Date |
Page |

Course Title: Parallel and Distributed Computing
Course No: C.Sc. 544
Nature of the Course: Theory + Lab

Full Marks: 45+30
Pass Marks: 22.5+15
Credit Hrs: 3

Course Description:

- This course covers a broad range of topics related to parallel and distributed computing, including foundations, models, and algorithms.

Course Contents:

Unit 1: Foundations

1.1 Parallel and Distributed Computing: the Scene, the Props, the Players (2hrs)

A Perspective, Parallel Processing Paradigms, Modeling and Characterizing Parallel Algorithms, Cost vs. Performance Evaluation, Software and General Purpose PDC

1.2 Semantics of Concurrent Programming (2hrs)

Models of Concurrent Programming, Semantic Definitions, Axiomatic Semantic Definitions, Denotational semantics Definition Operational Semantic Definitions

Date
Page

1.3 Formal Methods : A Petri Nets Bases Approach (2hrs)
Process Algebras, PETRI Nets, High-Level New Models

1.4 Complexity Issues in Parallel and Distributed Computing (3hr
Introduction, Turing Machines as the Basis and Consequences, Complexity Measures for Parallelism, Parallel Complexity Models and Resulting Classes, VLSI computational Complexity, Complexity Measures for Distributed Systems, Neural Networks and complexity Issues.

1.5 Distributed Computing Theory (3hrs)
The computational Model, A simple Example, Leader Election, Sparse Network Covers and Their Applications, Ordering of Events, Resourse Allocation, Tolerating Processor Failures in Synchronous systems, Tolerating Processor Failures in Asynchronous Systems, Other Types of Failures, Wait-Free Implementations of Shared Objects.

Unit 2 : Models

2.1 PRAM Models (5hrs)
Introduction; Techniques for the Design of Parallel Algorithms; The PRAM Model; Optimality and Efficiency of Parallel Algorithms; Basic PRAM Algorithms; The NC-Class; P-Completene Hardly Parallelizable Problems, Randomized Algorithms and Parallelism; List Ranking Revisited; Optimal $O(\log n)$ Deterministic List Ranking; Taxonomy of Parallel Algorithm Deficiencies of the PRAM Model.

Date |
Page |

2.2 Broadcasting with Selective Reduction: A Powerful Model of Parallel Computation (3 hrs)
Introduction; A Generalized BSR Model; One Criterion BSR Algorithms; Two criteria BSR Algorithms; Three Criteria BSR Algorithms; Multiple Criteria BSR Algorithms

2.3 Dataflow Models (3 hrs)
Kinds of Dataflow; Data-Driven Dataflow Computing Models; Demand-driven Dataflow Computing Models; Unifying Data-Driven and Demand-Driven

2.4 Partitioning and Scheduling (4 hrs)
Program Partitioning; Task scheduling; scheduling System Model; Communication Models; Optimal Scheduling Algorithm Scheduling Heuristic Algorithms; Scheduling Nondeterministic Task Graphs; Scheduling Tools; Task Allocation; Heterogeneo Environments

2.5 Checkpointing in Parallel and Distributed System (3 hrs)
Introduction; Checkpointing Using Task Duplication; Techniques for Consistent checkpointing

2.6 Architecture for Open Distributed software System (2 hrs
Introduction to Open Distributed System Architecture; Computational Model; Engineering Model; OOP Application

Date
Page

Unit 3 : Algorithms

3.1. Fundamentals of Parallel Algorithms (8 hrs)

Introduction; Models of Parallel Computation; Balanced Trees; Divide and Conquer, Partitioning; Combining

3.2. Parallel Graph Algorithms (5 hrs)

Graph-Theoretic Concepts and Notation. Tree Algorithms. Algorithms for General Graphs; Algorithms for Particular Classes of Graphs

3.3. Data Parallel Algorithms (5 hrs)

Chapter Overview; Machine Model; Impact of Data Distribution; CU/PE Overlap; Parallel Reduction Operations; Matrix and Vector Operations; Mapping Algorithms onto Partitionable Machines; Achieving Scalability Using Set of Algorithms

Recommended Books

Unit-1

Date |
Page |

Unit 1.1 : Parallel and Distributed Computing

Introduction

- In parallel computing, all processors may have access to a share memory to exchange information between processors.
- In distributed computing, each processor has it's own private memory, where information is exchanged by passing message between processors (RPC, RMI)

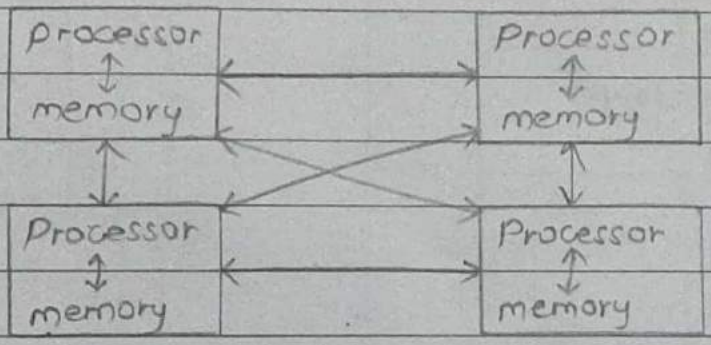


Fig: Distributed computing

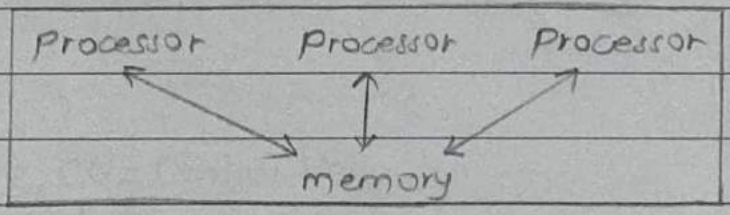


Fig: Parallel computing

§ Why do we need Parallel Computing?

- Serial computing is too slow
- need for more computing power (e.g cryptography, data mining, search engine)

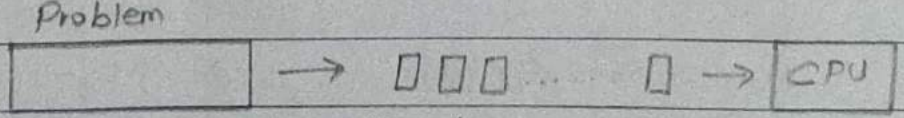


Fig: Traditional approach

Date
Page

Problem

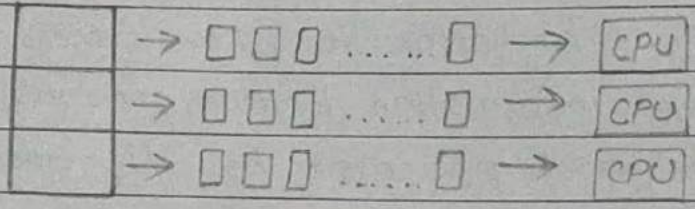
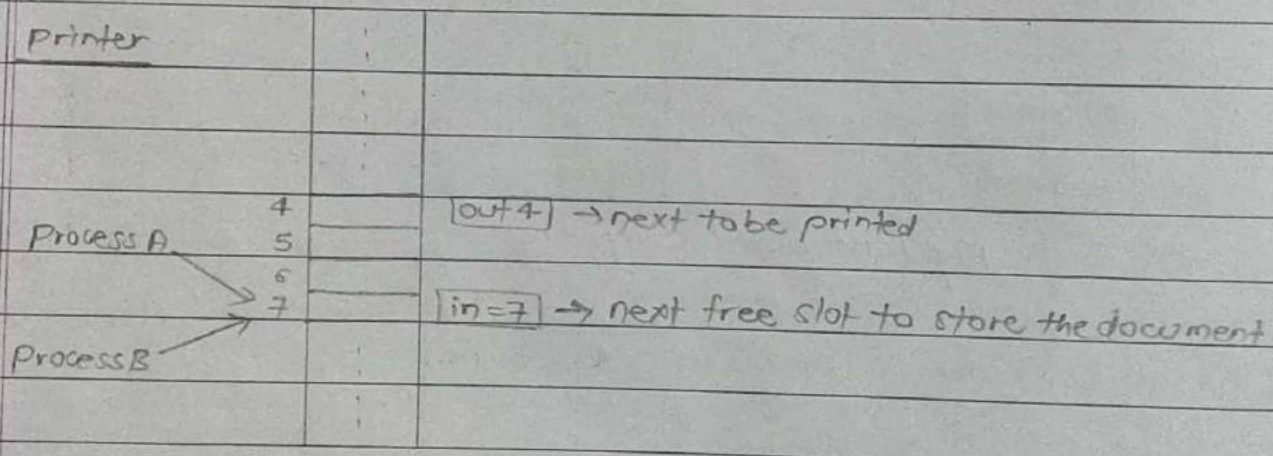


Fig: Parallel computing approach

Issues in Sharing resources

Spooler Directory



Date |
Page |

Parallel Processing Paradigms

- Flynn's Taxonomy
  1. SISD (Single Instruction Single Data)
  2. SIMD (Single Instruction Multiple Data)
  3. MISD (Multiple Instruction ~~M~~ Single Data)
  4. MIMD (Multiple Instruction Multiple Data)

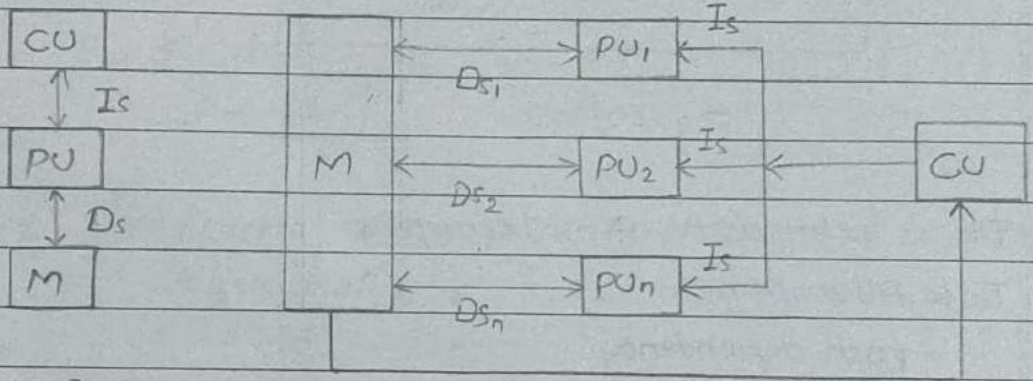


Fig: SISD

Fig: SIMD

Where, CU = Control Unit
PU = Processing Unit
M = Memory
Is = Instruction Stream
Ds = Data stream

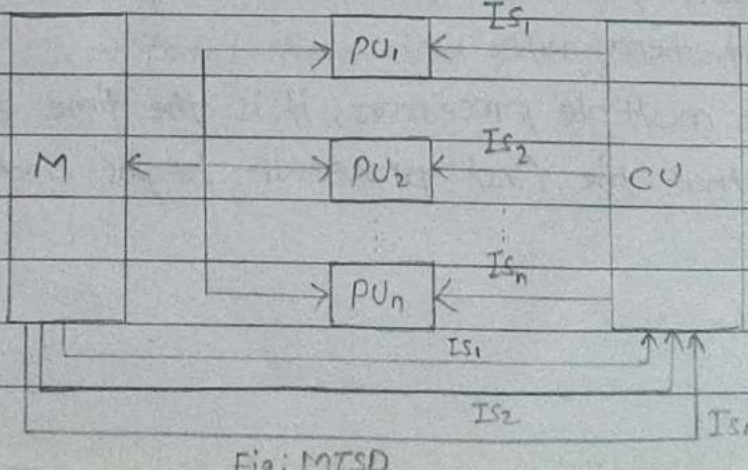


Fig: MISD

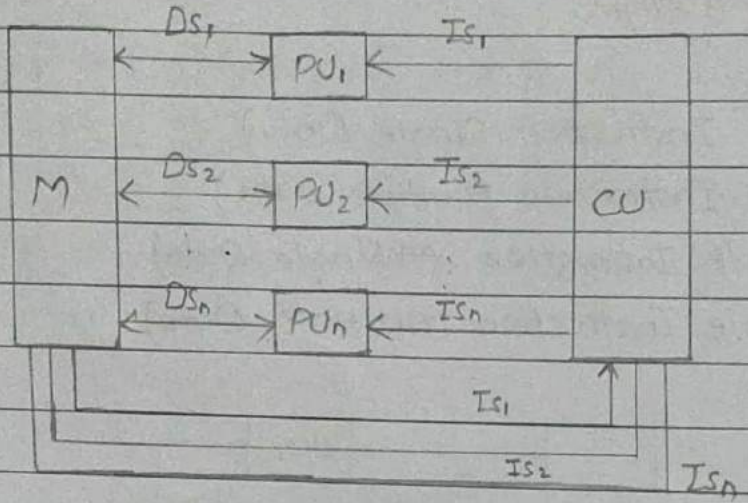


Fig: MIMD

Issues related to parallelization

1. Task Allocation
   - task dependency
   - Load balancing

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

3. 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 and the last one terminates.

Date |
Page |

4. Speedup

- Speedup (S) = $\frac{\text{Running time of the best sequential algorithm}}{\text{Running time of the parallel algorithm}}$

- it is not always possible to decomposes 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, $f \rightarrow$ fraction of computation task must be done sequentially.

no speedup
$S_{max} = 1$, Where $f = 1 \rightarrow$ every job sequentially
$S_{max} = 0$, Where $f = 0 \rightarrow$ every job parallelly
full speedup

5. Communication Penalty

$$CP_i = \frac{E_i}{C_i}$$

Where, $E_i \rightarrow$ total execution time spent by P_i
$C_i \rightarrow$ total time used for communication by P_i.

Date
Page

Unit 1.2 : Semantics of Concurrent Programming

- Concurrent programming refers to activation of more than one instruction at a time
- Semantics is the field of concerned with the mathematical study of the meaning of the programming language.
- Syntactic vs semantic structure.

eg.

