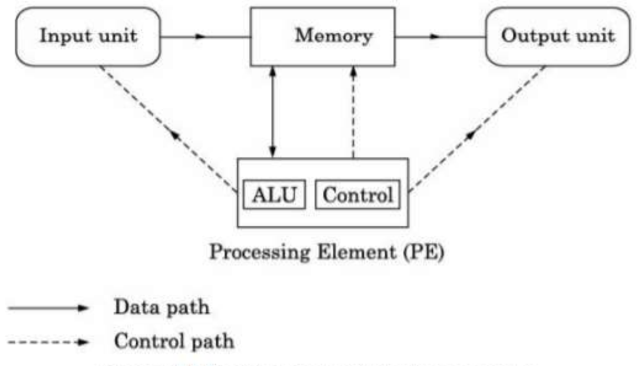


Figure 1.1 Von Neumann architecture computer.

SERIAL COMPUTING

- Program broken down into discrete instructions
- Instruction executed one by one
- One instruction at a time

PROBLEM WITH SERIAL COMPUTING:
The speed of the sequential computer is thus limited by the speed at which a PE can retrieve instructions and data from the memory and the speed at which it can process the retrieved data.

To increase the speed of processing of data one may increase the speed of the PE by increasing the clock speed. The clock speed increased from a few hundred kHz in the 1970s to 3 GHz in 2005. It was difficult to increase the clock speed further as the chip was getting overheated.

Does that mean we cannot achieve greater performance if we are unable to develop more efficient processors?
NO, WE CAN
The number of transistors which could be integrated in a chip could, however, be doubled every two years. Thus, processor designers placed many processing “cores” inside the

processor chip to increase its effective throughput. The processor retrieves a sequence of instructions from the main memory and stores them in an on-chip memory. The “cores” can then cooperate to execute these instructions in parallel. HENCE CAME PARALLELISM.

HENCE, LIMITATIONS OF SERIAL COMPUTING

- Waits for input output events: slow operation: weak performance
- Single point of failure
- Complex applications: real time simulations: gaming, imaging, database applications
- Limited RAM

PARALLEL COMPUTING

- Use multiple cores to perform several operations at once
- Much faster than sequential for doing repetitive calculations on vast amount of data
- Single computation problem divided into pieces for simultaneous work: Parallel Processing
  Multiple computational units for multiple unrelated problems: Multiprocessing

ADVANTAGES OF PARALLEL COMPUTING:

- Faster
- Complex computational applications
- Power consumption
- Reliable
- Better quality of solution. When arithmetic operations are distributed to many computers, each one does a smaller number of arithmetic operations. Thus, rounding errors are lower when parallel computers are used.
- Better algorithms. The availability of many computers that can work simultaneously leads to different algorithms which are not relevant for purely sequential computers. It is possible to explore different facets of a solution simultaneously using several processors and these give better insight into solutions of several physical problems.
- Better storage distribution. Certain types of parallel computing systems provide much larger storage which is distributed. Access to the storage is faster on each computer. This feature is of special interest in many applications such as information retrieval and computer aided design.

CHALLENGES OF PARALLEL COMPUTING:

- Synchronization and communication between the processors and sub-processes
- Algorithms need to be designed so as to handle parallelism, must have low coupling and high cohesion, and skilled programmers hence needed

BUT HOW WILL PARALLELISM WORK?
(Different ways in which a job can be solved using parallelism)

Method 1: TEMPORAL PARALLELISM
This method breaks up a job into a set of tasks to be executed overlapped in time. (different subtasks simultaneous at one point of time)
This method of parallel processing is appropriate if:
1. The jobs to be carried out are identical for all data
2. A job can be divided into many independent tasks

3. The time taken for each task is the same.
4. The time taken to send a job from one teacher to the next is negligible compared to the time needed to do a task.
5. The number of tasks is much smaller as compared to the total number of jobs to be done.

