Page No.
Date : / /

Unit 1: Foundation

1.1: Parallel & Distributed Computing: the scene, the props, the players.

** A perspective:

① Parallel computing:

Parallel computing is also called parallel processing. There are multiple processors in parallel computing. Each of them performs the computations assigned to them. In other words, in parallel computing, multiple calculations are performed simultaneously.

In parallel computing, all processors may have access to a shared memory to exchange information between processors.

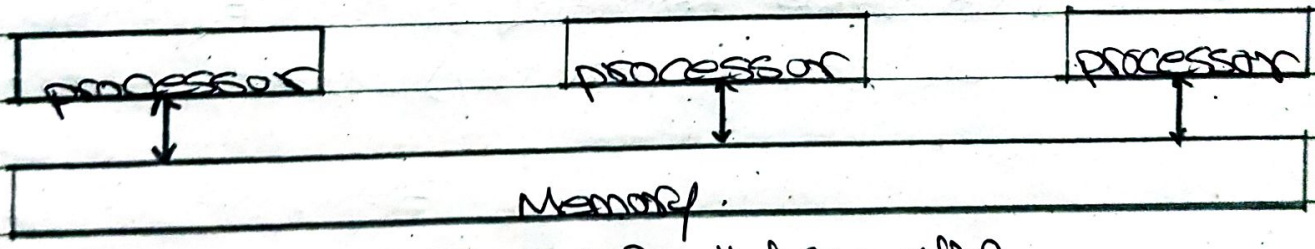


Fig of: Parallel computing.

② Distributed Computing:

Distributed computing divides a single tasks between multiple computers. All computers work together to achieve a common goal. Thus, they all work as a single entity. In distributed computing, each processor has its own private memory, where information is exchanged by passing message between processors (RPC, RMI).

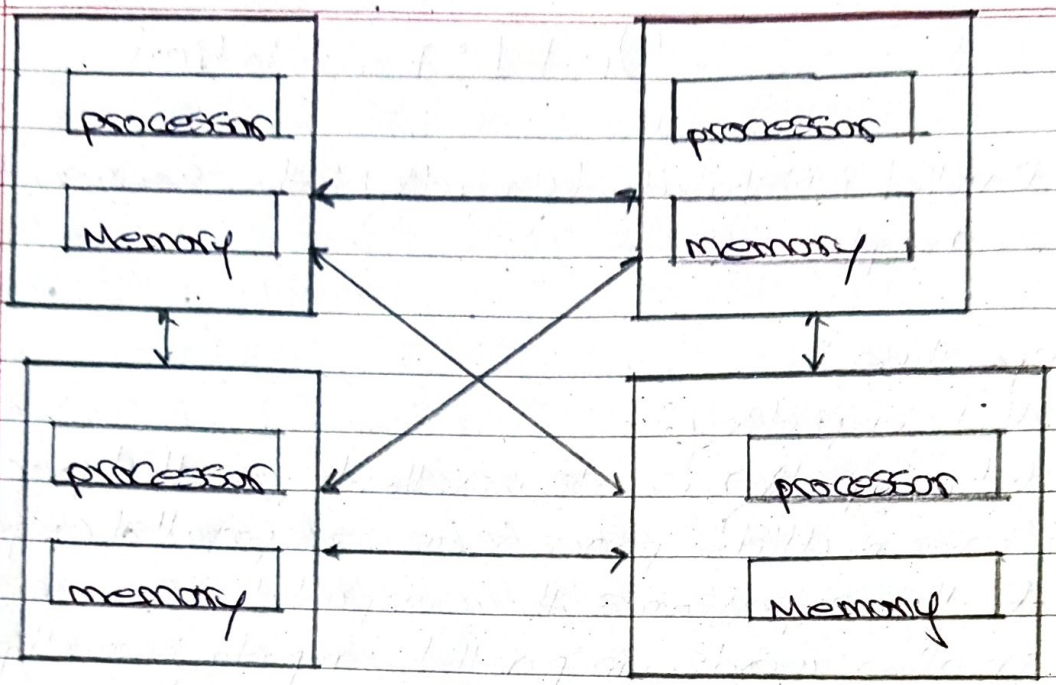


Fig of :- Distributed computing.

* Why do we need parallel computing?
- Serial computing is too slow,
- Need for more computing power (E.g:- Datamining, Search engine, Cryptography)

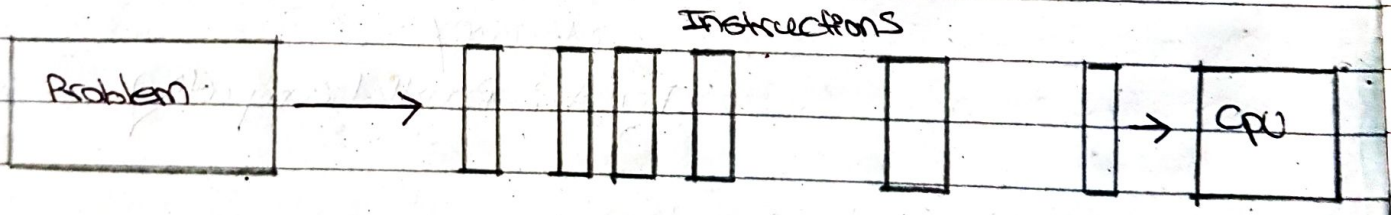


Fig of :- Traditional Approach.

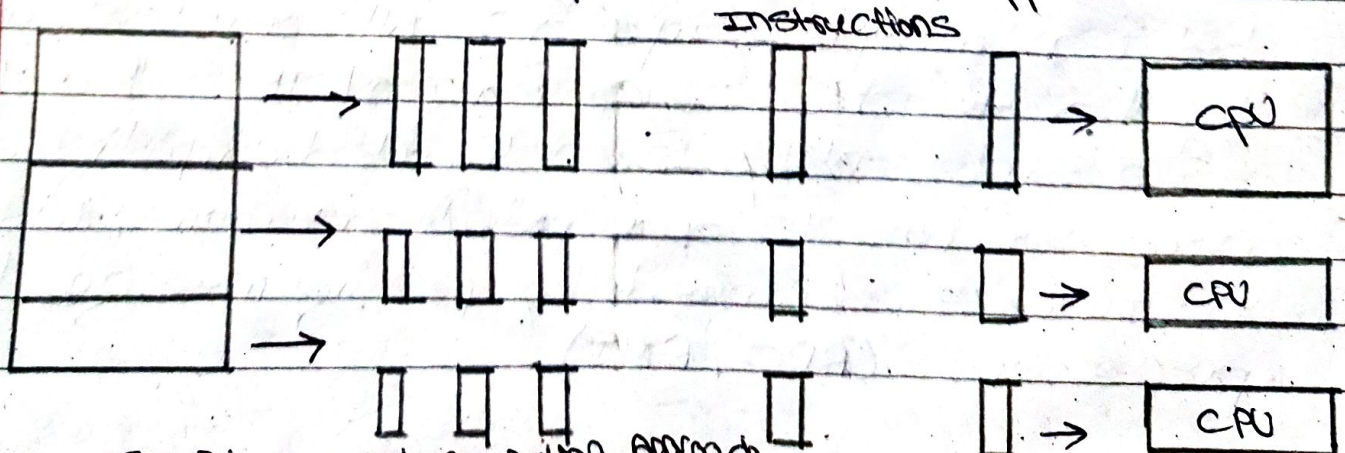


Fig of :- Parallel computing Approach.

Page No.
Date : / /

V.U.I

* Issues on shared Resources:

One of the issues faced while sharing the Resources by multiple processes is the result in inConsistent results also known as Race Condition.

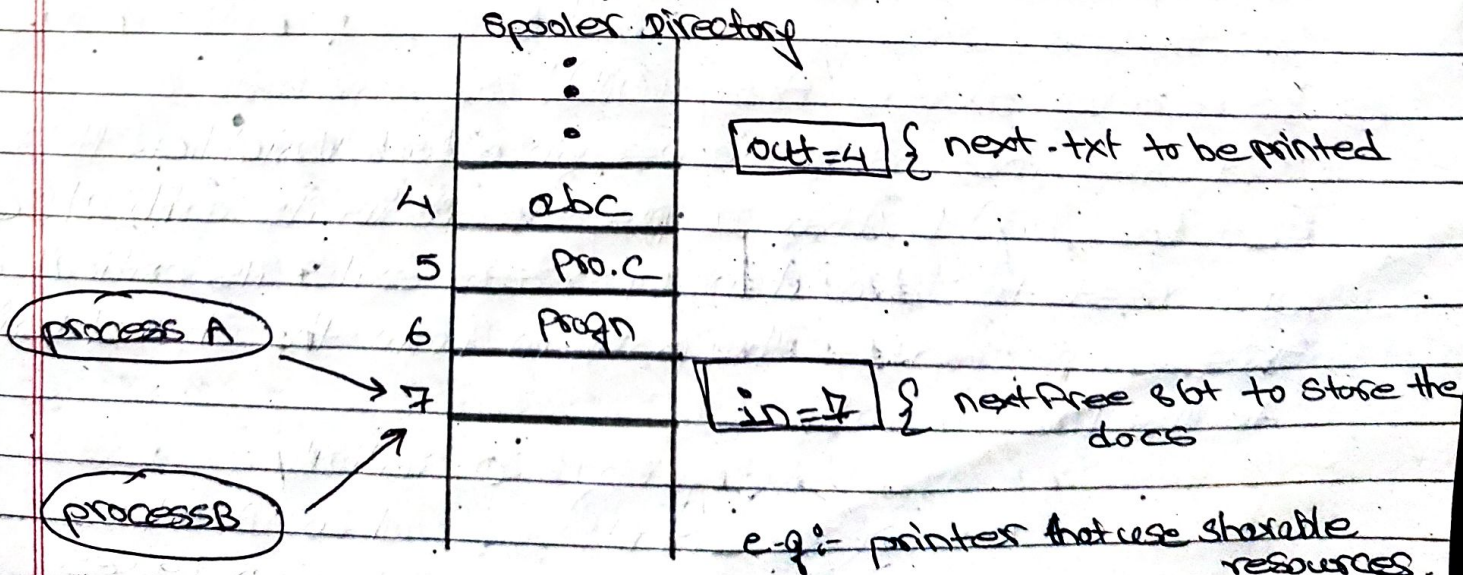
* Race condition:

Race condition is a situation where

i) The final output produced depends on the execution order of instructions of different processes.

ii) Several processes compete with each other.

At the time when more than one process is either executing the same code or accessing the same memory or shared resources, in that condition there is a possibility that the output is wrong so for that purpose all the processes are doing the race to say that their output is correct. As several processes access & process the manipulations over the same data concurrently, then outcomes depends on the particular order in which the access takes place commonly known as a race conditions.



e.g:- printer that use shareable resources.

Page No.
Date : / /

* How to avoid Race Condition?

To prevent the race conditions from occurring, we can lock shared variables, so that only one thread at a time has access to the shared variable i.e. Mutual Exclusion.

* Mutual Exclusion:

Mutual exclusion is a property of process synchronization which states that "no two processes can exist in the critical section at any given point of time".

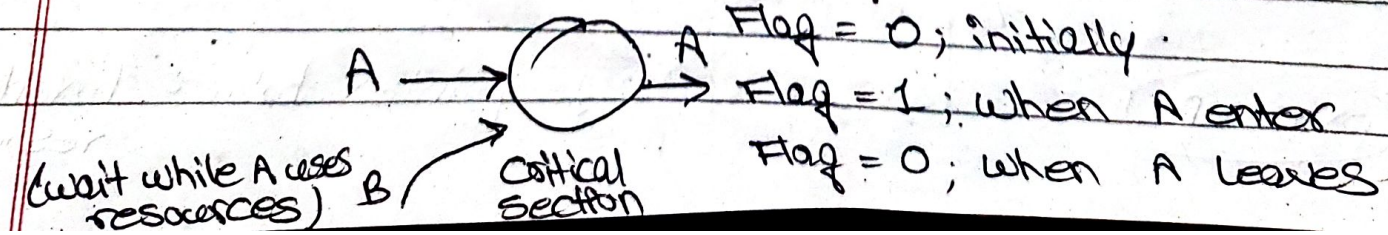
* Implementing Mutual Exclusion with Busy Waiting:

① Disabling Interrupts (Busy wait):

Perhaps the most obvious way of achieving mutual exclusion is to allow a process to disable interrupt before it enters its critical section and then enable interrupts after it leaves its critical section. By disabling interrupts the CPU will be unable to switch processes, which guarantees that the process can use the shared variables without another process accessing it.

② Locked variables (Busy wait):

Another method is to assign a lock variable. This is set to (say) 1 when a process is in its critical section and reset to zero when a process exits its critical section. This simply moves the problem from the shared variable to the lock variable.



Page No.
Date : / /

(3) Strict Alternation:

```
process 0
while (TRUE) {
  while (turn != 0); // wait
  criticalSection();
  turn = 1;
  noncritical_Section(); }
```

```
process 1
while (TRUE) {
  while (turn != 1); // wait
  criticalSection();
  turn = 0;
  noncritical_section(); }
```

These code fragments offers a solution to the Mutual exclusion problem.
Assume the variable turn is initially set to zero. Process 0 is allowed to run. It finds that turn is zero & is allowed to enter its critical region. If process 1 tries to run, it will also find that turn is zero & will have to wait until turn become 1. When process 0 exit its critical section it sets to turn to 1, which allows process 1 to enter its critical section. If process 0 tries to enter its critical region again it will be blocked as turn is no longer zero.

(4) Sleep and wakeup (Busy wait Solution):

When a process is not permitted to access its critical section, it uses a system call known as sleep, which causes that process to block. The process will not be scheduled to run again, until another process uses the wakeup system call. In most cases, wakeup is called by a process when it leaves its critical section if any other processes have blocked.

Page No.
Date : / /

V.V.I

* Parallel Processing Paradigms:

Parallel systems are more difficult to program than computers with a single processor because the architecture of parallel computers varies accordingly and the processes of multiple CPUs must be coordinated & synchronized.

The difficult part of parallel processing are CPUs. Based on the number of instruction and data streams that can be processed simultaneously, computing systems are classified into four major categories:

* Flynn's Taxonomy:

1. Single-Instruction, Single-data (SISD),
2. Single Instruction, Multiple-data (SIMD),
3. Multiple Instruction, Multiple data (MIMD),
4. Multiple Instruction, Single data (MISD).

1. Single Instruction, Single data (SISD):

An SISD computing system is a uniprocessor machine which is capable of executing a single instruction operating on a single data stream. In SISD, machine instructions are processed in a sequential manner & computers adopting this model are popularly called sequential computers. All the instructions & data to be processed have to be stored in primary memory.

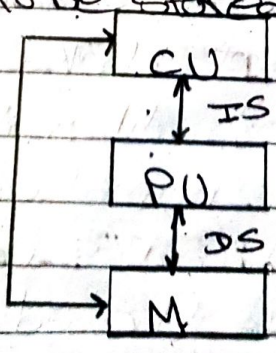


Fig 1: SISD

Page No.
Date : / /

② Single Instruction, multiple Data (SIMD):

An SIMD system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams.

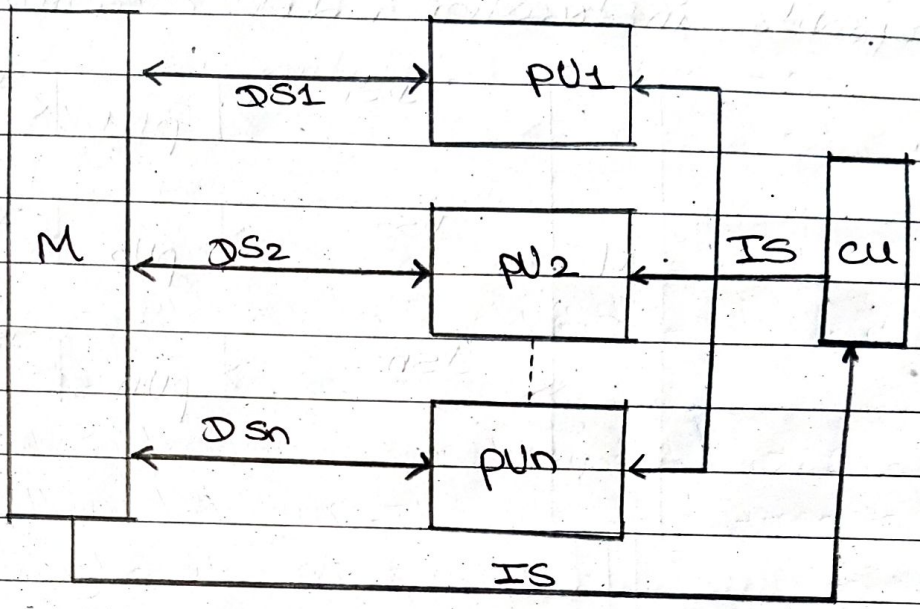


Fig ② : SIMD

③ Multiple Instruction, Single Data (MISD):

An MISD ~~inst~~ computing system is a multiprocessor machine capable of executing multiple or different instructi on different processing Elements (PEs) but all of them operating on the same dataset.

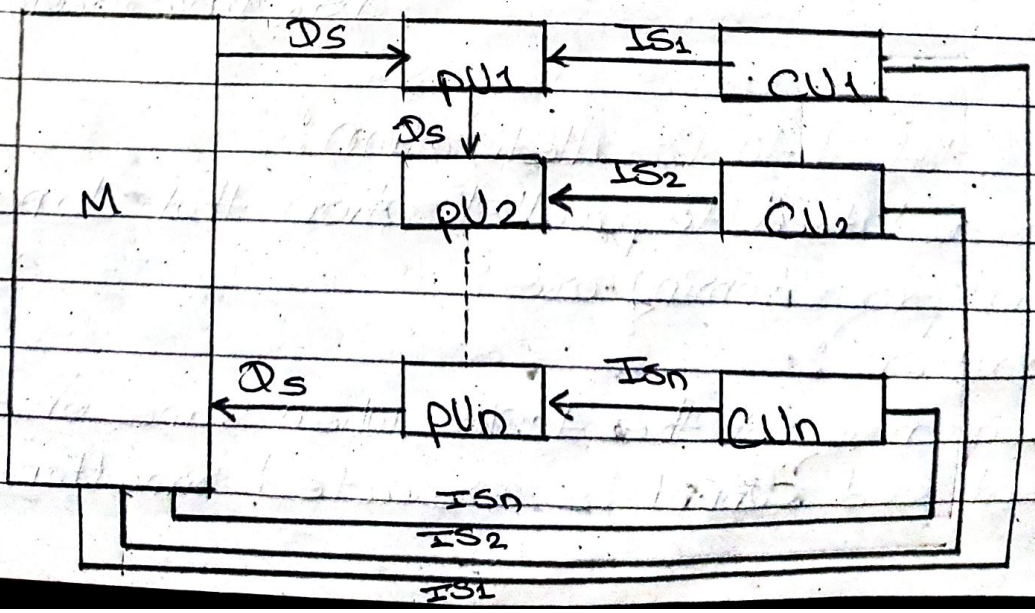


Fig ③ : MISD

④ Multiple Instruction, Multiple Data (MIMD):

An MIMD System is a multiprocessor machine which is capable of executing multiple instructions on multiple data sets. Each processing Elements in the MIMD model has seperate instruction & data streams and work Asynchronously

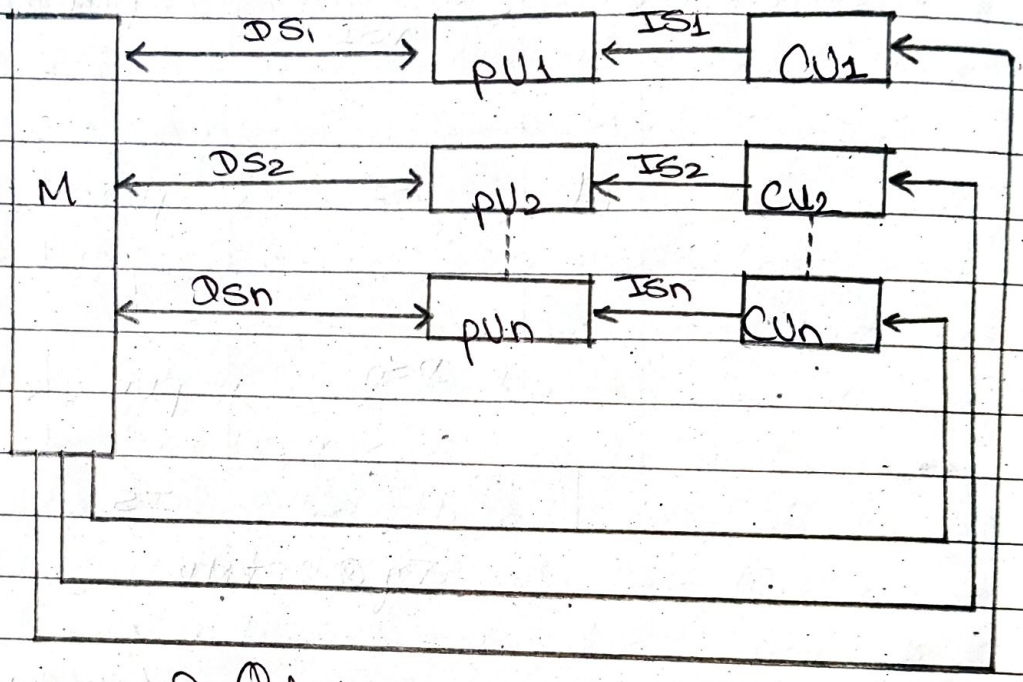


Fig ④ : MIMD

M: Memory
CU: Control unit
PU: Processing unit
IS: Instruction unit
DS: Data stream.

* ISSUES related to Parallelization:

Issues related to parallelization that donot arise in sequential programming are:

① TASK Allocation:

- Proper sequencing of the tasks when some of them are dependent and cannot be executed simultaneously

- Load balancing or scheduling.

② Communication Time:
- Communication time between the processors
- Serious when the number of processors increases to hundreds or thousands.

* Cost VS Performance Evaluation:

① Execution Time:
- Time elapsed from the moment the algorithm starts to the moment it terminates.
- In case of multiple processors, it is the time elapse between the time that the first processor begins & the last one terminates.

② Speed up:

$$\text{Speed} = \frac{\text{Running time of the best available Sequential algorithm}}{\text{Running time of the parallel algorithm}}$$

- It is not always possible to decompose the task.
- So maximum speed of N processor system in executing algorithm is

$$S_N \leq \frac{1}{F + \frac{(1-F)}{N}} \leq \frac{1}{F}$$ where, F → fraction of task or computation that must be done sequentially.

- $S_{max} = 1$ where $F = 1$ (no speed up that is every task must be done sequentially)
- $S_{max} = 0$, where $F = 0$ (full speedup that is all computations must be parallel)

Page No.
Date : / /

VVI

* Performance Evaluation:

① Communication Penalty:

$$C_{pi} = \frac{E_i}{C_i}$$

where, E_i → total execution time spent by P_i
C_i → total time used for communication by P_i

Page No.
Date : / /

1.2 Semantics of Concurrent Programming:

- Concurrent Programming refers activation of more than one instruction at a time.
- In programming language theory, semantics is the field of concerned with the mathmatical study of the meaning of the programming language.
- Syntatic Structure VS semantic Structure.

① English Language:

- Subject + verb + object (Grammatical pattern but language dependent)

*example:
- I eat rice. (syntactically & semantically correct)
- I rice eat. (syntactically wrong)
- I eat car. (Syntactically correct but Semantically wrong)

② Programming Language:

- CFG (Context Free Grammar) consisting of a finite set of grammar rules in a quadruple (N, T, P, S) where
  - N is a set of non-terminal symbols,
  - T is a set of terminals where $N \cap T = NULL$
  - P is a set of rules; $P: N \rightarrow (N \cup T)^*$
  - S is a start symbol.

- Declaration → Type variables
- Type → int

* example: int a, a = 2;
  int a, a = 3.1 (semantically wrong)

Page No.
Date : / /

* Models of Concurrent Programming: V.V.I

We have identified a number of attributes of concurrency models, each model can be defined by the values that it takes for these attributes.

① Level of Granularity:

- Consider two agents (eg bank teller) who are accessing a centralized database simultaneously.
- Observe their activities at

① Transaction level:

We find that their activities are concurrent, i.e they are performing their transaction at same time.

② CPU cycle level:

We find that their activities are sequential, because the CPU takes turns Serving their queries.

The activities of these agents can be considered as concurrent or sequential depending on the level of granularity at which we observe them.

② Sharing the clock:

When several processes operate concurrently and must eventually interact, (eg to exchange information), it is important to determine whether they share the same clock.
eg: P_1, P_2 is dependent on P_1.

③ Sharing the Memory:

When several process operate concurrently and must eventually interact, it is important to determine whether they share some memory space. i.e, Access to share

memory may have to be mutually exclusive.

④ Pattern of Interaction:

There exist two forms of interactions between processes;

① Synchronization:

Synchronization defines a chronological order between events taking place within different processes. There exists two patterns of synchronization

• Mutual exclusion:

It is defined by an encapsulated sequence of actions whose execution is indivisible; once a process starts executing this sequence, it may not be intercepted until the sequence is completed.

• Mutual Admission:

It is defined by an encapsulated sequence of actions which must be executed by two processes simultaneously if one process is ready & the other is not; it must wait until both are ready to give their undivided attention

② Communication:

Communication defines a transfer of information from one process to another. There exists two modes of communication.

• Synchronous Communication:

All parties involved in the communication are present at the same time. e.g:- phone calling.

• Asynchronous Communication:

All parties doesn't requires to be present at the same

time during communication. eg: E-mail).

** Semantic Definitions: V.V.I

A programming language is defined by two features its syntax, which defines the set of legal sentences in the language; and its semantics, which assigns meaning to legal sentences of the language. Three broad techniques or categories are known for the definition of programming language semantics:

1. Axiomatic Semantic definition, **
2. Operational Semantic definition and **
3. Denotational Semantic definition **

(1) Axiomatic Semantic definition:

- Defines the meaning of language construct by making statements in the form of axioms or inference rules.
- Based on the Hoare's program (Hoare Triple), which is in the form, :

  $P\{S\}Q$, where P → Pre-condition
  Q → Post-condition
  S → Statement
- If S statement is executed in state where P holds, then it terminates and Q holds.
- Pre condition describes the states of the program before execution (like Input)
- Post Condition defines the states after the execution (like output)
- Here P and Q are predicates that holds Boolean value either True or False.

Page No.
Date : / /

example:

① $\{x=5\}$ $x = x+1$ $\{x=6\}$

P S Q

True After Execution Must be true

② $\{i=3 \text{ AND } K=4\}$ $i = i+K$ $\{i=7 \text{ AND } K=4\}$

③ $\{x \geq 0\}$ while $(x \neq 0)$ do $x = x-1$ $\{x=0\}$

④ $x = x+1$ $\{x \leq N\}$

$\{x+1 \leq N\}$

$\{x \leq N-1\}$

$\therefore$ P ∧ S terminates → Q (after Execution)

* Partial and Total Correctness:

① Partial Correctness:

- A program is partially correct with respect to pre condition and post condition, If the program is started with the values that makes the preCondition true, and the resulting value make the post condition true, when the program halts (if ever).
- i.e. If the answer is returned, it will be correct.

② Total Correctness:

- A program is total Correctness if the program terminates when started with values satisfying the pre conditions, the program is called totally correct.
- i.e. It requires that the algorithm terminates.

Page No.
Date : / /

① Sequential Statement rule:

The semantics of sequence statements is defined by means of the following rules:

has the form:

$$\frac{H_1, H_2, \ldots, H_n}{H}$$

i.e., if $H_1, H_2, \ldots, H_n$ have been verified then H is also valid.

e.g.: • $\frac{P \to Q, P}{Q}$

• $\frac{\{p\}\ S_1\ \{Q\}, \{Q\}\ S_2\ \{R\}}{\therefore \{P\}\ S_1, S_2\ \{R\}}$

• $\{x>1\}\ x = x+1\ \{x>2\}$
$\{x>2\}\ x = x+1\ \{x>3\}$

$\{x>1\}\ x = x+1, x = x+1\ \{x>3\}$

• $\{P\}\ S_1\ \{int\}$
$\{int\}\ S_2\ \{Q\}$

$\{p\}\ S_1, S_2\ \{Q\}$

To prove the conclusion of this rule, it suffices to find predicate int such that both premises hold.

Page No.
Date : / /

② Alternation Statement Rule:

The Axiomatic semantics of alternation statement is defined by means of following inference rule:

① IF-THEN:

$\{P$ and $B\}$ S $\{Q\}$

$\{P$ and NOT $B\} \supset$ (Logical implies) $\{Q\}$

$\therefore \{p\}$ IF B THEN S END IF $\{Q\}$

i.e p IF (con) $\{S1\}$ else S2-Q

② IF-ELSE:

$\{p$ and $B\}$ S_1 $\{Q\}$

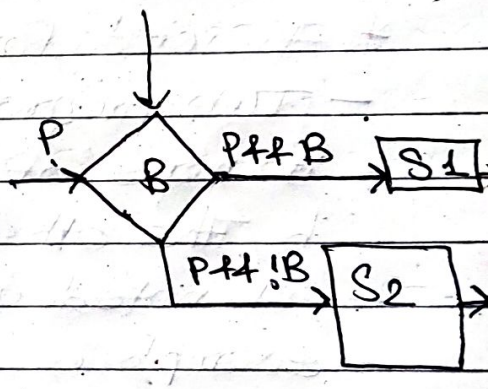
$\{P$ and NOT $B\}$ S_2 $\{Q\}$

$\therefore \{P\}$ IF B THEN S1 ELSE S2 END IF $\{Q\}$

$\{p \wedge t\}$ S $\{q\}$

$\{p \wedge \neg t\}$ T $\{q\}$

$\therefore \{p\}$ if t then S else T $\{q\}$



To establish the conclusion of this rule, it suffices to prove that the premises hold about statement S & T; this rules defines the meaning of an alternation statement.

Page No.
Date : / /

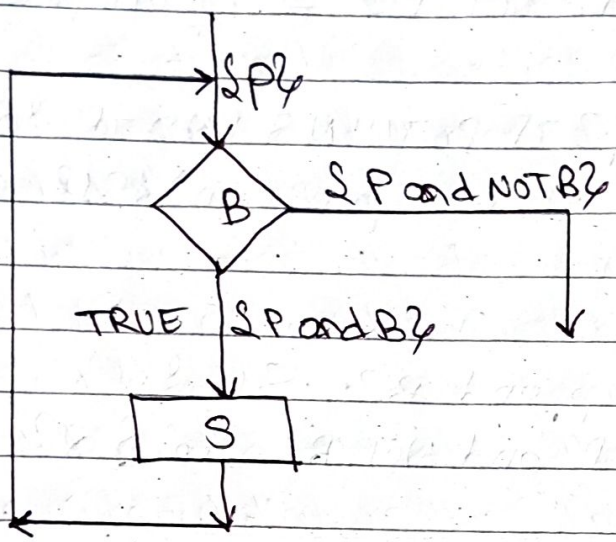
③ Loop:

- WHILE

• $\{p\}$ and $B\}$ S $\{p\}$

$\therefore \{p\}$ while B do S END while $\{p$ and NOT $B\}$

i.e,



* Disjoint Parallel program:

- Two programs S1 and S2 are said to be disjoint if none of them can change the variables accessed by the other.
- Denoted as $[S_1 || S_2]$.
- example:
  - $S_1 : x = z$
  - $S_2 : y = z$

But not → $S_1 : x = y + 1$, $S_2 : x = z$

- Semantics of this statement can be denoted as:

$\{p1\}$ S_1 $\{Q1\}$

$\{P2\}$ S_2 $\{Q2\}$

$\therefore \{p1$ and $P2\}$ $[S_1 || S_2]$ $\{Q1$ and $Q2\}$

Page No.
Date : / /

* Awaits Then Rule:

- Awaits T then B, where T is a condition and B is a block of code.
- IF condition T holds then block B is executed else the process is suspended until T becomes True.

example: The bank has an automated program that sends an emails if the balance of customer is less than 1000... here B is the block of code that sends the emails and has to await until the balance < 1000

- Semantically, it can be denoted as:

$$\frac{\{P \text{ and } T\}\ B\ \{Q\}}{\therefore \{P\}\ \text{Await } T \text{ then } B\ \{Q\}}$$

Page No.
Date : / /

② Operational Semantics:

- While axiomatic semantics capture the meaning of programming language constructs by focusing on the effect of these constructs on the program state, operational semantics focuses on how the state of the program is affected.

