HPC

physical problem
↓
mathematic mode
↓
numerical solv
↓
Stimulate

HPC.

performance metrices → power consumption, FLOPS

① Theoretical peak performance (R_ theory)
② Real performance (R_ real)
↳ flops for specific operations.

sustained performance : for a particular period of time, $\frac{\text{total flops}}{t}$ (R_ sustained)

R_ sustained < 10% R theory

R sustained << R_ real << R_ theory

CPU not given data ⇒ CPU is idle
⇒ performance ↓

CPU → ALU
↳ CU

Dual core
→ 2 ALU
→ 1 CU

Quad core
→ 4 ALU
→ 1 CU

fast CPU requires fast memory

Computer time ⇒ CPU cycle
↳ min time to execute one instruction

⇓

Time to execute the programme

T ⇒ ~~time for 1 cycle * no of instructions~~

Computer time measured in CPU cycle

↓

time to execute given programme

↓

min time to execute 1 instruction.

$$T = n_c \times t_c \Rightarrow n_i \times \frac{n_c}{n_i} \times t_c \Rightarrow n_i \times CPI \times t_c$$

n_c → no of CPU cycles
t_c → time of the cycle

Date

n_i : no of instructions
CPI → no of clocks required for 1 instr n

Reduce
1) cycle time (↓) increasing frequency
2) Reduce n_i (# of instr^ns)

eg c = a+b → inst 1 → better compiler, interpreter, assembler
↓
load a
load b
Add a and b
store C

(3) Reduce CPI clock per instructions

parallelism & pipelining

Reduce CPI → interleaved execution, overlapping execution, overlapping, execution of # of instn.

parallelism → in single processor computer
① pipelining
② ↑ no of cores
③ functional units

→ overlap operation of diff units of computer.

→ Increase speed of ALU by exploiting data/temporal parallelism.

How to make computers fast.

Reduce CPI : key is ||ism.

multiple CPU :→ 2 CPU's → ||ism (parallelism) in true sense

Simultaneous execution → ||ism.
Interleaved execution → one task at a time but CPU is never idle.

FLYN classification of || ARCH's.

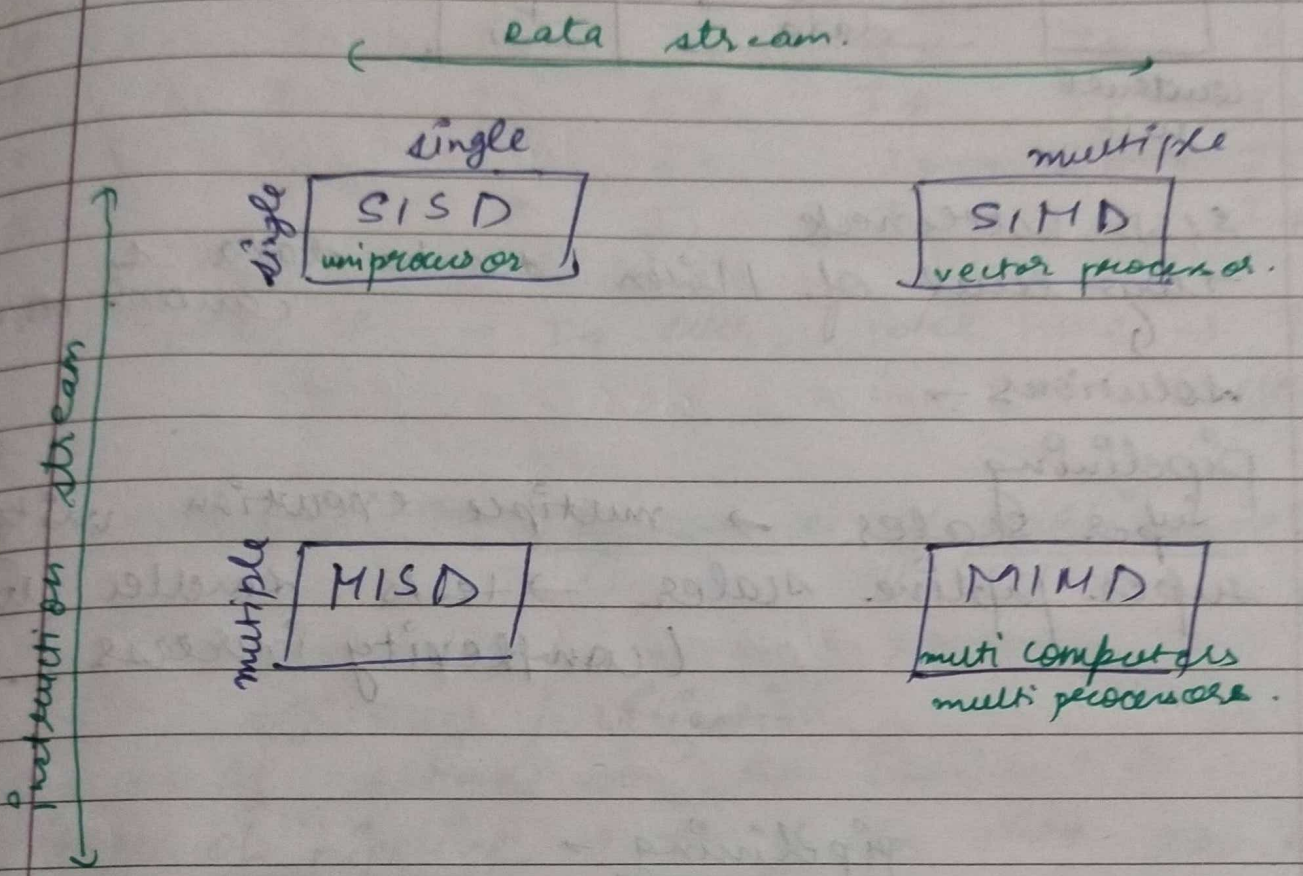
if you are able to read/write 2 data items @ time → called ||ism.