Calculations:
If n jobs, p time to complete a job, and each job into k subtasks
Assuming time taken for completing each task: p/k
Time taken to complete all jobs without parallelism: np
Time taken to complete all jobs with temporal parallelism: p + (n-1)*p/k
SpeedUp: without/with =

$$= \frac{np}{p(k + n - 1)/k} = \frac{k}{1 + \frac{k-1}{n}}$$

If n >> k then (k – 1)/n = 0 and the speedup is nearly equal to k

Problem with this method of parallelism:

1. Synchronization: to have identical times for doing each task
2. Bubbles in pipelines: some tasks absent in a job. For example, if there are some answer books with only 2 questions answered, two teachers will be forced to be idle during the time allocated to them to correct these answers.
3. Fault Tolerance: one task (stage) fails, whole pipeline is abrupt
4. Inter-task communication: should be small
5. Scalability: n>>k must hold

Advantage:
Efficient: each stage in the pipeline can be optimized to do a specific task well.

*It was the main technique used by vector supercomputers such as CRAY to attain their high speed.

Method 2: DATA PARALLELISM
In this method, the input data is divided into independent sets and processed simultaneously.

Calculations:
If n jobs, p time to complete a job, k teachers for data to be distributed between
Assuming time taken to distribute the data into k processors: kq (proportional to k)
Time taken to complete all jobs without parallelism: np
Time taken to complete all jobs with parallelism: kq + (n/k)*p
SpeedUp: without/with =

$$\frac{np}{kq + \frac{np}{k}} = \frac{knp}{k^2q + np} = \frac{k}{1 + (k^2q/np)}$$

If k^2q << np then the speedup is nearly equal to k, the number of teachers working independently. Observe that this will be true if the time to distribute the jobs is small. And the number of processes is also small. (because distribution time also increases)

Efficiency: actual speedup/maximum possible speedup = actual speedup/k

Observe that the speedup is not directly proportional to the number of teachers as the time to distribute jobs to teachers (which is an unavoidable overhead) increases as the number of teachers is increased

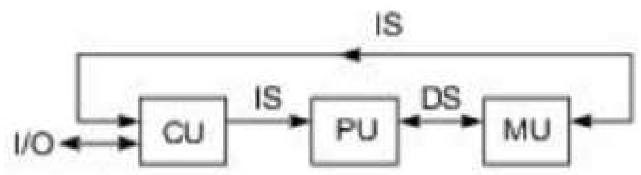
Advantages:
1. Synchronization: Not required, each teacher can check on his/her own pace
2. Bubbles: absent, no need to sit idle
3. Fault Tolerance: More fault tolerant, doesn't affect the whole work
4. Inter-task communication: not required (hence no delay), processors are independent

Disadvantages:
1. No specialization, each teacher must be able to correct all answers
2. Division into mutually independent jobs with same time
3. Distribution time must be small, k should be small
4. The assignment of jobs to each teacher is pre-decided. This is called a static assignment. Thus, if a teacher is slow then the completion time of the total job will be slowed down. If another teacher gets many blank answer books he will complete his work early and will thereafter be idle. Thus, a static assignment of jobs is not efficient.

Method 3: MIXED PARALLELISM
Combination of both temporal and data. Called Parallel Pipeline Processing.
More than one pipeline is formed and data distributed to each.
The method is effective only if the number of jobs given to each pipeline is much larger than the number of stages in the pipeline.
Multiple pipeline processing was used in supercomputers such as Cray and NEC-SX as this method is very efficient for numerical computing in which a number of long vectors and large matrices are used as data and could be processed simultaneously