  In operational semantics, each statement of language is defined by describing the process that a computer goes through to execute that statement, typically, the process is described in terms of a lower-level language whose semantics is predefined.

- Focuses on how the state of the program is affected.
- i.e. how a computation is performed.
- Meaning of for program phrases defined in terms of the steps of computation they can take during program execution.
- example:

  To execute $X = y$, first determine the value of y, and assign it to x.

- uses the notation $\langle P, S \rangle$, means semantics of program P at states S.

  i.e

  ① $\langle z=x, x=y, y=z, [x\to5, y\to7, z\to0] \rangle \Rightarrow \langle P_1, S_1 \rangle$

  $\langle x=y, y=z, [x\to5, y\to7, z\to5] \rangle \Rightarrow \langle P_2, S_2 \rangle$

  $\langle y=z, [x\to7; y\to7, z\to5] \rangle \Rightarrow \langle P_3, S_3 \rangle$

  $\langle , [x\to7, y\to5; z\to5] \rangle$

So, before starting the program, the state of the memory is [x→5, y→7, z→0], and after the program has terminated, it is [x→7, y→5, z→5], which is meaning of the program i.e. z=x, x=y, y=z in the state [x→5, y→7, z→0].

- Syntax directed translation can also be represented as operational semantics.

e.g:

Production	Semantic rule
$E \to E + T$	&print ('+')&
$E \to E - T$	&print ("-")&
$E \to T$	& - &
$T \to 0$	&print ("0")&
$T \to 1$	&print ("1")&
$T \to 9$	&print ("9")&

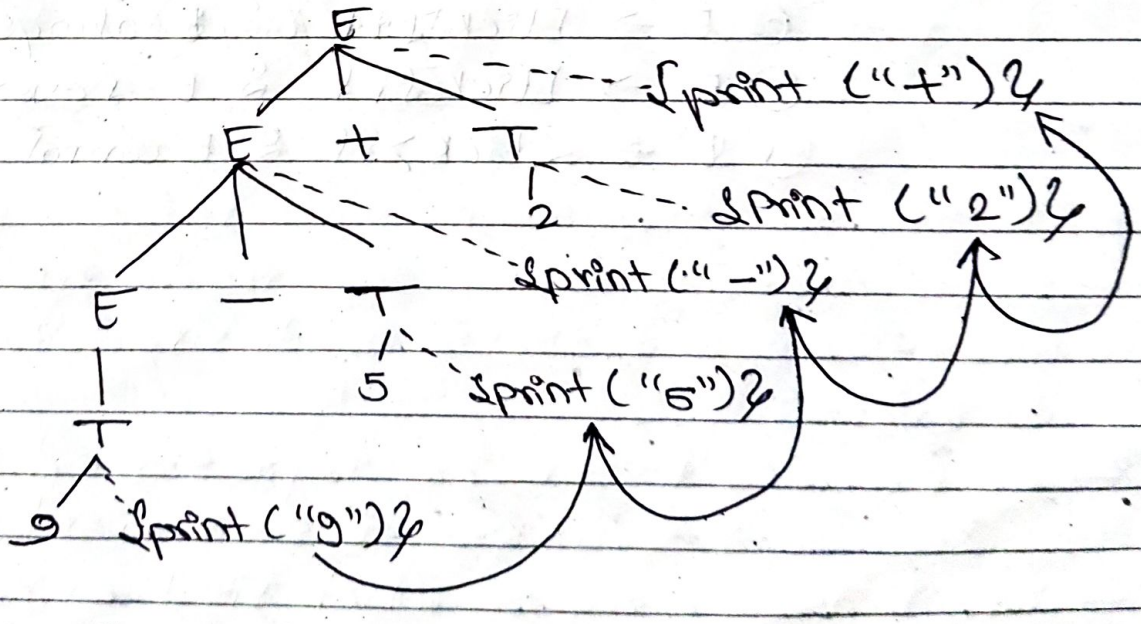


Fig: Action translating 9-5+2 into 95-2+.

Page No.
Date : / /

* Hennessy Milner Logic:

- Used to specify properties of a Labeled transition System.
- Introduced by Matthew Hennessy and Robin Milner.
- Syntax
  - $\phi \rightarrow tt$ & always true %
  - $\phi \rightarrow ff$ & always false %
  - $\phi \rightarrow \phi_1 \wedge \phi_2$ & conjunction %
  - $\phi \rightarrow \phi_1 \vee \phi_2$ & disjunction %
  - $\phi \rightarrow [L]\phi$ & after every execution of event L, the result has property ϕ %
  - $\phi \rightarrow \langle L \rangle \phi$ & if there exists an event L such that after its execution the result has property ϕ %

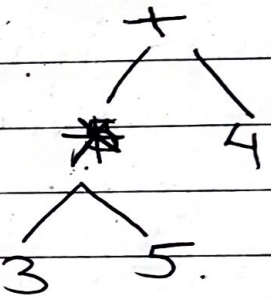
example:

1. $E \rightarrow \langle tick \rangle tt$ & E can do a tick %
2. $E \rightarrow [tick]\, tt$ & E always tick %
3. $E \rightarrow [tick]\, ff$ & E never tick %
4. $E \rightarrow \langle tick \rangle ff$ & E cannot do a tick %

Page No.
Date : / /

③ Denotational Semantic Definitions:

- Each construct of the Language is mapped into a mathmatical object that defines its meaning. These objects vary according to the Language at hand, the computation model underlying the language, and the properties that we want the semantic definition to capture.
- Idea of denotational semantics is to associate an appropriate mathmatical object with each phrase of the language.
- example:
  - Syntax tree for 3*5+4



- Traditionally, the emphatic bracket ([,]), is used to represent the denotational semantics definition.
- e.g: $E_1 + E_2$ = plus (Evaluate [E1], Evaluate [E2])
- Also, the ordered pair can be used to represent the program,
  e.g:- fact(n) = IF (n=0) then 1 else n*fact(n-1) can be viewed as an ordered pair for denotational semantics as <n, n!>,
  $\{<0,1>, <1,1>, <2,2>, <3,6>, <4,24>, \ldots\}$

- example: denotational specification of language of non negative integer number:

Number → digit | Number digit
digit → 0|1|2|3|4|5|6|7|8|9

value [ND] = plus (times (10, value [N], digit [D]
value [D] = digit [D]
digit [0] = 0
digit [1] = 1
---
digit [9] = 9

So, eg:-

value [65] = plus (times (10, value [6]), digit [5])
= plus (times (10, digit [6]), digit [5])
= plus (times (10, 6), digit [5])
= plus (60, digit [5])
= plus (60, 5)
= 65

* Communicating Sequential Process:

- Concurrent systems are made up of processes where each process is defined by a sequence of events.
- A process P can be represented by the notation $x \rightarrow Q$, where x is an event and Q is a process
- i.e. P as a process which first engages in the event x then behaves exactly as described by Q.
- Each process is defined with a particular alphabet

Page No
Date : / /

which represent the event.
- Small letter (alphabet, i-e. set of events), Capital letter (process)
- It is logically impossible for an alphabet to engage an event outside an alphabet.
- i-e. a machine designed to sell chocolate cannot deliver toy.

* example: A simple vending machine that serves a single customer then stop can be represented as,

1. $VM_0 \stackrel{def}{=} (coin \rightarrow (candy \rightarrow stop))$ (single customer)
2. $VM_1 \stackrel{def}{=} (coin \rightarrow (candy \rightarrow VM_1))$ (infinite custom
3. $VM_2 \stackrel{def}{=} (coin \rightarrow (candy \rightarrow VM_2)) \square (notebill \rightarrow (toffee \rightarrow VM_2))$

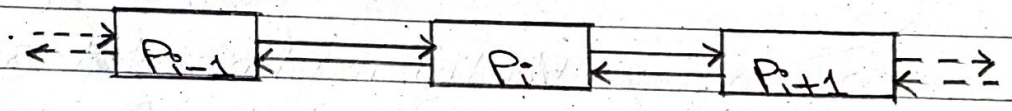
→ Vending machine that serves a candy for a coin deposit & a toffee for a notebill deposit

Page No.
Date : / /

1.3 Formal Methods : A Petri Nets Bases Approach :

* Process Algebras :

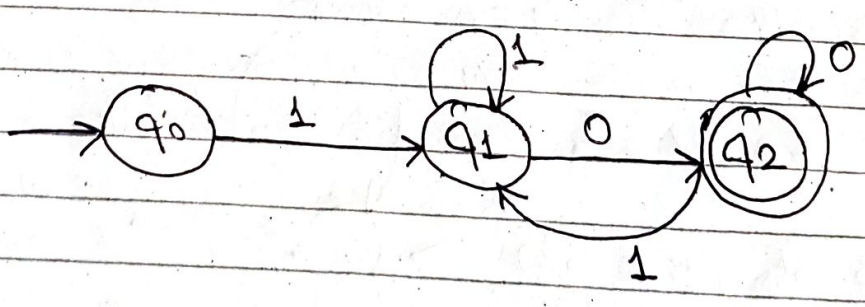
- System behavior generally consists of Process & data.
- Process are the control mechanism for the manipulation of the data.
- Process are dynamic and Active, while the data are static and passive.
- System behavior tends to be composed of several processes that are executed concurrently, where those processes exchange data in order to influence each other's behavior.



- example :

DFA (Deterministic Finite Automata)

$1(0+1)^*0$ → Process Algebra



process Graph.

Page No.
Date : / /

- Process Algebra is basically a labeled Transition System in which one state is selected as a root state.
- If the labeled Transition system contains an edge $s \xrightarrow{a} s'$, then the process graph can evolve from state s to s' by the execution of action (event) a i.e. s & s' are states. & a is an event.
- For the mathematical reasoning, process graph are expressed algebraically called process algebra
- So process algebra is basically a mathematical framework in which system behaviour is expressed in the form of algebraic terms.
- Process Algebra studies two types of action:
  - observable Action (Explicit)
  - Un-observable Action (Implicit)

① observable Action:
- If the process is ready to perform the action and some controlling environments allows it to occe
- occurs with any explicit event within a process.

② Un-observable Action:
- occurs without any explicit event.
- occurs implicit within a process.
- Denoted by τ (tow).

Page No.
Date : / /

* Example: Observable Action
  Memory = 2 items q_0, q_1 (1+1), q_2 (2-1)

- Consider, two-place buffer, the buffer is operated through two actions in and out, respectively putting one item in the buffer and taking one item from it.
- It is impossible to put one item into a full buffer & impossible to take one item from the empty buffer.
- Initially, buffer is empty, i.e. only one operation (IN) is possible.
- After IN is performed, it is possible to perform both IN & OUT.
- If IN is performed, the buffer will be full & only out is possible,
- If out is performed, the buffer is empty & only IN is possible.

We can represent the two-places buffer in graphical terms as a transition system as:

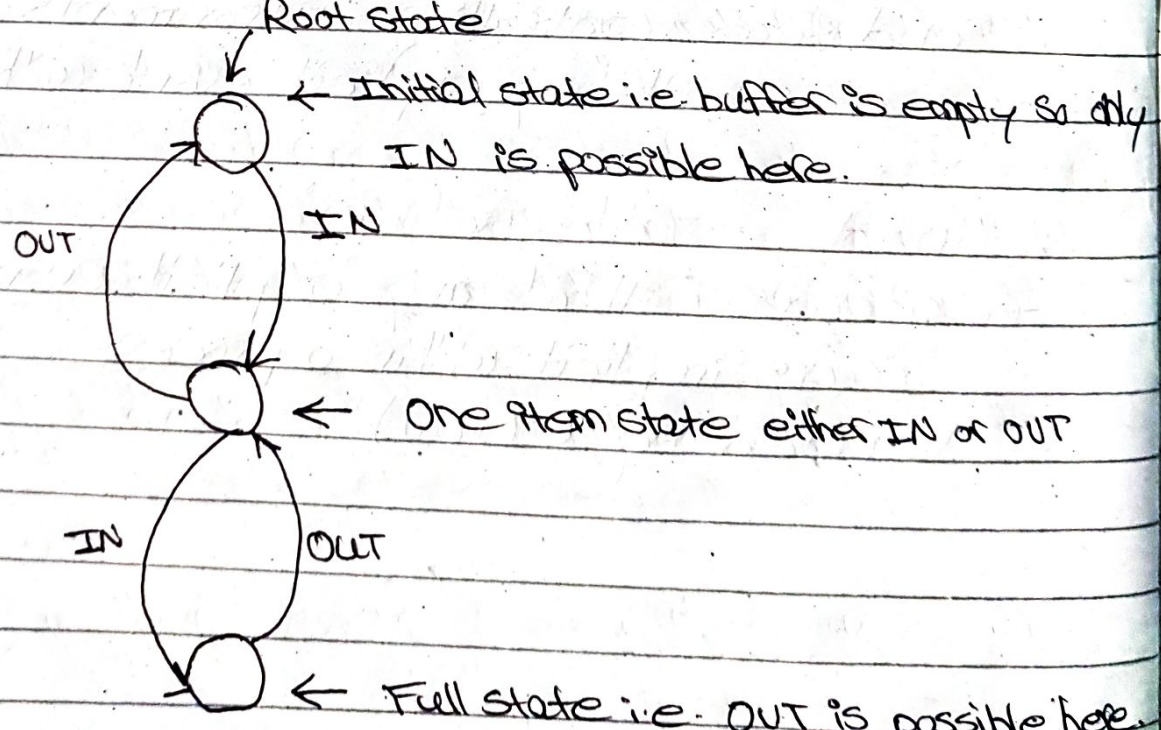


Fig: Transition system representing two place buffer.

Page No.
Date : / /

* Example ②: un-observable Action.
- Let us assume that at the beginning both one-place buffer are empty and at first only the in operation on the first buffer is possible. i.e.

q_0 : b-emp C-emp

- After IN is performed (the first buffer contain one item) only the unobservable transfer of the item from the first buffer to the second is possible. i.e.

q_1 : b-full C-emp

- Once the transfer is performed the first buffer is again empty, while the second one is full; it is possible to perform both in on the first buffer & out on the second.

q_2 : b-em C-full

- If in is performed afterwards both the buffers are ~~empty~~ full and only ~~in~~ out (on the second buffer) is possible.

q_3 : b-full C-full

In graphical terms (where the unobservable transfer of the item from the first to the second buffer is represented by a τ) the system can be represented by the transition system.

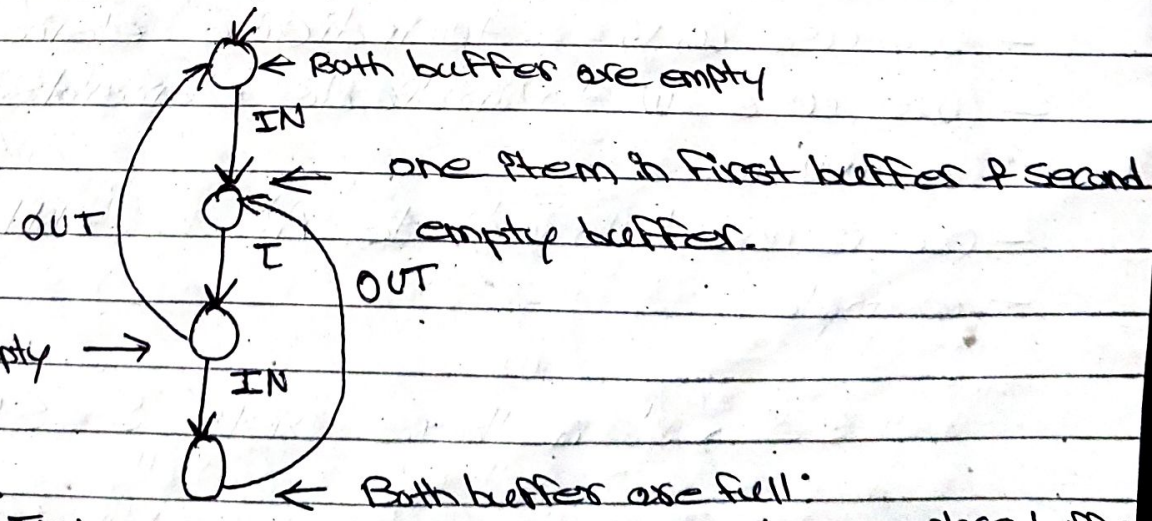


Fig:- Transition system representing a two-one place buffer

Page No.
Date : / /

* Transition Systems and observation bisimilarity:

- We will always consider systems that are able to perform both observable & unobservable actions.
- The observable actions will be represented by the symbols of an alphabet A, while any unobservable action (i.e., considered as an internal action the observer cannot see) will be represented by τ.
- Whenever we say that the alphabet of a system is A, we are considering the possibility that the system performs an action from $A \cup \{\tau\}$ i.e. observable ∪ un-observable.

① * **Definition:** Transition System.

A transition system is a quadruple and can be defined as

$$A' = (A, S, \rightarrow, S_0), \text{ where,}$$

A: is an alphabet,
S: is a set of states,
→: is a transition relation
S_0: is initial (root) state.

* **Conventions:**

- We use the letters a, b, c, ... to indicate elements of A,
- We use u, v, ... to indicate elements of $A \cup \{\tau\}$.
- We use w to indicate elements of $(A \cup \{\tau\})^*$
- A^ω is used to indicate A^* abbbccaa1b1
- example:

$s \xrightarrow{w} s'$ if these exists $s = s_1 \xrightarrow{u_1} s_2 \xrightarrow{u_2} s_3 \rightarrow \ldots s_n = s'$

Page No.
Date : / /

* Reachable States:

- Let $A' = (A, S, \rightarrow, S_0)$ be a transition system.
- Then $s' \in S$ is a reachable state of A' if there exists $w = u_1 u_2 \ldots u_n \in (A \cup \{\tau\})^*$ such that $S_0 \xrightarrow{w} s'$
- Set of all reachable states of A' is denoted by reach (A').
- Transition systems can also be represented through graphs. example:

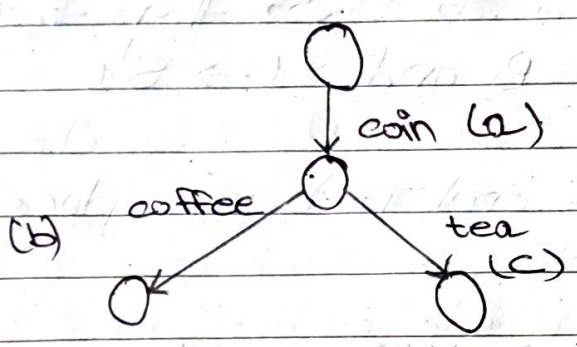


Fig a: ~~Non~~-Deterministic

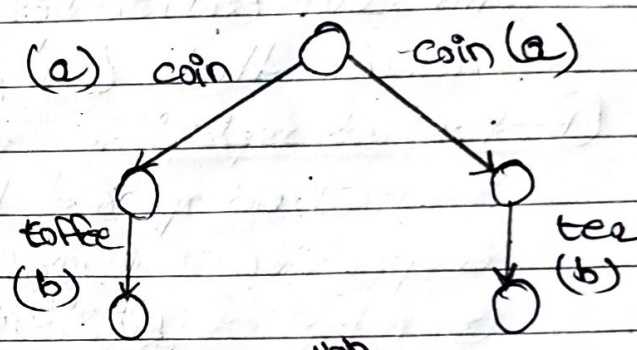


Fig b: Non-Deterministic

There are some cases in which the comparison of observable paths from the initial state is not enough. Considering above two transition systems from the viewpoint of the set of observable paths from the initial state they are equivalent, since for both it is $\{\varepsilon, a, ab, ac\}$, But in Fig (a), after ~~th~~ ~~entering~~ execution of a, is ready to execute both b & c, where in Fig (b), it is either ready to execute b (but not c) or ready to execute c (but not b). The second transition system is not deterministic in the sense that after the execution of a it can reach two different states, opening different sets of possibilities

Page No.
Date : / /

② * observation Bisimilarity:

- Let $A_i' = (A_i, S_i, \rightarrow_i, S_{0i})$, $i = 1, 2$ be two transition systems.
- A_1' and A_2' are observation bisimilar, denoted by $A_1' \cong A_2'$ if there is a relation $B \subseteq S_1 \times S_2$ such that,
  - i) $(S_{01}, S_{02}) \in B$
  - ii) If $(S_1, S_2) \in B$ and $S_1 \xrightarrow{a} S_1'$ then there is S_2' such that $(S_1', S_2') \in B$ and $S_2 \rightarrow S_2'$
  - iii) If $(S_1, S_2) \in B$ and $S_2 \xrightarrow{a} S_2'$ then there is S_1' such that $(S_1', S_2') \in B$ and $S_1 \rightarrow S_1'$

① ~~for example:~~

Two place buffer and Two one place buffer in previous pages i.e.

①

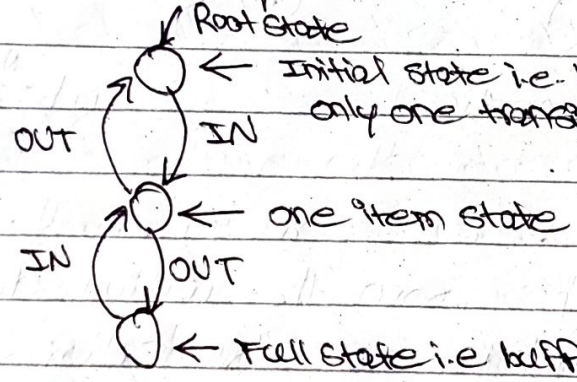


fig: Transition system representing a two place buffer.

②

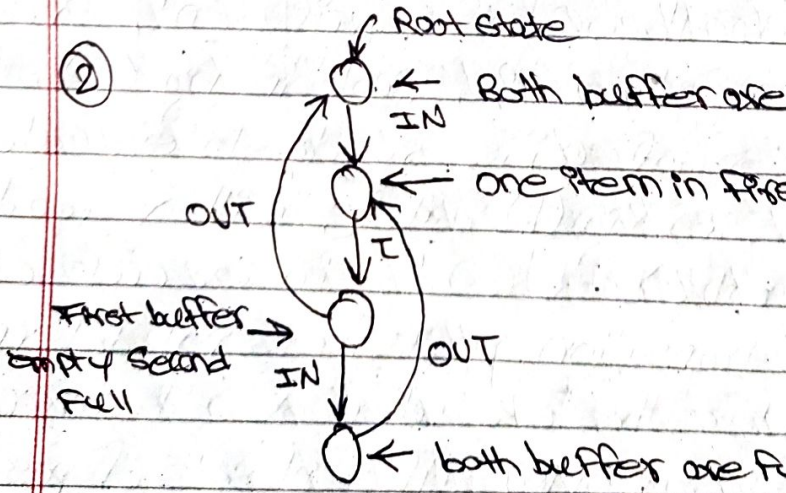


Fig: Transition system representing two one place buffer.

Page No.
Date : / /

② Example:

Consider the following two process:

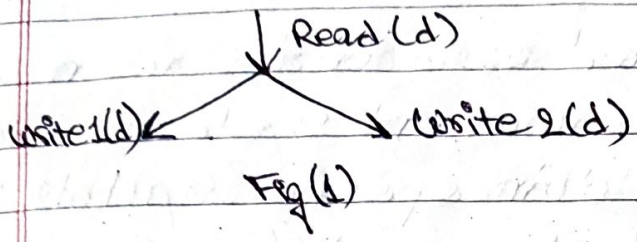
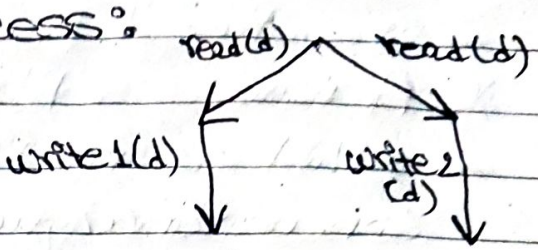


Fig (1)



Fig(2)

- In fig (1), the process read the datum d & decides whether to write d on disc 1 or disc 2
- In fig(2), the process determines whether to write either on disc 1 or disc 2, before reading the datum d
- Both processes display the same string of actions.
- i.e. read (d) write 1(d) OR read (d) & write 2(d)

So, It is easy to see that in example ① two-place buffer & two place one buffer are observation bisimilar where in example ② they are not observation bisimilar because

- in fig (1), in case of crash of disc 1 after executing read (d) & write 1 (d), there is second chance to write on disc 2 using write 2(d).
- But in fig (2), in case the disc 1 crashed after executing read (d) and write 1 (d) then it doesn't have any option to write the datum.

Observation bisimilarity is therefore a good equivalence notion with respect to concurrent systems.

Page No.
Date : / /

③ ~~Process Terms~~ A Process Algebra:

Let us now introduced a second view on concurrency, whereby concurrent systems are recursive term over certain operator symbols, i.e. they form the smallest class of terms generated by the operator symbols & closed under recursion (without parameters). System of this type are generally called process algebras.

3. process terms:

- Let Act be defined as $A \cup \{\tau\}$, where A is a finite alphabet of (observable) actions, whose elements will be denoted by a, b and τ is the symbol of the unobservable actions, not belonging to A.
- The set of process terms, proc, whose elemet will be denoted by P, Q, R and described as:

i) $P \to a.p$ (Prefix) ⇒ denotes a process that behaves like P after executing event a.

ii) $P \to stop$ (Deadlock) ⇒ denotes a process which doesn't perform any action.

iii) $P \to P + Q$ (choice) ⇒ denotes process that behaves like either P or Q.

iv) $P \to P \,||\, Q$ (parallelism) ⇒ where P & Q run independently.

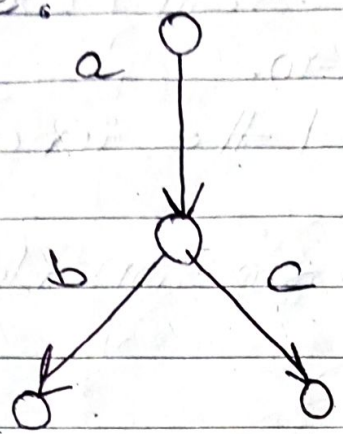
v) $P \to P[a/b]$ (Renaming) ⇒ denotes a process that behave like P, except for the fact it executes b

Page No
Date : / /

whenever P executes b.

vi) $P \rightarrow P \setminus \{b\}$ (Restriction) → is the process without action b.

example:

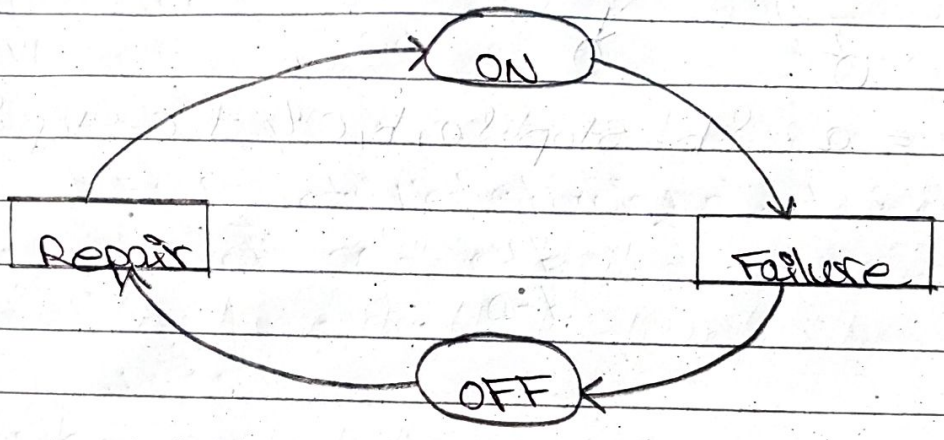


- $S = a \cdot \{b \cdot Stop\{a,b,c\} + c \cdot Stop\{a,b,c\}\}$

Page No.
Date : / /

* Petri Nets:

- Developed by Carl Adam Petri in 1962 as his PhD research
- It is the mathematical modeling Language for the description of the system.
- i.e. It is used to model the functional behaviour of the System.
- Abstract model of information Flow.



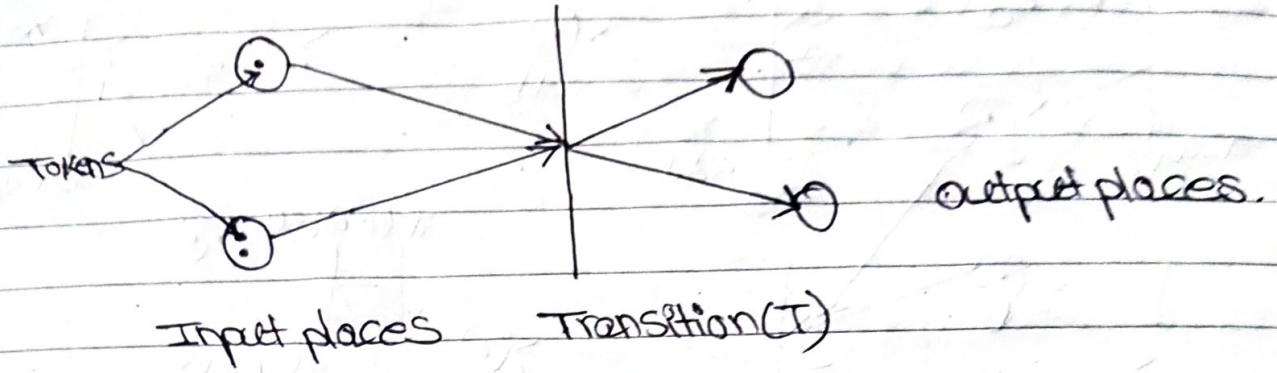
- Petri Nets (PN) are represented through directed graph (bi-partite graph)
- It consists of two types of nodes:
  1) Place:
     - Buffer that holds something
     - represented by circle.

  2) Transition (Event / activity that takes places)
     - Represented by straight line (|), or rectangle (□), or box (■)

- A place can be marked with a finite number (possibly zero) of tokens

Page No
Date : / /

- Token circulates in the system between places.