English language	programming language

Categories

1. Axiomatic semantic Definition
2. Operation semantic Definition
3. Denotational semantic Definition

Models of Concurrent Programming

1. Level of granularity
2. Sharing the clock
3. Sharing the memory (mutually exclusive)
4. Pattern of interaction
   - 4.1 Synchronization → mutual exclusion/admission
   - 4.2 Communication → Synchronous / Asynchronous

1. Axiomatic Semantic Definition

- defines the meaning of language construct by making statement 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, $P \rightarrow$ pre-condition
$S \rightarrow$ statement
$Q \rightarrow$ Post-condition

eg. $\{x=5\}\ x=x+1\ \{x=6\}$

eg. 1. $\{j=3 \text{ AND } K=4\}\ j=j+k\ \{j=7 \text{ AND } K=4\}$
2. $\{x \geq 0\}$ while $(x \neq 0)$ do $x=x-1\ \{x=0\}$
3. $\{x \leq N-1\}\ x=x+1\ \{x \leq N\}$

Partial Correctness
- if answer is returned, it will be correct

Total Correctness
- it requires that the algorithm terminates

Sequence Statement Rule
- It has the form

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

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

eg.
$$\frac{x>1\ \{x=x+1\}\ x>2 \qquad x>2\ \{x=x+1\}\ x>3}{x>1\ \{x=x+1, x=x+1\}\ x>3}$$

Date
Page

i.e $\dfrac{\{P\}\ S1\ \{Q\},\ \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1, S_2\ \{R\}}$

Alternation Statement Rule

IF - THEN

$$\frac{\{P \text{ and } B\}\ S\ \{Q\} \quad \{P \text{ and NOT } B\} \supset \{Q\}}{\{P\}\ \text{IF } B \text{ THEN } S \text{ END IF } \{Q\}}$$

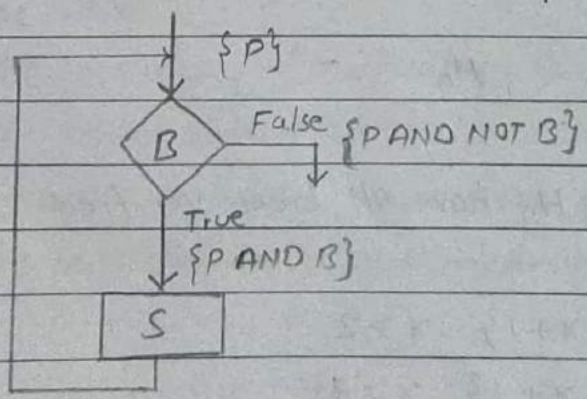
IF - ELSE

$$\frac{\{P \text{ and } B\}\ S_1\ \{Q\} \quad \{P \text{ and NOT } B\}\ S_2\ \{Q\}}{\{P\}\ \text{IF } B \text{ THEN } S_1 \text{ ELSE } S_2 \text{ END IF } \{Q\}}$$

Loop

While

$$\frac{\{P \text{ and } B\}\ S\ \{P\}}{\{P\}\ \text{While } B \text{ do } S \text{ END While } \{P \text{ and NOT } B\}}$$



eg.

$$\frac{x>0 \wedge x<10\ \{x=x+1\}\ x>0}{x>0\ \{\text{While } x<10 \text{ do } x=x+1\}\ x \geq 10 \wedge x>0}$$

Disjoint Parallel Program

- Two programs S_1 and S_2 are said to be disjoint if none of them can change the variables accessed by the other.
- Denoted by $[S_1 \parallel S_2]$

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

$$\frac{\{P_1\}\ S_1\ \{Q_1\} \quad \{P_2\}\ S_2\ \{Q_2\}}{\{P_1 \text{ AND } P_2\}\ [S_1 \parallel S_2]\ \{Q_1 \text{ AND } Q_2\}}$$

Await then Rule

- Await 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.
- Semantically, it can be denoted as,

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

Operational Semantics

- Focus on how the state of the program is affected.
- i.e how a computation is performed.
- Example, use the notation $\langle P, s \rangle$, means semantic of program P at state s

Date
Page

$\langle z=x,\ x=y,\ y=z,\ [x \to 5,\ y \to 7,\ z \to 0] \rangle$

$\langle x=y,\ y=z,\ [x \to 5,\ y \to 7,\ z \to 5] \rangle$

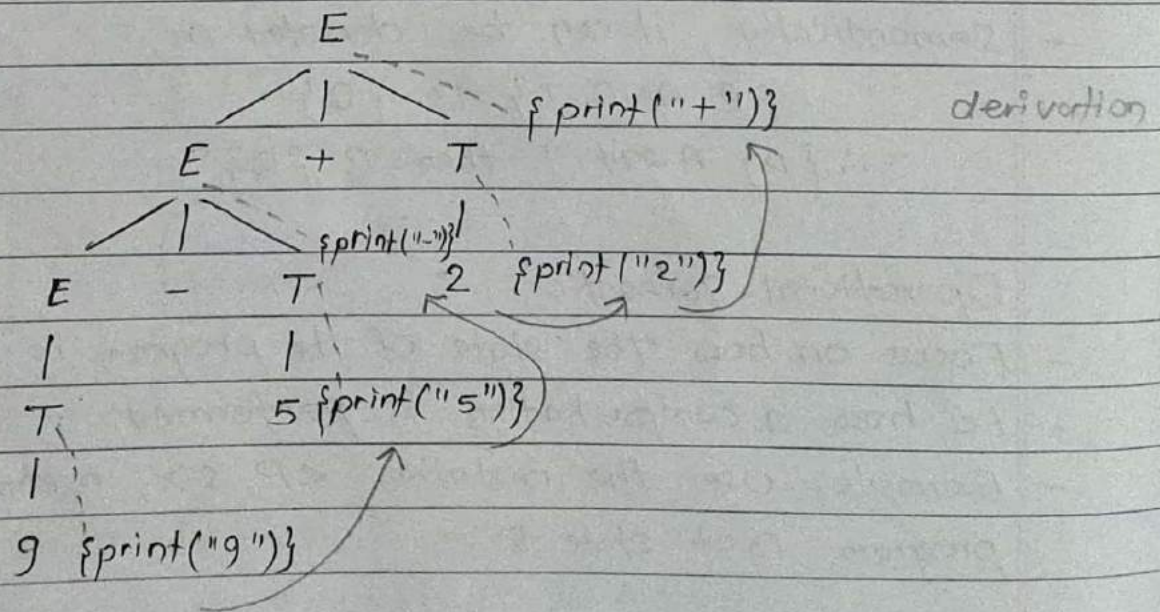
$\langle y=z,\ [x \to 7,\ y \to 7,\ z \to 5] \rangle$

$\langle \epsilon,\ [x \to 7,\ y \to 5,\ z \to 5] \rangle$

- Example, Syntax directed translation can also be represented as operational semantics

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")}

input string ⇒ 9 - 5 + 2



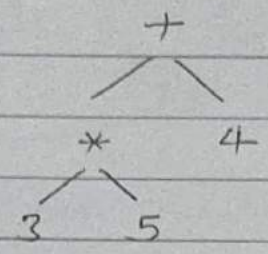
Date
Page

Denotational Semantics

- Each construct of the language is mapped into a mathematical objed that defines its meaning.

Example

$3 * 5 + 4 \Rightarrow$



- Example; also the ordered pair can be used to represent the program

Eg. fact(n) = if (n == 0) the 1
else n * fact(n-1)

- can be viewed as in ordered pair for denotational semantics as $<n, n!>$
- $<0,1>, <1,1>, <2,2>, <3,6>, <4,24>$.

Example: Communicating Sequential Process

- 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 process which first engage in that event x and behaves exactly as described by Q
- Example: a vending machine
  1. $VM_0 \overset{def}{=} (coin \rightarrow (candy \rightarrow stop))$
  2. $VM_1 \overset{def}{=} (coin \rightarrow (candy \rightarrow VM_1))$
  3. $VM_2 \overset{def}{=} (coin \rightarrow (candy \rightarrow VM_2)) \,||\, (note\ bill \rightarrow (toffee \rightarrow VM_2))$

Date
Page

Unit 1.3 Process Algebra

Transition system

- Formally, it can be defined as

  $A' = (A, S, \rightarrow, S_0)$, where $A \rightarrow$ set of alphabet
  $S \rightarrow$ set of states
  $\rightarrow \rightarrow$ transition $S \times A \rightarrow S$
  $S_0 \rightarrow$ starting state

Conventions

- $a, b, c, \ldots$ are used to indicate elements of A
- $u, v, \ldots$ are used to indicate $A \cup \{\tau\}$
- ω is used to indicate $(A \cup \{\tau\})^*$
- $a\omega$ is used to indicate A^*

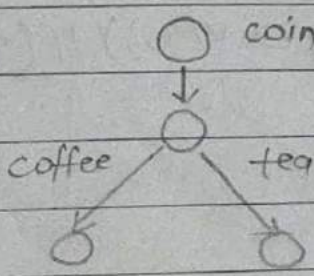
Example:

$S \xrightarrow{a\omega} S'$ means $S = S_1 \xrightarrow{a_1} S_2 \xrightarrow{a_2} S_3 \xrightarrow{a_3} \ldots \xrightarrow{a_n} S'$

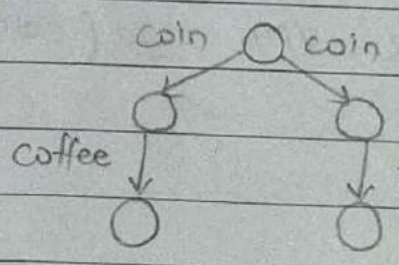
Reachable states

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

Example:



fig(i) Non-deterministic



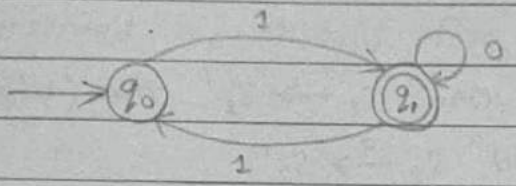
fig(ii) Deterministic

Date
Page

Process Algebra

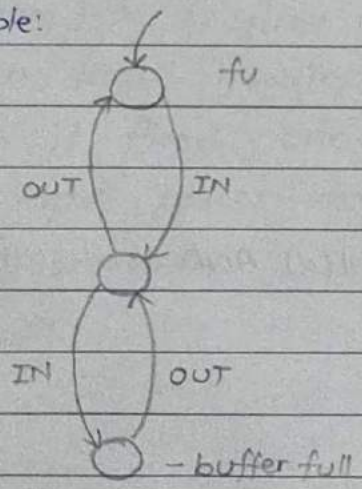
- System behavior generally consists of process and data
- Process are the control machanism for the manipulation of data.

Eg. DFA

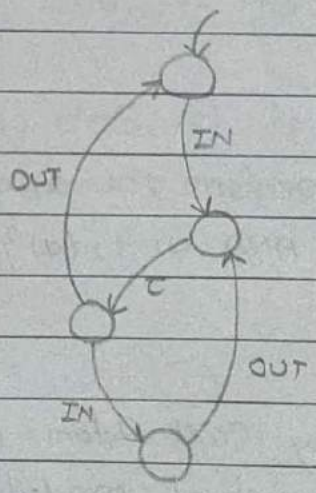


- is a labelled transition system in which one state is considered as root state
- for the mathematical reasoning, process graphs are expressed algebraically called Process Algebra.
- Consists of two types of actions
  1. Observable action → explicit
  2. Unobservable action → implicit and denoted by τ.

Example:



Fig(i) Transition system representing two place buffer



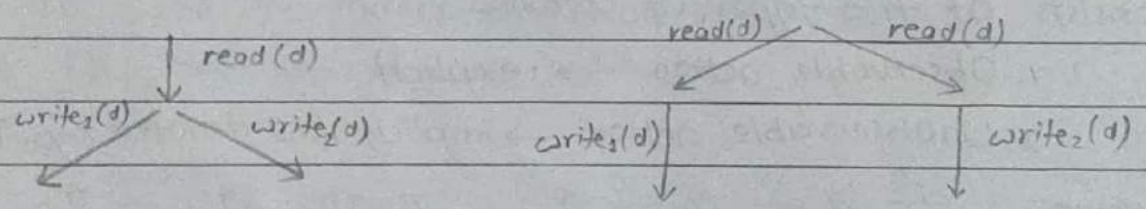
Fig(ii) Transition system representing two one place buffer

Date
Page

Observation bisimilarity

- Let $A_i' = (A_i, S_i, \rightarrow_i, S_{0i})$, $i = 1, 2, \ldots$ be two transition system
- 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,
  1. $(S_{01}, S_{02}) \leftarrow B$
  2. If $(S_1, S_2) \leftarrow B$ and $S_1 \rightarrow S_1'$ then there is S_2', such that $(S_1', S_2') \leftarrow B$ and $S_2 \xrightarrow{a} S_2'$
  3. If $(S_1, S_2) \leftarrow B$ and $S_2 \xrightarrow{a} S_2'$ then there is S_1', such that $(S_1', S_2') \leftarrow B$ and $S_1 \xrightarrow{a} S_1'$

Example: Consider the following two processes



Fig(I) Fig(II)

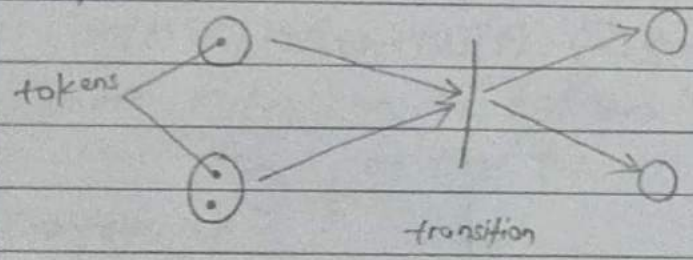
- Both system perform the same action
- i.e {read(d) AND $write_1(d)$} OR {read(d) AND $write_2(d)$}

Petri Nets

- Developed by Carl Adam Petri
- is the mathematical modelling language for the description of the system.
- Petri Nets (PN) are represented through directed graph (bi-partite graph)

- consists of two types of nodes
  1. Place (Buffer that holds something (sth))
     - represented by circle
  2. Transition (event)
     - represented by straight line (|) or rectangle (▭) or box (■)

Example:



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

Example:

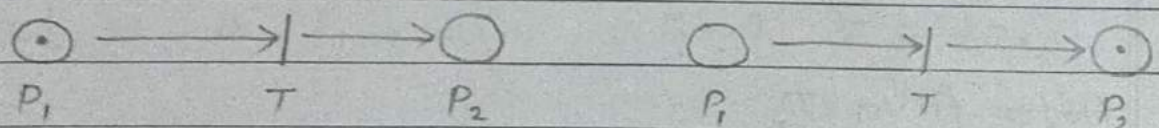


Fig: Before firing T

Fig: After firing T

Example:

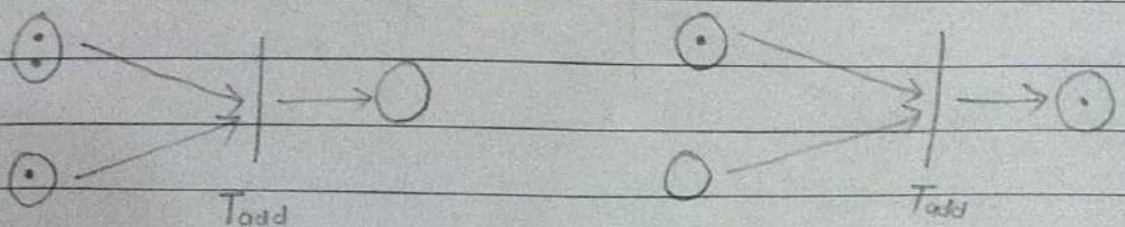
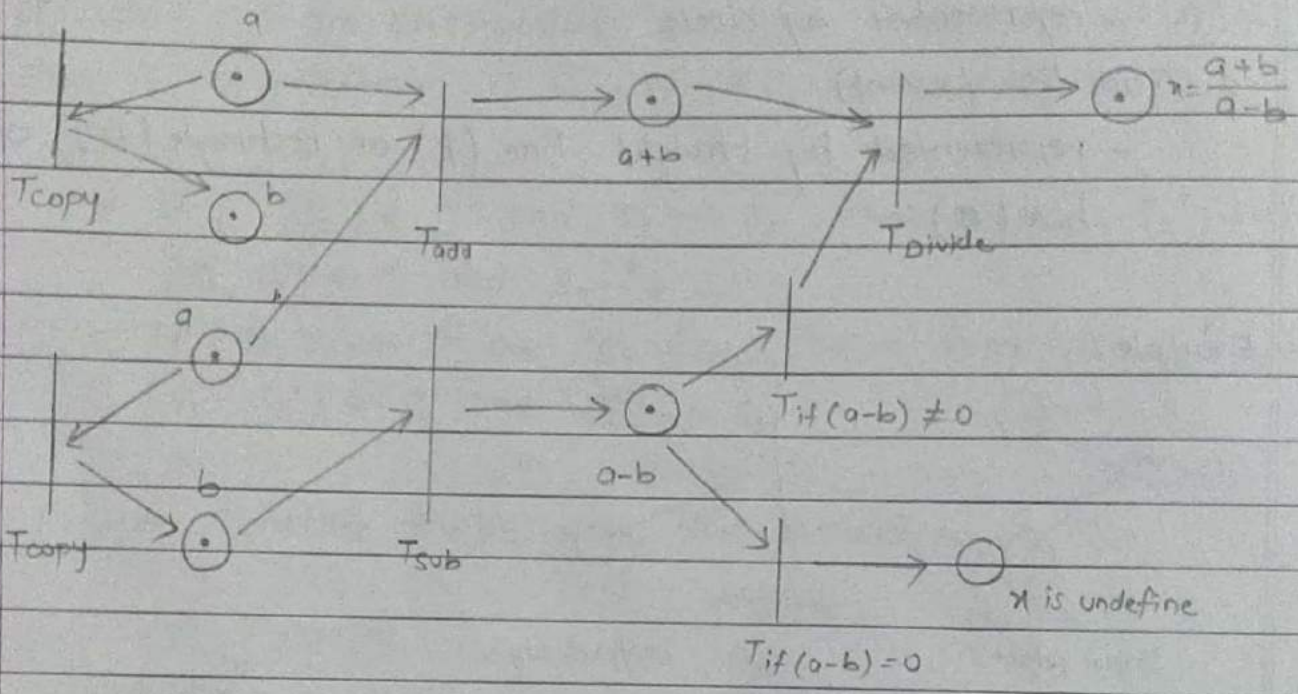


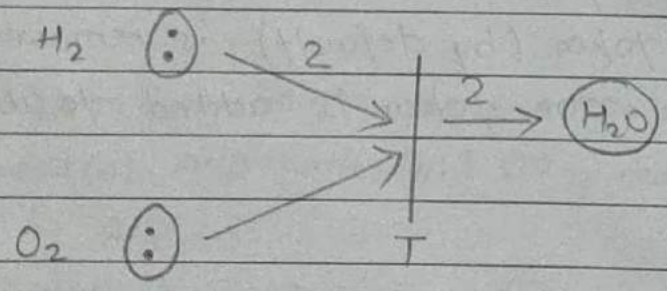
Fig: Before firing of T

Fig: After firing of T

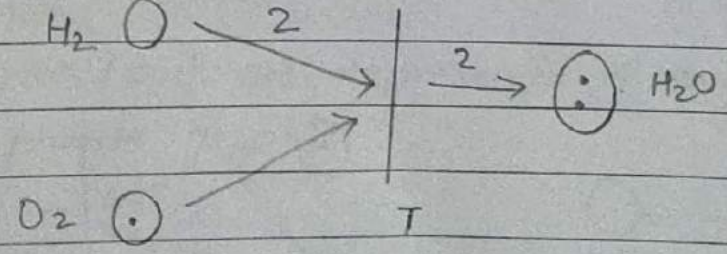
Example: $x = \frac{a+b}{a-b}$



Example: $2H_2 + O_2 \rightarrow 2H_2O$



After firing T,



Date
Page

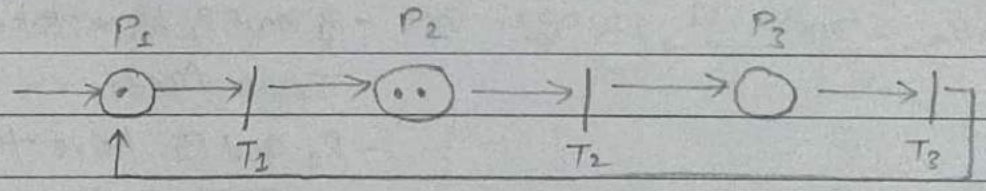
Formal Definition of Petri nets

- $PN = (P, T, I, O, M_0)$ where
  - $P \rightarrow$ set of places $= \{P_1, P_2, \ldots, P_m\}$
  - $T \rightarrow$ set of transitions $= \{T_1, T_2, \ldots T_n\}$
  - $I \rightarrow$ is an input matrix of $n \times m$
  - $O \rightarrow$ is an output matrix of $n \times m$
  - $M_0 \rightarrow$ represents initial marking of ~~matrix~~ machine.
- a marking of a PN at time t,

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

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

Example:



$P = \{P_1, P_2, P_3\}$

$T = \{T_1, T_2, T_3\}$

$$I = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \end{array} \begin{array}{c} \begin{array}{ccc} P_1 & P_2 & P_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}_{3\times3}$$

$$O = \begin{array}{c} \\ T_1 \\ T_2 \\ T_3 \end{array} \begin{array}{c} \begin{array}{ccc} P_1 & P_2 & P_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array}_{3\times3}$$

$$M_{t_0} = \begin{array}{c} \begin{array}{ccc} P_1 & P_2 & P_3 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \end{array}$$

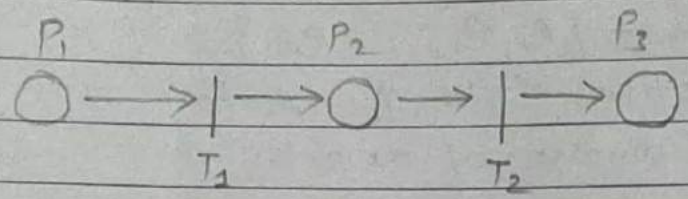
suppose T_2 is fired

$M_{t_1} = (1 \quad 1 \quad 1)$

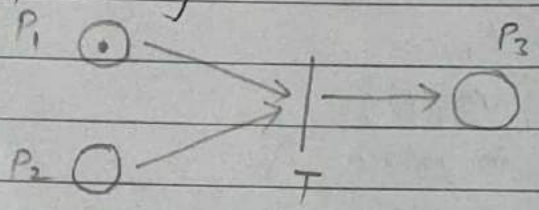
Date
Page

Variations in PN

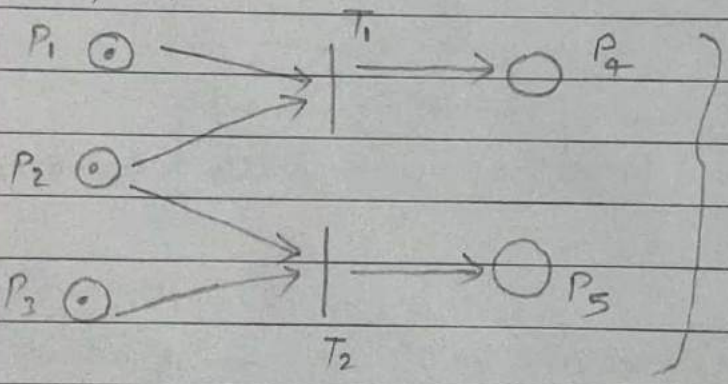
1. Sequential action



2. Dependency

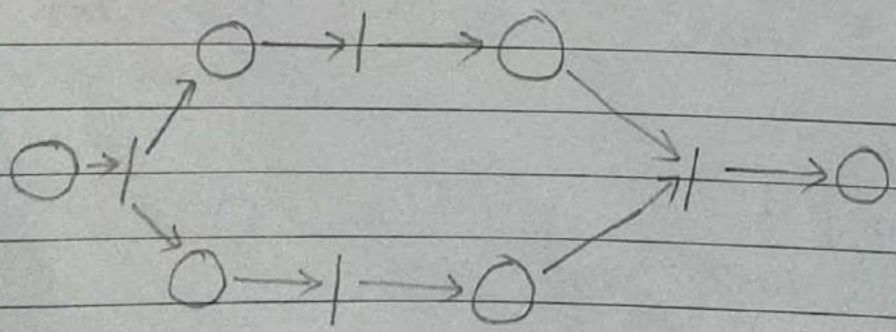


3. Conflict

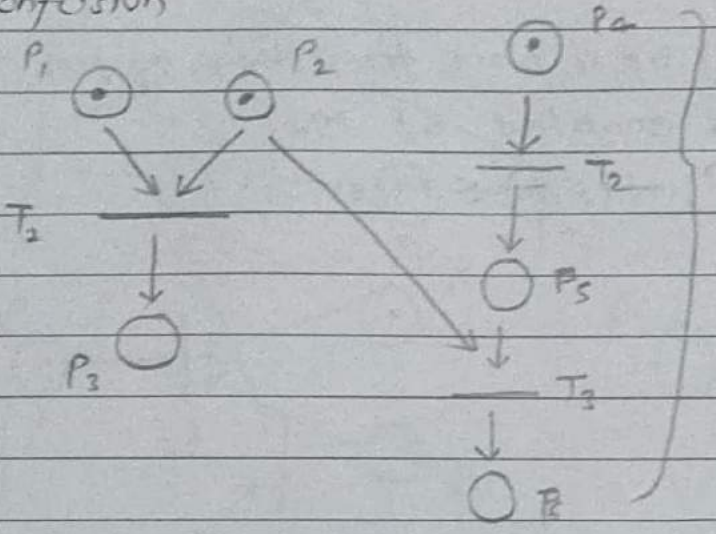


- P_1 and P_2 have tokens but not P_3, T_1 is fired
- P_2 and P_3 have tokens but not P_1, T_2 is fired.
- If all have tokens which one to fire (conflict).

4. Concurrency



5. Confusion



Two possible interpretation

1. T_1 occurred first without being in conflict with other events then T_2 occurs.
2. T_2 occured first and then T_1, T_3 got in conflict.

Pure Petri Net

- Let $PN = \{P, T, F\}$, PN is pure if $\forall x \in X : {}^\bullet x \cap x^\bullet = \phi$
- i.e no self loop

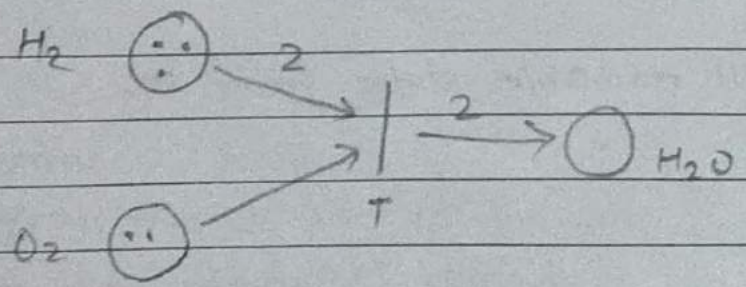
Eg.

not pure Petri Net

Place/Transition Net (P/T net)

- defined by 4 tuples (P, T, F, W) where (P, T, F) is a petri net and W is a weight function.

Eg.



Date
Page

Transition rule

- Let $M = (P, T, F, W)$ be a place transition system, then a transition $t \leftarrow T$ is enabled at M_0,

  $M_0[t > \text{if } \forall p \in P, W(p,t) \leq M(s)$

Eg.