2 read / 2 execution - - : etc.

collection of instructions → Instruction stream.

data stream → { CPU → memory : write
memory → CPU : Read

parallelism can happen on both data & instruction stream.



SISD :→ Scaler processors → pipelining only solution to achieve ||sm here.

→ Deterministic execution → tells what instructions are being executed when

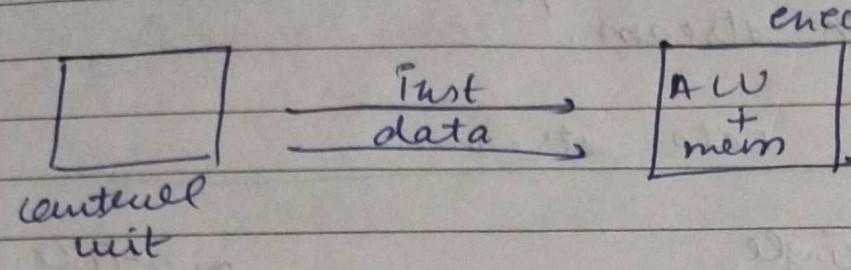
SIMD → single Instruction multiple data,

MISD → less significant.

MIMD → achieve ||sm in true sense.

Page No.
Date

SISD bottleneck



SISD bottleneck
→ low level of Ilism : due to data & control dependency
↳
solutions →

① pipelining
② Super scaler → multiple execution units
③ super pipline scaler → tasks smaller chunks (complexity increase).

pipelining →

1 instruction per cycle → goal

→ overlapped / interleaved execution → pipelining.

→ 1 instruction can be divided into 4 phases of stages

i) instruction fetch → memory to register
ii) Read/decode → memory → register.
iii) execution → ALU
iv) write back → reg → memory

Page No.
Date

for 7 instructions → 1×4 = 28 cycles.
→ without pipelining → 3×4 → 12 (3 instructions)

	1	2	3 time	4	5	6	7	8
IF	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
ID	X	I_1	I_2	I_3	I_4	I_5	I_6	I_7
EX	X	X	I_1	I_2	I_3	I_4	I_5	I_6
WB	X	X	X	I_1	I_2	I_3	I_4	I_5

CPI = 1

after 7 → I_4 out total time → 7.
↳ after pipelining.

measure the performance of pipelines →

1. speedup
2. efficiency / utilization

Depth of pipeline → no of stages in an instruction.
no of ops that can be performed in 1 cycle.

$$CPI = \frac{n + \text{depth} - 1}{n} \Rightarrow \frac{\text{total no of CPU cycles}}{\text{total no of instructions.}}$$

$$\text{efficiency} \rightarrow \frac{\text{no of boxes utilised}}{\text{Total no of pipelines.}}$$

$$\text{speed up} \rightarrow \frac{\text{Time to execute without pipelining}}{\text{Time with pipelining}}$$

⇒

Date

→ in general.

max efficiency → speed up.

no of stages / depth.

$0 < \text{efficiency} < 1$

$\text{speedup} < m.$

$\text{speedup} = \frac{D}{CPI} \rightarrow \text{depth.}$

T_{wop} → Time ⇒ $1 \times \text{depth}/m + (n-1) \times 1$

without pipeline.

$m \rightarrow$ depth. , $n \rightarrow$ no of instructio

T_{wp} Time with pipeline ⇒ $n \times m.$

now speedup → $\frac{T_{wop}}{T_{wp}} \Rightarrow \frac{n \times m}{m + n - 1}$

Date

Inhibitors of pipelining

1. controlled dependencies
2. data dependencies
3. structural dependencies

↳ due to branch statements. → branch penalties happen.

when next operation is jump statements. Then jump should run first we have to wait for the upper instructions not start next

↳ use NOP operation
↳ done by smart compilers.
↳ not hardware support

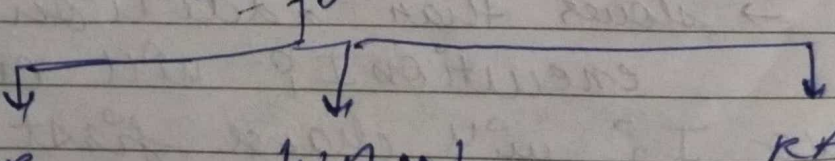
pipelining + stalls → Hardware support

↓
different from NOP.