- A transition is called enabled (fired), whenever there is atleast one token in all the input places.
- When it is fired, one token (by default) is removed from all the input places and one token is added to all its output places.

e.g:
- Input places → preCondition
- output places → post condition.

e.g-1:

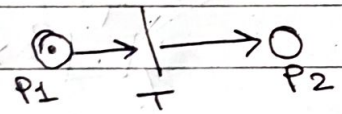


Fig:- Before Firing of T

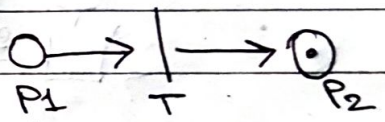


Fig:- After Firing of T

e.g-2:

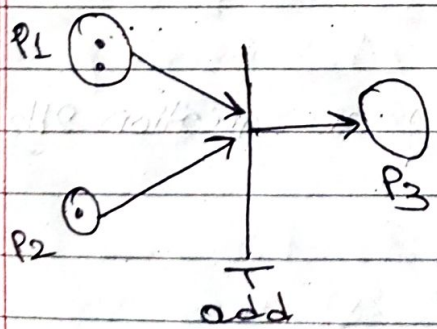


Fig: before Firing of T

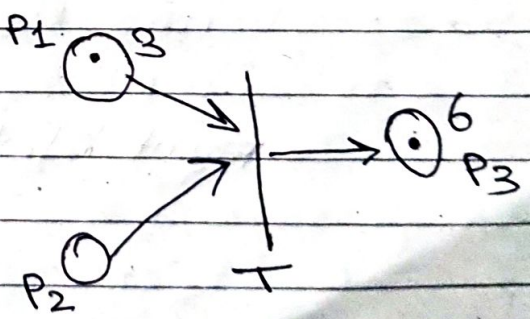


Fig: After Firing of T

Page No.
Date : / /

eg 3: V.V.I

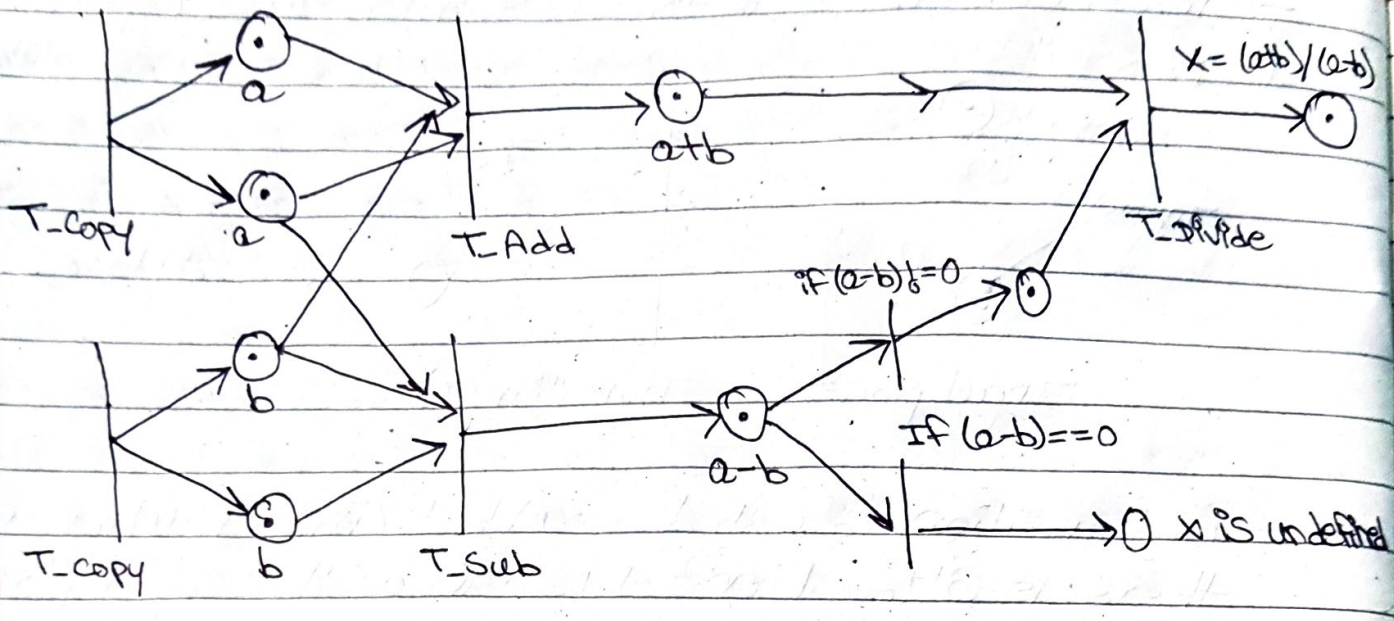


Fig :- Petri Net showing a data flow comparison of $x = \frac{a+b}{a-b}$

eg 4:

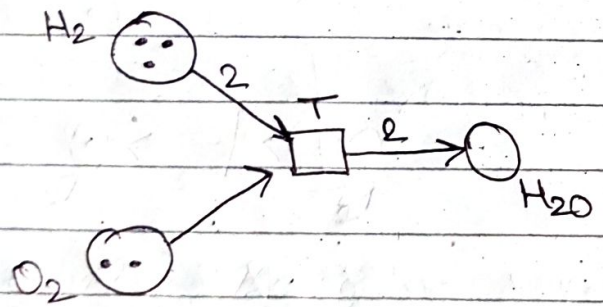


fig : before firing of T

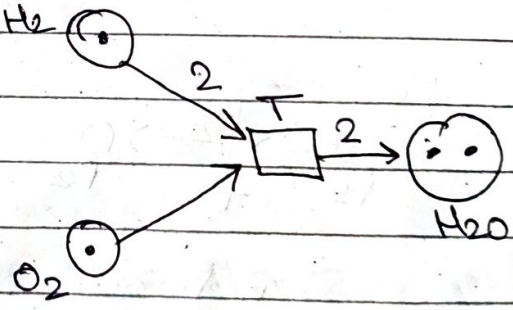


fig: After firing of T

Fig of : Petrinet showing an equation $2H_2 + O_2 \rightarrow 2H_2O$

Page No.
Date : / /

* Formal Definition:

- PN = (P, T, I, O, M_{to}) where
  - P → set of places = {$P_1, P_2, \dots, P_m$}
  - T → set of Transitions = {$T_1, T_2, \dots, T_n$}
  - I → is an input matrix of n×m
  - O → is an output matrix of n×m
  - M_{to} → represents initial marking of machine

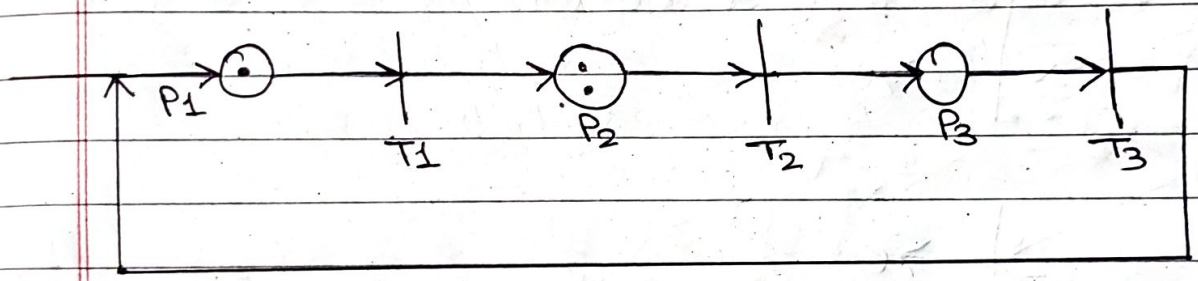
* Note:

A marking of a PN at time t,

$M(t) = \{M_1(t), M_2(t), \dots M_m(t)\}$

where,

$M_i(t)$ = no. of tokens in place i at time t.



- P = {P_1, P_2, P_3}
- T = {T_1, T_2, T_3}
- I =

	P_1	P_2	P_3
T_1	1	0	0
T_2	0	1	0
T_3	0	0	1

3×3 — In case of input

- O =

	P_1	P_2	P_3
T_1	0	1	0
T_2	0	0	1
T_3	1	0	0

3×3

- M_{to} = (1, 2, 0)

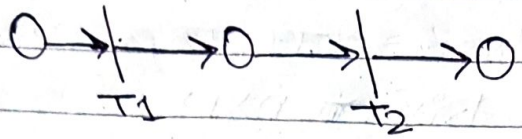
$t_1 \rightarrow t_2 \rightarrow$ fire

- M_{t_1} = (1, 1, 1)

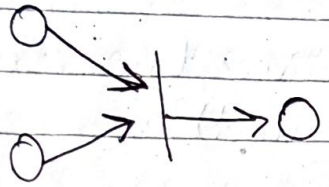
Page No.
Date : / /

* Variations in Petri Nets:

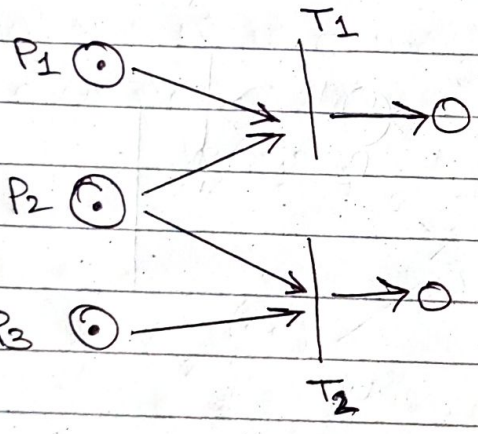
① Sequential Actions:



② Dependency:

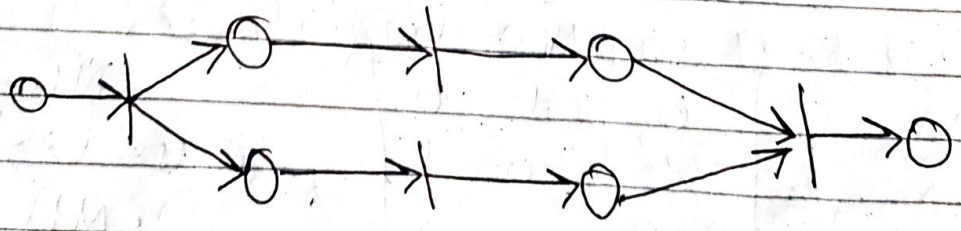


③ Conflict:



- P1 & P2 have tokens but not P3 then T1 is fired.
- P2 & P3 have tokens but not P1 then T2 is fired.
- If all have tokens which one to fire then conflict happens & solution may be priority.

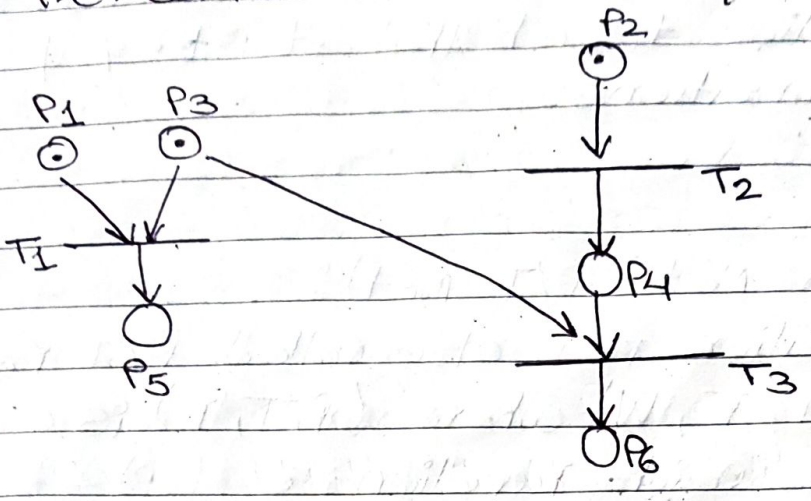
④ Concurrency:



Page No.
Date : / /

⑤ Confusion:

i.e. Conflict + Concurrency.



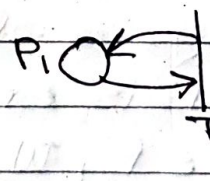
- Two possible interpretation

1. T1 occurred first without being in Conflict with other events then T2 occurred.
2. T2 occurred first & then T1, T3 got in conflict.
   i) Conflict $(T_1, C_1) = \phi$ where $C_1 = \{P_1, P_2, P_3\}$
   ii) Conflict $(T_1, C_2) = \{T_3\}$ where $C_2 = \{P_1, P_3, P_4\}$

* Pure & Simple Net:

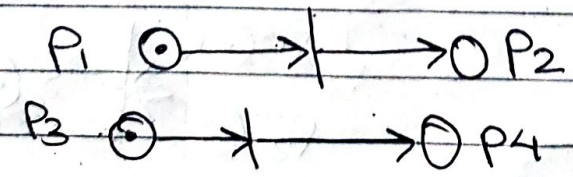
- Let $PN = (P, T, F)$ be a Net:

1. PN is pure if $\forall x \in X : {}^\bullet x \cap x^\bullet = \phi$
   (No self loop)



2. PN is simple if $\forall x, y \in X : [({}^\bullet x = {}^\bullet y \wedge x^\bullet = y^\bullet) \Rightarrow x = y]$

i.e.



Page No.
Date : / /

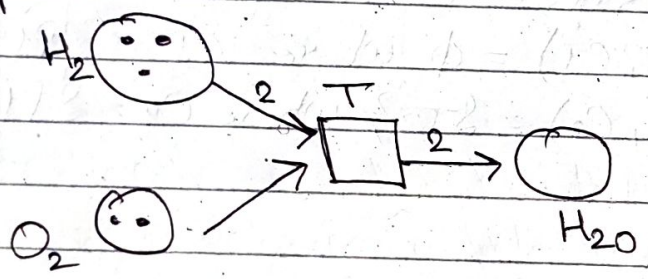
① In case of not pure pure nets, the places P such that $\bullet P \cap P\bullet \neq \phi$ are called side conditions.
② and the Transition t such that $\bullet t \cap t\bullet \neq \phi$ are called impure transitions.

* Place/Transition Net (P/T Net):

A place / transition net, also called P/T net, is a 4-tuples (P, T, F, W) where (P, T, F) is a Petri Net and W is a weight Function:

- (P, T, F) is a net, where P-elements are called places & T-elements are called Transitions.
- $W: F \to N^+$ is a weight Function.

example:



* Transition rule:

- Let $M = (P, T, F, W, M_0)$ be a Place transition system. Let M be a marking, i.e. Function $M: P \to N$.

1) A transition $t \in T$ is enabled at M

$M[t> \text{ if } \forall p \in P, W(P,t) \leq M(s)]$

e.g:

$M_0[t_1>M_1 \quad m_0 = 2,1,0 \quad t_1 = t \quad m_1 = 1,0,1]$.

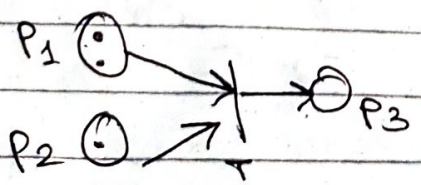


Fig: Before Firing T

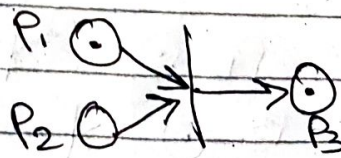


Fig: After Firing T

Page No
Date : / /

2) A transition t enabled at M may fire yielding a new marking M', $M[t>M'$, such that $\forall p \in P$.

$$M'(p) = M(p) - W(p,t) + W(t,p)$$

* Properties of Petri Nets:

1. Reachability,
2. Boundness,
3. Liveness,
4. Reversibility.

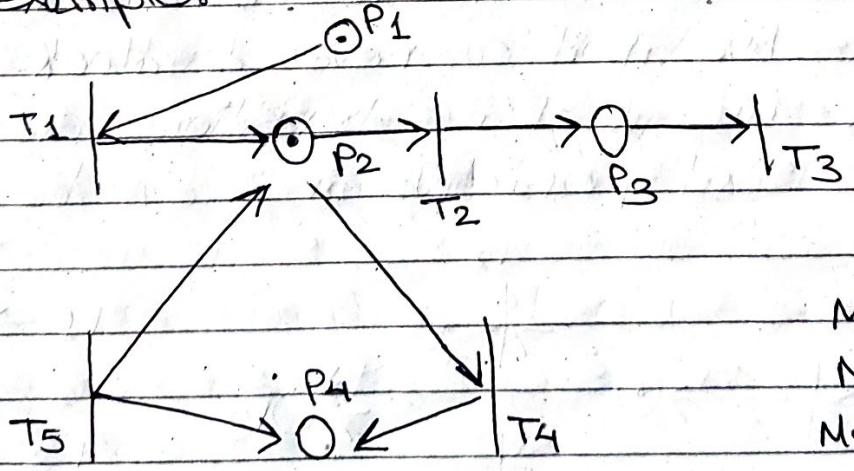
1. Reachability:

- A marking M is said reachable from another marking M', if there exists a sequence of transitions (σ), such that

$$M'[\sigma>M \quad \text{i.e.} \quad \text{Reach}(M') = M$$

$M' \xrightarrow{t_1} M'' \xrightarrow{t_2} M''' \xrightarrow{t_3} \cdots \xrightarrow{t_n} M$, where $\sigma = t_1 t_2 \cdots t_n$

- $R(M_0)$ → set of markings, reachable from the initial marking M_0.
- example:



P1 P2 P3 P4

$M_0 = (1, 1, 0, 0)$

$M_1 = (0, 2, 0, 0)$ T_1 Fire

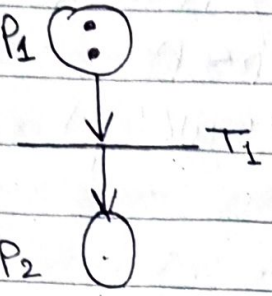
$M_2 = (1, 0, 1, 0)$ T_2 Fire

$M_3 = (1, 0, 0, 1)$ T_4 Fire

$R(M_0) = \{(0,2,0,0)$
$(1,0,1,0)$
$(1,0,0,1)\}$

Page No.
Date : / /

eg:-



$M_0 = (2,0)$
$M_1 = (1,1)$ or $(0,2)$
So, $R(M_0) = \{(1,1)(0,2)\}$
i.e $(2,0)[T_1 > (1,1)$

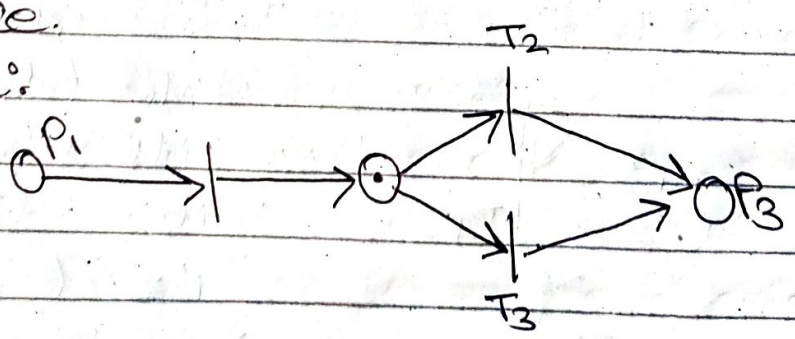
② Boundness:

- No. of tokens in a place is bounded.
- A place P is said to be K-bounded, if the number of tokens in P never exceed K, i.e $M(P) \leq K$
- A Petri net is said to be K-bounded if all places are K-bounded.
- A Petri net is said to be bounded if it is K-bounded for some $K > 0$

③ Liveness:

- A transition is said to be live if it can always be made from any reachable marking.
  i.e: $\forall M \in R(M_0), \exists M' \in R(M)$ such that $M'[t>$
- A transition is deadlocked if it can never fire.
- A transition is live if it can never deadlock.
- A Petri net is live if all its transitions are live.
- A transition is said quasi live if it can be fired at only once.

example:



$T_1 \rightarrow$ dead
$T_2 \rightarrow$ live
$T_3 \rightarrow$ live

Page No.
Date : / /

④ Reversibility:

A Petri net is said to be reversible, if the initial marking remains reachable from any reachable marking.

i.e. $M_0 \in R(M), \forall M \in R(M_0)$

* High-Level Petri Net:

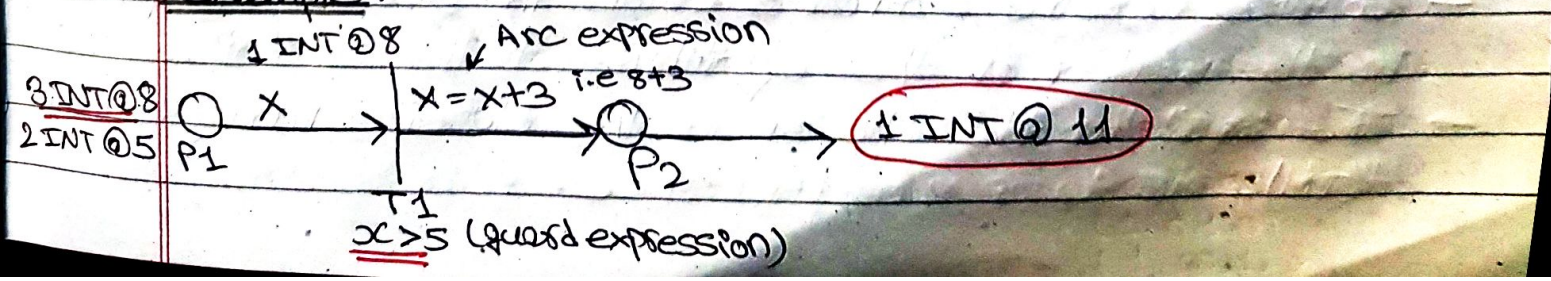
The Language presented in the previous sections are suitable for representing Concurrent systems where the control flow is independent of data manipulation i.e, fully hidden in action labels. This limited expressive power is not adequate for specifying real-world system whose description requires the explicit data representation.

To fulfill this requirement, Combinations of basic models of concurrency with various approaches to data representation have been proposed known as High Level Petri Nets.

High level petri Nets consists of the following features:

① Petri Nets,
② Marking Scheme (like color petri net)
③ Transition Guard (condition)
④ Arc expression (eqn to be satisfied)

example:



Page No.
Date : / /

1.4. Complexity Issues in Parallel & Distributed Computing

* Introduction:

The complexity theory estimates the amount of computational resources (such as the computing time and the storage space) needed to solve a problem, ignoring the details of implementation of the algorithm in a specific computing machine.

This theory provides a functional relationship between the input size of a solvable problem and the amount of resources needed for solving that problem on a universal computational model.

The three basic aims of complexity theory are:

1. **Introducing a notation to specify complexity:**
   The first aim is to introduce a mathmatical notation to specify the functional relationship between the input size of the problem and the consumption of computational resources, eg., Computational time & memory space. Notation like Big oh (O), Big omega (Ω), Big theta (Θ)
2. **Choice of machine model to standardize the measure:**
   - The second aim is to specify an underlying machine model to prescribe an associated set of measures for the consumption of resources.
   - So, that machine cannot exceeds the resources prescribed by the writer of the algorithm.
   - The basic model chosen is the classic Turing machine (TM).

Page No.
Date : / /

3. Refinement of the measures of parallel computation:
- The third aim is to understand how fast we can solve certain problems when a large number of processors are put together to work in parallel.

V.V.I

* Turing Machine as a basis of Consequences.
1. Simplicity of abstraction,
2. Dual nature of TM,
3. Inclusion of non-determinism
4. Realization of unbounded Parallelism
5. Provision of an easy abstract measure for resources.

1. Simplicty of abstraction:
- The Turing machine is the simplest model of an abstract machine. It consists of three basic compon
i) A finit state machine,
ii) A read/write head,
iii) A infinite tape memory.
- Mathematically Turing machine can be defined by
$TM = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, whose
$Q \rightarrow$ set of states,
$\Sigma \rightarrow$ set of input alphabet,
$\Gamma \rightarrow$ Set of Tape Symbol,
$\delta \rightarrow$ Transition function,
$q_0 \rightarrow$ Starting State,
$B \rightarrow$ blank Symbol,
$F \rightarrow$ set of final states.

Page No.
Date : / /

2. Dual nature of TM:

- A Turing Machine can simulate any special-purpose algorithm, this is specialty property.
- Also it can simulate any other Turing machine; this is the generality of universality property.
- i.e Universal TM whose input consists of program & a dataset for the program to process.
- These dual features, namely specialty & universality, make it a standard model for studying the theory of computation.

3. Inclusion of non-determinism:

- It is possible to add non-determinism in a TM by providing several alternative moves and allowing the TM to choose one of these alternatives nondeterministically.
- The essential difference between the NDTM & DTM is the manner in which the strings are accepted or recognized.

i) A DTM computation accepts a string if and only if there is a straight line sequence of configuration to a final state.

ii) NDTM computation can only be modeled by a branching Tree. i.e NDTM carries out non-deterministic moves and performs a search through the tree of possibilities, called an OR-tree.

- NDTM uses alternative moves i.e. NDTM solves a problem if there are some sequences of guesses or a path in the tree of possibilities to reach the solution.
- A computation halts, if and only if there is a sequence that leads to an accepting state.

- NDTM rejects an input if and only if there is no possible sequence of moves that leads to an acceptance.
- NDTM doesnot enlarge the class of solvable problems, but can help speed up computation of a solvable problem through guess work which provides for unbounded parallelism.

4. Realization of unbounded parallelism:
- The NDTM can be simulated using DTMs by allowing unbounded parallelism in computation.
- Each time a choice is made, the DTM makes several copy of itself.
- one copy is made for each possible choice
- Thus many copies of DTM are executing at the same time at different nodes in the tree.
- The copy of DTM that reaches a solution early halts all other DTM, if a copy fails, it only terminates.
- Hence, to simulate a NDTM, we need to use K^d copies of DTM, where K is the number of branches of the OR-tree at each level, & d is the depth of the tree.
- This means the NDTM is equivalent to an exponential number of DTMs working in parallel.

5. Provision of an easy abstract measure for resources:

(1) Case i) In case of DTM:
The measurement of abstract resources in DTM can be carried out by counting each moves as one

unit time & each tape cell scanned as one space time

② Case II: In case of NDTM:

The time complexity is the smallest number of moves required for an accepting sequence of moves, since ~~there~~ there can be more than one accepting sequence. If there is no accepting sequence it is undefined.

The time used is the depth of the tree and the space is the maximum (overall configurations in the Tree) of the number of cells in use.

* Describing the classes P and NP:

The DTM classifies the two fundamental complexity classes of Problems i.e P and Np.

① P class:

The class P consists of those problems for which the DTM takes polynomial time to solve them i.e time T(n) that is a polynomial in n. These problems are called tractable problems or polynomial time computable problems

$$P = DTIME(poly(n))$$

② Np-class:

When a decision problem cannot be answered "yes" by a DTM in polynomial time but can be answered "yes" by a NDTM in polynomial time, we say that the problems belong to the NP-class. Once the solutions are obtained by a NDTM using guesswork, their truth is verifiable by a DTM in polynomial Time. Every problem in this class

Note: The NP problems are set of problems whose solution are hard to find but easy to verify and are solved by NDM in polynomial time.
*NDM :- Non-Deterministic Machine.

Page No
Date : / /

can be solved in exponential time using exhaustive search.

$$DTime(poly(n)) \subseteq NTime(poly(n))$$
$$NTime(poly(n)) \subseteq DTime(Expoly(n))$$

(3) Co-Np:

- The class co-Np consists of those problems that are complement of the problems in Np.
- Co-Np is the class of decision problems for which a "no" answer can be given in polynomial time.
- It is not known whether NP = co-Np, nor it is known whether the problems in the intersection class

$$Np \cap Co\text{-}Np = P$$

- An example of a decision problem in NP is "whether a given number is a prime" and An example of a decision problem in Co-Np "whether a given number is composite".

(4) Np-Hard & Np-Complete:

i) Np-complete:

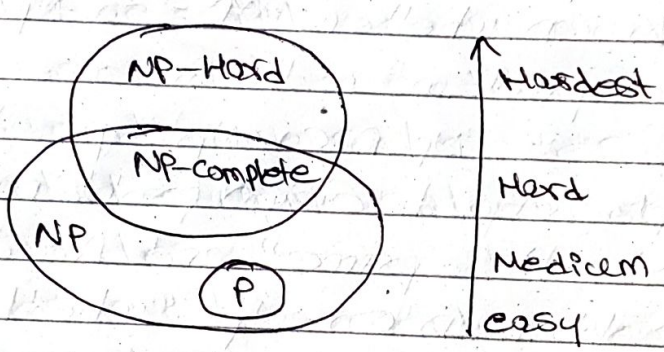
A problem X is NP-complete if there is a NP problem Y, such that Y is reducible to X in polynomial time. Np-complete problem are class of computation problems for which no efficient solution algorithm has been found. A problem is NP-complete if it is a part of both NP and NP-Hard problem. A non-deterministic Turing machine can solve NP-complete problem in polynomial time. for eg: Travelling salesman problem, satisfiability problems, & graph-covering problem.

Page No.
Date : / /

ii) NP-Hard:

A Problem X is NP-Hard if there is an NP-complete Problem Y, such that Y is reducible to X in polynomial time NP-Hard problem need not to be in NP class.

- Easy → p class problem,
- Medium → Np class problem,
- Hard → Np-complete problem,
- Hardest → Np-Hard problem.



V.I.T

* Satisfiability Problem:

Let E be a Boolean expression in the Conjunctive normal form (CNF); i.e. It is an expression consisting of "and" of several "or" clauses. for example:

Consider E by:

(x or y or z) and (x or ¬y or ¬w) and (¬z or w) and (y

- A truth assignment for E is called Satisfiable is there is some assignment of truth values True (T) and False (F) to the variables in E such that the resulting value of E is True (T).
- Consider the above example, This E is satisfiable under the assignment x = z = w = False (F) & y = True (T).

Page No
Date : / /

(x or y or z) and (x or ¬y or ¬w) and (¬z or w) and (¬x)

= (F or T or F) and (F or ¬T or ¬F) and (¬F or F) and (¬F)

= T and T and T and T

= T ⇒ satisfiability.

- On the other hand, the expression ϕ (x or y) and (¬x) and (¬y) is not satisfiable for any assignment of True(T) & False(F) to x & y. as it always results in the expression assuming the value F.
- The satisfiability problem is to determine whether a given Boolean CNF expression is satisfiable.
- If there are n literals per conjunct in the satisfiability problem, we call it n-satisfiability problem. If n=2 we call it 2-satisfiability problem, & if n=3, we call it 3-satisfiability problem.
- n-satisfiability for n ≥ 3 is NP-complete. However 2-satisfiability problem is in P.

Page No.
Date : / /

* Generalized TM relating time & space measures:

ATM (Alternating Turing Machine) :

To relate the time and space measures, we can generalize the NDTM to a more powerful TM called an Alternating Turing Machine (ATM). The ATM is very similar to NDTM except for the rules used for acceptance of the input strings. The ATM time complexity classes closely correspond to TM and NDTM space complexity classes.

- An ATM consists of infinite memory tape & a finite state control as in a TM.
- The states of ATM are partitioned into normal, existential, and universal states.
- The existential & universal states have two successor states.
- Normal states can be accepting, rejecting or undefined.
- Given, an initial state of the finite control and the input tape position for the head, the state of machine is completely determined.

i) Universal State :

It leads to an acceptance, if and only if all of its descendants lead to an accepting states. i.e. AND tree.

ii) Existential State :

A computation is accepting, if and only if at least one of its descendant is accepting i.e. OR tree.

Page No.
Date : / /

iii) Normal State:

A computation leads to an acceptance if and only if its unique descend is accepting.

If none of the above rules are applicable, it can leads to an undefined state that corresponds to an infinite computation.

* Parallel Complexity Models and Resulting classes:

We will discuss about Synchronous parallel system:

- Synchronous parallel system is the system where all the identical processors work in synchrony under the control of a Centralized Clock.
- It is easy to measure time in terms of clock cycles.
- In such system, the communication time among the processes and processors is much shorter than the computation time, since the interprocessor communication links are predetermined and directed hardwired.
- Synchronous parallel computing can be used to solve many numerical & scientific problems. Like matrix multiplication.

Page No.
Date : / /

* VLSI Computational Complexity:

- The practical uses of many algorithms depend upon the design of VLSI chips that can perform computation with maximum efficiency.
- To study the efficiency, different models are used. The main difference among these models is the manner in which the signal propagate time is modeled.
- Two fundamental parameters decides the efficiency of a VLSI chip:

i) The area of circuit:

The area gives the space complexity,

ii) The time taken to produce an output:

Time taken to produce an output for a given input gives the time complexity.

The product of area and time is used as a complexity measure of VLSI chips.

* Complexity Measures For Distributed Systems:

- Measuring time & space complexities for asynchronous distributed computation is difficult due to the intrinsic differences between the parallel & distributed computing.
- In Asynchronous distributed System, the communication time can be much longer than the computation time because the communication links are usually longer.
- In distributed systems, there are multiple processes and processors that communicate & co-operate with each other.
- Therefore, the complexity issues that arise are entirely of a different nature.