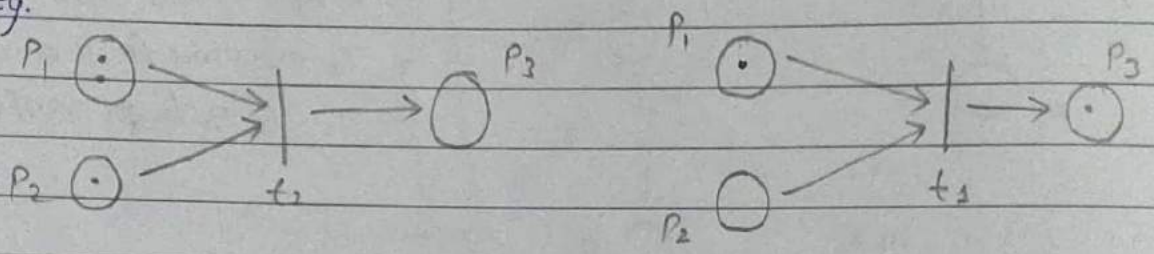


Fig: Before firing t_1 Fig: After firing t_1

- $M_0[t_1 > M_1$, $M_0 = (2, 1, 0)$, $M_1 = (1, 0, 1)$

Properties of Petri Net

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

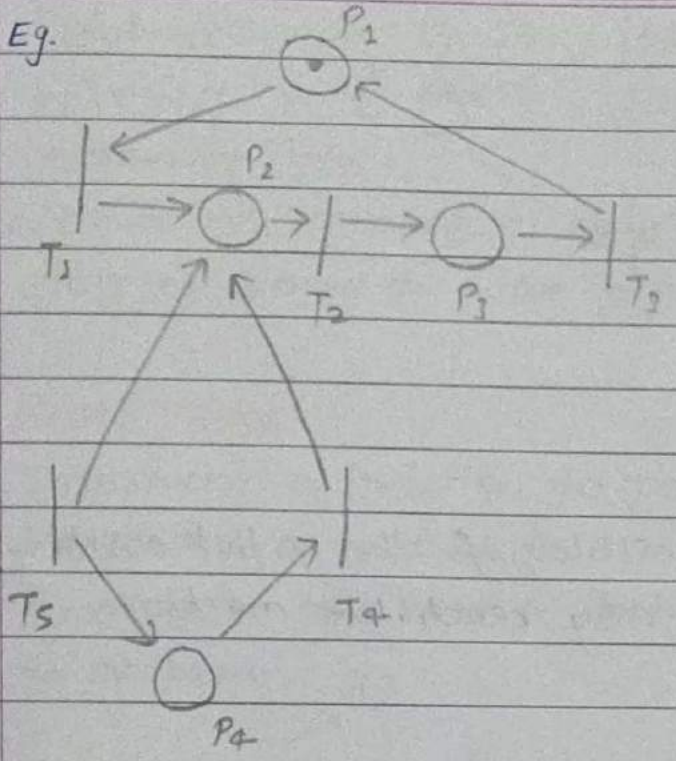
1 Reachability

- a marking M is said reachable from another marking M', if there exist a sequence, σ of transitions such that

  $M'[\sigma > M$

- $R(M_0) \rightarrow$ set of all reachable states from M_0.

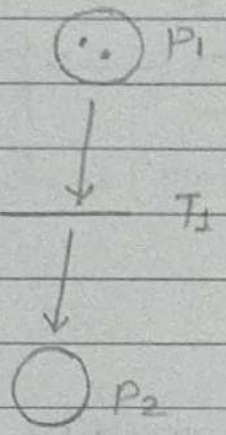
Eg.



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

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

Eg.

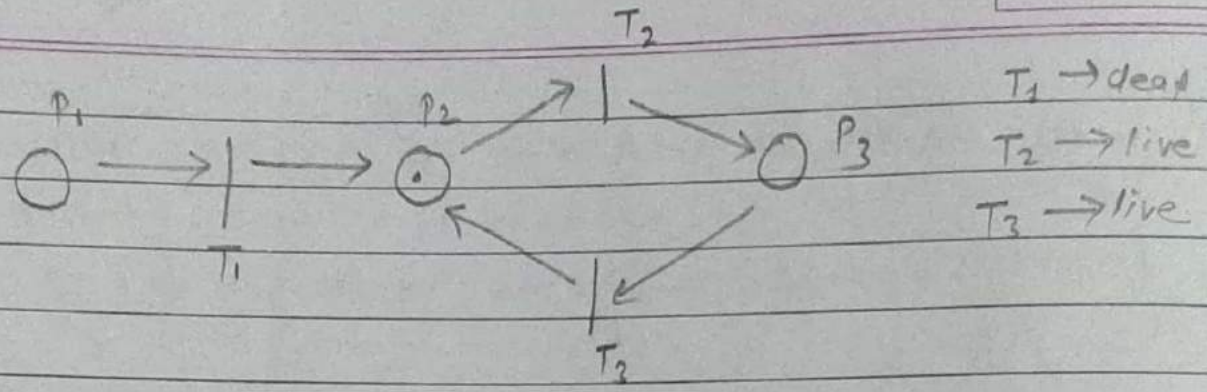


$M_0 = (2, 0)$

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

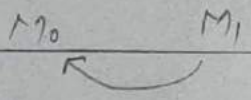
2. Boundness
- no. of tokens in a place is bounded
- a place is said to be k-bounded, if the ~~no~~ number of tokens in P never exceed k.
- a petri net is k-bounded if all of its places are k-bounded.

3. Liveness
- a transition is said to be live it can always be made from any reachable marking
- a transition is deadlock if it can be never fire



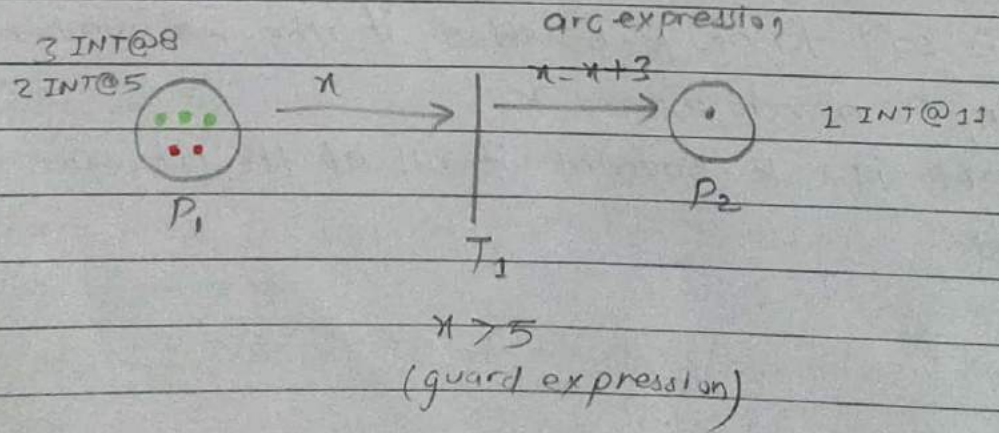
4. Reversibility

- a petri net is said reversible, if the initial marking (M_0) remains reachable from any reachable marking.



High level Petri Net

- consists of following properties
  1. Petri Net
  2. Marking Scheme (color)
  3. Transition Guard
  4. Arc expression



Assignment-1

Date
Page

Unit 1.4: Complexity Issues in Parallel and Distributed Computing

Complexity Theory

- estimate the amount of computational resources (time and storage) need to solve a problem.

Basic Terms

1. Introducing a notation to specify complexity
   - introduce a mathematical notation to specify the functional relationship between the input and size of the problem.

2. Choice of machine model to standardize the measure
   - specifying the underlying machine
   - RAM, PRAM

Sequential Algorithm Parallel Algorithm

3. Refinement of the measure of parallel computation
   - how fast can we solve the problem, when a number of processors are put together to work in parallel.

Turing machine as a basis on 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.

Date
Page

1. Simplicity of abstraction
- simple model of an abstract machine that consists of three components.
  a. Finite state machine
  b. A read/write head
  c. Infinite tape memory

DFA → RE
↑ x
CFG → PDA

- formally, it can be defined as
  $TM = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where
  Q = set of states
  Σ = set of input alphabets
  Γ = set of tape symbols
  δ = Transition function
  q_0 = starting state
  B = Blank symbol
  F = set of final states

$S \times \Sigma = S \times \Gamma \times \{S, L, R\}$

2. Dual nature of TM
- Universal Turing Machine (UTM)

TM → (TM)

3. Inclusion of non-determinism
- NDTM → alternative move,
- branching tree called OR tree
- a computation halts, if there is a sequence that leads to an accepting state
- rejects an input if and only if there is no possible sequences of moves.

Realization of unbounded parallelism

- NDTM can be simulated using DTMs by allowing unbounded parallelism in computation
- DTM that reaches a solution early, halts all other DTM, if it fails, it only terminates.

Provision of easy abstract measure for resources

- DTM ⇒ By counting each move as one unit time and each cell as scanned as one space time.

- NDTM ⇒ Complexity is the number of moves required for an accepting sequence.

ATM (Alternating Turing Machine)

- States of ATM can be partitioned as

1. Universal state → AND tree
2. Existential state → OR tree
3. Normal state → computation lead to acceptance if and only if its unique descendent is accepting.

Date
Page

Parallel Complexity Models

- Synchronous parallel computation can be used to solve problem like matrix multiplication
- It is because all identical processors work under the control of centralized clock.

VLSI computational complexity

- practical use of algorithm depends upon the design of VLSI.
- two fundamental parameters deciding the efficiency of VLSI is
  1. Area of circuit (space complexity)
  2. Time taken to provide an output for a given n input (Time complexity)

Date
Page

Unit 1.5

Distributed Computing

- Major abstacles is distributed computing is uncertainty due to differing processor speed and occarsional 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 state machine with state sets Q_i.
- The state set Q_i contains a distinguished initial state $q_{0,i}$.
- Each processor P_i, contains $buff_i$.

Leader Election

- In distributed computing, leader election is the process of designating a single processor as the leader of some task distributed among several computers.
- The existence of leader can simplify co-ordination among the processors and is helpful in achieving fault tolerance and saving resources.
- Election Algorithm
    1. Bully Algorithm
    2. Ring Algorithm

1. Bully Algorithm

- The biggest guy in town always win.
- When a process notices that the co-ordinator is not

responding to request, it initiates an election.

- Procedure
  1. Each process has a unique processor ID
  2. Processes know the IDs and address of every other processes
  3. Communication is assumed reliable
  4. Process initiates election if the co-ordinator failed
  5. Select the processor with highest ID.

Example:

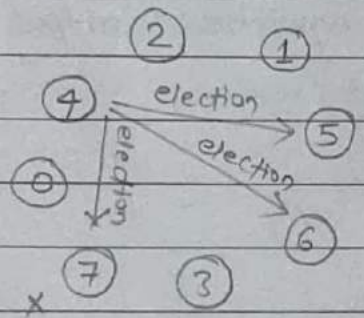


Fig (a)

- currently the leader is 7 and is not responding
- suppose 4 initiates the election

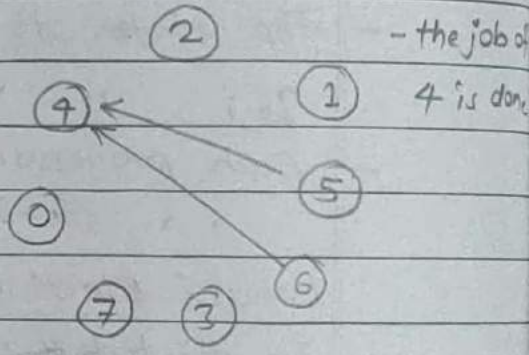


Fig (b)

- the job of 4 is done

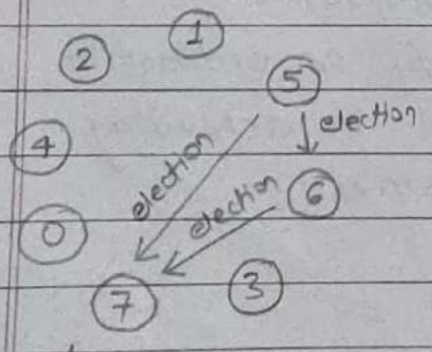


Fig (c)

- both 5 & 6 initiates election

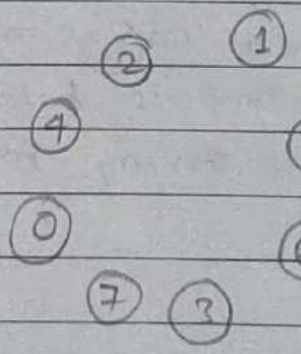


Fig (d)

- 6 is winner

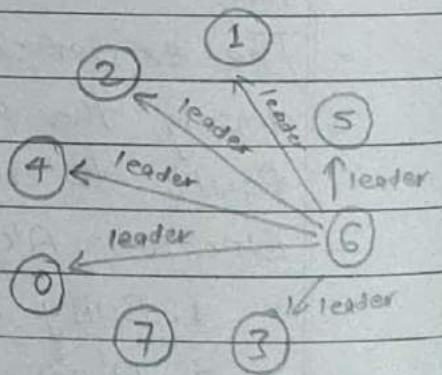


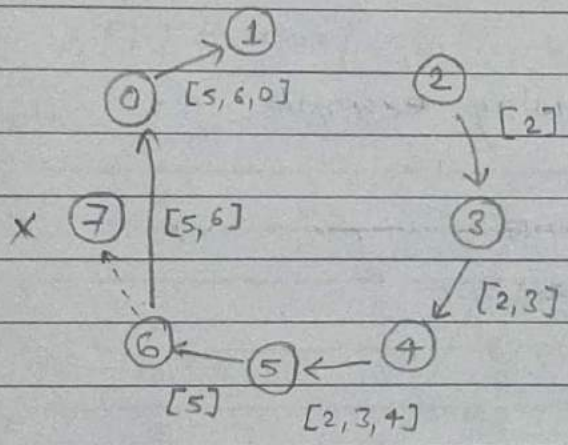
Fig (e)

- 6 is announcing

Date |
Page |

Ring Algorithm

- 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.
- When the election message reaches the originator, the election is complete
- Leader is choosen based on the highest numbered process.



step 1: 2 and 5 are initiation the election

step 2: they forward the election message to its neighbor until the circuit builds

step 3: 2 → [2, 3, 4, 5, ⑥, 0, 1]

5 → [5, ⑥, 0, 1, 2, 3, 4]

Both 2 and 5 elects 6 as new leader.