→ Rearrange instructions } → software
→ rescheduling instructions
→ Branch prediction → hardware

→ Data dependency →

→ when an instruction depends on the data coming from other instructions.



RAW → read after write [flow/true data dependency]

WAR → write after read [anti-data dependency]

WAW → write after write [output data dependency]

examples

RAW: read after write

I_i: $R_1 \leftarrow R_1 + R_3$

I_j: $R_4 \leftarrow R_1 + R_5$.

WAR:

I_i: $R_1 \leftarrow R_2 + R_3$ — R_2 value is not read here.

I_j: load m, R_2 — its value is updated earlier

WAW: write after write [output data dependency]

I_i: $R_1 \leftarrow R_2 * R_3$

I_j: $R_1 \leftarrow R_5 + R_6$

* → slower than addition.

execution I_i will more than I_j so I_j will change first (execute first).

Date

sequence not followed.

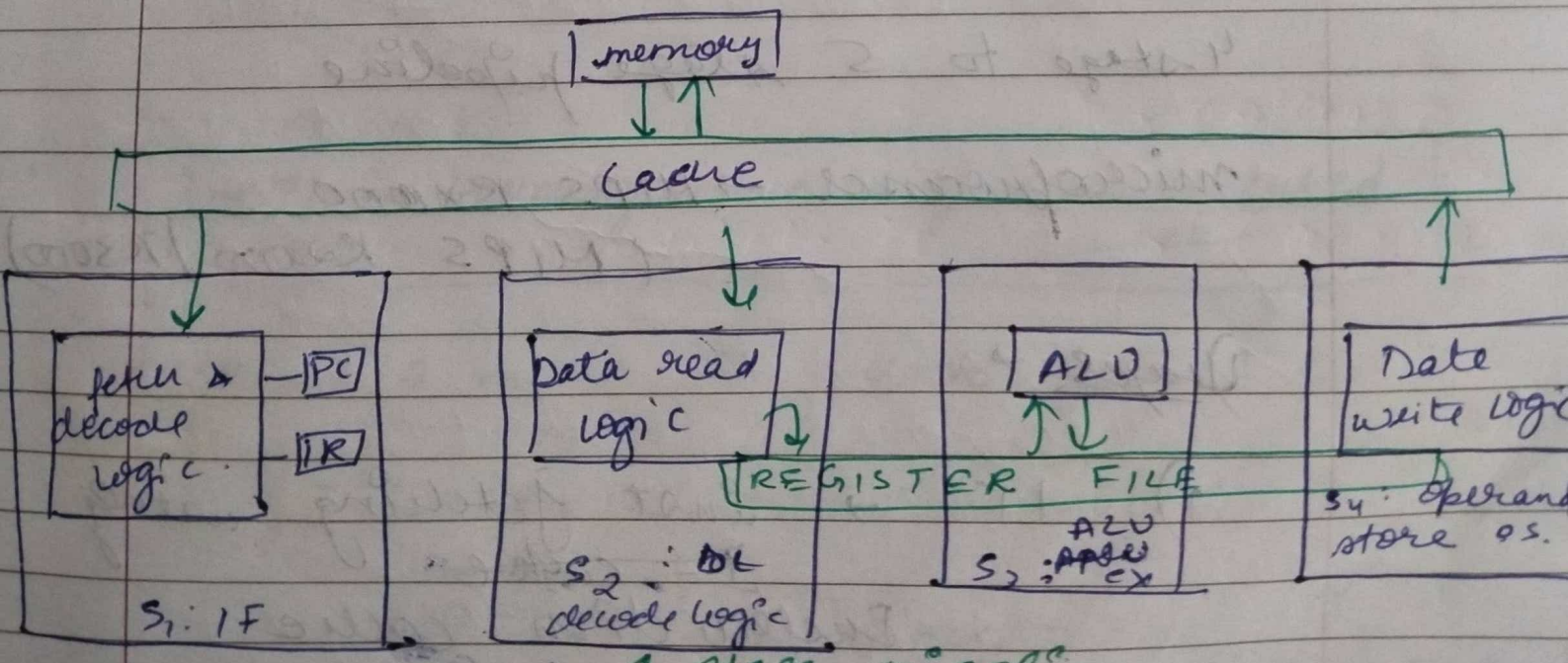
solutions →

lock → drawback → //ism
stall → no of.

Interface registers → hold intermediate outputs o/p.
↳ called Operand forwarding

structural dependence →

Resource conflict in the pipeline
eg → Bus
when more than one instruction tries to fetch the same resource in same cycle then : structural dependence.



4 stage pipeline

same time S1 & S2 can ask for cache
→ conflict

	T_1	T_2	T_3	T_4
I_1		2	3	4
I_2		1	2	3
I_3			1	2

Page No.
Date

Sol^n →

Rename

① Resource duplication (instead one have multiple copies of resource)

here divide cache into 2 parts

① instruction cache

② data cache

problem ?

instruction 3 at → S_2 } → both accessing
& 1 at S_4 } cache at time T_4.