Page No
Date : / /

1.5 Distributed Computing Theory:

* The Computational Model:

- We outline the basic elements of formal model of a distributed system. The model shares many ideas with other models.
- Major obstacles in distributed computing is uncertainty due to different processor speed, and occasional failure of components.
- A system consists of n processors $P_1, P_2, \ldots, P_n$ and the network net.
- Each processor P_i is modeled as a state machine with state set Q_i.
- The state set Q_i, contains a distinguished initial state $q_{0,i}$.
- We assume the state of processor P_i contains a special component, $buff_i$, in which incoming messages are buffered.
- Processor P_i's transition function takes as input a state of P_i and produces as output another state of P_i and a set of messages.
- Each messages consists of the body of the message and indicates the sender & receiver.
- The network net is modeled by the set of messages that are in transit.

Page No.
Date : / /

** Leader Election:

- Leader Election problem is a problem in distributed computing in which the processors must "choose" one of them as a leader.
- The existence of a leader can simplify coordination among processors and is helpful in achieving fault-tolerance and saving resources.
- Informally, the problem is for each processor to eventually decide whether it is the leader or not.
- In terms of our formal model, an algorithm is said to solve the leader election problem if it satisfies the following conditions:
  i) Each processor has a subset of elected states and a disjoint subset of not-elected states. Once a processor enters an elected state, its transition function will only move it to another elected state.
  ii) In every execution, exactly one processor enters an elected state, and the remaining processors enter a not-elected state.
- Election Algorithm:
  i) Bully Algorithm,
  ii) Ring Algorithm.

* Election Algorithm:

Election Algorithms choose a process from group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor.

Election algorithm assumes that every active

process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is send to every active process in the distributed system.

1) The Bully Algorithm:

This algorithm applies to system where every process can send a message to every other process in the system.

* **Algorithm:** Suppose process P sends a message to the coordinator.

i) If coordinator doesnot respond to it within a time interval T, then it is assumed that coordinator has failed.

ii) Now process P sends election message to every process with high priority number.

iii) It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.

iv) Then it sends a message to all lower priority number processes that it is elected as their new coordinator.

v) However, if an answer is received within time T from any other process Q,

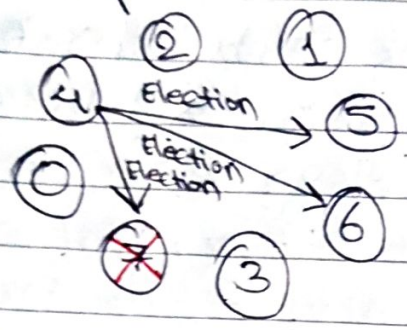
i) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.

Page No.
Date : / /

ii) If Q doesn't responds within time interval T then it is assumed to have failed and algorithm is restarted.

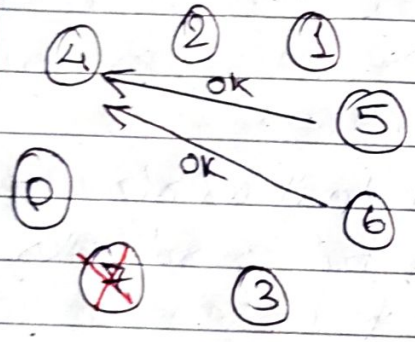
* example:

i)



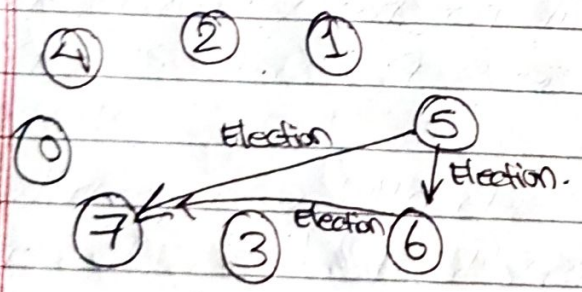
i) Since, 7 is crashed it doesnot received the message, Process 4 starts the election.

ii)



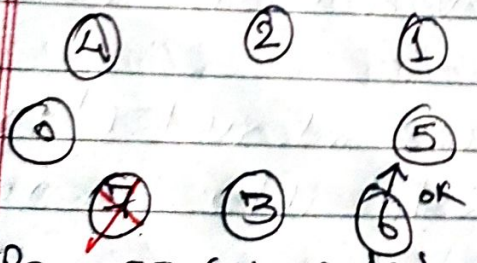
Process 5 & 6 responds to the Message since Process 7 previous coordinator has crashed.

iii)



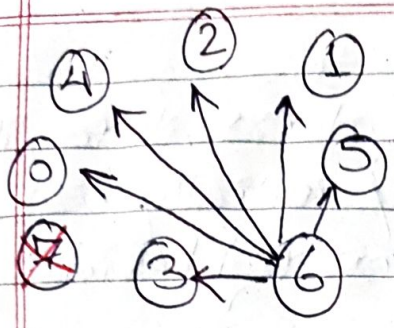
Both process 5 & 6 hold the election.

iv)



Process 6 has high priority after process 7, process 6 is new coordinator

v)



Process 6 announces to every other process that it is the new Coordinator.

2) Ring Algorithm:

- This algorithm organized a given System as a ring (logically or physically).
- If any process notices that the current Co-ordinator has failed, it starts an election by sending message to the first neighbor of the ring.
- The election message contains the node's identifier and is forwarded around the ring.
- Each process adds its own identifier to the message
- When the election message reaches the originator, the election is complete.
- Co-ordinator is chose based on the highest numbered process.
- In this algorithm we assume that the Link between the process are unidirectional and every process can message to the process on its right only.
- Data structure used by this algorithm is Active List, a list that has priority number of all active processes in the system.

Page No.
Date : / /

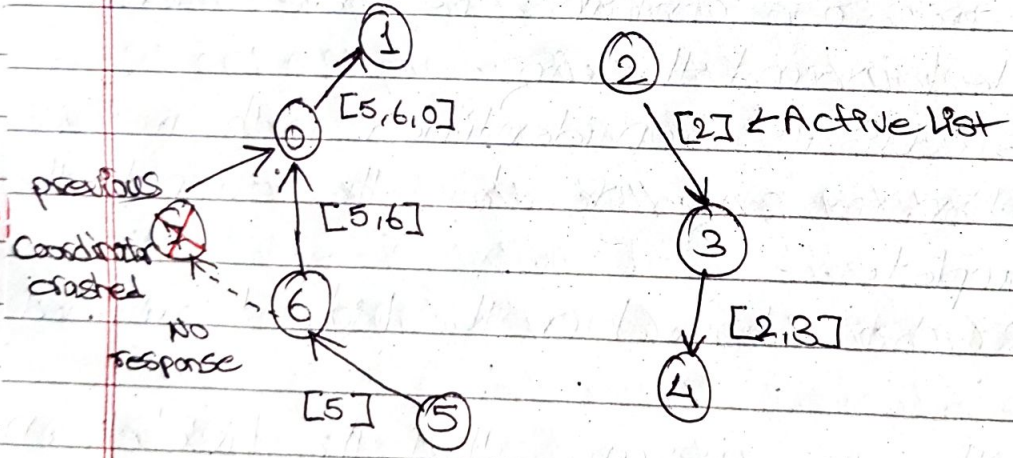
* Algorithm:

i) If Process P1 detects a coordinator Failure, it creates a new active list which is empty initially. It sends election message to its neighbor on right side and adds number, to its active list.

ii) If process P2 receives message elect from processes on left, in the same way it adds its number in active list and forwarded the message to its neighbor on right.

iii) If a process receives its own message, then it detects the process with highest number as a new Leader.

* example:



Step i): Initially, Process 2 & 5 initiate the election.

Step ii): They both forward the election message to its neighbor with its IDs and so on until the circuit builds

Step iii): The Active list for 2 & 5 will be

2 → [2, 3, 4, 5, 6, 0, 1, 2]

5 → [5, 6, 0, 1, 2, 3, 4, 5]

Step iv): Process 6 has highest number so selected as new coordinator.

Page No.
Date : / /

* Ordering of Events:

- In Distributed System; It is convenient to obtain an ordering between the events in the system. It helps to manage the order of events in which it happens in the system.
- For example: an ordering between events by different processors that request permission to use some shared resources can be used to grant the resource in a fair manner.
- In the synchronous model, there is very clear ordering between events at different processors, either they happen at the same round or at strictly ordered rounds.
- In the Asynchronous systems, events are unordered. We discuss various mechanisms for ordering events in an Asynchronous system.

* Logical clock in Distributed System:

- Distributed system may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.
- Logical clocks refer to implementing a protocol on all machines within a distributed system, so that the machines are able to maintain consistent ordering of events within some virtual Timespan.
- For example, suppose we have more than 10 PC's in a distributed system and every PC is doing it's own work but then how we make them work

together. There comes a solution to this, i.e. LOGICAL CLOCK.

* **Solution:**
Assign Timestamp to events based on ordering among them.

* **Case i:**
Taking single PC only if 2 events A and B are occuring one by one then TS(A) < TS(B). If A has Timestamp of 1, then B should have timestamp more than 1.

* **Case ii:**
Taking 2 PCs and event A in PC-1 and event B in PC-2 then also the condition will be TS(A) < TS(B). for example: we are sending message to someone at 2:00:00 pm, and other person is receiving it at 2:00:02 pm. Then it's obvious that TS(Sender) < TS(receiver)

We can derive following Relation from above two cases:

① **Transitive Relation:**
If, TS(A) < TS(B) and TS(B) < TS(C), then TS(A) < TS(C)

② **Causally ordered Relation:**
$a \rightarrow b$, this means that a is occuring before b & if there is any changes in a it will surely reflect on b.

③ **Concurrent Event:**
This means that not every process occurs one by one, some processes are made to happen simultaneously, i.e. A || B.

Page No.
Date : / /

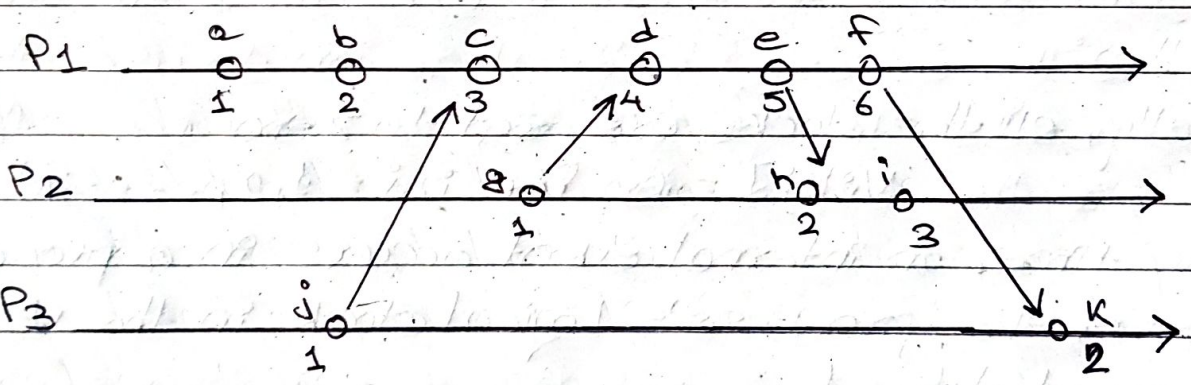
Also, if a and b occur on different processes, and donot exchange message, then neither $a \rightarrow b$ nor $b \rightarrow a$ are true, also called Concurrent.

* Lamport's Logical clock:

- It is a procedure to determine the order of events occurring.
- It provides a basis for the more advanced Vector clock Algorithm.
- Due to absence of a global clock in a Distributed operating System Lamport Logical clock is needed.

* Let us assume the following example:

- There are 3 processes with each processes have different set of events & Time stamp associated with each events i.e.



- Happened before relation ($\rightarrow$) : $a \rightarrow b$, means 'a' happened before 'b' so, Logical clock $TS(a) < TS(b)$
- $\rightarrow$ denotes message passing between different events within different processes. now, There are some bad ordering which doesn't satisfy our condition. for eg:

i) event (e, h) = $TS(e) < TS(h)$ = $5 < 2$ which is wrong. ✗

ii) event (f, k) = $TS(f) < TS(k)$ = $6 < 2$ which is wrong. ✗

iii) event (j, c) = $TS(j) < TS(c)$ = $1 < 3$ This is right. ✓

Page No.
Date : / /

- So, when a message arrives:
  1. If receiver's clock < (message_time_stamp)
     Set system clock to (message_time_stamp + 1)
  2. Else do nothing.

Note: Sender' clock < receiver clock
event (i, c) = 1 < 3 (True).

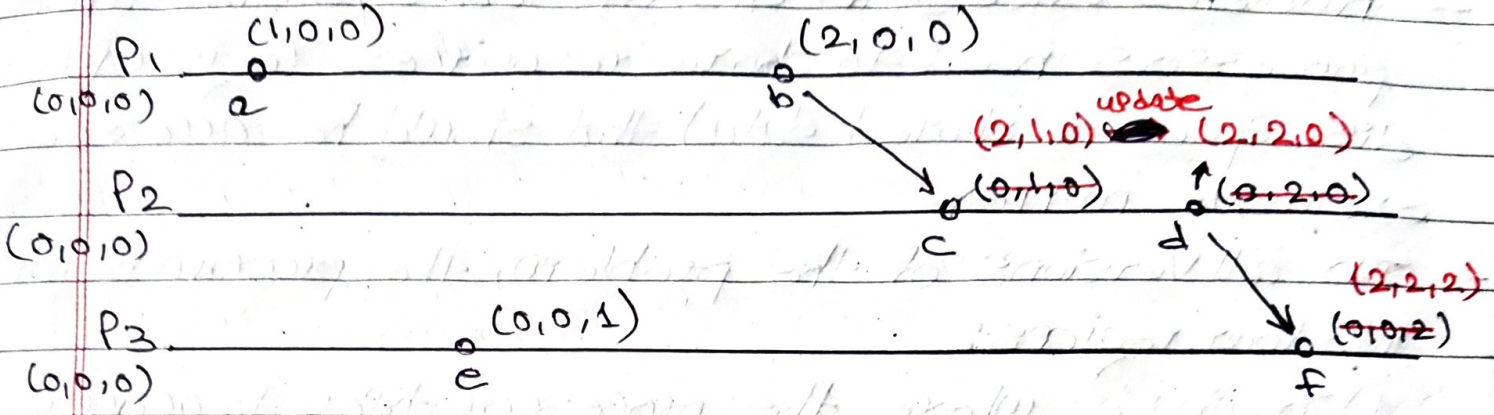
* Vector clock:

- Vector clock is an algorithm that generates partial ordering of events and also detects causality violations in a distributed system.
- This algorithm helps us label each process with a vector (a list of integer) i.e. An integer for each logical clock of every process within the system.

* Algorithm:

1) Initially, all the clocks are set to zero.
   $V_i[j] = 0$, for $i, j = 1, 2, \ldots, N$
2) Every time, an internal event occurs in a process, the value of the processe's logical clock in the vector is incremented by 1.
   $V_i[i] = V_i[i] + 1$
3) Every time a Process P_i sends a message with V_i attached to it.
4) When another P_i receives message, compares vector elements, & set Local Vector to higher (max) among two values
   $V_j[i] = \max(V_i[i], V_j[i])$.

example:

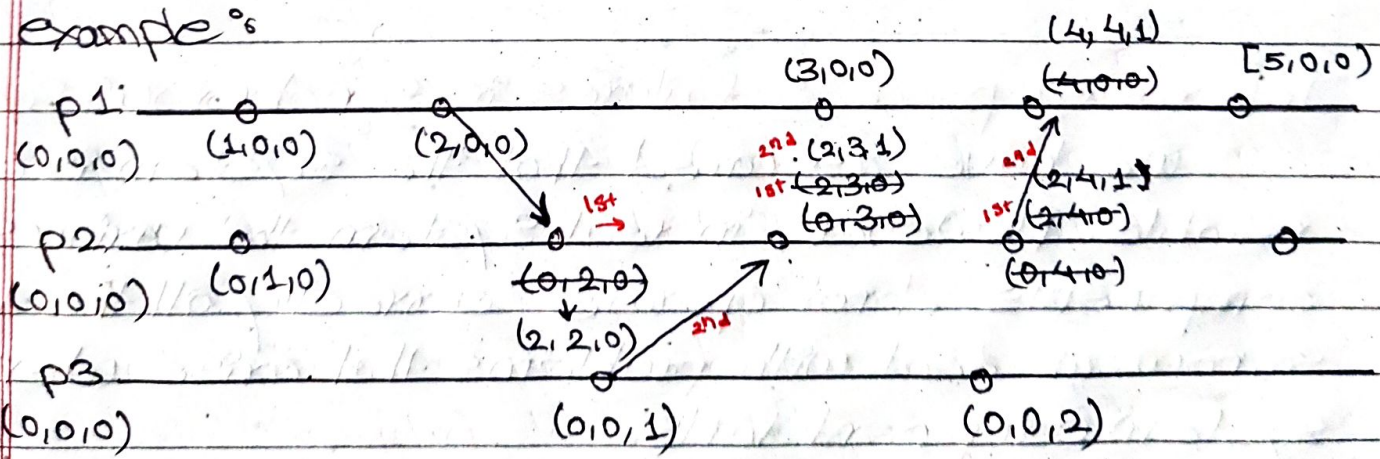


- Comparing vector timestamps
  1) $V = V'$ iff $V[i] = V'[i]$ for $i = 1, 2, \ldots, N$
  2) $V \leq V'$ iff $V[i] \leq V'[i]$ for $i = 1, 2, \ldots, N$

- For any two events e & e'
  1) If $e \rightarrow e'$ then $v(e) < v(e')$
  2) If $v(e) < v(e')$ then $e \rightarrow e'$

- Two events e and e' are concurrent iff neither $v(e) < v(e')$ is true nor $v(e') < v(e)$ is true.

example:



Page No.
Date : / /

* Resource Allocation in Distributed System:

- Problems arises in distributed systems when processors need to share resources (e.g.: CPU time, disk space or shared data) that should be accessed in an exclusive manner.
- In all versions of the problem, the program is partitioned into four regions:
  1) Trying: where the processor tries to acquire the resources.
  2) critical: where the resources are used.
  3) exit: where some cleanup is performed. (optional)
  4) Remainder: The rest of the code.

- When the processor is not interested in the shared resources, it is in the remainder
- To gain access to them it executes the trying region
- Eventually, entering critical region to use them.
- Afterwards, the processor may announces that it is done, in the exit region.

* Tolerating process failures in Synchronous System:

We have assumed that the system is completely reliable. However, in real system the various components donot operate correctly all the time. So, now we deal with problems that arise when system component fails.

- We start with the tolerating process failures in synchronous message passing systems.

- It is assumed that communication links are reliable, but processors are not reliable.
- We Concentrate on Coordination problems, which agree on outputs, based on their inputs. Such problems are typically very easy to solve in reliable systems
- Basically Two types of failure:
  i) Crash failure,
  ii) Byzantine failure.

i) Crash Failure:

- In Crash failure Processors halt during execution.
- Specifically, a processor crashes in round r, if it correctly operates in round 1, 2, ... r-1, . So it doesn't take a step in round r+1 and on.
- A processor that crashes is called faulty.
- Once a processor crashes, it is of no interest to the algorithm & no requirements are made on its decision.
- The validity condition implies that if all processors have the same input v, then all non faulty processors should decide on v.

ii) Byzantine failure:

- In Byzantine failure, failed processors can behave randomly.
- This is one of the most severe type of processor failure in distributed computing.
- The Faulty processors are often called Byzantine failure
- It can send different messages to different processors (or not send messages at all) when it is supposed to send.

the same messages.
- Moreover, the faulty processors may coordinate their actions.

* Sparse network Covers :

- In sparse network covers we take a collection of sub-graphs that covers all the nodes of the graphs and conduct computations inside each subgraph and between neighboring subgraphs.
- A cover is sparse if each node is covered by few sub-graphs.
- This method can be tuned to save alot on communication, time and memory.
- Sparse network covers are typically constructed during a pre-processing stage.
- Also the cost of an algorithm using a specific network cover is often proportional to the volume of the cover.

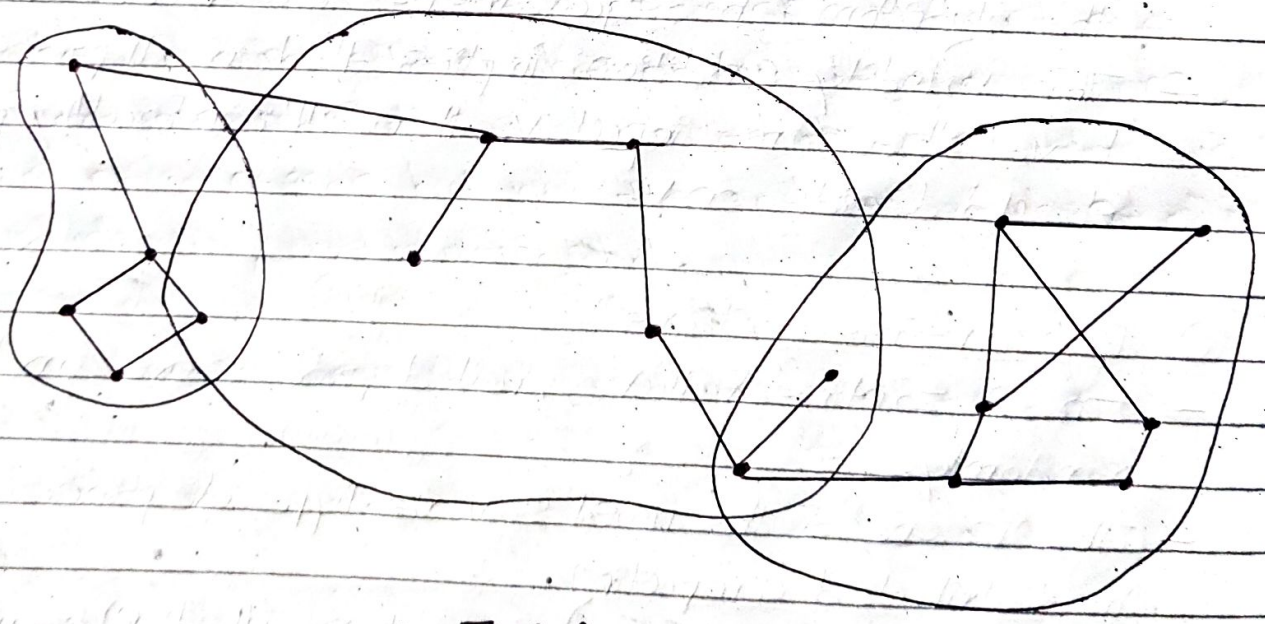


Fig : A cover

The two of the applications of a network as sparse

Page No.
Date : / /

covers are:
- To message routing,
- for reducing message complexity of algorithm,
- to obtain efficient synchronizers.

* Tolerating processor failures in Asynchronous Systems:

Let us assume a situation where a process P no longer perceives any actions from another process Q. However, can P conclude that Q has indeed come to a halt? In asynchronous system, ∴ No assumptions about process executions speeds or Message delivery times are made. So, Process P ~~be~~ when no longer perceives any actions from Q cannot conclude that Q crashed. Instead it assumes, it may just be slow or its messages may have been lost.

In Asynchronous system, consensus problem cannot be solved deterministically. If the system is asynchronous, then there is no deterministic consensus algorithm, even in the presence of single processor failure. This impossibility of Achieving consensus can be overcome by using Randomization or by solving weaker coordination problems. Problems with weaker requirements than consensus can help sidestep the impossibility results. Some of the methods used are approximate agreement, K-consensus & renaming.

Page No.
Date : / /

* Wait-free implementation of shared objects :

Informally, an implementation is a set of procedures, one for each operation of the high-level (implemented) object. The procedures use only operations supported by the low-level (implemented) objects. The operations generated by the procedures should look as if they were applied directly to the high level objects.

In past, for implementing shared objects relied on mutual exclusion which is very sensitive to processor failures and delays. If a processor inside the critical section fails or is very slow then all processor attempting to access this object are delayed.

Wait-free implementations guarantees that each processor completes each procedure (i.e, implementation of a high-level operation) within a finite number of low-level operations regardless of the behavior of other processor. If several high-level operations are in progress, at least one completes within finite number of low-level operations. Also, wait-free implementations of shared objects tolerates the failures of processes but not the failures of base object from which they are implemented.

Page No
Date : / /

Unit 2 : Models

2.1 PRAM Models :

* Introduction :

- Parallel processing can be defined as a technique for increasing the computation speed for a task, by dividing the algorithm into several sub tasks and allocating multiple processors to execute multiple sub tasks simultaneously.
- The main objective of exploration of parallel processing is to achieve higher efficiency of computation as compared to sequential computation.
- Parallel Random Access Machine models.
- example :

Finding the smallest value in the set of N numbers. There are two possible ways :

1) Algorithm 1 (Sequential) :

All possible comparisons of the pairs of the elements from set of numbers and carried out simultaneously each processor executing one operation of comparison. (We will not explain Sequential method rather focus on parallel i.e. Algorith 2)

2) Algorithm 2 (parallel) :

- Do in Parallel (where Actual required no. of processor is given by)

$$P = \frac{n(n-1)}{2}$$

- P_i derives the pair (i_1, i_2) of indices that corresponds to a different i.
- P_i reads value $L(i_1)$ and $L(i_2)$.
- IF $L(i_1) \geq L(i_2)$ then P_i sends negative outcomes to P_{i_1} else P_i sends negative outcomes to P_{i_2}.
- As this stage the only active processors is P_j, $1 \leq j \leq n$, which didnot receives a negative outcome
- P_i reads the values of $L(i)$ and write it into the output cell.

index = 1 2 3 4

example: Let Array $L = \{2, 6, 4, 8\}$

no. of elements = 4

So, no. of required processors $(P) = \frac{n(n-1)}{2}$

$= \frac{4*3}{2}$

$= 6$

Step i): P_i derives the pair (i_1, i_2) of indices that corresponds to a different i.

ie which process compares which two numbers in array is decided by formula $i = \frac{(i_2-1)(i_2-2)}{2} + i_1$ where, i is index of number

So,

1. $\text{index}(1,2) = \frac{(i_2-1)(i_2-2)}{2} + i_1 = \frac{(2-1)(2-2)}{2} + 1 = 1 \rightarrow P_1$
2. $\text{index}(1,3) = \frac{(i_2-1)(i_2-2)}{2} + i_1 = \frac{(3-1)(3-2)}{2} + 1 = 2 \rightarrow P_2$
3. $\text{index}(1,4) = \frac{(i_2-1)(i_2-2)}{2} + i_1 = \frac{(4-1)(4-2)}{2} + 1 = 4 \rightarrow P_4$
4. $\text{index}(2,3) = \frac{(i_2-1)(i_2-2)}{2} + i_1 = \frac{(3-1)(3-2)}{2} + 2 = 3 \rightarrow P_3$

Page No
Date : / /

5. index $(2,4) = \frac{(4-1)(4-2)}{2} + 2 = 5 \rightarrow P_5$

6. index $(3,4) = \frac{(4-1)(4-2)}{2} + 3 = 6 \rightarrow P_6$

step(ii) : Pi reads value $L(i_1)$ and $L(i_2)$ and step (3) assign -ve by comparing. $L(i_1) < L(i_2)$

1. $P_1 \rightarrow$ index $(1,2) \rightarrow$ value $(2,6) \rightarrow$ -ve $\rightarrow P_2$
2. $P_2 \rightarrow$ index $(1,3) \rightarrow$ value $(2,4) \rightarrow$ -ve $\rightarrow P_3$ ($L(i_1) < L(i_3)$)
3. $P_3 \rightarrow$ index $(2,3) \rightarrow$ value $(6,4) \rightarrow$ -ve $\rightarrow P_2$ ($L(i_2) > L(i_3)$)
4. $P_4 \rightarrow$ index $(1,4) \rightarrow$ value $(2,8) \rightarrow$ -ve $\rightarrow P_4$ ($L(i_1) < L(i_4)$)
5. $P_5 \rightarrow$ index $(2,4) \rightarrow$ value $(6,8) \rightarrow$ -ve $\rightarrow P_4$ ($L(i_2) < L(i_4)$)
6. $P_6 \rightarrow$ index $(3,4) \rightarrow$ value $(4,8) \rightarrow$ -ve $\rightarrow P_4$ ($L(i_3) < L(i_4)$)

step(iii) : As this stage the only active processors is P_i, $1 \le i \le n$, which didn't receives a negative outcomes = our total ~~process are~~ numbers are $(n) = 4$

where, P_2, P_3 & P_4 are negative (-ve) according to step(ii), The only processor which is not negative is P_1 and the index of P_1 is 1 which values is = 2

step(iv) : Pi reads the values of $L(i)$ & write it into the output cell.

So, the output is 2 which is smallest in the given array.

Page No.
Date : / /

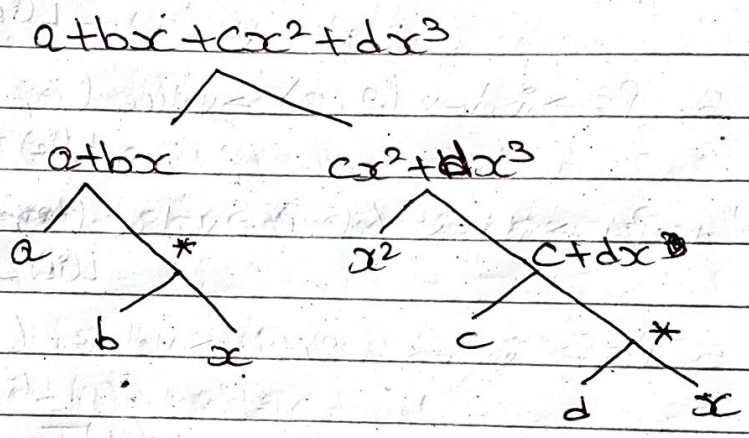
* Techniques for the design of parallel algorithm :

A number of basic techniques are employed in the design of parallel Algorithms. Some of them are:

1) Divide and conquer technique:

A given problem is divided into a number of independent subproblems that are dealt with recursively.

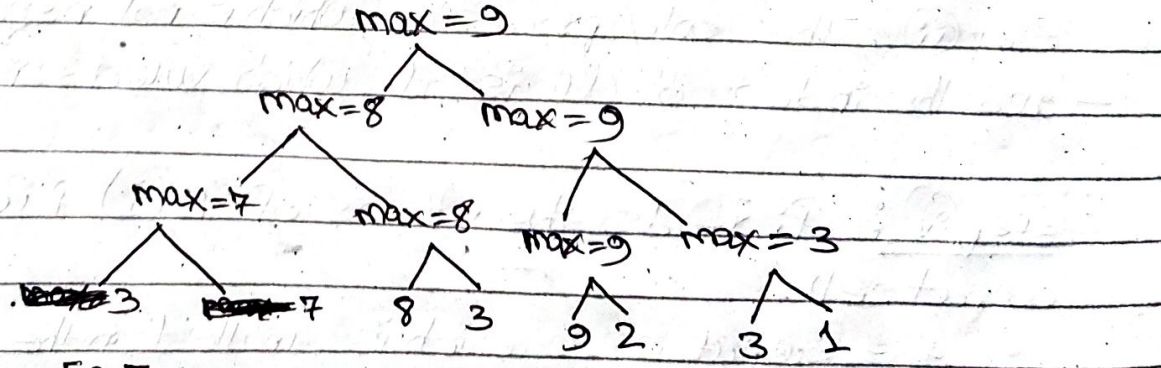
for e.g: Let the problem be $a+bx+cx^2+dx^3$ then.



2) Balanced binary tree method:

The problems are solved in bottom-up order, with those at the same depth in the tree being computed in parallel. here, each internal node corresponds to the computation of a subproblem.

for e.g: Find maximum of n elements.



[3,7,8,3,9,2,3,1] → [7,8,9,3] → [8,9] → [9]

Page No.
Date : / /

* PRAM Model :

PRAMs are classified according to restrictions on shared memory access. The read and write conflicts are resolved by allowing one of the following mechanisms:

i) Exclusive - read, exclusive write PRAM model (EREW):

The most common model, EREW PRAM, doesnot allow simultaneous access by more than one processor to the same memory location for read or write purposes.

ii) Concurrent - read, exclusive write PRAM model (CREW):

The CREW model allows multiple processors to read a memory cell but only one can write at a time.

iii) Exclusive read, concurrent write (ERCW) :

The ERCW model allows only one processor to read a memory cell while allowing multiple processors to write at a same time. This model is never considered.

iv) Concurrent read, Concurrent write (CRCW) :

The CRCW model allows concurrent access by more than one processor to the same memory location for both read and write purposes.