Date
Page

Ordering of Events

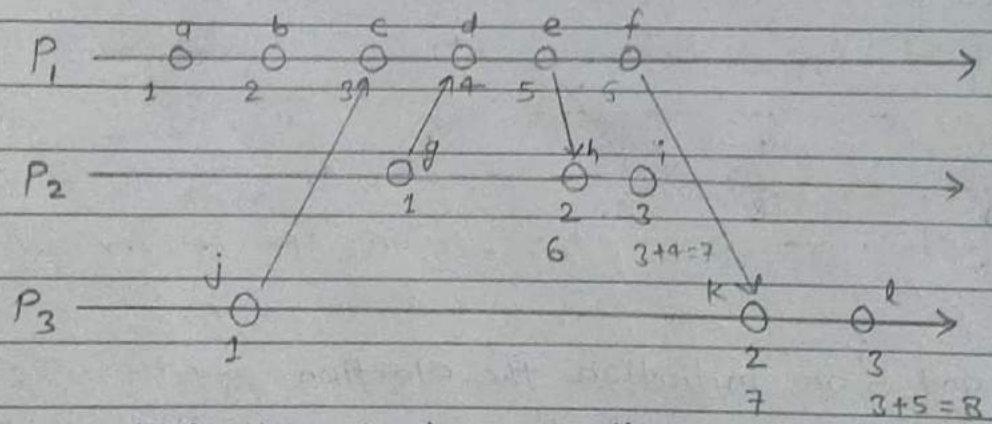
- Assign logical time stamps to events based on ordering among them
- if event a happened before event b, then it can be represented as

$$a \rightarrow b \quad OR \quad a \overset{h}{<} b$$

- if $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$
- if $a \overset{h}{<} b$ then $T(a) < T(b)$ where T is time stamp
- but if $T(a) < T(b)$, can we guarantee $a \overset{h}{<} b$.

Lamport's Logical clock

- Let us assume the following example.



i) $(j,c) \rightarrow (1,3)$ → do nothing
ii) $(g,d) \rightarrow (1,4)$ → do nothing
} good ordering

iii) $(e,h) \rightarrow (5,2)$
iv) $(f,k) \rightarrow (6,2)$
} bad ordering

- Each message carries a time stamp of the sender's clock
- When a message arrives
  1. If receives clock < message_time_stamp
     set system clock to message_time_stamp + 1
  2. else do nothing

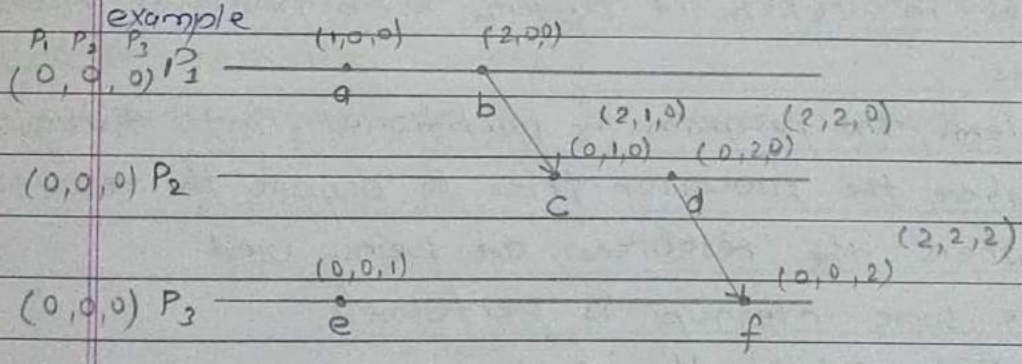
Date
Page

Vector Clock

Rules

1. Vector initialization to 0 at each process
   $V_i[j] = 0$, for $i, j = 1, 2, \ldots N$
2. Process increments its elements of the local vector before time stamping event.
   $V_i[i] = V_i[i] + 1$
3. Message is sent from process P_i with V_i attached to it.
4. When P_i receives message, compare vector elements and set local vector to higher of two values.
   $V_j[i] = \max(V_i[i], V_j[i])$

example



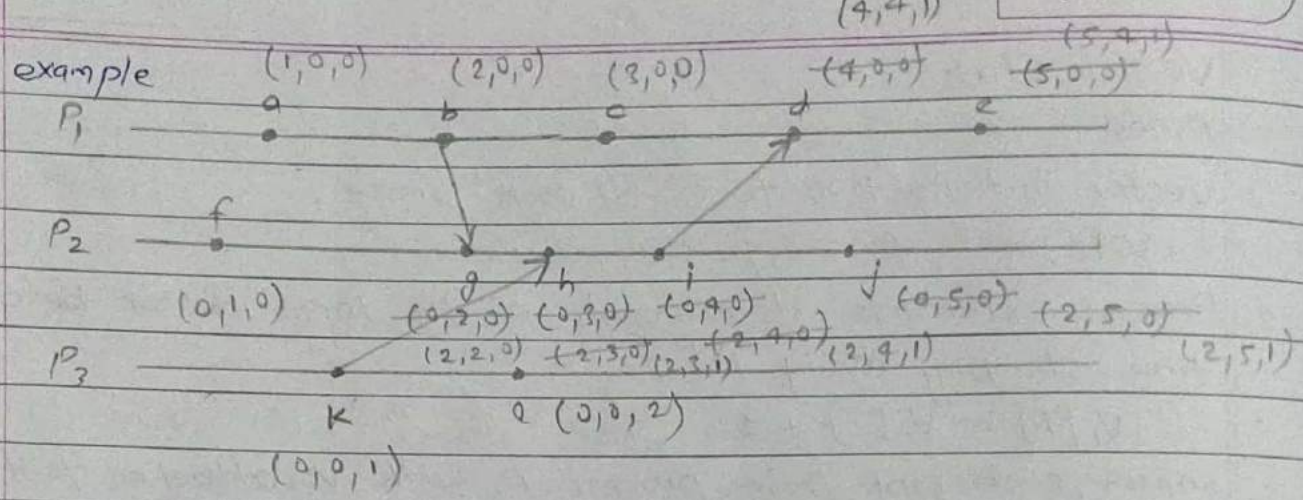
Concurrent events

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

Example

Events	Timestamp
a	(1, 0, 0)
b	(2, 0, 0)
c	(2, 1, 0)
d	(2, 2, 0)
e	(0, 0, 1)
f	(2, 2, 2)

- Which are concurrent events.
  (a, e), (b, e)



Resource Allocation

- problem arises in distributed systems when processor needs to share resources
- In such problem, the program is partitioned into 4 regions.
  1. Trying: where the processor tries to acquire the resources
  2. Critical: where the resources are being used.
  3. Exit: where some cleanup is performed.
  4. Remainder: the rest of the code.

Tolerating processor failure in synchronous system

- In real systems, the various components may not operate correctly all the time.
- It is assumed that communication links are reliable but processors are not reliable.
- Two types of failure
  1. Crash Failure
  2. Byzantine Failure

Date |
Page |

Crash Failure (Technical problem)

- A processor crashes in round r, it is correctly operates in round $1, 2, \ldots, r-1$, and does not take a step in round $r+1$.

Byzantine Failure (logical problem)

- Faulty processor (traitor) is called byzantine.
- It can send different message to different processors, that is not supposed to send by it.

Assignment 2:

- Tolerating processor failure in Asynchronous system
- sparse network covers
- Wait free implementation of shared object. - eg busy wait

Date
Page

Unit 2: Models

Unit 2.1. PRAM Models

PRAM Models

- Parallel Random Access Machine models
- Parallel can be defined as a technique for increasing the computation speed by allocating multiple processors to execute sub tasks simultaneously.

e.g.
Finding the smallest value in the set of N values.

Algorithm

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

Do in Parallel
- P_i derives the pair (i_1, i_2) of indices that correspond to different i
- P_i reads values $L(i_1)$ & $L(i_2)$
- If $L(i_1) \geq L(i_2)$ then P_i sends negative outcome to P_{i_1} else to P_{i_2}
- only one processor, P_j, $1 \leq j \leq n$, did not receive negative outcome
- P_j reads the value $L(j)$ and write it onto output cell

1 2 3 4 → index

eg. $L = \{2, 6, 4, 8\}$, $n = 4$

$P = \frac{n(n-1)}{2} = \frac{4 \times 3}{3} = 6$

Date
Page

→ P_1 → (1,2) P_2(-ve) (i_1, i_2)

→ P_2 → (1,3) P_3(-ve) ↓ P_j

→ P_3 → (2,3) P_2(-ve) $P_j = \frac{(i_2 - 1)(i_2 - 2)}{2} + i_1$

→ P_4 → (1,4) P_4(-ve)

→ P_5 → (2,4) P_4(-ve)

→ P_6 → (3,4) P_4(-ve)

$P_j = P_1 = 2$

Basic PRAM algorithms

1. Computing prefix sums
2. List Ranking
3. Parallel sorting algorithm

Computing Prefix sums

- given $A = \{a_0, a_1, a_2, \ldots, a_{n-1}\}$
- We have to compute

$\{a_0, (a_0 + a_1), (a_0 + a_1 + a_2), \ldots, (a_0 + a_1 + \ldots + a_{n-1})\}$

eg.

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

- the output is

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

Date
Page

Algorithm

- 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
      1. obtains $a[i - 2^j]$ from P_{i-2^j} through shared memory
      2. $a[i] = a[i - 2^j] + a[i]$

Eg. A $\{5, 3, -6, 2, 7, 10, -2, 8\}$

Here,

$n = 8$, $\log 8 = 3$ $j = 0, 1, 2$

For $j = 0$

$i = 2^j + 1$ to n $= 2$ to 8

$i - 2^j = 2 - i = 1$

$P_1 \rightarrow a[1] = 5$

$P_2 \rightarrow a[2] = a[1] + a[2] = 5 + 3 = 8$

$P_3 \rightarrow a[3] = a[2] + a[3] = 3 + (-6) = -3$

$P_4 \rightarrow a[4] = a[3] + a[4] = (-6) + 2 = -4$

$P_5 \rightarrow a[5] = a[4] + a[5] = 2 + 7 = 9$

$P_6 \rightarrow a[6] = a[5] + a[6] = 7 + 10 = 17$

$P_7 \rightarrow a[7] = a[6] + a[7] = 10 + (-2) = 8$

$P_8 \rightarrow a[8] = a[7] + a[8] = (-2) + 8 = 6$

{5, 8, -3, 4, 9, 17, 8, 6}

For j = 1

$i = 2^j + 1$ to n

= 3 to 8

$i - 2^j = 3 - 2^1 = 1$

~~$P_1 \rightarrow a[1]$~~

~~$P_2 =$~~

$P_3 \rightarrow a[3] = a[1] + a[3] = 5 + (-3) = 2$

$P_4 \rightarrow a[4] = a[2] + a[4] = 8 + (-4) = 4$

$P_5 \rightarrow a[5] = a[3] + a[5] = (-3) + 9 = 6$

$P_6 \rightarrow a[6] = a[4] + a[6] = (-4) + 17 = 13$

$P_7 \rightarrow a[7] = a[5] + a[7] = 9 + 8 = 17$

$P_8 \rightarrow a[8] = a[6] + a[8] = 17 + 6 = 23$

{5, 8, 2, 4, 6, 13, 17, 23}

For j = 2

$i = 2^j + 1$ to n

= 5 to 8

$i - 2^j = 5 - 2^2 = 1$

$P_5 \rightarrow a[5] = a[1] + a[5] = 5 + 6 = 11$

$P_6 \rightarrow a[6] = a[2] + a[6] = 8 + 13 = 21$

$P_7 \rightarrow a[7] = a[3] + a[7] = 2 + 17 = 19$

$P_8 \rightarrow a[8] = a[4] + a[8] = 4 + 23 = 27$

{5, 8, 2, 4, 11, 21, 19, 27} output

Date
Page

List Ranking

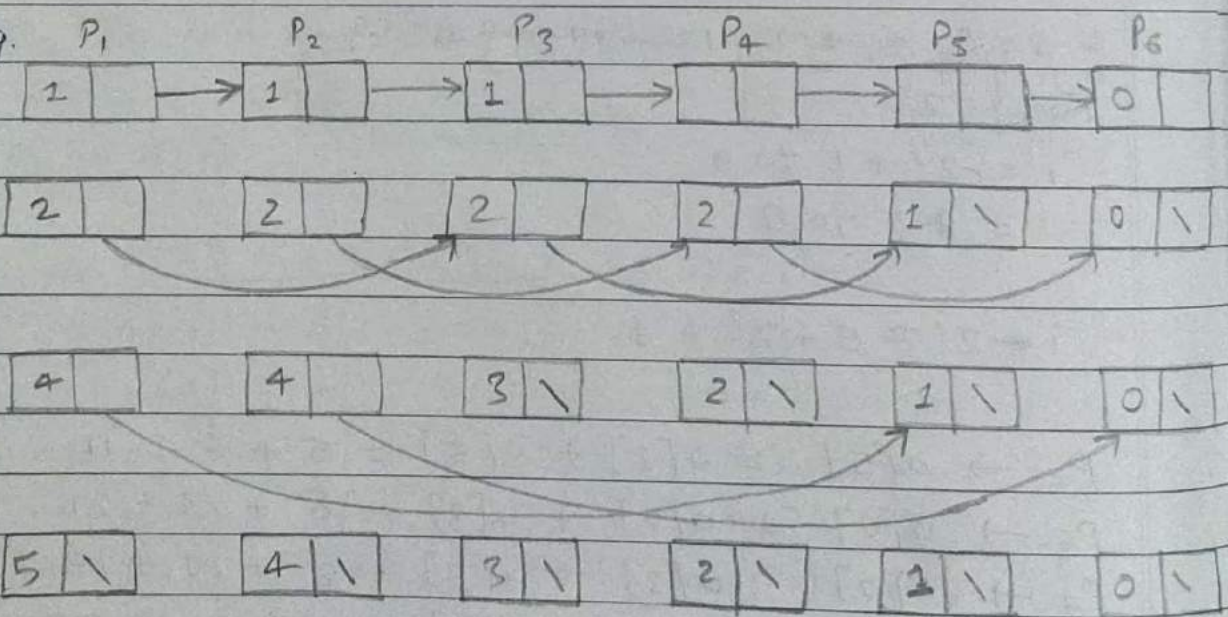
- The problem is that, given a singly linked list L, with N objects, for each node compute the distance to the end of the list.
- if d denotes the distance

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

Parallel Algorithm

- assign one processor for each node
- for each node i, do in parallel
  1. $i.d = i.d + i.next.d$
  2. $i.next = i.next.next$

Eg.