Solution →

4 stage to 5 stage pipeline

microprocessor → MIPS RX000
(MIPS R2000 / R3000)

Stages :

1) IF → Inst fetching using ~~I-cache~~ Instruction cache

2) RD : Operand loading (reading) from register file 'RF' while decoding the fetched instruction.. whatever operands you require are loaded into register file.

3) execute → data processing using ALU & register file. as needed

4) memory access → operand accessing (load / store) using data - cache

5) write back → operand storing (write back in register file).

types of ins →
now,

CPI = 1

what is throughput → no of inst executed / 1 sec

CPI = 1 → means in 1 clock cycle, 1 inst is execute

if CC = 1ns → 1ns, 1 inst is execute

10^{-6} sec = 1 inst

1 sec → $\frac{1}{10^{-6}}$ instruction

Date

Types of pipeline

1. Uniform delay pipeline
2. non-uniform delay pipeline

→ every stage will take same time to complete an operation

normal no stalls: cycle time (T_p) = stage delay

↳ cycle time (T_p) → stage delay + buffer delay

after stall adding.

if we have → stages let's say

4sec 3sec 2sec.

all will be given 4 sec → now

↓

add stalls

non-uniform delay → diff stage

Cycle Time (T_p) = max(stage delay)

not for uniformity ← Buffer.

T_p = Max (stage + Buffer delay)

↓

not necessary that after buffer all are same.

Date

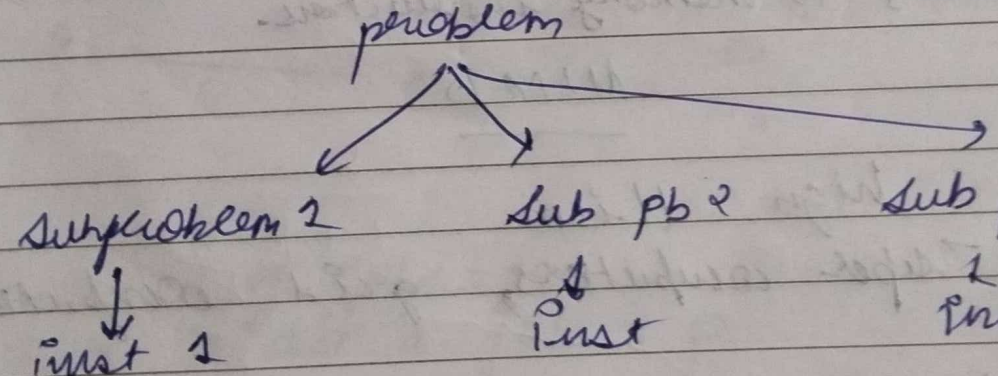
SIMD

instruction same → data different.

deterministic data → we know the output

synchronous $S_1 \rightarrow S_2 \rightarrow S_3$ } → data source
behavior

in SIMD T_1 should know at what time which data is getting from where
eg at T_1 → data from S_1.



eg → addition is a subproblem and mutiple data → (1, 2), (3, 4) are coming
so instruction → single
data → mutiple.

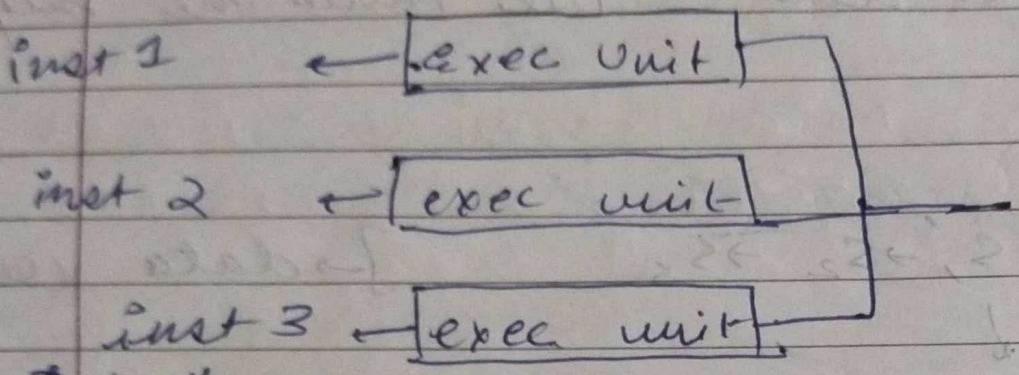
slide 32 ; read.

Page No.
Date

MISD

MISD bottleneck

→ low level of ||ism.
→ high sync required synchronization.



→ bottleneck.
→ limited ||'sm, memory access, microcode, resource contention. instruction decoding.

→ CISC → complex instruction set computers
→ rich instruction set, variable length instruction, memory to memory instructions.

MIMD

high ||ism
super computers, grid computers

→ no sync necessarily required but can be both.

→ no data is shared → loosely coupled.
→ if sharing then sync.