ISSUES : On concurrent write it can be assumed that all the processors are writing the same value otherwise, priority can be assigned to solve the write conflicts.

Page No.
Date : / /

* Optimality and Efficiency of Parallel Algorithms?

We will consider the most obvious and most important computing resources i.e Time and number of processor p

- Let A be a problem instance of size N.
- Assume that Problem A can be solved on a PRAM by a Parallel Algorithm P_A in time T(N) by employing p(N) processo
- The work of a parallel algorithm P_A is the product of its time, T(N) and the number of processors p(N) i.e

$$W(N) = T(N) * P(N)$$

The two most important criteria for evaluation of parallel algorithms are Speedup and efficiency of a parallel algorithm.

where,

$$\text{Speedup} = \frac{t(N)}{T(N)} \quad \text{where,}$$

t(n) = Sequential time
T(N) = parallel time

$$\text{Efficiency} = \frac{\text{Speedup}}{P(N)}$$

Page No.
Date : / /

* Basic PRAM algorithms :

1. Computing Prefix Sums,
2. List Ranking
3. Parallel Sorting Algorithms.

1. Computing prefix sums :

- let a given Array, $A = \{a_0, a_1, a_2, \ldots, a_{n-1}\}$
- we have to compute,
  $\{a_0, (a_0+a_1), (a_0+a_1+a_2), \ldots, (a_0+a_1+\ldots+a_{n-1})\}$
- for example :
  $A = \{5, 3, -6, 2, 7, 10, -2, 8\}$

Then output is

$= \{5, 8, 2, 4, 11, 21, 19, 27\}$

* Algorithm : Parallel Prefix Sum

- Input an array $[a_1, a_2, \ldots, a_n]$
- IF ($n == 1$) then $S_1 \leftarrow a[1]$

ELSE,

```
for j = 0 to log n - 1 do
    for i = 2^j + 1 to n do in parallel
        processor Pi
    1. obtain a[i - 2^j] from P(i - 2^j) through shared
       memory.
    2. a[i] = a[i - 2^j] + a[i]
```

* For example :

Let an Array $A = \{5, 3, -6, 2, 7, 10, -2, 8\}$

here,

$n = 8$ So, $\log_2 8 = 3 \rightarrow \log n - 1 = 3 - 1 = 2$

So, for j = 0 to log n - 1 we have,

J = {0, 1, 2}

* Case (i): J = 0

for i = $2^j + 1$ to n A = {5, 3, -6, 2, 7, 10, -2, 8}
(1) (2) (3) (4) (5) (6) (7) (8)

= $2^0 + 1$ to 8

= 2 to 8.

a[i] = a[i - 2^j] + a[i] where i = 2 to 8 So,

i) a[2] = a[2 - 2^0] + a[2] = a[1] + a[2] = 5 + 3 = 8
ii) a[3] = a[3 - 2^0] + a[3] = a[2] + a[3] = 3 - 6 = -3.
iii) a[4] = a[4 - 2^0] + a[4] = a[3] + a[4] = -6 + 2 = -4
iv) a[5] = a[5 - 2^0] + a[5] = a[4] + a[5] = 2 + 7 = 9
v) a[6] = a[6 - 2^0] + a[6] = a[5] + a[6] = 7 + 10 = 17
vi) a[7] = a[7 - 2^0] + a[7] = a[6] + a[7] = 10 - 2 = 8
vii) a[8] = a[8 - 2^0] + a[8] = a[7] + a[8] = -2 + 8 = 6

So, After 1st Iteration new Array is
A = {5, 8, -3, -4, 9, 17, 8, 6}

* Case (ii): J = 1

for i = $2^j + 1$ to n new Array,
(1) (2) (3) (4) (5) (6) (7) (8)
A = {5, 8, -3, -4, 9, 17, 8, 6}

= $2^1 + 1$ to 8

= 3 to 8

a[i] = a[i - 2^j] + a[i] where i = 3 to 8

Page No.
Date : / /

i) a[3] = a[3 - 2^1] + a[3] = a[1] + a[3] = 5 - 3 = 2
ii) a[4] = a[4 - 2^1] + a[4] = a[2] + a[4] = 8 - 4 = 4
iii) a[5] = a[5 - 2^1] + a[5] = a[3] + a[5] = -3 + 9 = 6
iv) a[6] = a[6 - 2^1] + a[6] = a[4] + a[6] = -4 + 17 = 13
v) a[7] = a[7 - 2^1] + a[7] = a[5] + a[7] = 9 + 8 = 17
vi) a[8] = a[8 - 2^1] + a[8] = a[6] + a[8] = 17 + 6 = 23

So, After 2nd iteration new Array is
A = {5, 8, 2, 4, 6, 13, 17, 23}

* Case (iii) : J = 2
for i = 2^J + 1 to n
= 2^2 + 1 to 8
= 5 to 8

new Array;
1 2 3 4 5 6 7 8
A = {5, 8, 2, 4, 6, 13, 17, 23}

a[i] = a[i - 2^J] + a[i] where i = 5 to 8

i) a[5] = a[5 - 2^2] + a[5] = a[1] + a[5] = 5 + 6 = 11
ii) a[6] = a[6 - 2^2] + a[6] = a[2] + a[6] = 8 + 13 = 21
iii) a[7] = a[7 - 2^2] + a[7] = a[3] + a[7] = 2 + 17 = 19
iv) a[8] = a[8 - 2^2] + a[8] = a[4] + a[8] = 4 + 23 = 27

So, After 3rd iteration new Array is

A = {5, 8, 2, 4, 11, 21, 19, 27}
which is our final answer.

Page No.
Date : / /

2. List Ranking :

- The Problem is that, given a singly linked list L, with N objects, compute the distance between each node and the last node in the end of the list.
- If d denotes the distance,

$$node.d = \begin{cases} 0 & \text{if } node.next = nil \\ node.next.d + 1, & \text{otherwise} \end{cases}$$

* Algorithm : Parallel List Ranking

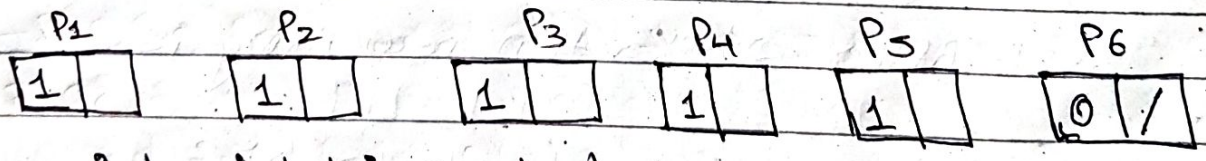
- Assign one processor for each node
- for each node i, do in parallel
  - ~~i.d = i.d + i.next.d~~
  - ~~i~~
  - i.d = i.d + i.next.d
  - i.next = i.next.next where, i = index of n
    d = distance.

* Illustration :

Let no. of Processor = 6

then,

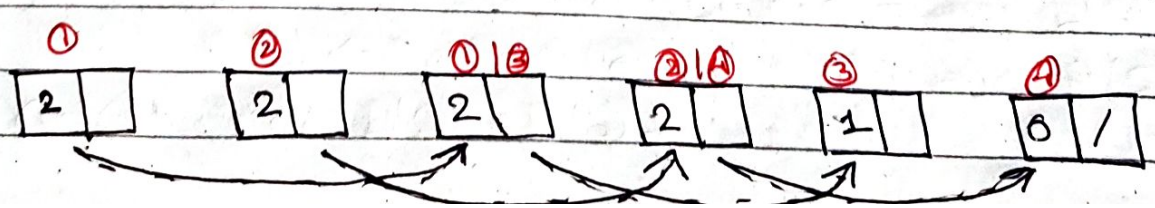
1)

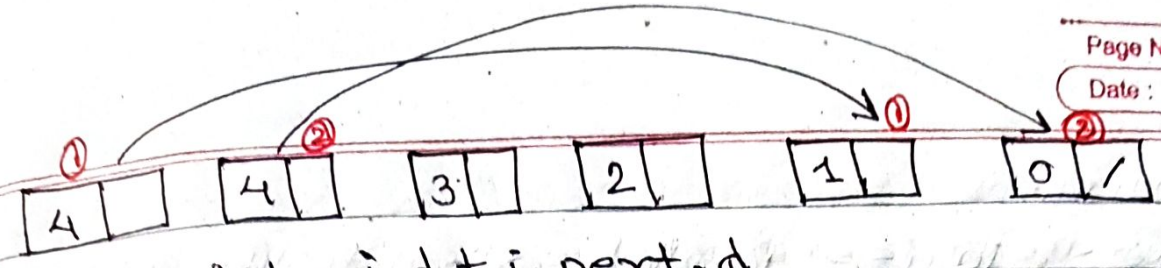


i.d = i.d + i.next.d

i.next = i.next.next So,

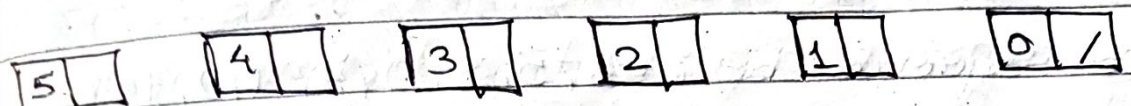
2)





3) i.d = i.d + i.next.d

i.next = i.next.next



4)

which is the final result with respective distance of each node from the final node in the list.

3. Parallel Sorting Algorithm:

- Bitonic Sort i.e. consists of two sequences, one increasing and one decreasing with even number of elements.
- for eg:- 5, 6, 7, 8, (1st seq) 4, 3, 2, 1 (2nd seq)
- It is a sequence of elements $\langle a_0, a_1, \dots, a_{n-1} \rangle$ there exists $\langle a_0, a_1, \dots, a_i \rangle$ is increasing and $\langle a_{i+1}, a_{i+2}, \dots, a_{n-1} \rangle$ is decreasing
- If $a_{n/2}$ is the beginning of the decreasing sequence S, then

$$L(S) = \{ \min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \dots \min(a_{n/2-1}, a_{n-1}) \}$$
$$R(S) = \{ \max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \dots, \max(a_{n/2-1}, a_{n-1}) \}$$

i.e.

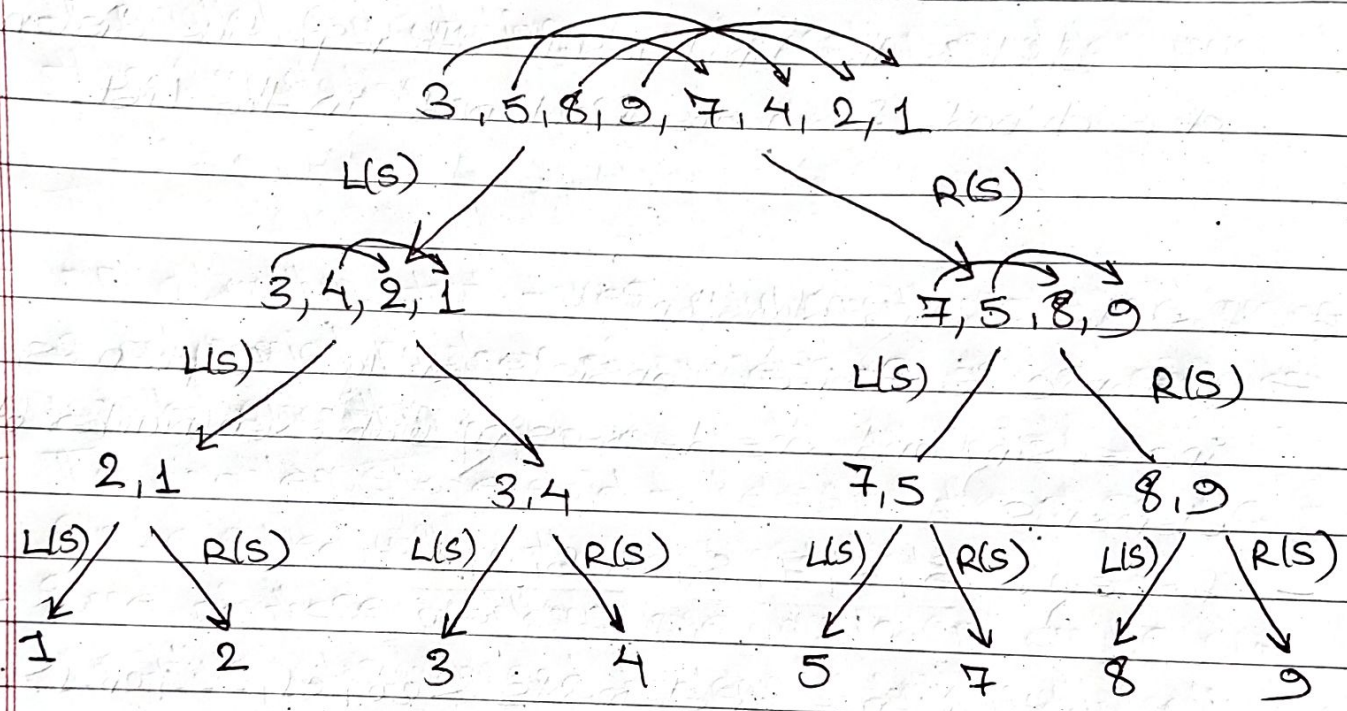
- Basically, compare 1st element of 1st sequence with first element of 2nd sequence and put
- ① L(S) = smallest among two in left side,
- ② R(S) = largest among two in right side.

Page No.
Date : / /

* Illustration:

Note: To use Parallel Sorting Algorithm the given sequence must be a bitonic sequence S

Let the sequence be: 3, 5, 8, 9, 7, 4, 2, 1
(1st seq | 2nd seq)



So the Last Seq = {1, 2, 3, 4, 5, 7, 8, 9}

* Algorithm:

- Input = a bitonic sequence S
- If S is the length of 1 then STOP
- ELSE
  - form L(S) & R(S)
  - Do in parallel
    - L(S) ← Recursive bitonic merge L(S)
    - R(S) ← Recursive bitonic merge R(S)
- Concatenate L(S) & R(S).

Page No.
Date : / /

* Randomized Algorithm:

- Randomized Algorithm is the algorithm whose execution is controlled at one or more points by randomly made choices.
- An early example of randomized algorithm is the work done on primality testing by Rabin.

Primality Testing.

- Fermat's Little Theorem

i.e $b^p \equiv b \bmod p$

where; $b \in Z_p$

* example:

$b^p \equiv b \bmod p$ where, P = Prime no.

$b \in Z_p$ also $Z_n = \{0, 1, 2, \ldots, n-1\}$

i.e set of residue

$Z_5 = \{0, 1, 2, 3, 4\}$

→ Any numbers divided by 5 gives remainder $\{0, 1, 2, 3, 4\}$

Let us assume $p = 5$

then, $b = \{0, 1, 2, 3, 4\}$

so,

$0^5 \bmod 5 = 0$

$1^5 \bmod 5 = 1$

$2^5 \bmod 5 = 2$

$3^5 \bmod 5 = 3$

$4^5 \bmod 5 = 4$

Page No.
Date : / /

* Deficiencies of PRAM algorithm:

The major drawback of the PRAM model are:

- It has no mechanism for representing for representing communications between the processors.
- It's storage management and communication issues are hidden from the algorithm designer.
- In parallel computation, communication is often much slower than computation.

* The NC-class:

- An algorithm that takes polylogarithmic time using a polynomial number of processors is an efficient parallel algorithm.
- The problems are said to belong to the class NC if they can be solved within these constraints.
- Some of these problems are basic arithmatic operations, transitive closure and Boolean matrix multiplication etc.

* P-Completeness : Hardly parallelizable Problems:

- P is the class of Problems that are solvable in polynomial sequential time.
- Efficiently parallelizable problems that we know of are in NC and in contrast, there have been identified certain problems which do not seem to admit efficient parallelization readily called "Hardly parallelizable" which form the class of P-completeness problem.
- A Problem $L \in P$ is said to be P-complete if every other Problem in P can be transformed to L in polylogarithmic parallel time using a polynomial number of processors.

Page No.
Date : / /

2.2 Broadcasting with Selective Reduction : A Powerful Model of Parallel Computations.

* Introduction :

- Broadcasting with Selective Reduction (BSR) is a model of parallel computation in which N processors share M memory locations.
- In BSR a additional type of memory access is permitted by which all processors may gain access to all memory locations at the same time for purpose of writing.
- At each memory location, a subset of the incoming broadcast data is selected (according to one or more appropriate selection criteria) and reduced to one value (using an appropriate reducing operator), which is stored in the memory location.
- Broadcast instruction consists of three phases :
  i) Broadcast phase,
  ii) Selection phase,
  iii) Reduction phase.

i) Broadcast phase :

- Allow all of the N processors to write concurrently to all of the M memory locations.
- Each processor P_i, $1 \leq i \leq N$, produced a record containing two fields, a tag t_i and a datum d_i, to be stored.

ii) Selection phase :

- After the data are received at each memory locations U_i, $1 \leq i \leq M$, a switch S_i will select the receiving data d_i by comparing tag value t_i by using a selection rule, $\sigma [<, \leq, =, >, \geq, \neq]$.

iii) Reduction phase:

- Selected data are reduced to a single value using reduction rule, $R \in \Sigma, \pi, \wedge, \vee, \oplus, \cap, \cup$ $\left(\max, \min\right)$ $\in$

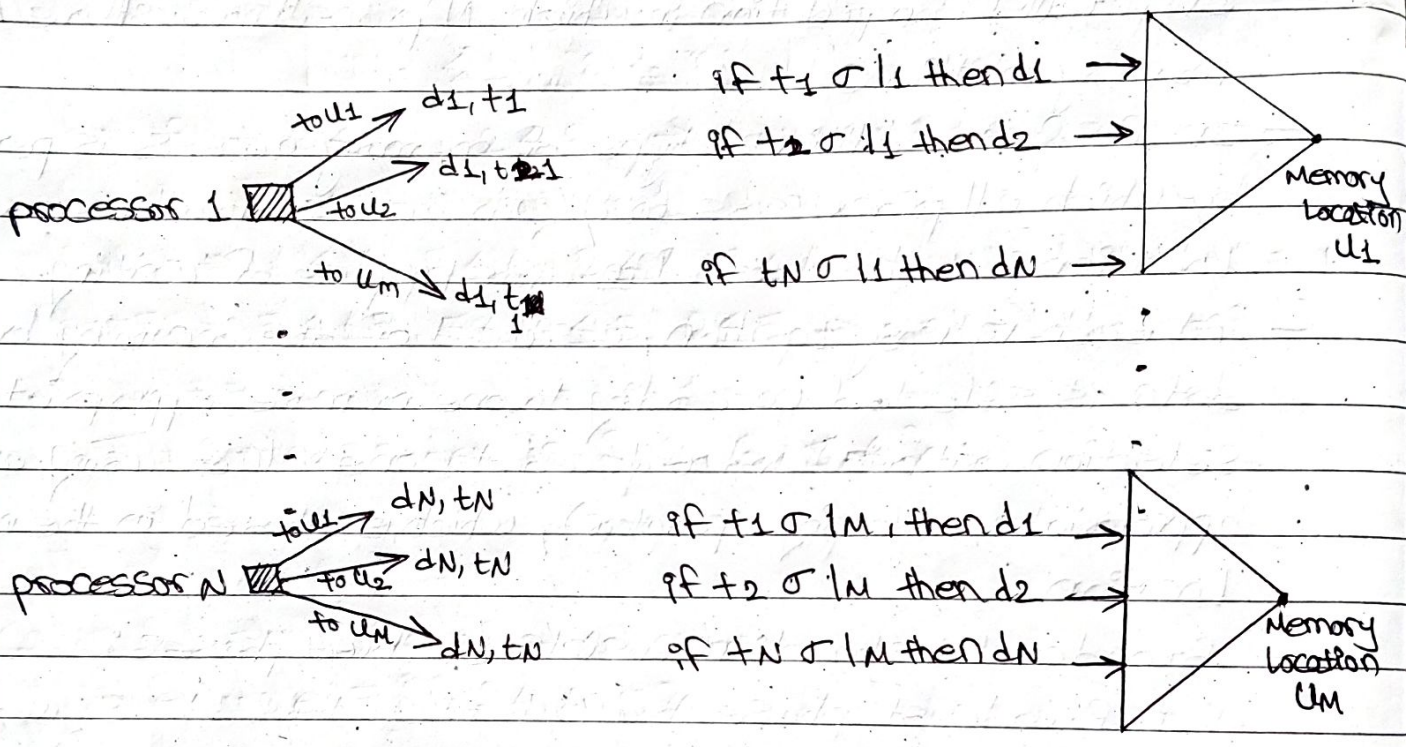


Fig of :- Broadcasting with selective reduction

* A Generalized BSR Model:

Mathmatically, it can be expressed as:

$$U_j \leftarrow R_{d_i} \left| \wedge_{t(i,h)} \sigma_h \right| (i,h)$$

$1 \le j \le M \quad 1 \le i \le N \quad 1 \le h \le K \quad 1 \le j \le M$

where,

- N → no. of processors
- d_i → datum broadcast by the processor P_i

Page No.
Date : / /

- σ_h → Selection operation, $1 \leq h \leq K$, where K is the number of criteria.
- t(i,h) → tag broadcast by processor i for criteria h
- l(i,h) → limit value i for criteria σ_h
- R → reduction operation, $R \in \{\Sigma, \pi, \wedge, \vee, \oplus, \cap, \cup\}$
- Vi → memory location to be stored for the single reduced value

* BSR Types:

- One criterion BSR algorithm,
- Two criterion BSR algorithm,
- Three criterion BSR algorithm,
- Multiple criterion BSR algorithm.

① One criterion BSR algorithm:

BSR algorithm requiring each broadcast datum to pass a single test before being allowed to participate in the reduction process.

example:

- Sorting,
- Parenthesis Matching,
- Optimal Sum Subsegment.

i) Sorting:

- This algorithm consists of three Broadcast instructions. The first instruction gives the rank r_i of element x_i in the array $x_1, x_2, x_3, \ldots, x_n$, i.e. the number of elements in the array that are smaller than or equal to x_i.

$$r_i = \Sigma 1 \mid x_i \leq x_j$$

- Number of processor needed = number of elements in array i.e. N

Page No.
Date : / /

case (i): When all the elements in the array are distinct, every datum x_j in it's position r_j are in the sorted array

for example: $A = \{3, 2, 6, 5\}$
$P_1 \quad P_2 \quad P_3 \quad P_4$

$$r_j = \leq 1 \mid x_i \leq x_j$$

for 3 → $3 \leq 3$, $2 \leq 3$ → rank = 1 + 1 = 2
for 2 → $2 \leq 2$ → rank = 1
for 6 → $6 \leq 6$, $5 \leq 6$, $2 \leq 6$, $3 \leq 6$ → rank = 1+1+1+1 = 4
for 5 → $5 \leq 5$, $3 \leq 5$, $2 \leq 5$ → rank = 1+1+1 = 3 So,

The sorted order is $\{2, 3, 5, 6\}$

case (ii): In case of duplicate,
for example: $A = \{4, 4, 5\}$
for 4 → $4 \leq 4$, $4 \leq 4$ → rank = 1+1 = 2
for 4 → $4 \leq 4$, $4 \leq 4$ → rank = 1+1 = 2
for 5 → $4 \leq 5$, $4 \leq 5$, $5 \leq 5$ → rank = 1+1+1 = 3

both 2nd rank incorrect

So,

$$S_j = \leq 1 \mid t_i \leq l_i \text{ where } t_i = r_i - \frac{1}{i}, l_i = r_j - \frac{1}{j}$$

for example: $A = \{3, 4, 3\}$

i) $A[1] = 3$

$t_1 = 3 - \frac{1}{1} \leq l_1 = 3 - \frac{1}{1}$ → True

$t_2 = 4 - \frac{1}{2} \leq l_1 = 3 - \frac{1}{1}$ → False

$t_3 = 3 - \frac{1}{3} \leq l_1 = 3 - \frac{1}{1}$ → False

Rank = 1st because only 1 True.

Page No.
Date : / /

ii) $A[2] = 4$

$t_1 = 3 - \frac{1}{1} \leq l_2 = 4 - \frac{1}{2}$ → True

$t_2 = 4 - \frac{1}{2} \leq l_2 = 4 - \frac{1}{2}$ → True

$t_3 = 3 - \frac{1}{3} \leq l_2 = 4 - \frac{1}{2}$ → True

All are True
So, Rank = 3rd

iii) $A[3] = 3$

$t_1 = 3 - \frac{1}{1} \leq l_3 = 3 - \frac{1}{3}$ → True

$t_2 = 4 - \frac{1}{2} \leq l_3 = 3 - \frac{1}{3}$ → False

$t_3 = 3 - \frac{1}{3} \leq l_3 = 3 - \frac{1}{3}$ → True

Two are True
So, rank = 2nd

ADA

Parenthesis Matching:

- One criteria BSR algorithm
- The problem is to find the pairs of matching parenthesis in a given legal sequences $l_1, l_2, \ldots, l_n$ of parenthesis.
- By legal means that every parenthesis has its matching parenthesis in a sequence
- example: output for the input sequence,

  () ((())) (()) is,

Page No.
Date : / /

(	)	(	(	(	)	)	)	(	(	)	)
1	2	3	4	5	6	7	8	9	10	11	12
2	1	8	7	6	5	4	3	12	11	10	9

- Can be solved by N processors.

* Algorithm :

- for each processor i do in parallel
1. - If $b_i == '('$ then $b_i = 1$ else $b_i = -1$
2. - $p_i = \sum b_j \mid i \le j$
3. - If $b_i == -1$ then $p_i = p_i + 1$
4. - $p'_i = p_i - \frac{1}{i}$
5. - $q_i = -1,\ t_i = 0,\ r_i = 0$
6. - $q_i = \cap p'_i \mid p'_i < p'_j$
7. - $t_i = \cap i \mid p'_i = q_i$
8. - $r_i = U_i \mid t_i = j$
9. - If $b_i = ')'$ then $m_i = t_i$ else $m_i = r_i$

Note: $\cap i$ & U_i look for the index.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
Sequence	(	)	(	(	(	)	)	)	(	(	)	)
2 index (i)	1	2	3	4	5	6	7	8	9	10	11	12
3 b_i	1	-1	1	1	1	-1	-1	-1	1	1	-1	-1
4 $p_i = \sum b_j$	1	0	1	2	3	2	1	0	1	2	1	0
5 p_i	1	1	1	2	3	3	2	1	1	2	2	1
6 p'_i	0	0.5	0.66	1.75	2.8	2.83	1.86	0.87	0.89	1.9	1.91	0.92
7 q_i (init)	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
8 q_i (act)	-1	0	0.5	0.92	1.91	2.8	1.75	0.66	0.87	1.86	1.9	0.89
t_i	0	1	2	12	11	5	4	3	8	7	10	9
r_i	2	3	8	7	6	0	10	9	12	11	5	4
m_i	2	1	8	7	6	5	4	3	12	11	10	9

Page No.
Date : / /

① If $d_i == '('$ then $b_i = 1$ else $b_i = -1$
→ if index has open bracket '(' then $b_i = 1$
→ if index has close bracket ')' then $b_i = -1$

② $P_i = \Sigma b_i \mid i \le j$
→ Consecutive sum of b_i from P_1 to P_{12}

$\Sigma b_1 = i \le 1 \quad = 1$
$\Sigma b_2 = i \le 2 \rightarrow b_1 + b_2 = 0$
$\Sigma b_3 = i \le 3 \rightarrow b_1 + b_2 + b_3 = 0 + 1 = 1$
$\Sigma b_4 = i \le 4 \rightarrow b_1 + b_2 + b_3 + b_4 = 1 + 1 = 2$
$\Sigma b_5 = 2 + 1 = 3$
$\Sigma b_6 = 3 - 1 = 2$
$\Sigma b_7 = 2 - 1 = 1$
$\Sigma b_8 = 1 - 1 = 0$
$\Sigma b_9 = 0 + 1 = 1$
$\Sigma b_{10} = 1 + 1 = 2$
$\Sigma b_{11} = 2 - 1 = 1$
$\Sigma b_{12} = 1 - 1 = 0$

③ If $b_i == -1$ then $P_i = P_i + 1$
Add +1 to every $\Sigma b_i (P_i)$ whose $b_i = -1$

④ $P'_j = P_j - \frac{1}{j}$ (whose P_i from 5th row)

$P'_1 = P_1 - \frac{1}{1} = 1 - \frac{1}{1} = 0$

$P'_2 = P_2 - \frac{1}{2} = 1 - \frac{1}{2} = 0.5$

$P'_3 = P_3 - \frac{1}{3} = 1 - \frac{1}{3} = 0.67$

$P'_4 = P_4 - \frac{1}{4} = 2 - \frac{1}{4} = 1.75$

$P'_5 = P_5 - \frac{1}{5} = 3 - \frac{1}{5} = 2.8$

$P'_6 = P_6 - \frac{1}{6} = 3 - \frac{1}{6} = 2.833$

$P'_7 = P_7 - \frac{1}{7} = 2 - \frac{1}{7} = 1.86$

$P'_8 = P_8 - \frac{1}{8} = 1 - \frac{1}{8} = 0.875$

$P'_9 = P_9 - \frac{1}{9} = 1 - \frac{1}{9} = 0.89$

$P'_{10} = P_{10} - \frac{1}{10} = 2 - \frac{1}{10} = 1.9$

$P'_{11} = P_{11} - \frac{1}{11} = 2 - \frac{1}{11} = 1.91$

$P'_{12} = P_{12} - \frac{1}{12} = 1 - \frac{1}{12} = 0.9167$

Page No.
Date : / /

(5) $q_j = -1,\ t_j = 0,\ r_j = 0$

Set, $q_i = -1$ to all as default value.

(6) $q_j = \cap p'_i \mid p'_i < p'_j$

→ use the row 6th for p'_j where select all value in 6th row which less than p'_j and select maximum($\cap$) among them assign to q_j i.e.