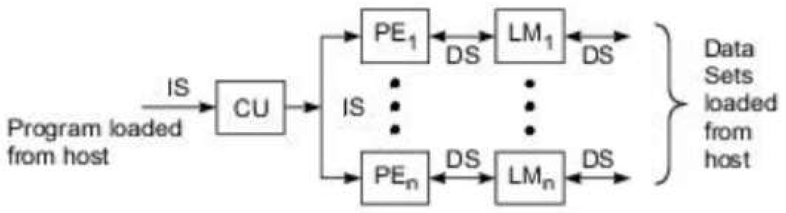
CLASSIFICATION OF PARALLEL COMPUTERS
Basis:
1. HOW DO INSTRUCTIONS AND DATA FLOW IN THE SYSTEM (How instructions process data): Flynn's Classification
2. WHAT IS COUPLING BETWEEN THE PEs
3. HOW DO PEs ACCESS MEMORY
4. WHAT IS THE QUANTUM OF WORK DONE BY A PE BEFORE IT COMMUNICATES WITH ANOTHER PE (What's the grain size of computation)

1. FLYNN'S CLASSIFICATION
   SISD: Computer with single processor, a single stream of instructions and a single stream of data are accessed by the PE from the main memory, processed and the results stored back in the main memory

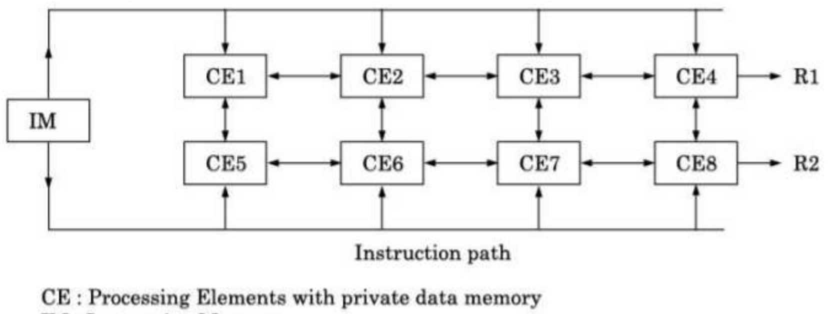


(a) SISD uniprocessor architecture

SIMD: There is no explicit communication among processors. However, data paths between nearest neighbours are used in some structures. SIMD computers have also been built as a grid with communication between neare
st neighbours. All processors in this structure are given identical instructions to execute and they execute them in a lock-step-fashion (simultaneously) using data in their memory.
All PEs work synchronously controlled by a single stream of instructions. An instruction may be broadcast to all PEs and they can process data items fed to them using this instruction. If instead of a single instruction, the PEs use identical programs to process different data streams, such a parallel computer is called a Single Program Multiple Data (SPMD) Computer



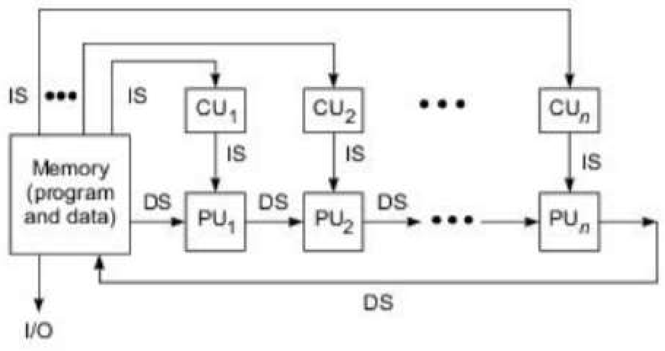
(b) SIMD architecture (with distributed memory)



CE : Processing Elements with private data memory
IM : Instruction Memory

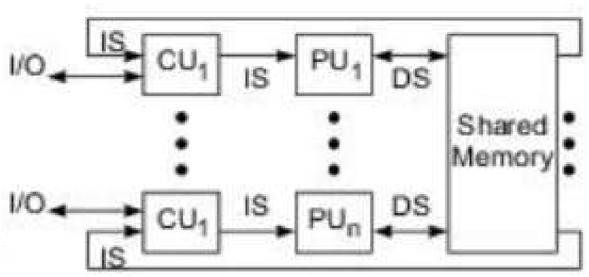
Figure 4.3 Regular structure of SIMD computer with data flowing from neighbours.