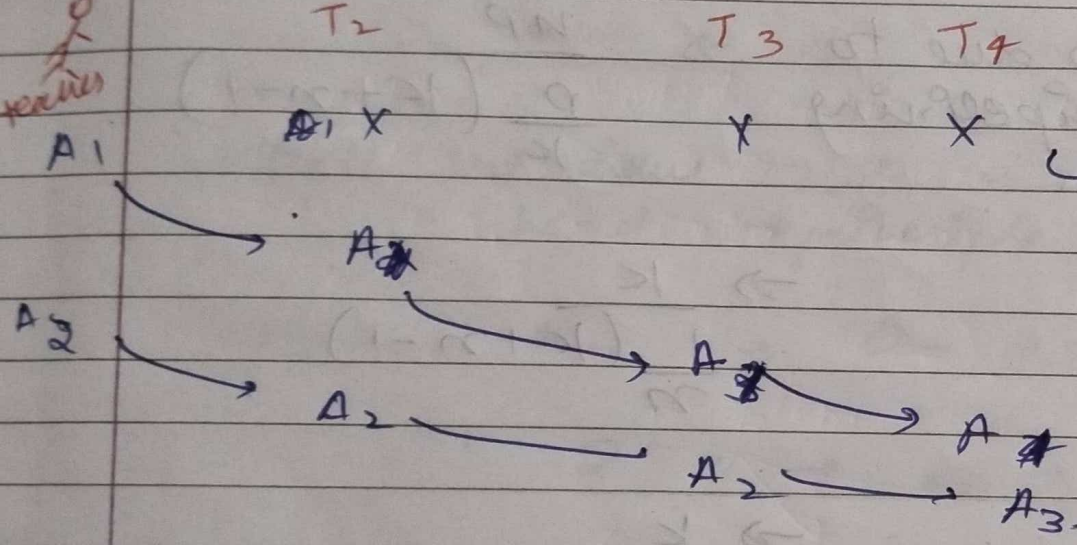
kinds

temporal | data | hybrid.

GPU → graphical processing units.
specific type of task is assigned to small components.

~~kinds~~ Kinds of parallelism

→ temporal ||ism → all tasks should be similary with no delay.
↓
no transfer time



let the no of jobs = n
let time to do a job = p

let each job be divide into k task &
let each task be done by a different individual.

let the time for each task ⇒ $\frac{p}{k}$.

time to complete → np.
n jobs with no pipelining.

Date

pipeline organisation of k individuals

$$\Rightarrow p + (n-1)\frac{p}{k}$$

$$\Rightarrow \frac{kp + np - p}{k}$$

$$\Rightarrow \frac{p(k+n-1)}{k}$$

speed up due to pipelining $\Rightarrow \frac{np}{\frac{p}{k}(k+n-1)}$

$$\Rightarrow \frac{k}{\frac{1}{n}(k+n-1)}$$

$$\Rightarrow \frac{k}{1+\frac{k-1}{n}}$$

problems to temporal limit.

→ Transfer time → synchronisation

→ we assumed this negligible.

Date

② Fault tolerance (all processors are equal) one does fault can't be sudden/fixed.

③ Scalability not possible

④ inter task ~~comp~~ communication

⑤ Bubbles in pipeline (one stage all other stages)

Data II ism.

suppose we have 1000 unsure fleets.
head → principle → divides.

teacher one → 0 - 250

" 2 → 250 → 500

" 3 → 500 - 750

" 4 → 750 - 1000

→ So all teachers are assigned tasks at one time together & again assigned together → (no overhead, Inter process communication

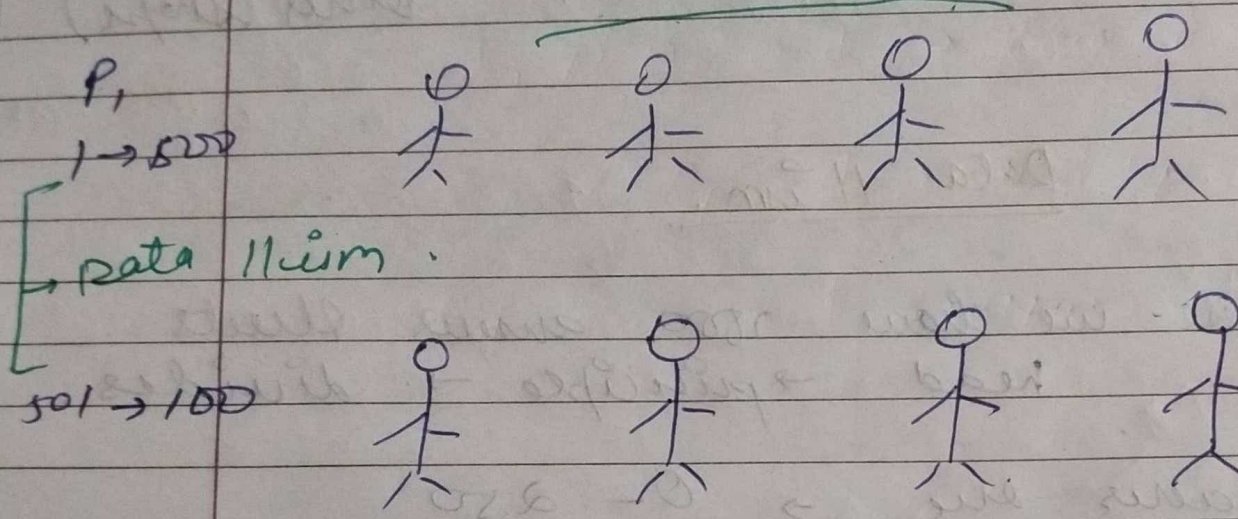
disadvantage →

→ single point of failure.