1. $q_1 = \cap p'_i \mid p'_i < p'_1$
   $= \cap p'_i \mid p'_i < 0$
   $=$ no $p'_i < 0$ so default (-1)

2. $q_2 = \cap p'_i \mid p'_i < p'_2$
   $= \cap p'_i \mid p'_i < 0.5$
   $= 0 < 0.5$
   $= 0$

3. $q_3 = p'_i < p_3'$
   $= \{0, 0.5\} < 0.66$
   $= 0.5$ (max)

4. $q_4 = p'_i < p_4'$ (1.75)
   $= \{0, 0.5, 0.66, 0.87, 0.89, 0.92\} < (1.75)$
   (max)
   $= 0.92$

5. $q_5 = p'_i < p_5'$ (2.8)
   $= \{0, 0.5, 0.66, 1.75, 1.86, 0.87, 0.89, 1.9, 1.91, 0.92\}$
   $=$ max (1.91)

Page No.
Date : / /

6. $q_6 = p'_i < p'_6$ (2-83)
= 2-8

7. $q_7 = p'_i < p'_7$ (1-86)
= 1-75

8. $q_8 = p'_i < p'_8$ (0-87)
= 0-66

9. $q_9 = p'_i < p'_9$ (0-89)
= 0-87

10. $q_{10} = p'_i < p'_{10}$ (1-9)
= 1-86

11. $q_{11} = p'_i < p'_{11}$ (1-91)
= 1-9

12. $q_{12} = p'_i < p'_{12}$ (0-92)
= 0-89

(7) $t_i = \cap i \mid p'_i = q_i$

→ If $p'_i = q_i$ then take the index of p'_j also if there are two values in p'_j which are equal to q_i take ~~index of maximum~~ maximum index. In this problem there is no two values to compose with so,

i) $t_1 = \cap i \mid p'_i = q_1$
$= \cap i \mid p'_i = -1$
= No (-1) in p'_j so take default value 0

ii) $t_2 = \cap i \mid p'_i = q_2$
$= \cap i \mid p'_i = 0$
= index (1) where $p'_i = 0$

iii) $t_3 = \cap i \mid p'_i = q_3$
$= \cap i \mid p'_i = 0.5$
= index (2) where $p'_i = 0.5$

iv) t_4 = index (12)

v) t_5 = index (11)

vi) t_6 = index (5)

vii) t_7 = index (4)

viii) t_8 = index (3)

ix) t_9 = index (8)

x) t_{10} = index (7)

xi) t_{11} = index (10)

xii) t_{12} = index (9)

Page No.
Date : / /

(8) $r_j = U_i \mid t_i = j$

→ If $t_i = j$ then take the index where $t_i = j$ i.e.

i) $r_1 = U_i \mid t_i = 1$
$= t_i = 1$ in index (2)
$r_1 = 2$

ii) $r_2 = t_i = 2$
$=$ index 3

iii) $r_3 = t_i = 3$
$=$ index (8)

iv) $r_4 = t_i = 4$
$=$ index (7)

v) $r_5 = t_i = 5$
$=$ index (6)

vi) $r_6 = t_i = 6$
$=$ index (0)

vii) $r_7 = t_i = 7$
$=$ index (10)

viii) $r_8 = t_i = 8$
$=$ index (9)

ix) $r_9 = t_i = 9$
$=$ index (12)

x) $r_{10} = t_i = 10$
$=$ index (11)

xi) $= t_i = 11$
$=$ index (5)

xii) $t_i = 12$
$:=$ index (4)

(9) If $l_j = ')'$ then $m_j = t_j$ else $m_j = r_j$

= If index has close bracket ')' then $m_j = t_j$
If index has open bracket '(' then $m_j = r_j$

Page No.
Date : / /

iii) Maximum Sum Sub Segment:

- Given an array of numbers $d_1, d_2, \ldots, d_m$, it is required to find a contiguous sub array of maximumal sum.
- example:

$A = \{-2, -3, 4, -1, -2, 1, 5, -3\}$

Maximal Sum $= 4 + (-1) + (-2) + 1 + 5 = 7$

* Algorithm:

- $S_i = \Sigma d_i \mid i \leq j$
- $m_j = \cap S_i \mid i \geq j$
- $e_j = \cap i \mid S_i = m_j$
- $m_i = m_i - S_j + d_j$
- $t = \cap m_i$
- $x = \cap i \mid m_i = t \rightarrow$ starting point
- $y = e_x \rightarrow$ end point

Note: $\cap i$ means highest index in the Array.

* example: given $A = \{\overset{d_1}{31}, \overset{d_2}{-41}, \overset{d_3}{59}, \overset{d_4}{26}, \overset{d_5}{-53}, \overset{d_6}{58}, \overset{d_7}{97}, \overset{d_8}{-93}, \overset{d_9}{-23}, \overset{d_{10}}{84}\}$

i) $S_i = \Sigma d_i \mid i \leq j$

$S_1 = \Sigma d_i \mid i \leq 1 = 31$

$S_2 = \Sigma d_i \mid i \leq 2 = 31 + (-41) = -10$

$S_3 = 31 + (-41) + 59 = 49$

$S_4 = 31 - 41 + 59 + 26 = 75$

$S_5 = 31 - 41 + 59 + 26 - 53 = 22$

$S_6 = 31 - 41 + 59 + 26 - 53 + 58 = 80$

$S_7 = 31 - 41 + 59 + 26 - 53 + 58 + 97 = 177$

$S_8 = 31 - 41 + 59 + 26 - 53 + 58 + 97 - 93 = 84$

$S_9 = 31 - 41 + 59 + 26 - 53 + 58 + 97 - 93 - 23 = 61$

$S_{10} = 31 - 41 + 59 + 26 - 53 + 58 + 97 - 93 - 23 + 84 = 145$

(2) $m_j = \cap S_i \mid i \geq j$

i) $m_1 = \cap S_i \mid i \geq 1$ (1 to 10 max)
$= \max(S_i)$
$= 177$

ii) $m_2 = \cap S_i \mid i \geq 2$ (2 to 10 max)
$= \max(S_i)$
$= 177$

iii) $m_3 = \cap S_i \mid i \geq 3$ (3 to 10 max)
$= \max(S_i)$
$= 177$

iv) $m_4 = \cap S_i \mid i \geq 4$ (4 to 10 max)
$= 177$

v) $m_5 = \cap S_i \mid i \geq 5$ (5 to 10 max)
$= 177$

vi) $m_6 = \cap S_i \mid i \geq 6$ (6 to 10 max) $= 177$

vii) $m_7 = \cap S_i \mid i \geq 7$ (7 to 10 max) $= 177$

viii) $m_8 = \cap S_i \mid i \geq 8$ (8 to 10 max) $= 145$

ix) $m_9 = \cap S_i \mid i \geq 9$ (9 to 10 max) $= 145$

x) $m_{10} = \cap S_i \mid i \geq 10$ (10 to 10 max) $= 145$

(3) $e_j = \cap i \mid S_i = m_j$ → ($\cap i$ → index which is highest)

i) $e_1 = \cap i \mid S_i = m_1$ (177)
$= 7$ (index)

ii) $e_2 = \cap i \mid S_i = m_2$ (177)
$= 7$ (index)

iii) $e_3 = \cap i \mid S_i = m_3$ (177)
$= 7$ index

iv) $e_4 = \min i \mid S_i = m_4 (177)$
$= 7$ index

v) $e_5 = \min i \mid S_i = m_5 (177)$
$= 7$ index

vi) $e_6 = \min i \mid S_i = m_6 (177)$
$= 7$ index

vii) $e_7 = \min i \mid S_i = m_7 (177)$
$= 7$ index

viii) $e_8 = \min i \mid S_i = m_8 (145)$
$= 10$ index

ix) $e_9 = \min i \mid S_i = m_9 (145)$
$= 10$ index

x) $e_{10} = \min i \mid S_i = m_{10} (145)$
$= 10$ index

④ $m_i = m_i - S_i + d_i$

i) $m_1 = 177 - 31 + 31 = 177$
ii) $m_2 = 177 - (-19) - 42 = 146$
iii) $m_3 = 177 - 49 + 59 = 187$
iv) $m_4 = 177 - 75 + 26 = 128$
v) $m_5 = 177 - 22 - 53 = 102$
vi) $m_6 = 177 - 80 + 58 = 155$
vii) $m_7 = 177 - 177 + 97 = 97$
viii) $m_8 = 145 - 84 - 93 = -32$
ix) $m_9 = 145 - 61 - 23 = 61$
x) $m_{10} = 145 - 145 + 84 = 84$

⑤ $t = \min i$ (highest value among m_i)
$= 187$

Page No.
Date : / /

(6) $x = ni \mid mi = t$ → starting point (ni → highest index)

$x = ni \mid mi = 187$

= index (3) = 187 in mi

(7) $y = ex$ → ending point

$x = 3$

$y = ex$

$y = e3$ (index 3 ma ekovalue)

$= 7$,,

putting all the value in table we get:

di	31	-41	59	26	-53	58	97	-93	-23	84
index(i)	1	2	3	4	5	6	7	8	9	10
Si	31	-10	49	75	22	80	177	84	61	145
ms	177	177	177	177	177	177	177	145	145	145
ei	7	7	7	7	7	7	7	10	10	10
mi	177	146	187	128	102	155	97	-32	61	84
t	187	187	187	187	187	187	187	187	187	187
x	3	3	3	3	3	3	3	3	3	3
y	7	7	7	7	7	7	7	7	7	7

So, Sum of contiguous sub Array is from index (3) to index (7)

i.e = (59 + 26 - 53 + 58 + 97)

= 187 ,,

Page No.
Date : / /

② Two Criterion BSR algorithm:

- BSR algorithm where broadcast data are tested for their satisfaction of two criteria before being allowed to take part in reduction process.
- We use the problem of Counting inversions in a permutations

* example: Counting inversion permutations

- Given a set S, a permutation π of S is a set S', containing all elements of S but in different order.
- Inversion in permutations are those numbers of pairs, which are in disorder. The problem is to count the number of inversions.
- Rules: $y_i = \{1 \mid \{i<j\} \wedge \{\pi(i) > \pi(j)\}\}$

index 1 2 3 4 5 6

Let, No. of inversions in {1, 6, 2, 9, 5, 3} =

We have two condition: ① $i<j$

② $\pi(i) > \pi(j)$

① Compare 1 & 6

① where, index (1) < index (6)

1 < 2 True ✓

② $\pi(i) > \pi(j)$

1 > 6 (False) ✗

② Compare 6 & 2

① $i<j$ = index (6) < index (2)

= 2 < 3 True ✓

② $\pi(i) > \pi(j)$ = 6 > 2 True ✓

③ Compare 6 & 5

① $i<j$ = index (6) < index (5) ② $\pi(i) > \pi(j)$ = 6 > 5 True ✓

= 2 < 5 True ✓

So, The pair of number satisfing both condition is 3 i.e
= {6,2}, {6,5}, {9,5}

③ Three criterion BSR algorithm:

- BSR algorithm where broadcast data are tested for their satisfaction of three criterion before being allowed to taken post in reduction process.
- Example: vertical segment visibility.
- Rules:

• Left top (lt_i) = $\cap i \mid x_i < x_j \wedge t_i \geq t_j \wedge b_i \leq t_j$ (1, 2, 3)

• Right top (rt_j) = $\cap i \mid x_i > x_j \wedge t_i \geq t_j \wedge b_i \leq t_j$ (1, 2, 3)

Where,
$x_i < x_j$ = defines the coordinates either lies left of right
$t_i \geq t_j$ = decides the top is always greater than center line.
$b_i \leq t_i$ = decides the bottom to decide left top.

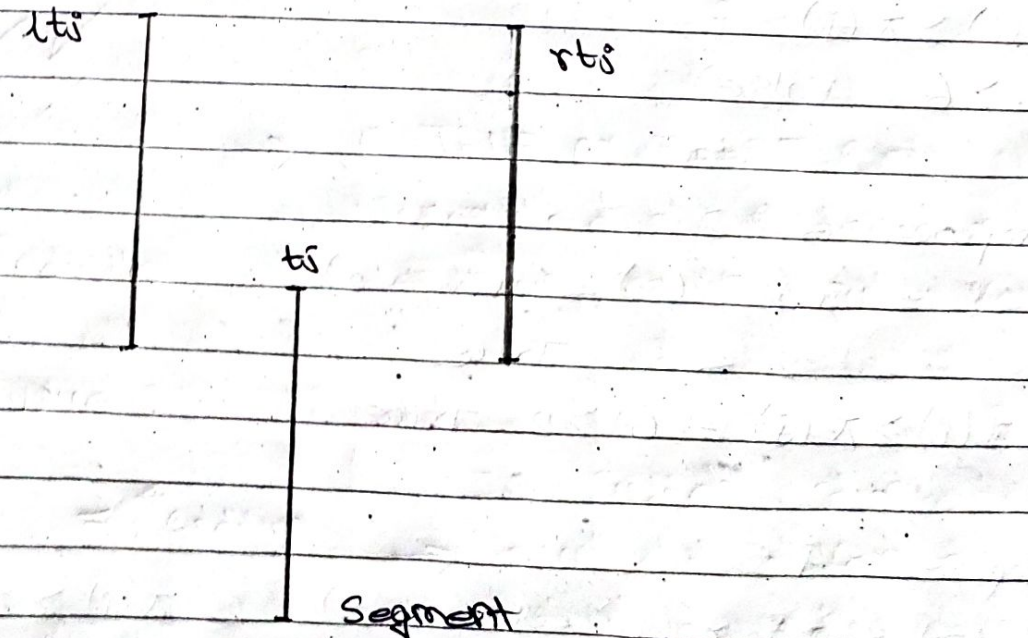


Fig:- 3 criterion BSR.

2.3 Dataflow Models:

* Introduction:

- The dataflow in software engineering refers to the flow of information between data processing entities.
- We will see two kinds of computing model to model the data flow and they are:

i) Control-Flow Computing model,

ii) Dataflow Computing model.

Consider a Example: if (n==0)

then

c = a+b — either this

else

c = a-b — or this

↓ control-flow

} both inst in data-driven

i) Control-Flow Computing model:

When using a control-flow computing model, the program will be translated into a series of instructions starting with an instruction comparing n to 0 and either transferring control to an instruction adding a to b or to an instruction subtracting b from a and in both case, executing only one instruction that stores the result in c.

- It specifies the next instruction to be executed, depending on what happened when executing the current instruction.

ii) Data-driven flow Computing model:

A dataflow computing model is not based on flow of control instead it is based on flow of data. It executes any operations as soon as there are necessary

operands.

For example: If n is available, the operation (==) can be applied to its operand, similarly, if a & b is available both of the operations (+) and (−) can be applied to a and b, even though only one of these results is needed.

* Basic Primitives of data Flow:

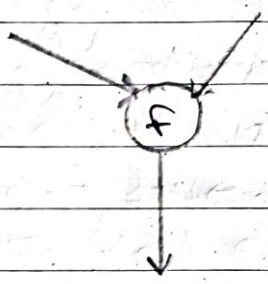
① 

Fig: operator

→ A data value is produced by an operator as a result of some operation f

②

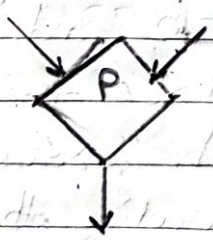


Fig: Predicate

→ a TRUE or FALSE control value is generated by a decider (predicate) depending on its input token.

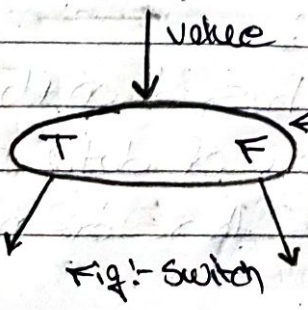
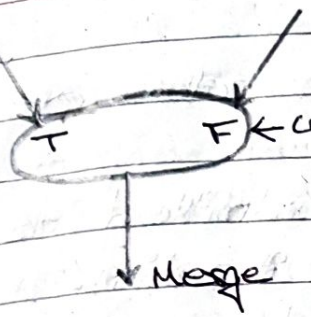
③ 

Fig:- Switch

→ a switch actor directs an input data token to one of its output depending upon the condition applied.

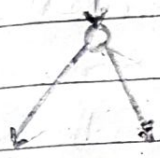
(A)



A merge actor passes one of its input tokens to the output depending on the input control token.

Fig: Merge

(B)



→ A copy is an identity operator which duplicates input tokens.

Fig: Copy

* Example: Draw a Data-Flow model for $S = \sum_{i=1}^{n} f(i)$

pseudo code:
```
Sum=0;
for(i=1; i<=n; i++)
  Sum = Sum + i;
```

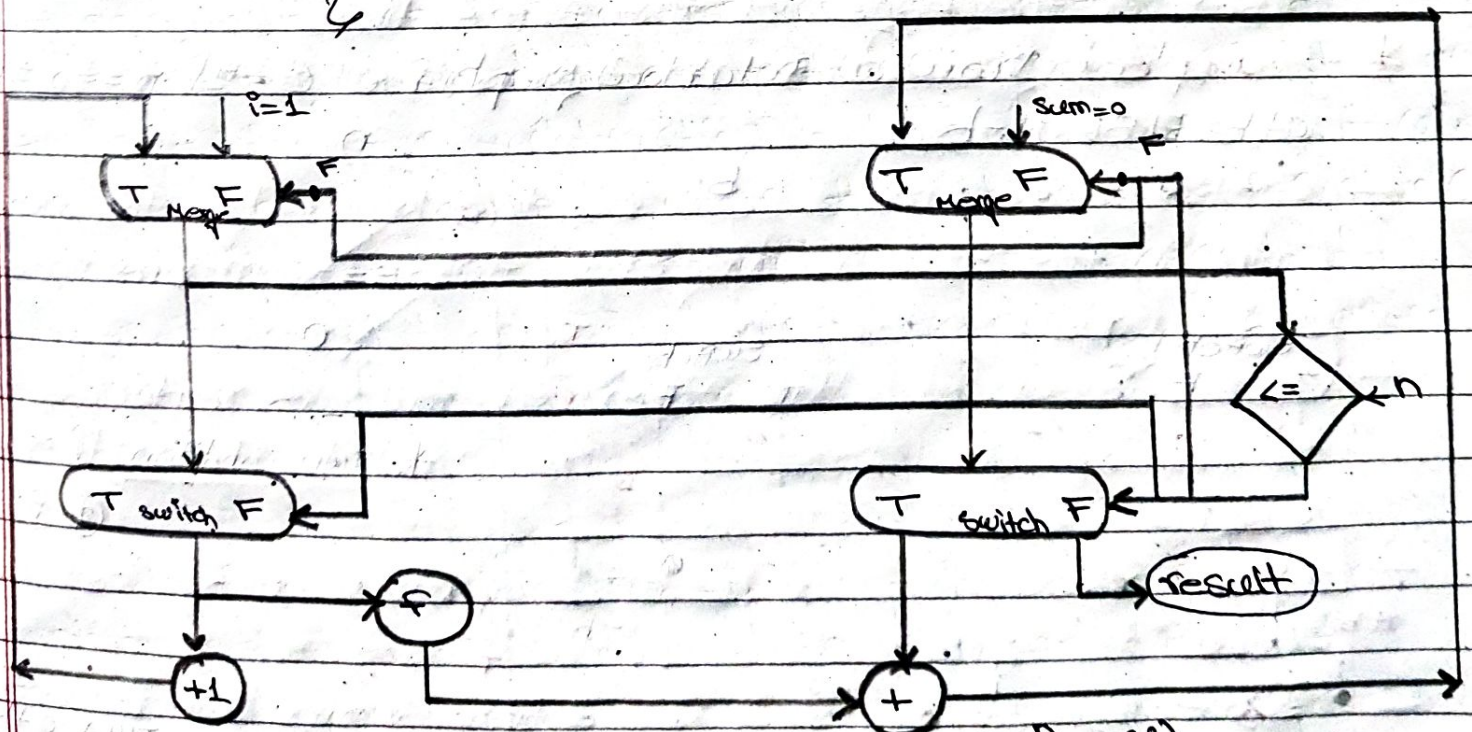
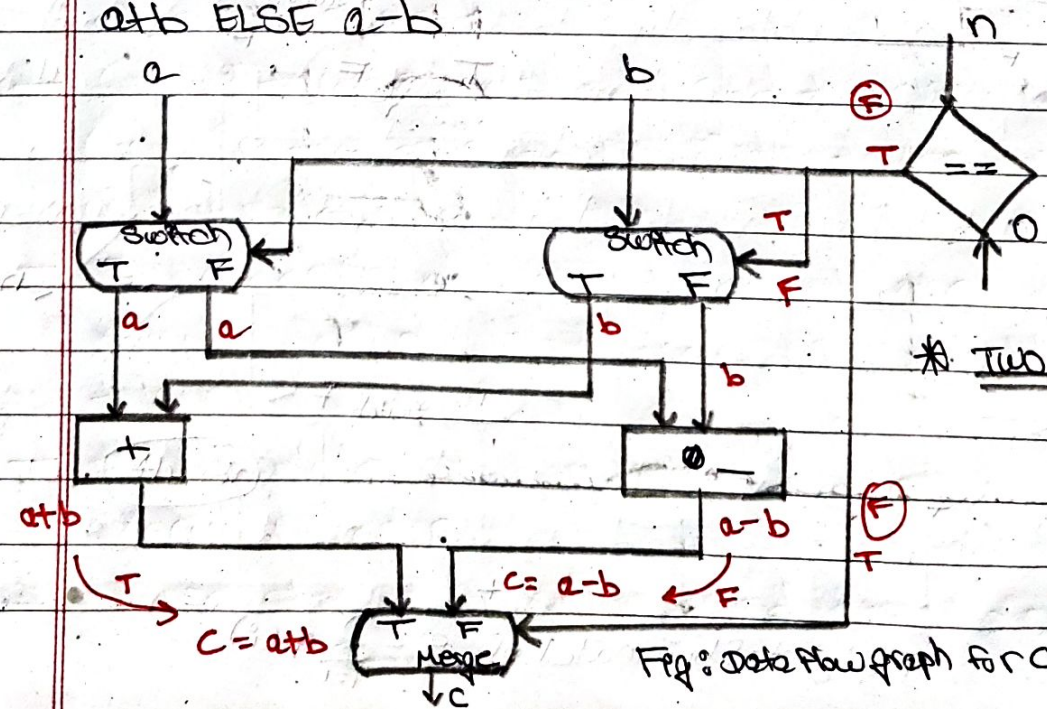


Fig:- Dataflow model of $S = \sum_{i=1}^{n} f(i)$

Page No.
Date : / /

* Description:
- Initially the conditions in the merge actors are set to false.
- The input values i and sum are admitted as i=1 & sum=0 as the initial values of the iteration.
- The predicate i≤n is then tested.
- If it is true, then the values of i & sum are routed to the TRUE (T) sides of switch
- This initiates the firing of function f as well as increment of i.
- Once the execution of the body of the loop completes, the next iteration initiated at the merge actors.
- The initiation continues until the condition i≤n is no longer true.
- The final sum is then routed out of the loop and the initial Boolean values at merge actors are restore to FALSE.

* Example: Draw a dataflow graph for C = If n == 0 THEN a+b ELSE a-b



* Two condition: ① T ② F

Fig: Data flow graph for C = If n = 0 THEN a+b a-b

* Note: Demand-driven & control flow models are same.

Page No.
Date: / /

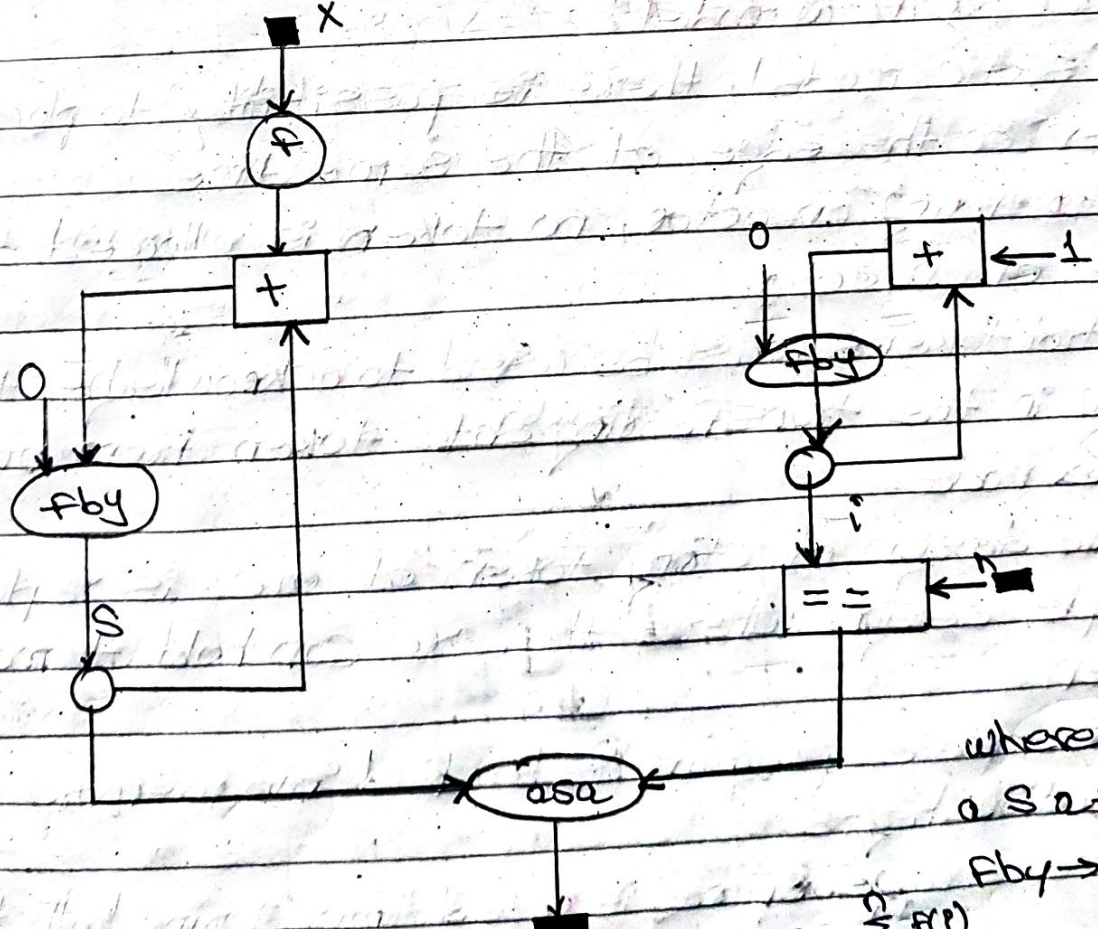
* Demand-Driven Data Flow Computing Model:

- The basic idea behind demand-driven execution is that an operation of a node will be performed only if there are tokens on all the input edges and there is a demand for the result of applying the operations.
- There are two programming models that demand-driven execution is used to evaluate : i) Operator nets ✓
  ii) Functional program ×

i) operator nets:

- They are quite similar to dataflow graphs.
- It is a programming models for demand-driven.
- consists of set of nodes, a set of arcs and set of equation that relate the output arcs of nodes to functions / operators applied to the input arcs.

example:



where,
a S a → as Soon as
Fby → Followed by

Fig:- operator nets for $S = \sum_{i=1}^{n} f(i)$

Page No.
Date : / /

* Static and Dynamic Data flow model:

The basic principle of any dataflow computer is data-driven and hence it executes a program by receiving, processing and sending out token: These token consists of some data and a tag. These tags are used for representing dependencies between instructions.

Also, The processing unit is composed of two parts matching unit that is used for matching the tokens and execution unit used for actual implementation of instruction When the processing elements get a token the matching unit perform the matching operation and when token matched processing begins by execution unit.

There are variety of static, dynamic & also hybrid dataflow models:

i) Static dataflow model:

- In static model, there is possibility to place only one token on the edge at the same time.
- When firing an actor, no token is allowed on the output edge of an actor.
- Control tokens must be used to acknowledge the proper timing in the transferring data token from one node to another node.
- FIFO design of storing token at arc is replaced by simple design where the arc can hold at most one data token.
- the complete program is loaded into memory before execution begins.
- Same storage space is used for storing both the instruction

as well as data.

ii) Dynamic dataflow model:

- In dynamic model, it allows placing of more than one token on the edge at the same time.
- The concept of tagging of tokens was used. Each token is tagged and the tag identifies the context in which particular token is used.
- No control tokens are needed to acknowledge the transfer of data tokens among the instructions.
- Program nodes can be instantiated at run time unlike in static where it is loaded in the beginning.
- Seperate storage is used for instruction & data.

Page No.
Date : / /

2.4 Partioning and Scheduling:

- Program partitioning and task scheduling are two distinguishing features of parallel versus sequential programming.
- The partitioning problem deals with how to detect parallelism. for example;

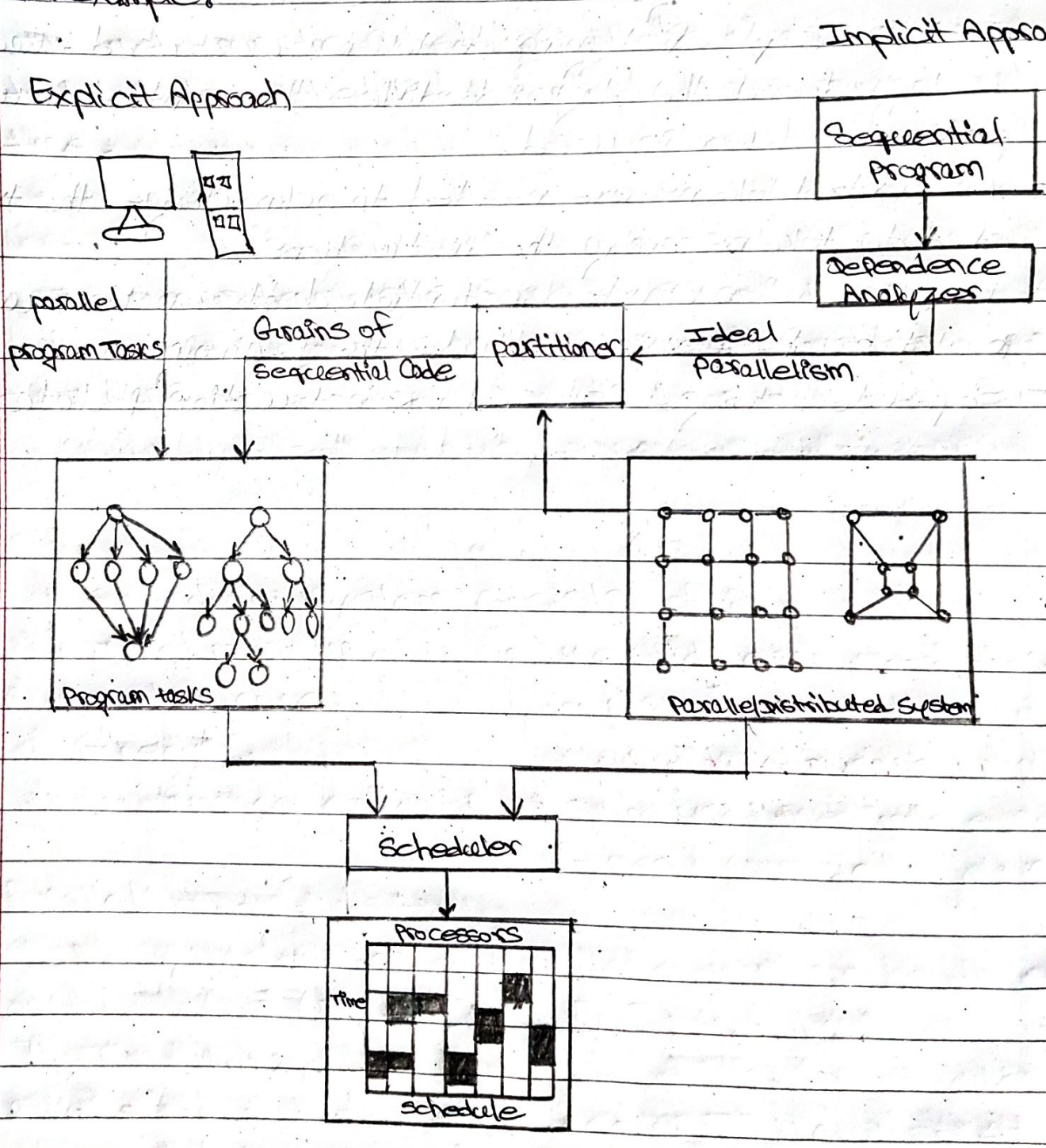


Fig: Partitioning & scheduling in implicit & explicit parallelism

- In the implict approach, the underlying computing environment is entirely concealed from the programmer. Compilers are required to analyze the application to expose embedded parallelism & perform partitioning. It removes the burden of dealing with the increased complexity from the shoulders of programmers.
- In the explict approach, existing languages are extended, or entirely new languages are introduced to express parallelism directly. This requires programmers to learn new kinds of constructs. It is programmers responsibility to identify parallelism within the application.

* Program Partitioning:

i) Data Partitioning:

- It perform the same computation on different sets of data concurrently.
- It is appropriate for applications that performs the same operations repeatedly on large collections of data.
- for eg: matrix multiplication, Fourier transformation etc.

ii) Function Partitioning:

- It perform different computations on same set of data concurrently.
- for eg: flight simulation etc.

* Task Scheduling:

- After program partitioning, tasks must be optimally scheduled on the processors such that the program execution time is minimized.
- Scheduling techniques can be classified based on the

availability of program task information as:
i) Deterministic and
ii) Non-deterministic.

i) Deterministic:
In deterministic scheduling, all the information about the tasks to be scheduled is entirely known prior to execution time.

ii) Non-deterministic:
In non-deterministic, some information may not be known before the program executes. For eg:- Conditional branches & for loops.