Date
Page

Parallel Sorting Algorithm

- bitonic sort
- consists of two sequences, one increasing and one decreasing

5, 6, 7, 8, 4, 3, 2, 1

- sequence of elements $< a_0, a_1, \ldots, a_{n-1} >$ there exists $< a_0, a_1, \ldots, a_i >$ is increasing and $< a_{i+1}, a_{i+2}, \ldots, a_{n-1} >$ is decreasing.
- if $a_{n/2}$ is the beginning of decreasing sequence s, then

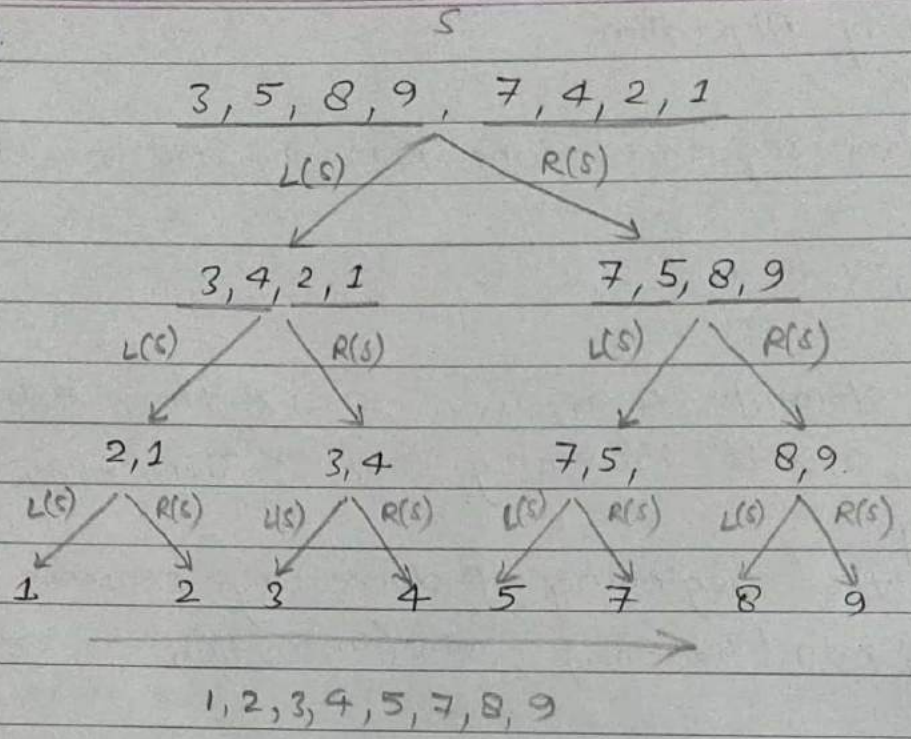
$$L(s) = \{ \min(a_0, a_{n/2}), \min(a_1, a_{n/2+1}), \ldots, \min(a_{n/2-1}, a_{n-1}) \}$$

$$R(s) = \{ \max(a_0, a_{n/2}), \max(a_1, a_{n/2+1}), \ldots, \max(a_{n/2-1}, a_{n-1}) \}$$

Algorithm

- Input → a bitonic sequences
- If s is the length 1 then STOP
- else
  - → Form L(s) and R(s)
  - → Do in parallel
    - L(s) ← recursive bitonic merge L(s)
    - R(s) ← recursive bitonic merge R(s)
    - concatenate L(s) and R(s)

Eg.



Randomized Algorithm

- is the algorithm where execution is controlled at one or more points by randomly made choices.

eg. Fermat's little theorem

- primality testing

$b^p \equiv b \pmod p$ → $2^7 = 2 \pmod 7$

where $b \in Zp$

p = prime

Z_n = set of residue

Deficiencies of PRAM algorithm

- no mechanisms for representing communication between the processors.
- storage management and communication issues are hidden from the algorithm designer.

Date
Page

Unit : 2.2

Broadcasting with Selective Reduction (BSR)

- Broadcast instruction consists of three phases
  1. Broadcast phase
  2. selection phase
  3. Reduction phase

Broadcast Phase

- allows all of the N processors to write concurrently to all of the M memory locations.
- each processor P_i produced a record containing two fields, a tag t_i and datum d_i.

selection Phase

- after the data are received and each memory locations U_j, $1 \leq j \leq m$, a switch S_j will select the receiving data d_i by comparing tag value t_i by using a selection rule.

$$\sigma \{ <, \leq, =, >, \geq, \neq \}$$

Reduction Phase

- selected data are reduces to a single value using reduction rule, $R \{ \Sigma, \Pi, \wedge, \vee, \oplus, \cap, \cup \}$

Date
Page

Formal Definition of BSR.

- Mathematically it can be expressed as

$$U_j \leftarrow R(d_i) \mid \wedge t(i,h)\ \sigma_h\ L(j,h)$$
$$1 \le i \le N \qquad 1 \le h \le k \qquad 1 \le j \le M$$

Where,

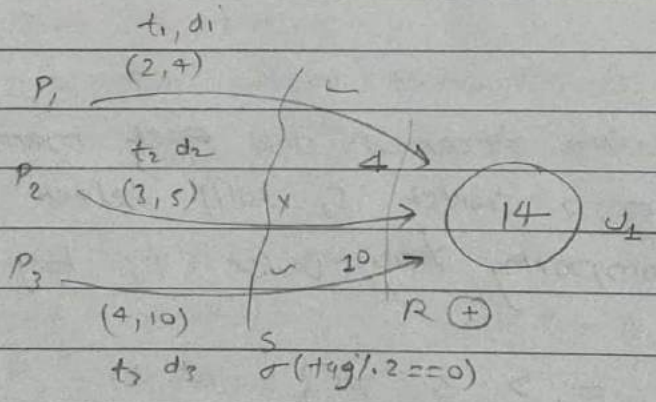
$N \Rightarrow$ no. of processor

$d_i \rightarrow$ datum broadcast by P

$\sigma_h \rightarrow$ selection operation

$t(i,h) \rightarrow$ tag broadcast by P_i for criteria σ_h

$L(j,h) \rightarrow$ limit value j for criteria σ_h.



Date
Page

Parenthesis Matching

- One criterion 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

(	)	(	(	(	)	)	)	(	(	)	)
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 j do in parallel
- If I_j == "(" then $b_j = 1$ else $b_j = -1$
- $P_j = \Sigma b_j \mid i \leq j$
- If $b_j == -1$ then $P_j = P_j + 1$
- $P_j' = P_j - \frac{1}{j}$
- $q_j = -1,\ t_j = 0,\ r_j = 0$
- $q_j = \cap P_i' \mid P_i' < P_j'$
- $t_j = \cap i \mid P_i' = q_j$
- $r_j = \cup i \mid t_i = j$
- If I_j = ')' then $m_j = t_j$ else $m_j = r_j$

	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	(	)	(	(	(	)	)	)	(	(	)	)
Index(j)	1	2	3	4	5	6	7	8	9	10	11	12
b_j	1	-1	1	1	1	-1	-1	-1	1	1	-1	-1
$\Sigma b_i \mid i \leq j$	1	0	1	2	3	2	1	0	1	2	1	0
updated P_j	1	1	1	2	3	3	2	1	1	2	2	1
$1-\frac{1}{1}, 1-\frac{1}{2}$ P_j'	0	0.5	0.66	1.75	2.8	2.83	1.86	0.87	0.89	1.9	1.91	0.92
default value q_j	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
q_j	-1	0	0.5	0.92	1.91	2.8	1.75	0.66	0.87	1.86	1.9	0.89
t_j	0	1	2	12	11	5	4	3	8	7	10	9
r_j	2	3	8	7	6	0	10	9	12	11	5	4
m_j	2	1	8	7	6	5	4	3	12	11	10	9

~~Maximal Sub Sum Set~~

Maximal 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 maximal sum.

Example:

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

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

Algorithm

$S_j = \Sigma d_i \mid i \leq j$

$m_j = \min S_i \mid i \geq j$

$e_j = \min i \mid S_i = m_j$

$m_j = m_j - s_j + d_j$

$t = \cap m_i$

$x = \cap i \mid m_i = t \rightarrow$ starting point

$y = e_x \rightarrow$ ending point

$A = \{31, -41, 59, 26, -53, 58, 97, -93, -23, 84\}$

s_j	31	-10	49	75	22	80	177	84	61	145
m_j	177	177	177	177	177	177	177	145	145	145
e_j	7	7	7	7	7	7	7	10	10	10
m_j	177	146	187	128	102	155	97	-32	61	84

$t = \cap m_i = 187$

$x = 3 \rightarrow$ starting point

$y = 7 \rightarrow$ ending point

$\therefore 59 + 26 - 53 + 58 + 97 = 187$

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.

Example:

- Counting inversions in a permutation
- Given a set S, a permutation π of S is a set S', containing all elements of S, but in different order.
- Inversions in permutations are those numbers of pairs, which are in disorder.

Eg.

1 2 3 4 5 - i

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

They are {6,2}, {6,5}, {9,5}

Rules

- $y_i = \Sigma 1 \mid \{i < j\} \wedge \{\pi(i) > \pi(j)\}$

2<3 ∧ 6>2 true 4<5 ∧ 9>5 true

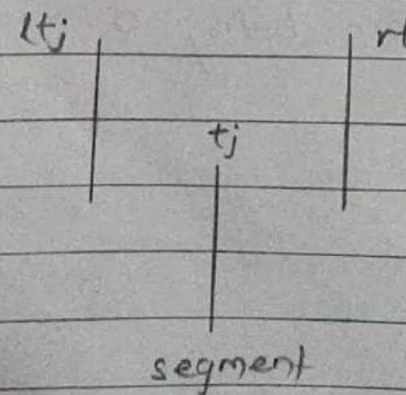
(1,6)	(6,2) ✓	(2,9)	(9,5) ✓
(1,2)	(6,9)	(2,5)	
(1,9)	(6,5) ✓		
(1,5)	2<5 ∧ 6>5 true		

Three criterion BSR algorithm

- Have to pass three criterion to go to reduction process.

Eg:

vertical segment visibility



Date
Page

$lt_j = \cap i \mid x_i < x_j \wedge t_i \geq t_j \wedge b_i \leq t_j$ (left top)

$rt_j = \cap i \mid x_i > x_j \wedge t_i \geq t_j \wedge b_i \leq t_j$ (right top)

Unit : 2.3

Data Flow Models

- Flow of information between data processing entities
- control flow and data flow
- Control flow assumes that a program in a series of instructions, each of which specifies
  - Either an operation with memory location
  - OR specifies transfer of control to another instruction

Example:

C = if (n==0) then a+b else a-b

- But in data flow computing model, it executes any operation as soon as necessary operations are available

Basic primitives of data flow

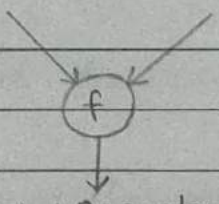


Fig: Operator

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

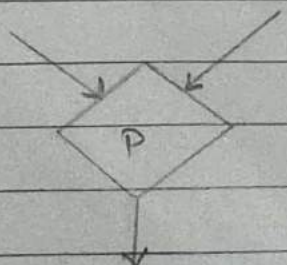


Fig: Predicate

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

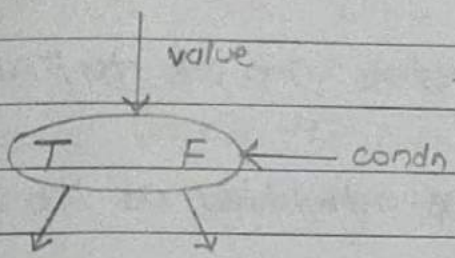


Fig: switch

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

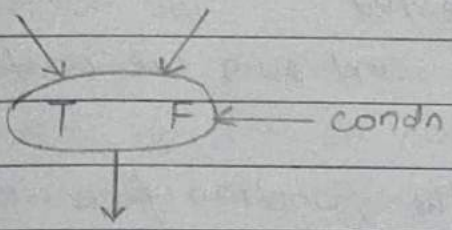


Fig: Merge

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

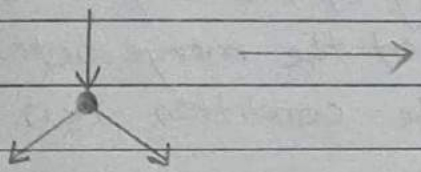
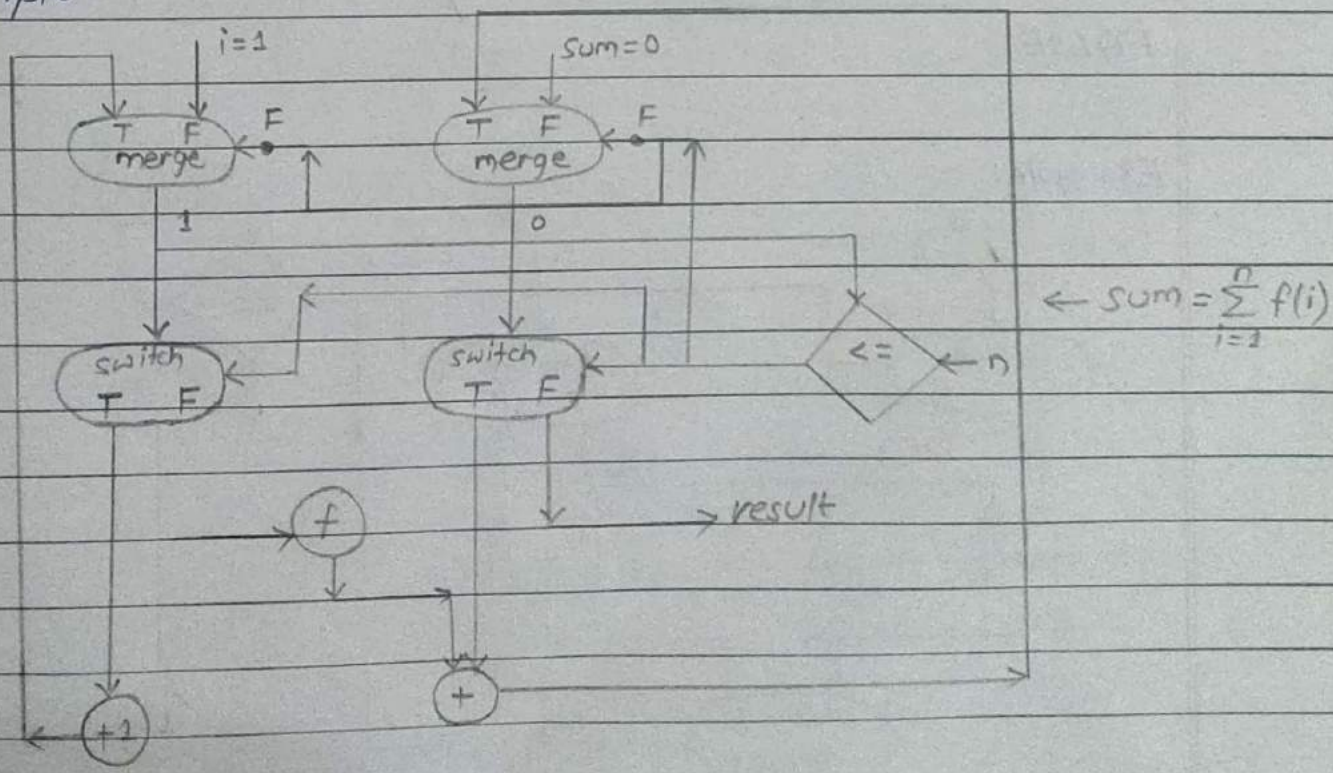


Fig: Copy

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

Example:



Description:

- Control units to the merge actors are set to FALSE for initialization.
- The input values i and sum are admitted as the initial values of the iteration.
- The predicate $i \leq n$ is then tested
- If it is true, the values of i and sum are routed to the TRUE sides of switch.
- This initiates the firing of the 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 $i \leq n$ is no longer true.
- The final sum is then routed out of the loop and the initial Boolean values at merge actors are restored to FALSE.