→ task should be ~~mutually~~ independent.

→ cost more (head takes more salary more than 4 teachers).

Mixed II-ism → temporal



eg → NEC - SX
CRAY.

Data II-sm → with dynamic assignment instead giving 1-250 at same time give next if previous job is done by the processor. (1 given at a time)

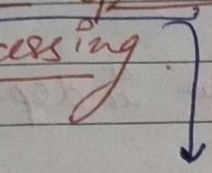
Quazi - dynamic → now one time in dynamic giving 4-5 at a time.

Date

specialist data // ism →

specialist person → who knows who can do what!

1. Couse Grained specialist temporal parallel processing



in ways
out ways of the processor.

Detecting // ism

→ can this run on a // arch

F/p Prism.
dependent → cannot
independent → can run.

Bernstein's condition we define →

I_i → input set → from have to fetch from memory
read

O_i → represent write set → write into memory

process →
Read set
write set

P_1 and P_2 will be independent when

P_1 // P_2 only when they are independent.

$I_1 \cap O_2 \Rightarrow \phi$ → flow independence.

$I_2 \cap O_1 \neq \phi$ → anti independence

$O_1 \cap O_2 = \phi$ → output independence

eg →

P_1 : $c = D \times E$

P_2 : $M = G + C$

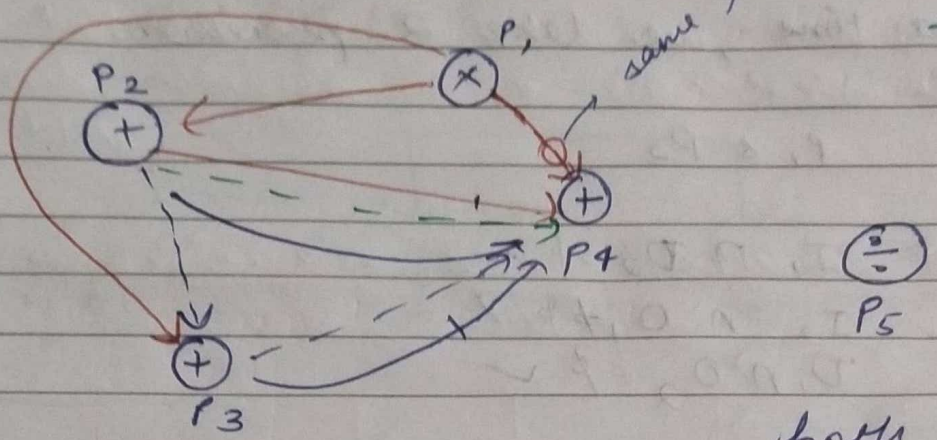
P_3 : $A \Rightarrow B + C$

P_4 : $C = L + M$

P_5 : $F = G \div E$

Page No.
Date

dependence graph →



→ out
→ anti
→ flow

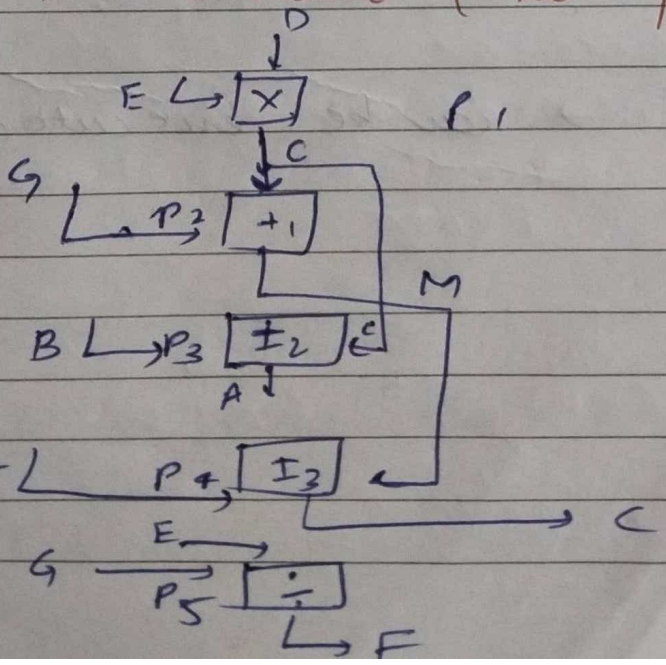
both use + ↳ so resource dependency

there are 3 addition units ($+_1, +_2, +_3$).

solid arrows → data dependence.
Dotted arrows → resource dependence.

for paper → flow, anti, output

sequential execution (no pipelining)