* Scheduling System Model:

A scheduling system consists of:
i) parallel program,
ii) target machine,
iii) Schedule,
iv) Performance criterion.

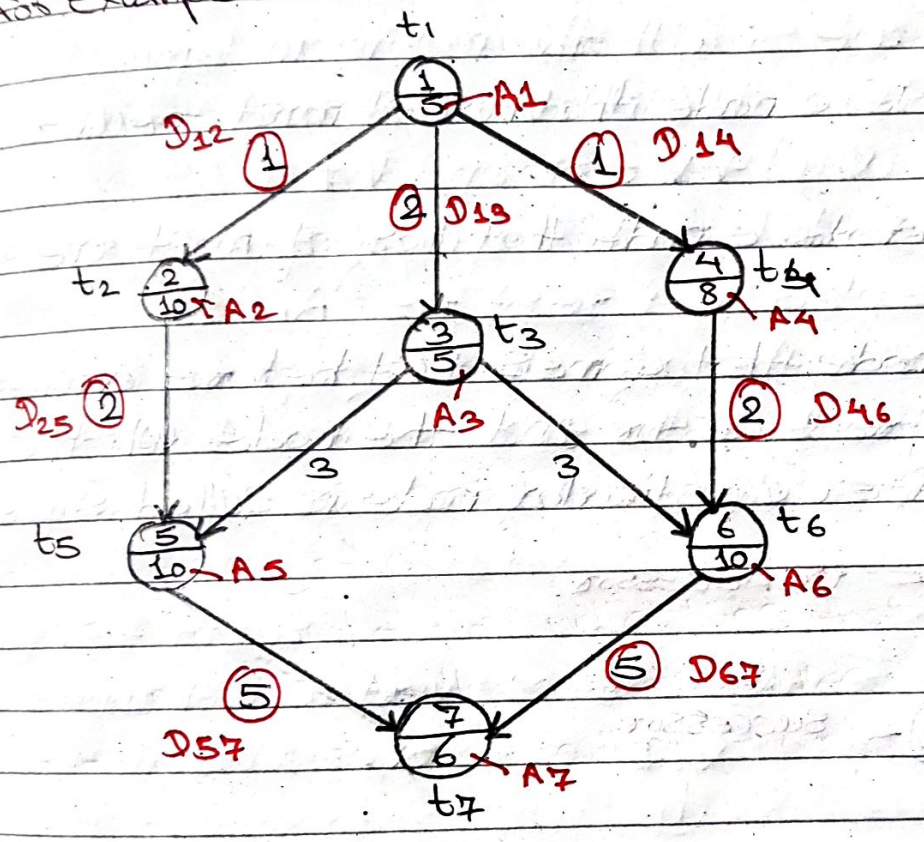
} we study each one of these four components & show how the program & target machine parameters can be used to estimate execution times & communication delays.

i) Parallel program tasks:
— The characteristics of a parallel program can be defined as the system (T, <, [Di], [Ai]) as follows:
1. T = {t1, t2, ..., tn} is a set of tasks to be executed
2. < = is a partial order defined on T, i.e. if ti < tj means that ti must be completed before tj start.
3. [Di] = is nxn matrix of communication data, where Di ≥ 0

is the amount of data required to be transmitted from task t_i to t_j

4. A_i → is n vector of the amount of computations, i.e. $A_i > 0$ is the number of instructions required to execute t_i.

for example:



here $\{t_1 < t_2, t_1 < t_3, t_1 < t_4\}$ shows t_1 must be completed first.

Fig: A task graph. ($n = 7$)

Above figure shows an example of a task graph consisting of 7 nodes ($n = 7$), where each node represents a task.

- The numbers shown in the upper portion of each node is the node number,
- The number in the lower portion of a node i represents the parameter A_i (the amount of computation needed by task t_i),
- The number next to the edges (i, j) represents the parameter D_{ij}. for eg: $A_2 = 10$, $D_{13} = 2$.

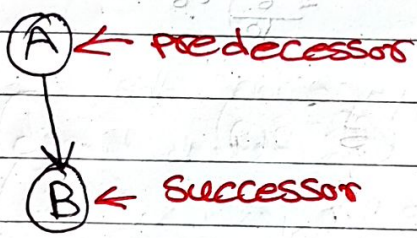
Page No.
Date : / /

* Two types of Process Scheduling:
  1. Scheduling in-Forest / out-Forest with communication
  2. Scheduling Interval ordered Task

1. Scheduling In Forest / out-Forest with Communication:

* Basic terms:
- In-Forest denote a node that has at most one immediate successor.
- out-Forest denote a node that has at most one immediate predecessor.

  where, The node that comes just before another node is called the predecessor and the node which comes immediately after a particular node is called successor.



- Precedence relations between activities signify that the activities must takes place in a particular sequence.

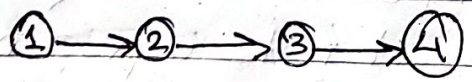


Fig: Illustrative set of four Activities with precedences

- The level of each node in the task graph is calculated and used as each node's priority. Higher the level Higher the priority to assign processor.

Page No.
Date : / /

- The algorithm is based on the idea of adding new precedence relations to the task graph in order to compensate for communication.
- The task graph after adding the new precedence relations is called the Augmented task graph.
- Scheduling the augmented task graph without considering communication is equivalent to scheduling the original task graph with communication.

Scheduling (Augmented Task graph) without communication
=
Scheduling (Task graph original) with communication.

- Initially we need to define: i) node-depth
ii) operation swap all.

① **node-depth:**
- The depth of a node is defined as the length of the longest path from any node with depth zero to that node.
- A node with no predecessors has a depth of zero.

i.e.
depth (u) = 0 if predecessor (u) = ϕ .
depth (u) = 1 + max { depth (v) } ∀ v ∈ predecessor (u)

② **Operation Swap-all:**
- Given a schedule f, the operation swapall (f, x, y) where x and y are two tasks in f scheduled to start at time t_1 on processors i and j
- The effect of this operation is to swap all the task pairs scheduled on processors i and j in the scheduled time $t_1, \forall t_1, t_1 \geq t$

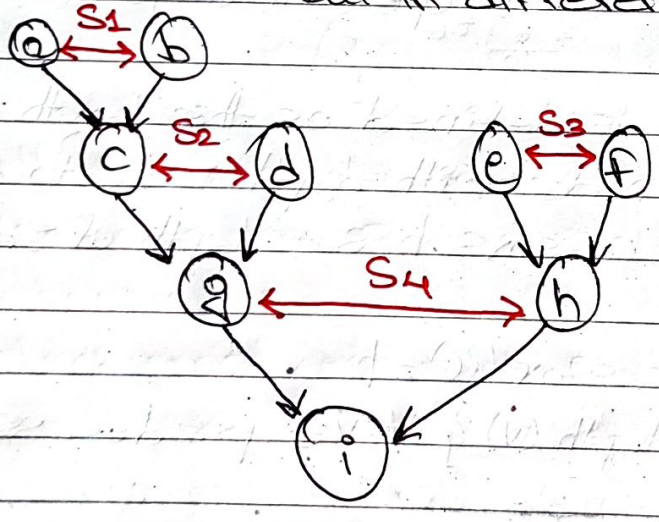
Page No.
Date : / /

* Algorithm:

1) Given an in-forest $G_1 = (V, E)$, identify the set of siblings $S_1, S_2, \ldots, S_k$ where S_i is the set of all nodes in V with common child (S_i).
2. $A_1 \leftarrow A$
3. for every set S_i
   - pick node $u \in S_i$ with the maximum depth
   - $A_1 \leftarrow A_1 - (v, \text{child}(S_i))\ \forall v \in S_i$ and $v \neq u$
   - $A_1 \leftarrow A_1 \cup (v, u)\ \forall v \in S_i$ and $v \neq u$
4. Obtain the schedule F by applying previous algorithm on the augmented in-forest $F = (A, A_1)$
5. for every set S_i in the original task graph G_1, if node u is scheduled in the time slot immediately before child (S_i) but in different processor, apply swap all.

* example:

(1)



Note: Select the pair of vertex with common child as S_i.

Fig: Task graph with respective S_i (original)

(2) $S_1 = \{a, b\}$ = both have 0 predecessor so, choose any → a, b
$S_2 = \{c, d\}$ = c has 1 predecessor & d has 0 → c
$S_3 = \{e, f\}$ → both have 0 predecessor, choose any → f
$S_4 = \{g, h\}$ → g has 2 predecessor, h has 1 → g

$(v, child(v)) = (a, c)$

$(v, u) = (a, b)$

So, the Augmented task graph is

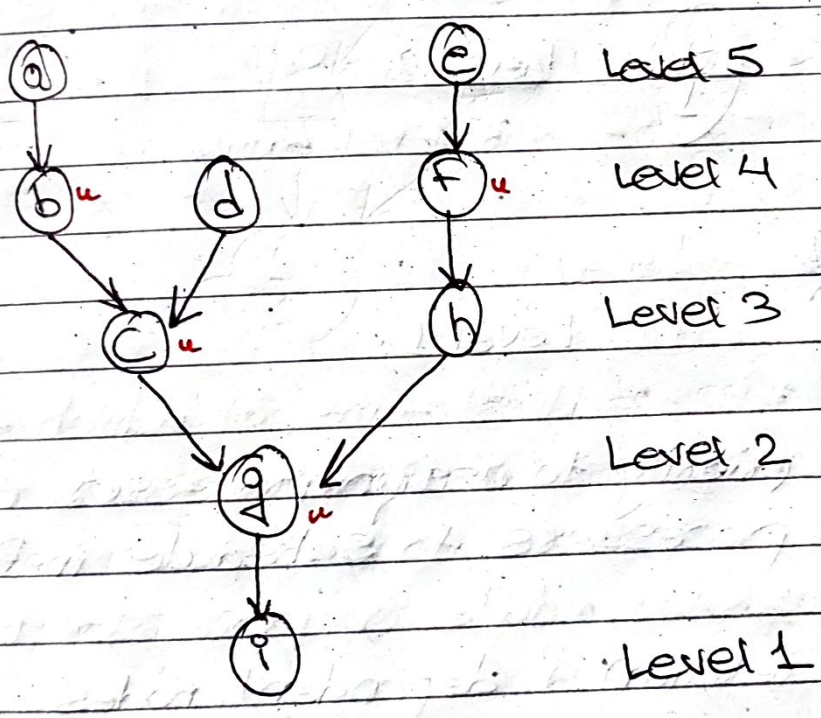


Fig: Augmented task graph.

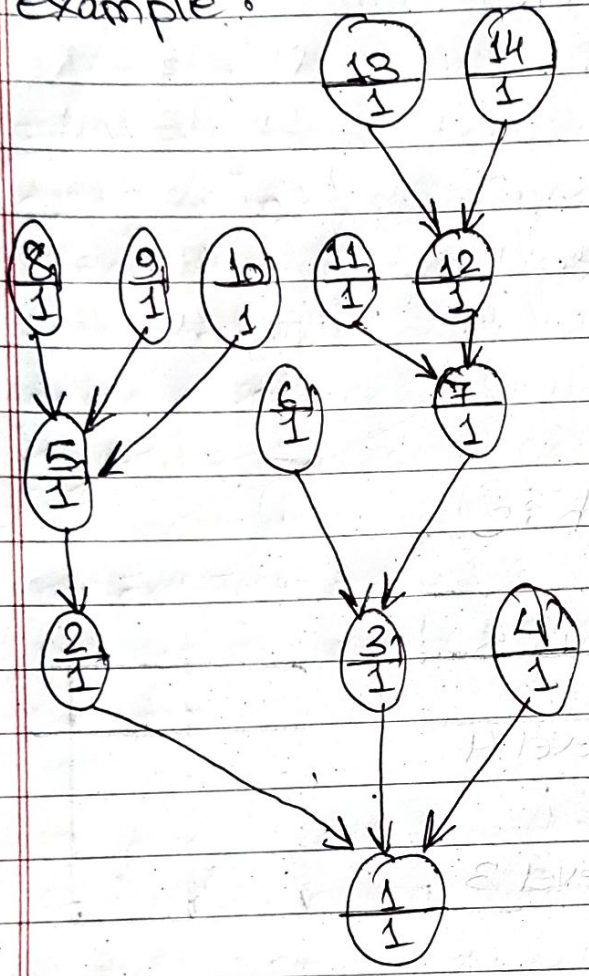
Node	Level
a	5
b	4
c	3
d	4
e	5
f	4
g	2
h	3
i	1

Fig:- Task Priority

Time	P_1	P_2
0	a	e
1	b	d
2	f	c
3	h	
4	g	
5	i	

Fig:- Task scheduling

* example :



P_1	P_2	P_3
14	13	11
12	10	9
8	7	6
5	4	3
2		
1		

- Higher level higher priority to assign processor,
- We assume 3 processors to schedule in-forest without communication.
- check precedence relation & dependent nodes.

② Scheduling interval ordered tasks :

- The term interval ordered tasks is used to indicate the task graph which describes the precedence relations among the system tasks in an interval order.
- A task graph is an interval order when it elements can be mapped into intervals on the real line and two elements are related if and only if the corresponding intervals do not overlap.

i) Without Communication,
ii) With Communication.

i) Scheduling interval orders without Communication:

- The number of all successors of each node is used as each node's priority.
- Whenever a processor becomes available, assign it, with the unexecuted ready task with the highest priority.

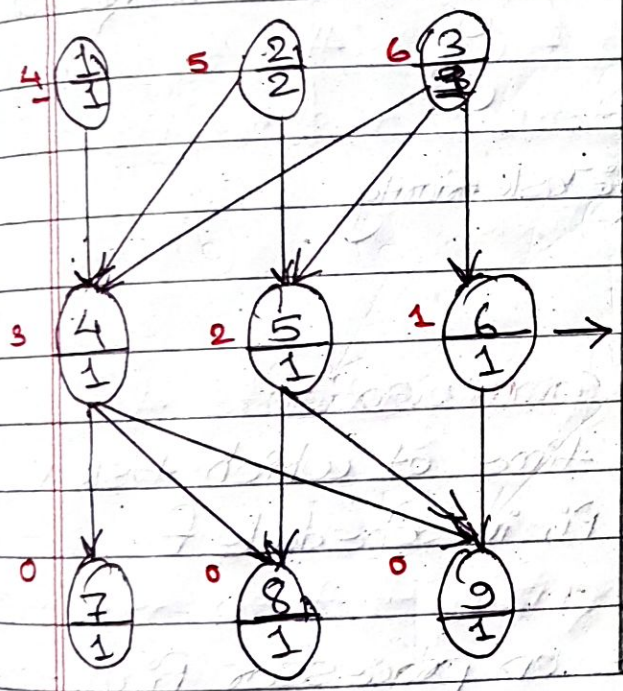


Fig:- Task Graph

Node	No of successor
1	4
2	5
3	6
4	3
5	2
6	1
7	0
8	0
9	0

Fig:- Task priority

Time

	P1	P2	P3
0	3	2	1
1	4	5	6
2	7	8	9
3			

Fig:- Task Scheduling

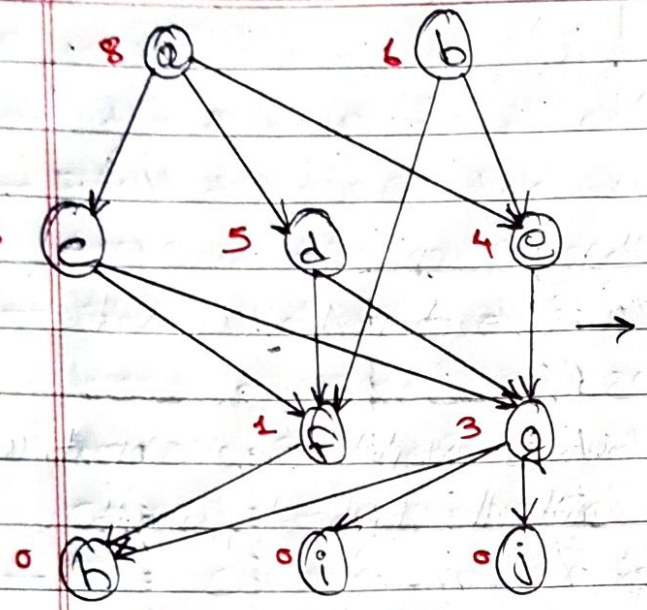


Fig:- Task graph

Node.	No of Successors
a	8
b	6
c	5
d	5
e	4
f	1
g	3
h	0
i	0
j	0

Fig:- Task priority

Time

	P1	P2	P3
0	a	b	
1	c	d	e
2	g	f	
3	h	i	j
4			

Fig:- Task Scheduling.

ii) Scheduling interval orders with Communication:

- start_time (v, i, f) : earliest time at which task v can start execution on processor Pi, in schedule f

- task (i, t, f) : task schedule on processor Pi at time t in schedule f.

* Algorithm:

- The number of all successors of each node is used as each node's priority.
- Nodes with highest priority are scheduled first.
- Each task v is assigned to processor Pi with the earliest time start.
- If start_time(v,i,f) = start_time (v,j,f), $1 \le i, j \le m$, task v is assigned to processor Pi if task (i, start_time (v,i,f)-1, f)

has smaller priority

- Communication delay is considered as one unit of time.

* example:

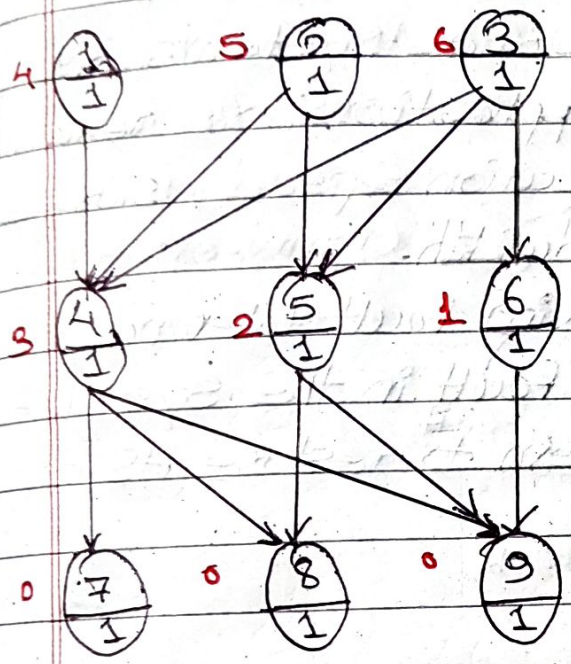


Fig: Task graph

Node	No of successor
1	4
2	5
3	6
4	3
5	2
6	1
7	0
8	0
9	0

Fig:- Task priority

Time	P1	P2	P3
0	8	2	1
1	6		
2	4	5	
3	7		
4		8	9
5			

Fig:- Task Scheduling

Page No.
Date : / /

2.5 Checkpointing in Parallel and Distributed Systems:

* Introduction:

- Fault tolerance techniques enable systems to perform tasks in the presence of faults.
- The likelihood of faults grows as systems are becoming more complex and applications are requiring more resources including execution speed, storage capacity & communication bandwidth.
- The common approach in achieving fault tolerance consists of detection and location of a fault in the system followed by reconfiguration of the system to restore its operational condition.

* Terminology:

Fault-tolerance schemes typically consists of following steps:

① **Fault detection:**

It is the process of recognizing that an error has occurred in the system.

② **Fault Location:**

After the error in the system is detected, the location of the part in the system that caused the error is identified.

③ **Fault containment:**

After the fault is located, the faulty part is isolated to prevent a possible propagation of the fault to the rest of the system.

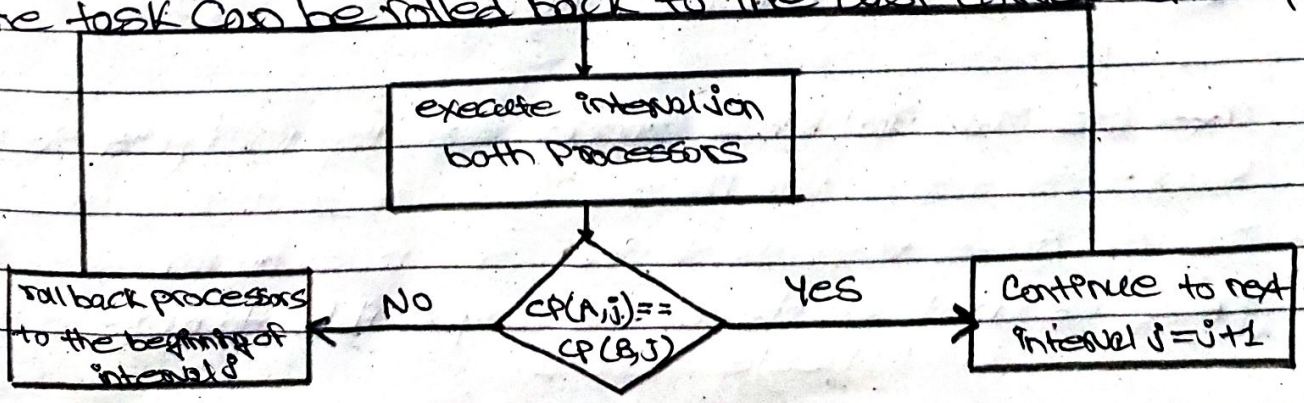
Page No.
Date : / /

Fault recovery or resilience :

It is the process of restoring the operational states of the system to a previous fault free consistent states.

* Checkpointing using Task duplication :

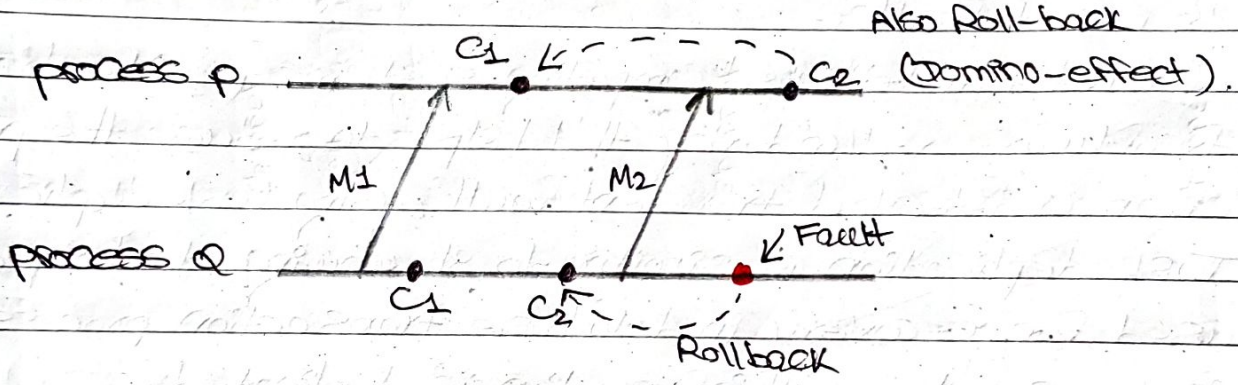
- The dominant cause for errors in computer systems are transient faults.
- Parallel & distributed computing systems provide hardware & software redundancy that helps to reduce the probability of an undetected transient fault, using task duplication.
- Task duplication is similar to shadowing techniques used for recovery in database transaction processing systems, where all information is duplicated.
- The detection of a transient faults is enabled by comparing the states of the same task that was executed in parallel on different processors.
- Fault detection is achieved by duplicating the task into two or more processors & comparing the states of the processors at the checkpoints.
- Two matching states is an indication of a correct execution.
- By saving the state of the task at each checkpoint, we avoid the need to restart the task after each fault, instead the task can be rolled back to the last correct checkpoint.



Page No.
Date : / /

* Techniques for consistent checkpointing:

- When a fault is detected in one of the processors, the process is rolled back to its last sane check point.
- In order to maintain consistent checkpoint, some other processes that communicates with this task (where fault occur) also need to be rolled back, called domino-effect.



Page No.
Date : / /

2.6 Architecture for open Distributed Software System:

* Introduction:

- Distributed systems are complex structures, composed of many types of hardware and software components.
- In some systems, components are developed separately by different implementors and then combined, resulting in a heterogeneous system.
- Some examples are: Telecommunication systems, Computer communication networks, client-server systems.

* Components of open-distributed system Architecture:

1) End-user viewpoint: Information content by view of user.
2) Enterprise viewpoint: Describing the needs of the users of an information system.
3) Computational viewpoint: Distributed system seen by application designer or programmer.
4) Information viewpoint: Focuses on the information content of the enterprise (Information Semantics of distributed system).
5) Enginering viewpoint: System support for distributed system
6) Technology viewpoint: Technical component information.

* Engineering Model:

- The engineering model is an abstract model designed to express the concepts of the engineering viewpoint.
- It contains the concepts such as operating systems, distributed transparency mechanisms, communication systems, processors & storage.

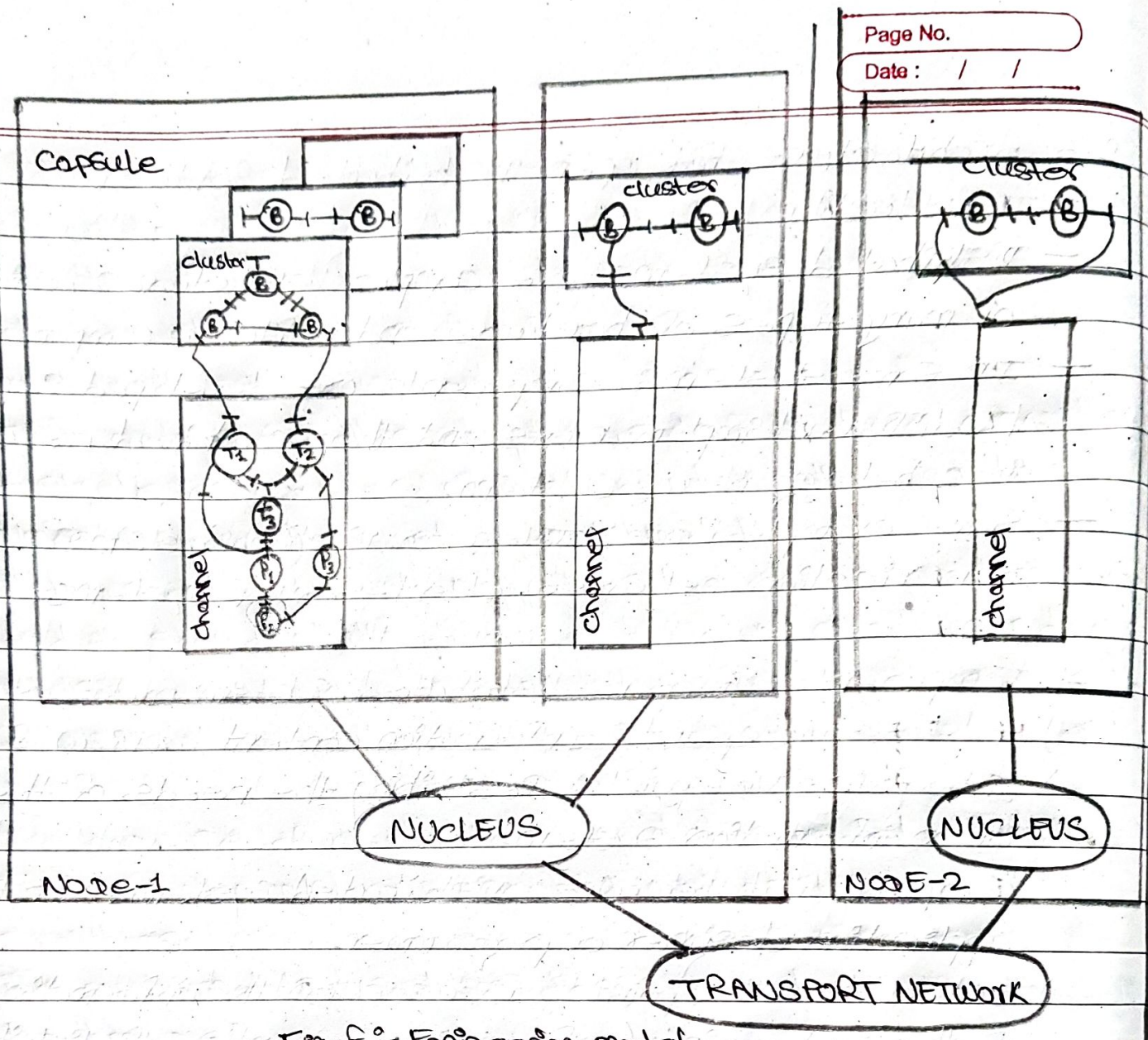
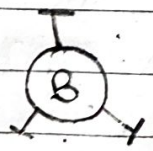
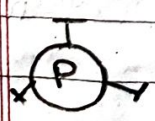


Fig of ir Engineering model.

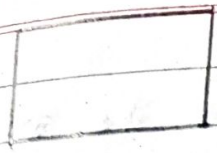
- A nucleus is an object that provides access to basic processing, storage and communication functions of a node for use by BEO, To and PO.

i)  = Basic Engineering objects are the run time representation of computational objects.

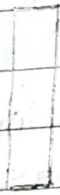
ii)  = Transparency object are special purpose model

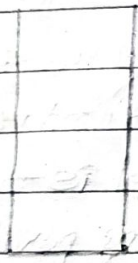
iii)  = Protocol object are responsible to make a bridge to communicate objects.

Page No.
Date : / /

iv)  = cluster are concepts of related modules.

v)  = capsule is the collection of BEO, IO, PO in a node.

vi)  = channel are runtime binding between objects.

vii)  = Node is a single computer system.

Page No.
Date : / /

Unit-3
Algorithms

3.1 Fundamentals of Parallel Algorithms

* Introductions

- The purpose of this chapter is to introduce a number of strategies for designing efficient parallel algorithms on almost any of the existing parallel model.
- It means that a strategy to solve a given problem can lead to substantially different efficient algorithms on different models.

* Models of Parallel Computations: (imp for short notes)

- The purpose of a parallel computation model is to give a simple framework for describing and Analyzing parallel algorithms.
- The major models used for designing and analyzing parallel algorithm are:
  i) PRAM model,
  ii) Shared memory model,
  iii) The network model,
  iv) The $\log^P$ model
  v) The block distributed memory model.

i) PRAM Model:

- The PRAM model is a natural extension of the sequential model of computation.
- It assumes the presence of an arbitrary number of processors that can access synchronously a large

shared memory.
- Each processor has its own local memory & can execute its own local program.
- Different versions of the PRAM model:
  i) EREW (Exclusive read, Exclusive write)
  ii) CREW (Concurrent read, Exclusive write)
  iii) ERCW (Exclusive Read, Concurrent write)
  iv) CRCW (Concurrent Read, Concurrent write)

  } 2-1 explain

- The main complexity measure used for evaluating the performance of a PRAM algorithm is

$$T_p(n) = \frac{T(n)}{P}$$

where, $T(n)$ = is the best sequential Time
P = no. of processors.

Also, PRAM algorithm is optimal if the work defined as the product $T_p(n) \times p$, is $O(T*(n))$

ii) Shared memory model:
- The shared memory model retains the single address space of the PRAM but assumes that the processors execute their operations asynchronously.
- Each processor has its own Local memory and control unit and executes its program asynchronously.
- The programming style used is typically the single program multiple data (SPMD) model, which results in the same program being run on all the processors that operate asynchronously on different portion of the problem.