Example:

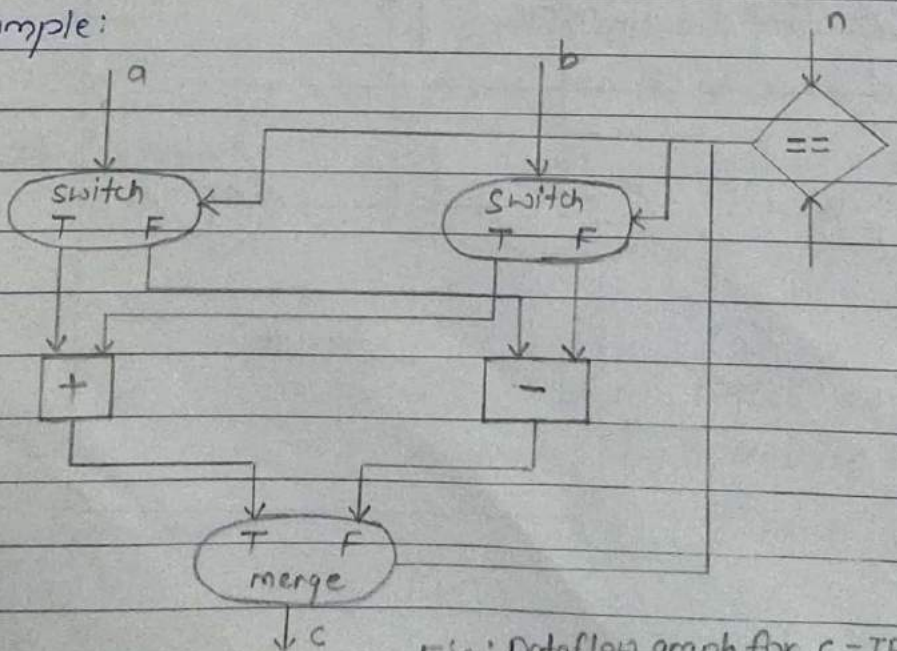


Fig: Dataflow graph for C=IF n==0 THEN a+b ELSE

Date
Page

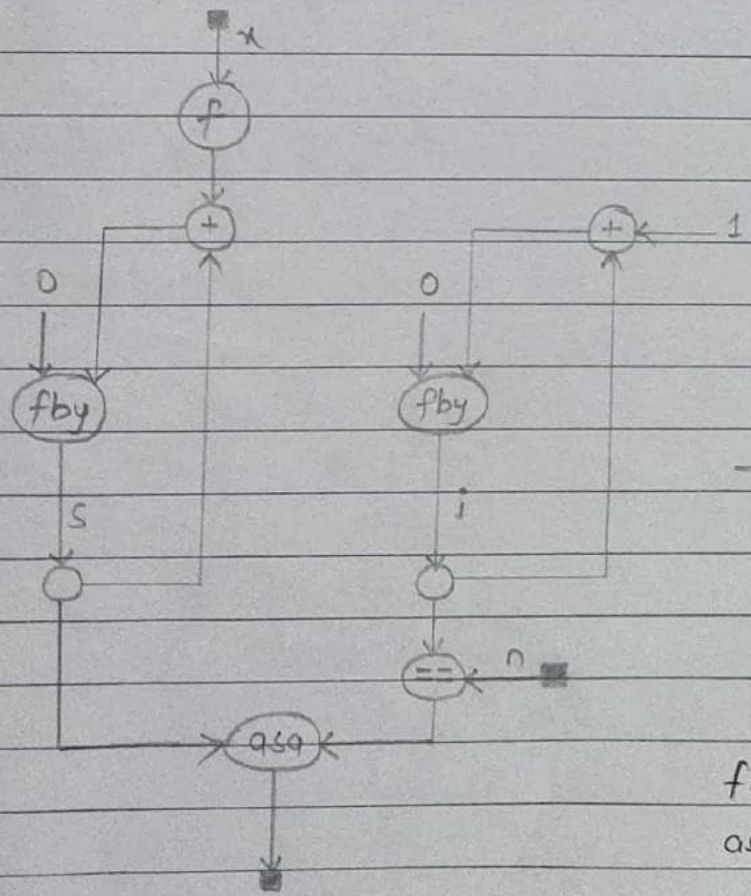
Demand Driven Data Flow Computing Model

- 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 operation

Operator net

- Programming model 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:



$\rightarrow S = \sum_{i=1}^{n} f(x)$

fby → followed by

asa → as soon as

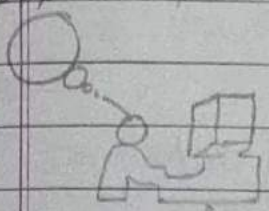
Date |
Page |

Unit 2.4: Partitioning and Scheduling

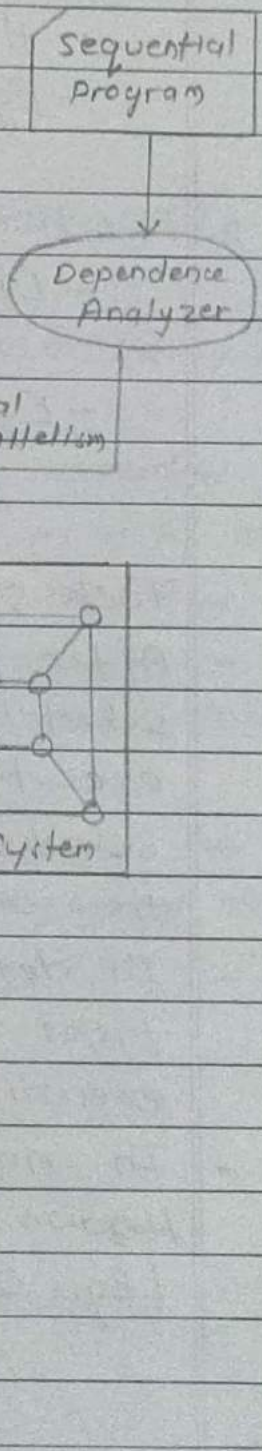
Partitioning and Scheduling

- Partitioning deals with parallelism

Explicit Approach



Implicit Approach



Date
Page

Program Partitioning

1. Data Partitioning
   - same computation on different set of data concurrently
   - Eg. matrix multiplication

2. Function partitioning
   - Different computation on same set of data concurrently
   - Eg. flight simulation

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 as deterministic and non-deterministic
- In deterministic scheduling, all the information about tasks to be scheduled is entirely known prior to execution time.
- In non-deterministic, some information may not be known before program execution (Eg. conditional branches)

Date
Page

Scheduling System Models

- consists of
  1. Parallel program tasks
  2. Target machine
  3. Schedule
  4. Performance criteria

1. Parallel program tasks

Parallel program tasks can be defined as the system $(T, <, D_{ij}, A_i)$, where

1. $T = \{t_1, t_2, \ldots, t_n\}$ is a set of tasks to be executed
2. $<$ is a partial order defined on T, i.e $t_i < t_j$ means t_i must be completed before t_j can start execution
3. D_{ij} is an $n \times n$ matrix of communication data, where $D_{ij} \geq 0$ is the amount of data required to be transmitted from task t_i to t_j.
4. A_i is a vector of the amount of computations i.e $A_i > 0$, is the number of instructions required to execute t_i.

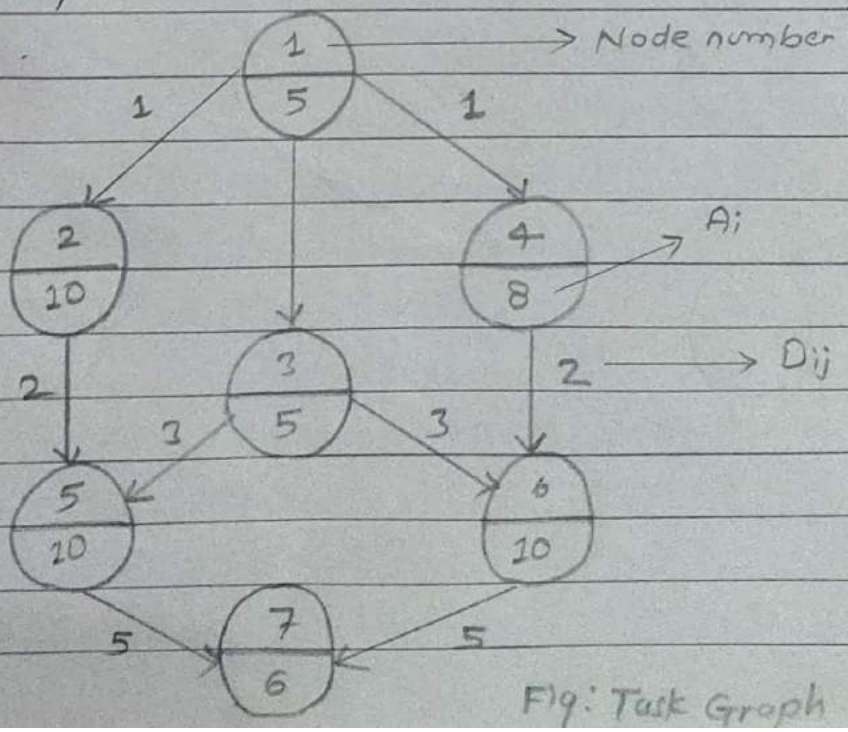


Fig: Task Graph (n=7)

Date
Page

1. $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$

2. $< = \{ t_1 < t_2,\ t_1 < t_3,\ t_1 < t_4,$
$t_2 < t_5,\ t_3 < t_5,\ t_6 < t_3,$
$t_6 < t_4,\ t_5 < t_7,\ t_6 < t_7 \}$

3. D_{ij}

	1	2	3	4	5	6	7
1	0	1	2	1	0	0	0
2	0	0	0	0	2	0	0
3	0	0	0	0	3	3	0
4	0	0	0	0	0	2	0
5	0	0	0	0	0	0	5
6	0	0	0	0	0	0	5
7	0	0	0	0	0	0	0

4. $A_1 = 5$
$A_2 = 10$
$A_3 = 5$
$A_4 = 8$
$A_5 = 10$
$A_6 = 10$
$A_7 = 6$

Date
Page

2. Target Machine

- Assumed to be made up of "m" heterogeneous processing elements connected using an arbitrary interconnection network
- Formally target machine can be defined as $(P, P_{ij}, S_i, T_i, B_i, R_{ij})$, where
  1. $P = \{P_1, P_2, \ldots, P_m\}$ is a set of m processors forming the parallel architecture
  2. P_{ij} is an mxm interconnection topology matrix
  3. S_i specifies the speed of processor P_i.
  4. I_i specifies the startup cost of initiating message m on processor P_i
  5. B_i specifies the startup cost of initiating a process on processor P_i.
  6. R_{ij} is the transmission rate over link connecting P_i and P_j

- Now execution time of task i on processor j (T_{ij}) can be computed as,

$$T_{ij} = \frac{A_i}{S_j} + B_j$$

A_i: no. of instruction; B_j: start (process j); T_{ij}: task, processor; S_j: speed

Schedule

Formally defined as

$$f : v \rightarrow \{1, 2, \ldots, m\} \times \{0, 1, \ldots, \infty\}$$

- i.e $f(v) = (i, t)$ for some v means task v is scheduled to be processed by processor P_i starting at time t
- Task 1 starts on processor P_2 at time 0 and finishes at time 10
- Task 5 starts on processor P_3 at time 0 and finishes at time 30
- Shaded area is communication delay.