// execution →

at a time, we take 2 processors

P_1 & P_2

$I_1 \cap O_2 = \phi$ ✓

$I_2 \cap O_1 \neq \phi$ ✗

$O_1 \cap O_2 = \phi$ ✓

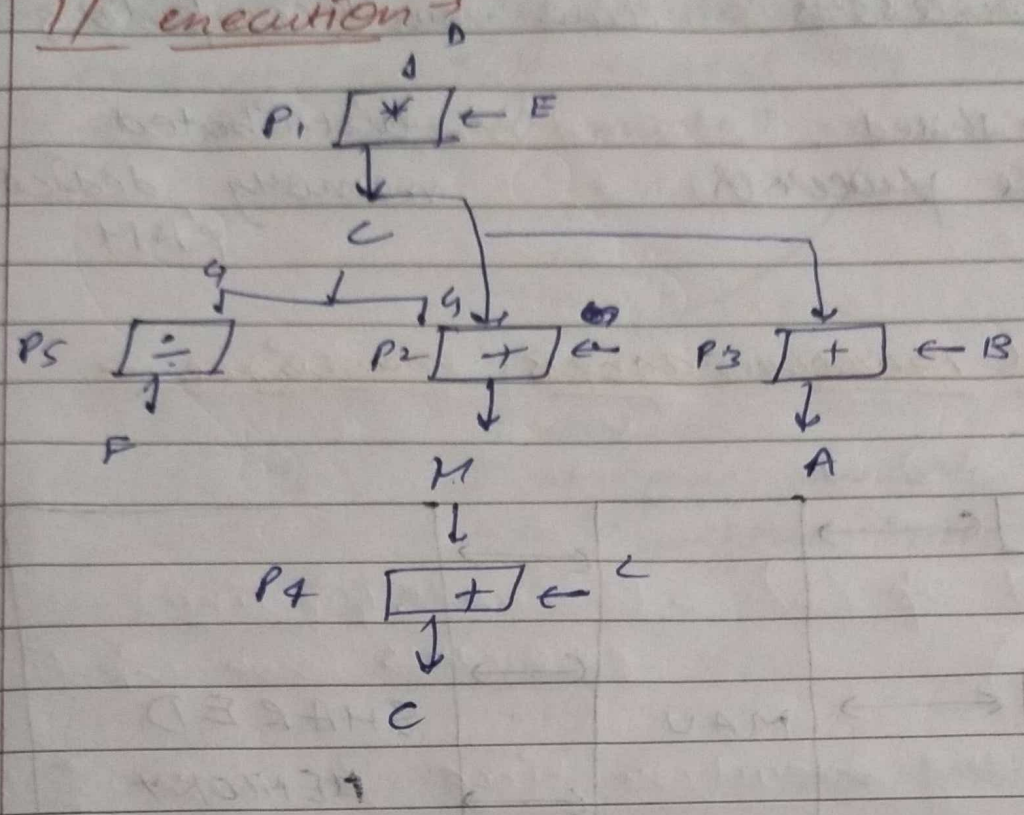
for pairs (all) when all 3 conditions are true then they can run parallely.

P_1 & P_5
P_2 & P_3
P_2 & P_5
P_3 & P_5
P_4 & P_5

→ these will run //.

P_2, P_3, P_5 → can be executed //.

// execution →



Processor types

vector & scalar

ppt ← more cores

CISC
RISC
} architecture

locality of reference → Basic principle on which cache works.