Page No.
Date : / /

iii) The network model:

- The network model takes the topology of the interconnected network as the basis for modeling the communication between different processors.
- A network can be viewed as a graph $G = (V, E)$, where each node $i \in V$ represents a processor, and each edge $(i, j) \in E$ represents a two-way communication link between processors i and j.
- Each processor is assumed to have its own local memory and no shared memory is available.
- It uses message passing model in which processors communicate explicitly by sending and receiving messages.

iv) The LogP model:

- The LogP model captures performance of distributed memory multiprocessors in which processors communicate by point-to-point messages.
- The parameters of the models are L, O, g & P defined as follows:

i) L = The parameter L is an upper bound on the latency incurred in communicating a message from a source processor to a destination processor.

ii) O = The parameter O is defined to be the overhead during which a processor is busy in the transmission or reception of a message.

iii) g = The parameter g refers to the minimal time interval between consecutive message transmissions or reception.

iv) p = The parameter P is the number of processors.

v) The block distributed memory model:

- It is a collection of powerful processors defined by a cluster and connected by a fast and robust communication network.
- The structure of the communication network for massively parallel systems is hierarchical.
- Collection of processors defined as a single cluster and the communication between each cluster is modeled by a complete graph.

* Balanced Trees, Divide & conquer

- Same as in 2-1 PRAM.

* Partitioning:

- The most natural strategy for designing parallel algorithms.
- It consists of preprocessing the inputs so that the initial problem can be broken into a set of completely independent sub problems.
- The induced sub-problems are recursively solved in parallel, and a method is specified to merge the solutions of the subproblems into a solution of the original problem.
- Similar strategy to the divide-and-conquer.

* Combining:

To obtain an optimal algorithm, it is sometimes necessary to use two or more algorithms, each of which is suitable for handling an input size that falls within a certain range.

Page No.
Date : / /

3.3 Data parallel Algorithms: (SIMD)

- Data parallel Algorithms is a model of parallel computing in which the same set of instructions is applied to all the elements in a dataset.

* Machine Model:

- It is based on SIMD (Single Instruction Multiple Data) Machine model
- A single sequence of instruction is executed simultaneously in an arbitary set of processors with each processor operating on its own data.
- It consists of N processors, N memory unit, and a control unit and an interconnected network

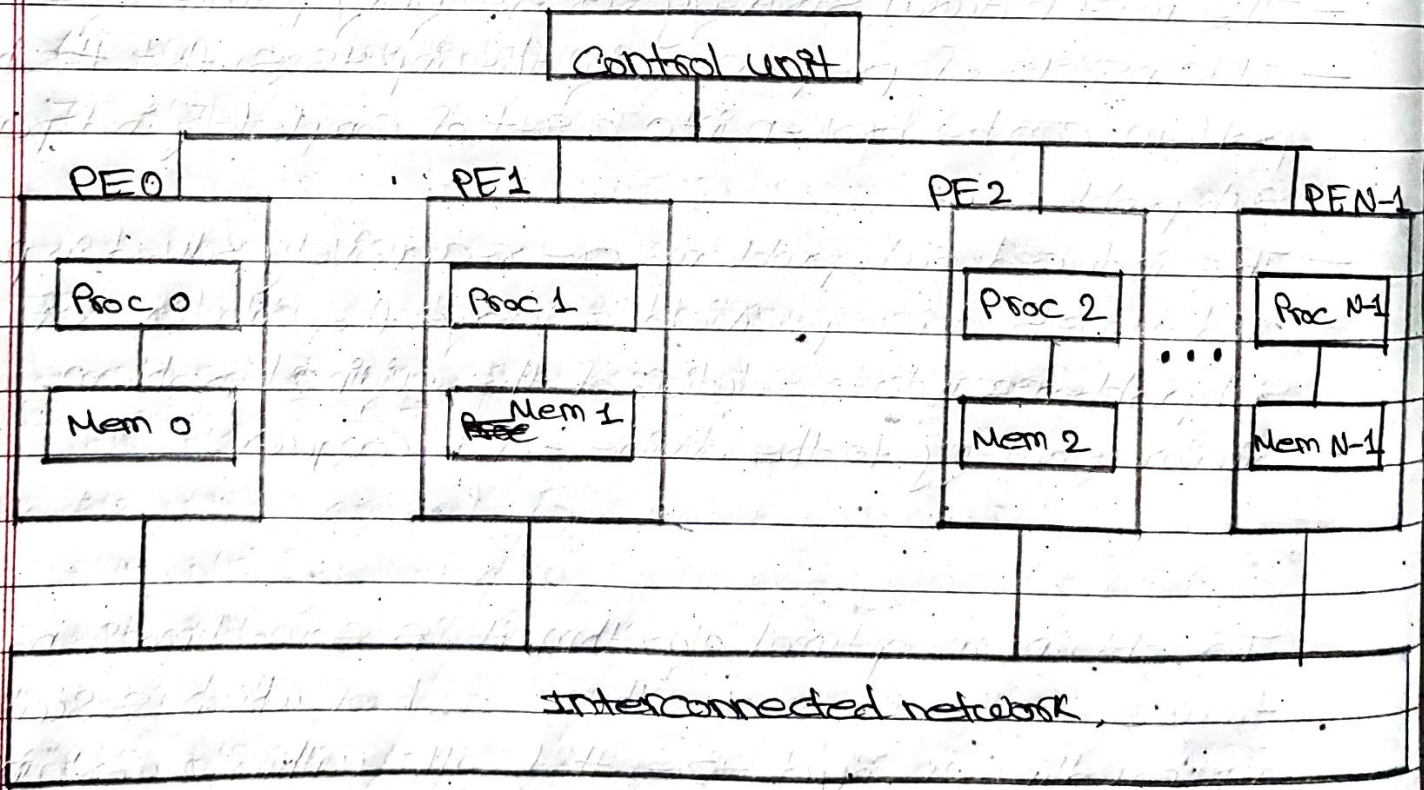


Fig: Distributed memory model of an SIMD machines.

- The CU broadcasts instructions in sequence of PEs and all enabled PEs execute the same instruction at the same time.

* CU/PE overlap:

- The impact of CU/PE overlap on execution time in SIMD mode is examined.
- The CU initiates parallel computations by sending blocks of SIMD code to the CU instruction broadcast queue.
- Once in the queue, each SIMD instruction is broadcast to all the PEs while CU can be performing its own computations.
- This property is called CU/PE overlap. It can improve the overall performance of a program because CU execution and PE execution can occur concurrently.

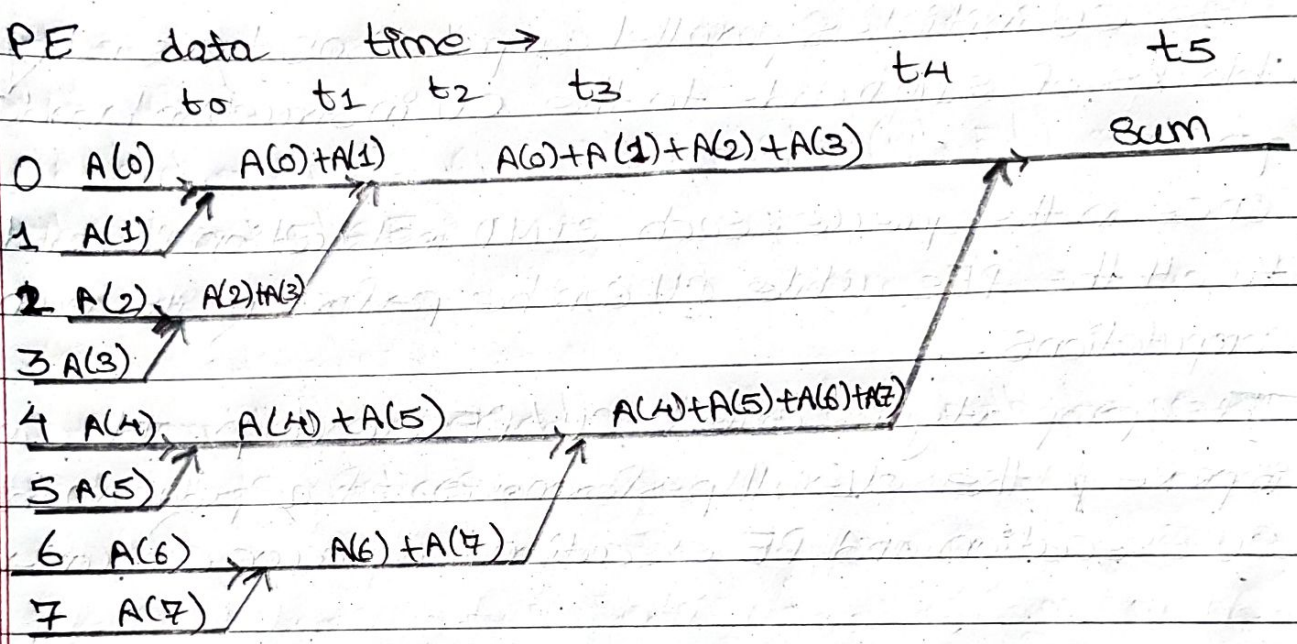
* Parallel Reduction operations:

Many problems requires that all the elements of a data set be combined in some fashions, e.g., the sum or product of all elements of an array. These operation are known as reduction operations i.e., a single value is computed as the result of an operation on the data set. Parallel reduction operations can only applied to associative operations, e.g:- sum, product, min and max. It is important to study parallel algorithms for reduction because data elements are generally distributed across the PEs on parallel machines: i) Single reduction operation on a single set of data,
ii) Multiple reduction on a single set of data.

Page No.
Date : / /

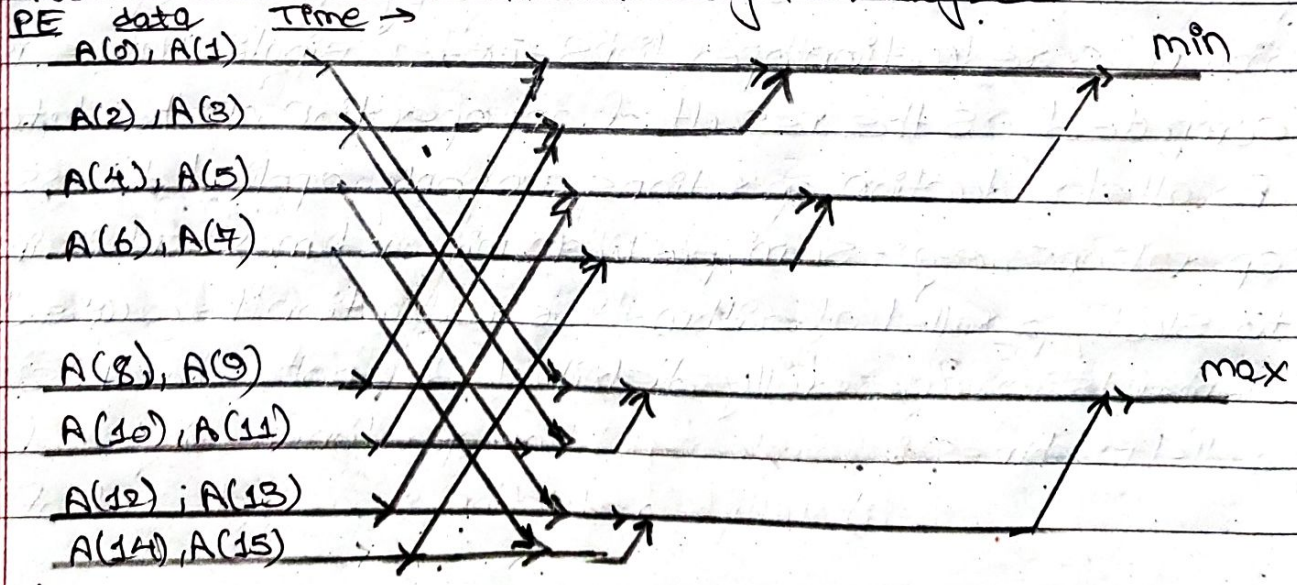
i) Single Reduction operation on a single set of data:

- Recursive doubling is used to demonstrate the concept of a single reduction operation on a single set of data.
- Recursive doubling is an algorithm that combines a set of operands distributed across PEs.
- Example: Finding the sum of M numbers.



ii) Multiple Reduction operations on a single set of data:

- The problem of finding the minimum and the maximum values from a set of data simultaneously is Analyzed.



Page No.
Date : / /

* Matrix & Vector operations :

i) Matrix transposition:

Given an MXM matrix A, the transpose of A, A^T is

$$A^T(i,j) = A(j,i), \text{ where } 0 \leq i,j < M.$$

ii) Matrix Multiplication:

Given an MXM matrix A and MXM matrix B, the product of A and B is an MXM matrix C = AXB whose elements are Given By:

$$C(q,r) = \sum_{h=0}^{M-1} A(q,h) \times B(h,r), \quad 0 \leq q,r \leq M-1$$

* Mapping Algorithms onto partitionable machines:

- Partitionable parallel machines are parallel processing systems that can be divided into independent or communicating subsystems, each having the same characteristics as the original machine.
- The ability to form multiple independent subsystems to execute multiple tasks in parallel helps in achieving better performance.
- Eg: MIMD can be partitioned under a system control.

* Achieving Scalability Using a Set of Algorithms:

- A Parallel algorithm is scalable if it is capable of delivering an increase in performance proportional to the increase of the number of processors utilized.
- Scalability can be better achieved by selecting one algorithm or some algorithm combination from a suite of algorithms to perform the task effectively.
- i.e. Performance (PA) & Processors utilized.

Page No.
Date : / /

3.2 Parallel Graph Algorithms

* Graph Theoretic Concepts and Notations:

① **Un-directed graph:**
It is an ordered pair (V, E) consisting of a set V of vertices and a set E of edges,

② **Sub-graph:**
A subgraph of G is a graph $G' = (V', E')$ such that $V' \in V$ and $E' \in E$.

③ **Connected graph:**
A graph is connected if there is a path between any pair of its vertices.

④ **Bi-connected graph:**
A graph $G = (V, E)$ is biconnected if the graph obtained from G by removing an arbitrary vertex along with all edges incident to it is still connected. For example, the sub-graph induced by {2, 4, 7} is biconnected.

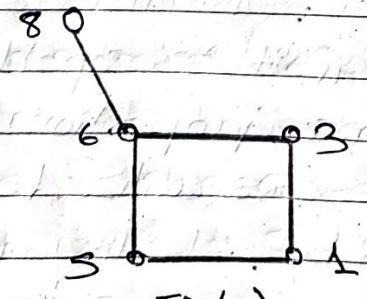


Fig (a)

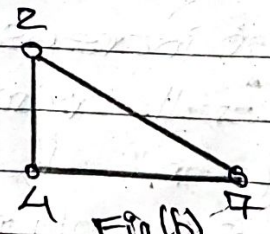
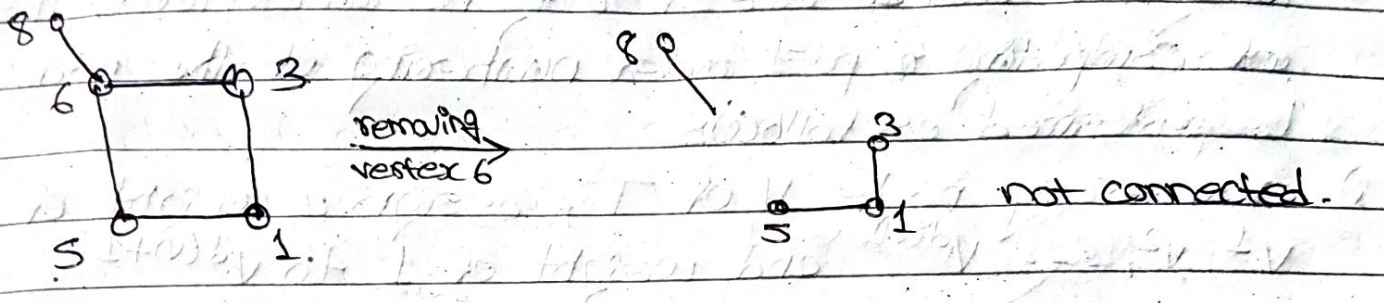


Fig (b)

Page No.
Date : / /

(5) Cut-vertex:

A vertex V of G is said to be a cut vertex if the graph induced by $V - \{v\}$ is not connected. For example, vertex 6 is a cut vertex in the graph induced by $\{1, 3, 5, 6, 8\}$



(6) Bridge (Cut-edge):

An edge $\{x, y\}$ of G is said to be a bridge if the graph obtained from G by removing the edge $\{x, y\}$ is not connected.

(7) Cliques:

A clique in a graph is a set of pairwise adjacent vertices. A clique cover for a graph G is a set of cliques that contain all the vertices of the graph. For example, $\{6, 8\}$, $\{6, 3\}$, $\{5, 1\}$ & $\{2, 4, 7\}$ constitute a clique cover.

(8) Matching:

A matching is a graph is a set of edges sharing no common vertex. For example, the set $(\{6, 8\}, \{1, 5\}, \{2, 7\})$ is a matching.

Page No.
Date : / /

* Tree Algorithms :

① Computing a Post order numbering : #(Left, Right, Root)

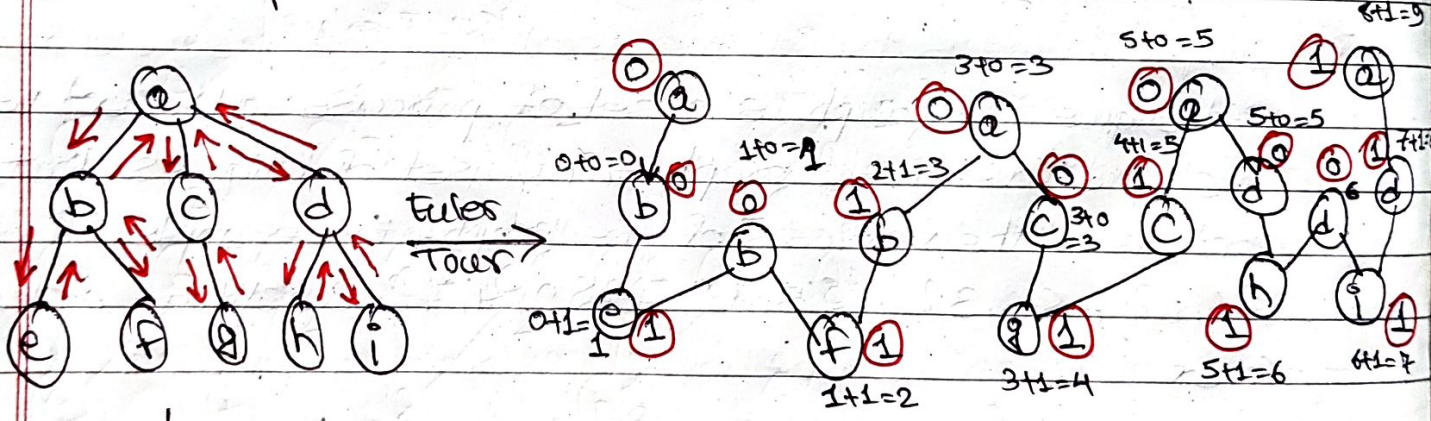
Given, a rooted tree T along its Euler Tours, the task of computing a post order numbering of the nodes T can be performed as follows

i) For every node V of T, assign a weight of 0 to $V^1, V^2, ---, V^{d(v)}$ and weight of 1 to $V^{d(v)+1}$

ii) Compute the weighted rank of every node in the Euler Tour.

iii) For every node $V^{d(v)+1}$, the corresponding weighted rank is the post order of V.

* for example :



post order (Left, Right, Root)

→ e f b g c h i d a

1 2 3 4 5 6 7 8 9

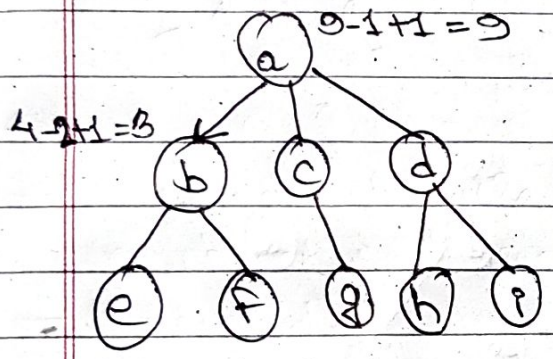
a = 9
b = 3
c = 5
d = 8
e = 1
f = 2
g = 4
h = 6
i = 7

Page No.
Date : / /

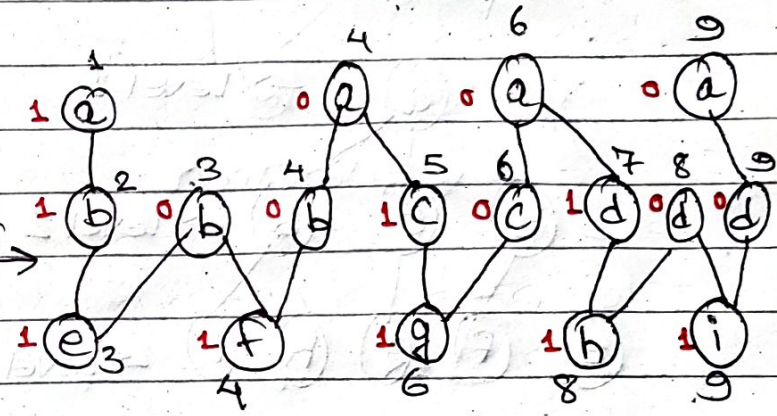
② Computing the number of Descendants:

- Given the rooted Tree T & Euler tour, assign a weight of 1 to u^1 & weight of 0, to $u^2, u^3, \ldots, u^{d(u)+1}$.
- Let $r(u^1)$ and $r(u^{d(u)+1})$ be the corresponding values of prefix sum at u^1 and $u^{d(u)+1}$ respectively.
- The number of descendants of u is exactly,

$$r(u^{d(u)+1}) - r(u^1) + 1$$



Euler Tour →



$a = r(u^{d(u)+1}) - r(u^1) + 1 = 9 - 1 + 1 = 9$

$b = r(u^{d(u)+1}) - r(u^1) + 1 = 4 - 2 + 1 = 3$

$c = r(u^{d(u)+1}) - r(u^1) + 1 = 6 - 5 + 1 = 2$

$d = r(u^{d(u)+1}) - r(u^1) + 1 = 9 - 7 + 1 = 3$

$e = r(u^{d(u)+1}) - r(u^1) + 1 = 1$

$f = 1$

$g = 1$

$h = 1$

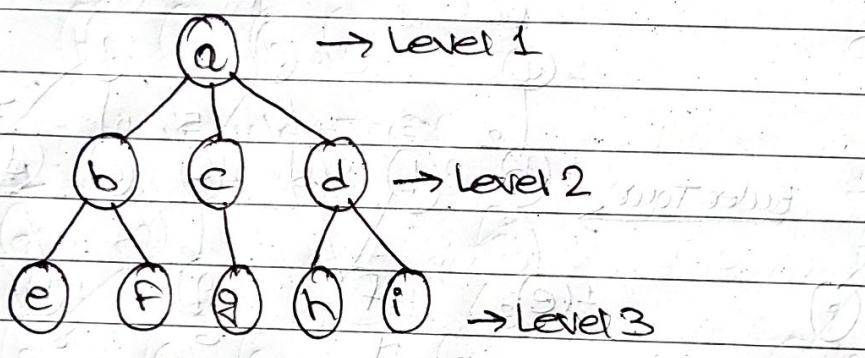
$i = 1$

Page No.
Date : / /

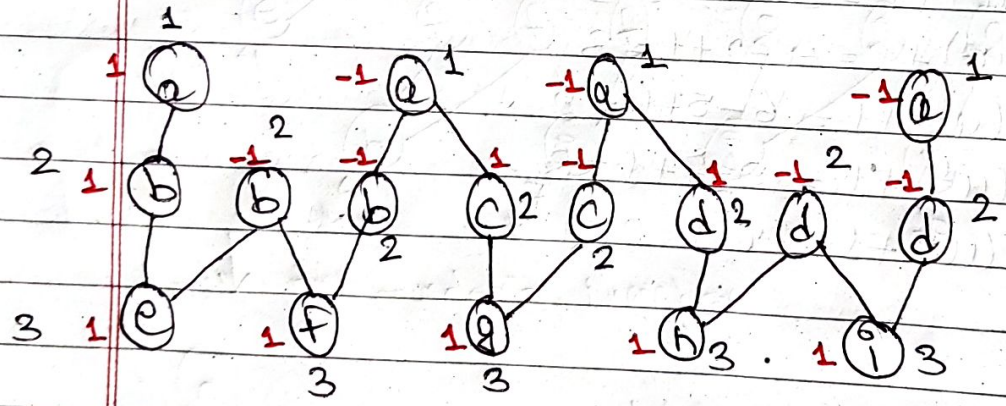
* Level computation:

- The Level of a node V of T is defined as the number of edges on the path joining V to the root.
- Consider a tree T with Euler Tour.
- Assign a weight of 1 to v^1 & weight of -1 to every v^i where $i \neq i$

* example:



↓ Euler Tour

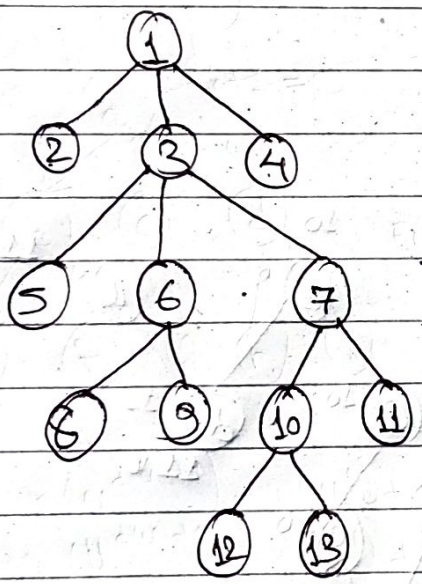


Page No.
Date : / /

* LCA (Lowest Common Ancestor) Computation:

- Let T be a rooted tree with n nodes.
- The lowest common ancestor between two nodes v & w is defined as the lowest node in T, that has both v & w as descendants.

* example:



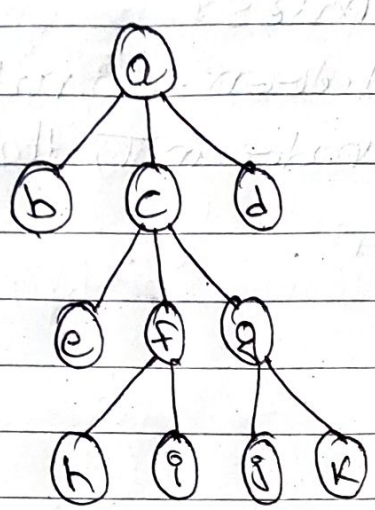
→ LCA(9, 12) = 3

* Algorithm:

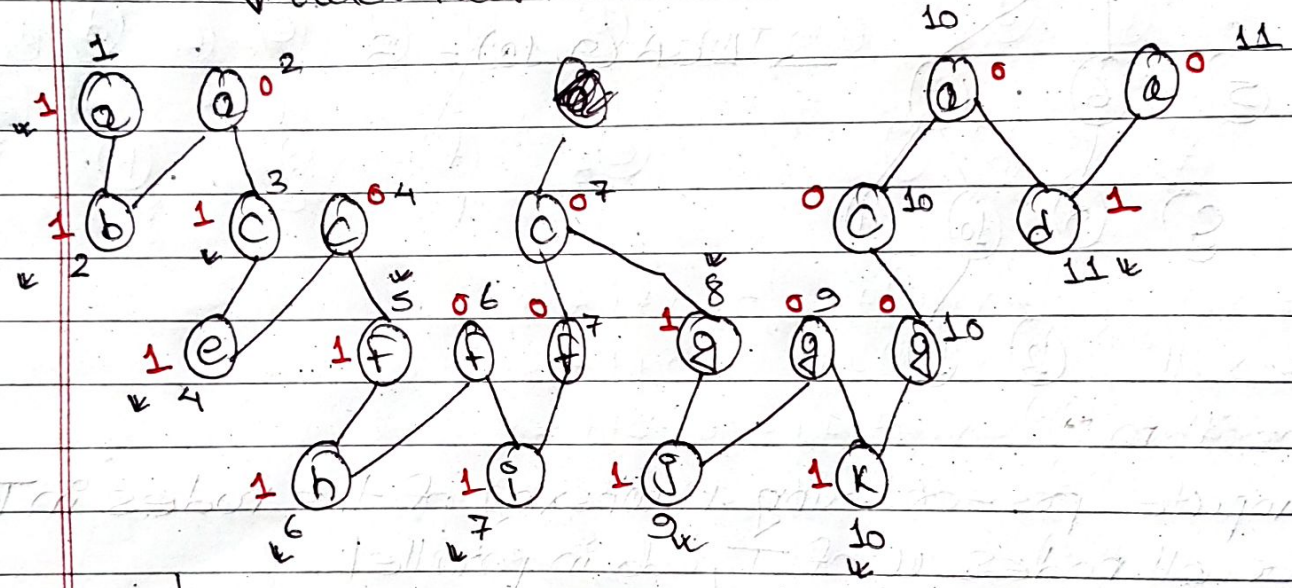
- Compute pre-ordering numbering of the nodes in T
- for all nodes u of T, do in parallel
  - Compute $l(u)$, $r(u)$
- For all $v \in [l(u), r(u)]$, do in parallel
  - $A[u, v] = u$
- For all $x \in [l(w_i), r(w_i)]$ and $y \in [l(w_j), r(w_j)]$, $1 \le i \ne j \le d(u)$ do in parallel.
  - $A[x, y] = u$
- Return A

Page No.
Date : / /

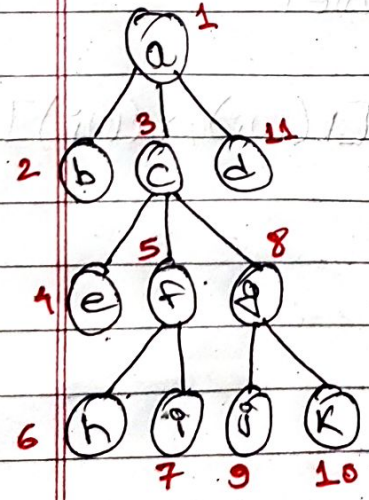
* example:



↓ Euler Tour



↓ pre-ordering Number



→

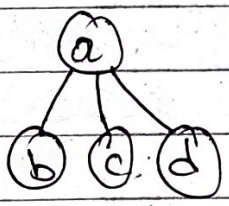
$l(a) = 1, r(a) = 11$
$l(b) = 2, r(b) = 2$
$l(c) = 3, r(c) = 10$
$l(d) = 11, r(d) = 11$
$l(e) = 4, r(e) = 4$
$l(f) = 5, r(f) = 7$
$l(g) = 8, r(g) = 10$
$l(h) = 6, r(h) = 6$
$l(i) = 7, r(i) = 7$
$l(j) = 9, r(j) = 9$
$l(k) = 10, r(k) = 10$

Page No.
Date : / /

	1	2	3	4	5	6	7	8	9	10	11
1	A	A	A	A	A	A	A	A	A	A	A
2	A	B	A	A	A	A	A	A	A	A	A
3	A	A	C	C	C	C	C	C	C	C	A
4	A	A	C	E	C	C	C	C	C	C	A
5	A	A	C	C	F	F	F	C	C	C	A
6	A	A	C	C	F	H	F	C	C	C	A
7	A	A	C	C	F	F	I	C	C	C	A
8	A	A	C	C	C	C	C	G	G	G	A
9	A	A	C	C	C	C	C	G	J.	G	A
10	A	A	C	C	C	C	C	G	G	K	A
11	A	A	A	A	A	A	A	A	A	A	D

Also,

$l(a) = 1, r(a) = 11$ For "a"



$bc = (2,2) \rightarrow (3,10)$

$bd = (2,2) \rightarrow (11,11)$

$cd = (3,10) \rightarrow (11,11)$

$bc = (2,2) \rightarrow (3,10) = (2,3)(2,4)(2,5)(2,6)(2,7)(2,8)(2,9)(2,10)$

$bd = (2,2) \rightarrow (11,11) = (2,11)$

$cd = (3,10) \rightarrow (11,11) = (3,11)(4,11)(5,11)(6,11)(7,11)(8,11)(9,11)(10,11)$

Note: Don't need to evaluate every values instead use the pre-ordering numbering graph.