MISD: Different PEs run different programs on the same data



(d) MISD architecture (the systolic array)

MIMD:



DEPENDENCES:
The ability to execute several program segments in parallel requires each segment to be independent of the other segments.
We use a dependence graph to describe the relations. The nodes of a dependence graph correspond to the program statements [instructions], and the directed edges with different labels show the ordered relations among the statements. The analysis of dependence graphs shows where opportunity exists for parallelization and vectorization.

Dependences are of various types:
DATA DEPENDENCE: Two statements have a data dependence if they cannot be executed simultaneously because of conflicting uses of the same variable. The ordering relationship between statements is indicated by data dependence.
Five types of data dependence:
Flow (RAW), Anti(WAR), Output(WAW), I/O(Access to same file), Unknown(LoI)

CONTROL DEPENDENCE:
RESOURCE DEPENDENCE:

PARALLEL ALGORITHMS

A parallel algorithm defines how a given problem can be solved on a given parallel computer, i.e., how the problem is divided into subproblems, how the processors communicate, and how the partial solutions are combined to produce the final result.

Parallel algorithms depend on the kind of parallel computer they are designed for. In order to simplify the design and analysis of parallel algorithms, parallel computers are represented by various abstract machine models.

Models:
RAM: Random Access Machine
Abstracts the sequential computer

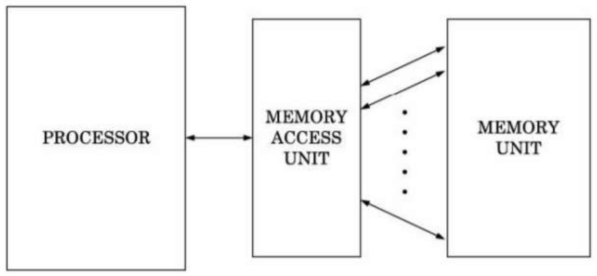


Figure 7.1 RAM model.

MAU is the address mapping unit from the logical address provided by the CPU to the physical address in main memory.

Any step of an algorithm designed for RAM Model consists of three basic phases:
Read (from memory and store in local register), Write, Execute (No role of memory)

PRAM: Parallel Random Access Machine
One of the popular models for designing parallel algorithms.

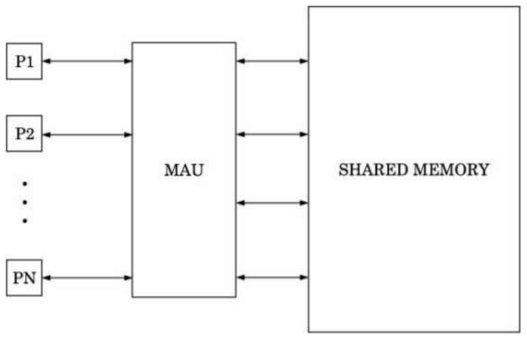


Figure 7.2 PRAM model.

(N identical processors)
Shared memory also functions as the communication medium for the processors.
A PRAM can be used to model both SIMD and MIMD machines.

Three basic phases for any step: Read, Write and Compute

In this model, one memory location can be accessed by more than one processor simultaneously. This is a problem. The PRAM model can be subdivided into four categories based on the way simultaneous memory accesses are handled:

- **Exclusive Read Exclusive Write (EREW)** PRAM: any memory access is exclusive, provides least amount of memory concurrency, hence weakest
- **Concurrent Read Exclusive Write (CREW)** PRAM: can concurrently read but write is exclusive, most commonly used
- **ERCW**: never used
- **CRCW**: most memory concurrency, most powerful

Concurrent Write: There are several protocols that are used to specify the value that is written to a memory in such a situation. They are:
- Priority CW: one with the highest assigned priority is assigned to write in case of conflict
- Common CW: allowed if same value to be written
- Combining CW: output of a function that maps those multiple values to a single value is written
- Arbitrary CW: any one of the processors may succeed in writing, and others fail