→ spatial / temporal locality

↓

→ cache
→ pipelining

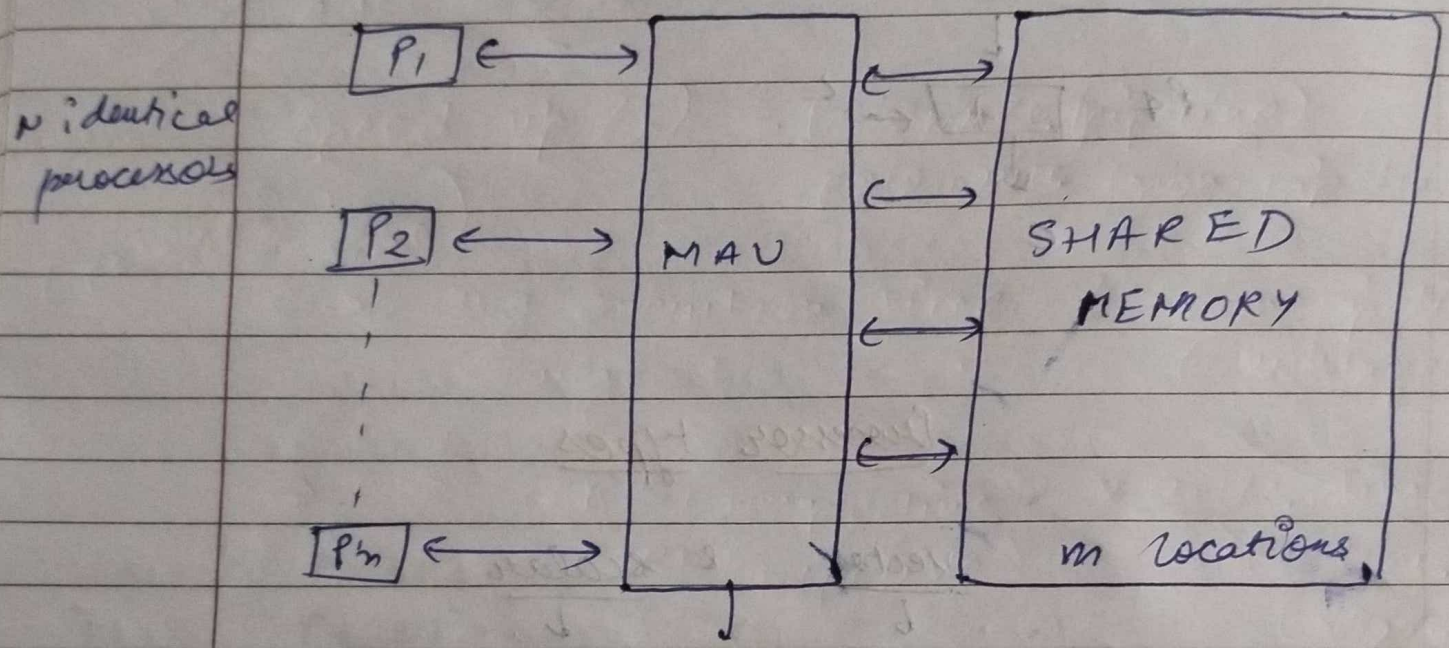
→ study all these terms.

multiprocessors and multicomputers

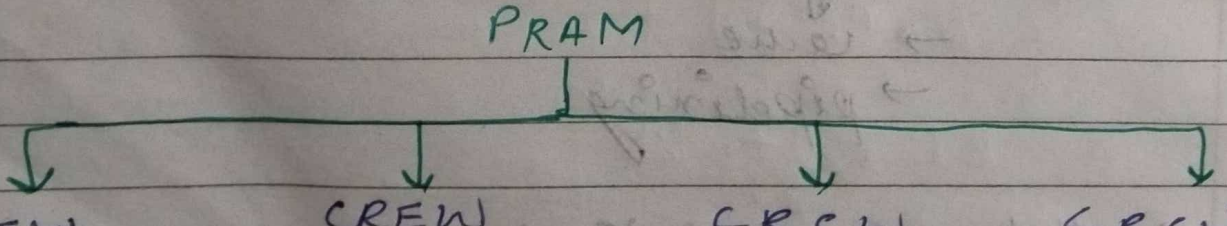
memory shared multiple processor

distributed memory dedicated RAM

(PRAM) parallel Random access machine →



all n processors has equal access to the shared memory:



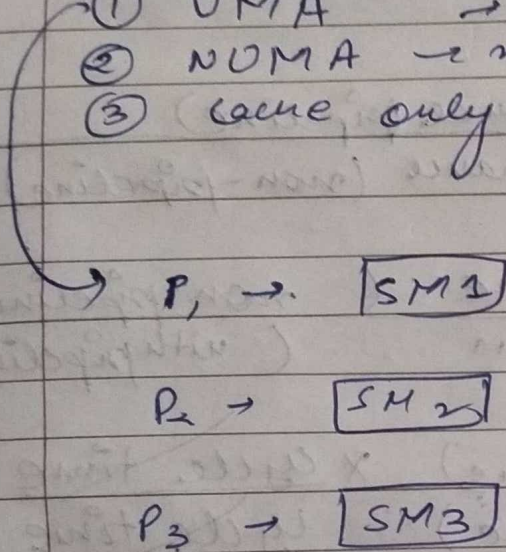
exclusive read exclusive write

concurrent read exclusive write

① shared memory model.
② message passing model.
③ data parallel model.

shared memory

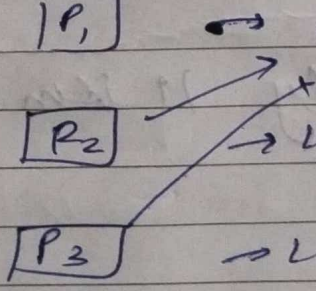
① UMA → uniform memory access
② NUMA → non-uniform memory access.
③ cache only



→ each processor has its own memory and but other processor can access the other's memory equal.

NUMA →

every P will have local RAM



eg → super computer

here P1 has more right on LR, (all do not have same right)
processores are loosely coupled.

Cache only (COMA) →

only cache shared b/w processors.
RAM not shared.

parallel programming paradigms →

unit 1

slide 32 →

Speed up (S) = Performance (pipeline) / Performance (non-pipeline)

(reciprocal $\frac{1}{Time}$)

S → avg execution time (non-pipeline) / avg " " (with pipeline)

S → CPI (non pipeline) / CPI (pipeline) × cycle time (non) / cycle time (with) same.

due to stalls →

S → CPI non pipeline / (1 + no of stalls per instruction)

because ideally in CPI → 1