Date |
Page |

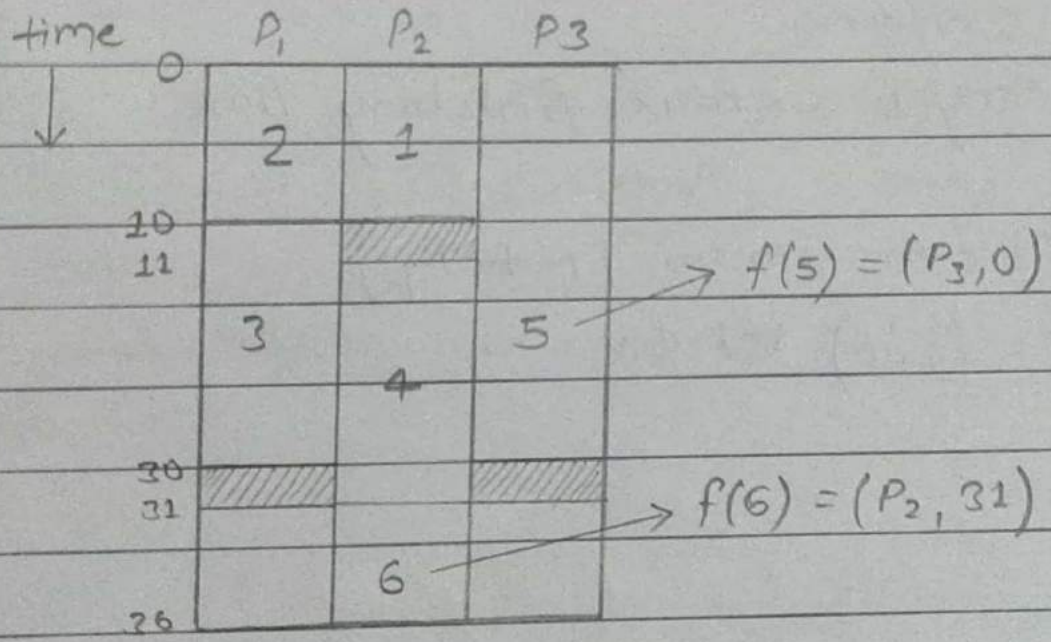


Fig: Gantt chart

Date
Page

Performance Criteria

- Scheduling length or max finishing time
- Formally

  length(f) = maximum $\{t + T_{ij}\}$
- Where $f(i) = (j, t)$ $\forall i \in V$

Communication Model

- Two key component that contribu to the total completion cost is
  1. Execution Time
  2. Communication Delay

Model A

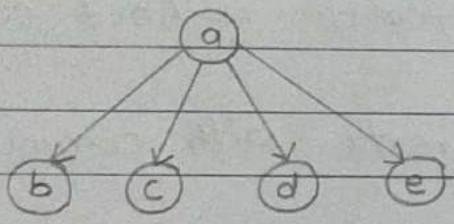
- Total cost = communication cost + execution cost
- where
  - Execution cost = schedule length
  - Communication cost = no. of message x cost per message
  - No. of message = the no. of node pairs (u, v) such that $(u, v) \in A$ and $proc(u) \neq proc(v)$

Date |
Page |

Model B

- If tasks u, v and $w \in V$ such that $(u,v), (u,w) \in A$, then the result of computing u must be communicated to the processor v and w
- This communication is counted twice in model A but one in model B

Example:



- Suppose that each of given task takes one unit of execution time in either P_1 or P_2
- Task a sends message to b, c, d, e
- Suppose a, b, d → allocated to P_1
  c, e → allocated to P_2

Model A

No. of message = $|(a,c), (a,e)| = 2$

Model B

No. of message = $|(a, P_2)| = 1$

Model C :

- This model assumes that the existence of an I/O processor that is associated with every processor in the system.

communication signal execution pair hunxa.

Date |
Page |

Optimal Scheduling Algorithm

- Deal in 3 cases
  1. When the task graph is a tree
  2. When the task graph is an interval order
  3. When there are only 2 processors available

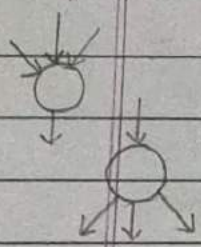
Scheduling tree structure task graphs

A. Scheduling in-forests / Out-forests without communication

B. Scheduling in-forests / out-forests with communication

Scheduling in-forests / out-forests without communication

- Task graph is either
  - In-forest → each node has at most one immediate successors OR
  - Out-forest → each node has at most one immediate predecessors



- General strategy → used the highest level first

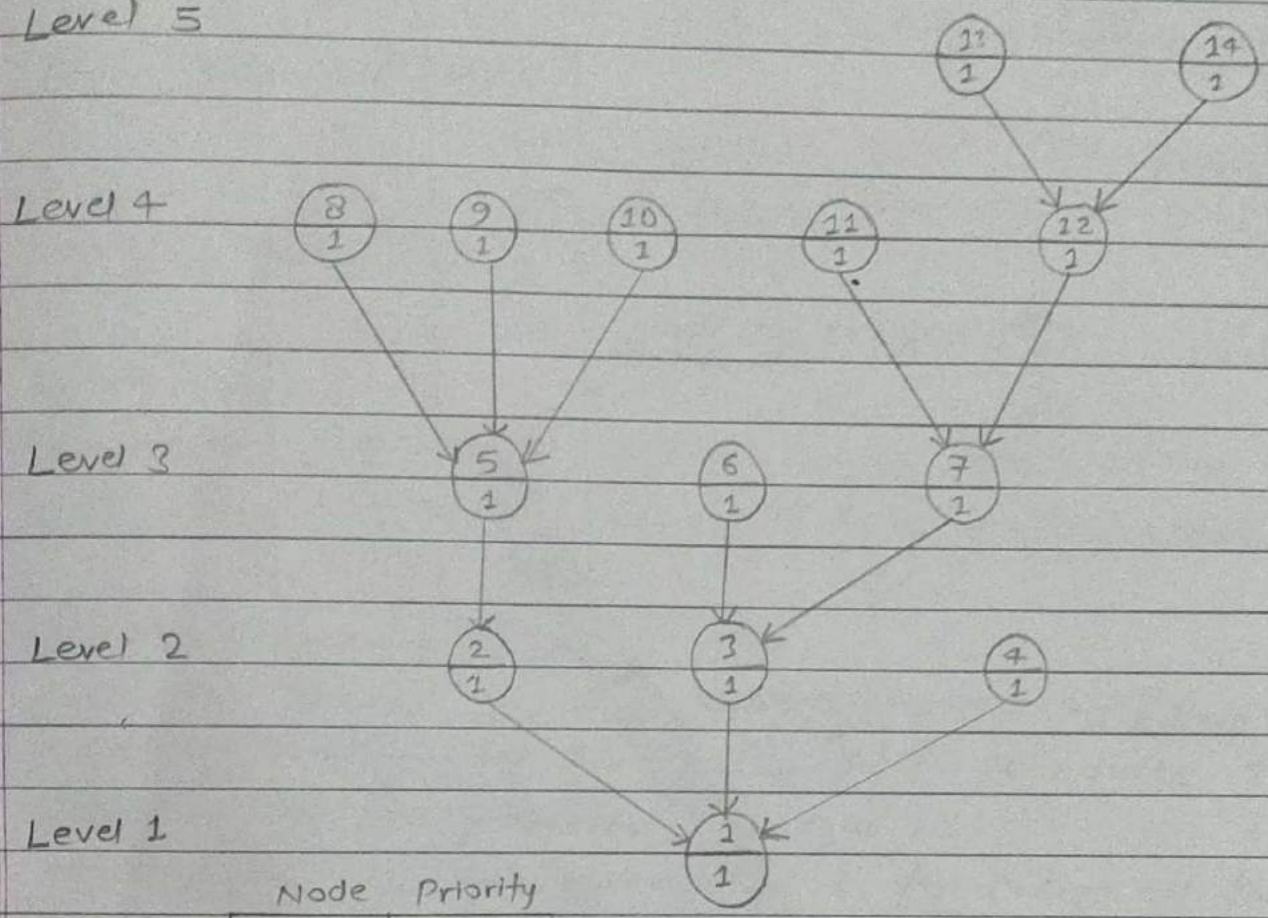
Algorithm

1. The level of each node in the task graph is calculated and used as each node priority

2. Whenever a processor becomes available, assign the unexecuted ready task with the highest priority

Date
Page

Example:

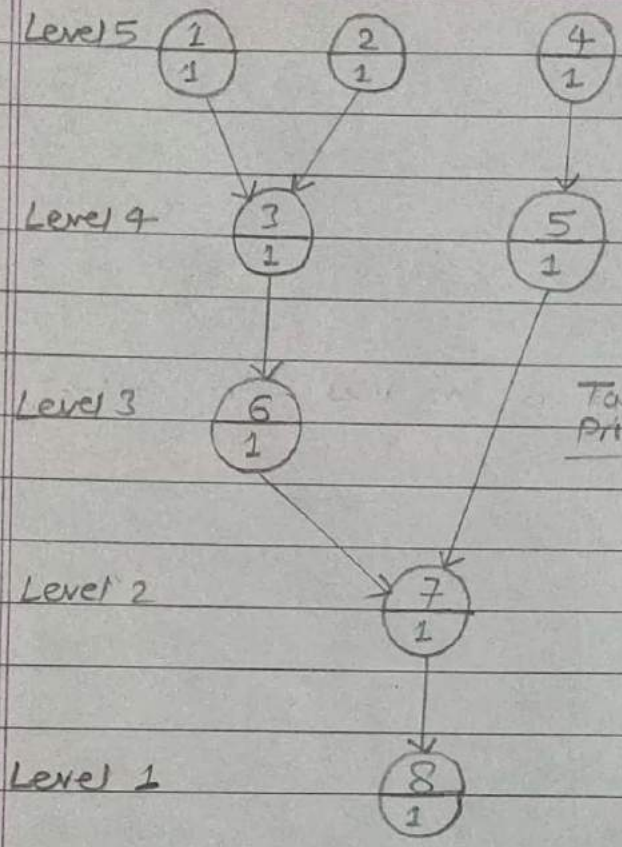


Node	Priority
14	5
13	5
12	4
11	4
10	4
9	4
8	4
7	3
6	3
5	3
4	2
3	2
2	2
1	1

Task Priority →

Task scheduling →

	P_1	P_2	P_3
0	14	13	11
1	12	10	9
2	8	6	7
3	5	3	4
4	2		
5	1		



Task Priority →

Node	Priority
1	5
2	5
3	4
4	5
5	4
6	3
7	2
8	1

Task scheduling →

	P_1	P_2	P_3
0	1	2	4
1	3		5
2	6		
3			7
4			8
5			

Date
Page

B. Scheduling in-forests / out-forests with communication

- Based on the idea of adding new procedence relations to the task graph
- scheduling the augmented task graph without communication is equivalent to scheduling the original task graph with communication.

Task graph (T) → Aug. task graph (T') → without C
with C

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.

operation swap-all

- given a schedule f, the operation swap-all (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 is the scheduled at time t_1, $\forall t_2$, $t_1 \geq t$

Algorithm

1. Given an in-forest $G = (V, E)$, identify the set of siblings $S_1, S_2, \ldots, S_k$ where S_i, is the set of all nodes in V with a common child (S_i)
2. $A_1 \leftarrow A$
3. For every set S_i
   3.1 Pick node $v \in S_i$ with the maximum depth.

3.2 $A_1 \leftarrow A_1 - (v, child(S_i))$, $\forall v \in S_i$ and $v \neq u$

3.3 $A_1 \leftarrow A_1 \cup (v, u)$ $\forall v \in S_i$ and $v \neq u$.

4. Obtain the schedule f on A_1 without communication.

5. For every set S_i in the original task graph G, if node v is scheduled in the time slot immediately before child (S_i) but in a different processor, apply swap-all.

Example:

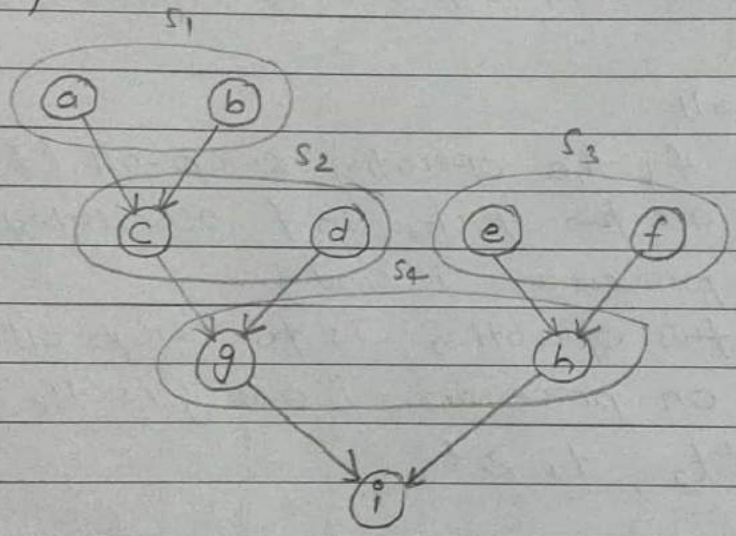


Fig1: Original task graph

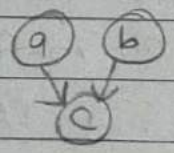
max depth (u)

$S_1 = \{a, b\} \Rightarrow b$ $v = a$

$S_2 = \{c, d\} \Rightarrow c$ $v = d$

$S_3 = \{e, f\} \Rightarrow f$ $v = e$

$S_4 = \{g, h\} \Rightarrow g$ $v = h$



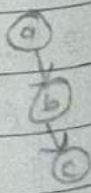
$u = b$

$v = a$

$(v, child(S_i))$

$= (a, c)$

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



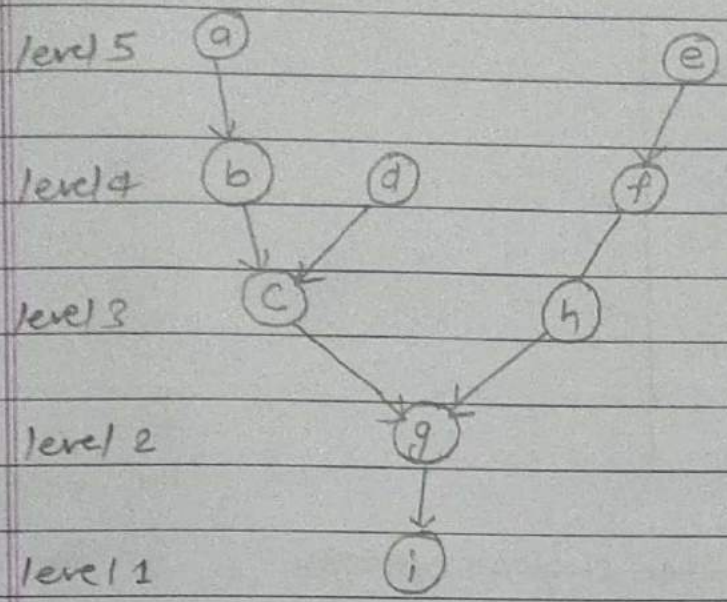


Fig2: Augmented task graph

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

Task priority →

Fig3 Task priority

Task scheduling →

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

Fig 4: Task scheduling

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

Fig5: Task schedule after swap all operation