Date 19th feb24 Monday

Quiz 1 :-

Uniprocessor are fast but some probs require : too much data
too much computation
too many parameters to explore

* Lecture 1 :-

Moore's Law : The num of transistors on microchips doubles every 2 years.

Moore's Law does not hold because ?

1.) multiple cores on single chip cause heat issues
2.) increasing num of cores may not be able to increase speeds (due to inter-process interactions)
3.) transistors would reach the limit of miniaturization

Memory / Disk Speed Argument :-

→ Clock Rates of processors have increased 40% per year over past decade, DRam has improved roughly 10% per year
→ Mismatch in speeds = performance bottlenecks
→ Parallel platforms :-
① Provide increased bandwidth
② provide higher aggregate caches

Data Communication Argument :-

→ Volume of data is too much in databases & data mining probs so it can not be moved => analysis of this data is done by using parallel techniques.

Date

① Distributed Systems:

collection of autonomous computers, connected thru a network and distribution middleware

→ computers coordinate their activities to share the resources of the system

→ system is perceived as a single, integrated computing facility

② Distributed Computing:

a specific use of dist sys, to split a large & complex processing into subparts & execute them in parallel to increase the productivity

① develop algos for dist cluster systems

② computers without any shared mem & have geographical dist

③ high latency & no shared clock

③ Parallel Computing:

Develop concurrent solutions for 2 types of systems

① Multi-Core Arch ② Many-Core Arch

Limitations of Parallel Computing:

① design proper communication & synchronization b/w process & sub-tasks

② program must have low coupling & high cohesion

③ requires more technical skills to code a parallel prog

④ exploring proper parallelism from a prob is a hectic process

See Applications of Parallel Computing from slides.

Date

Lecture 2 :-

Amdahl's Law :

- formulized in 1967
- shows an upper bound on the max-speedup that can be achieved

$$\text{Speedup } (p) = \frac{1}{F + \frac{(1-F)}{p}}$$

serial = F
parallel = 1 − F
processors = p

Derivation :-

Time for sequential code = T(s)
Time for parallel code = T(p)

T(p) = serial comp time + parallel comp time

$$T(p) = F.T(s) + \frac{(1-F).T(s)}{p}$$

$$\text{Speedup} = \frac{T(s)}{T(p)} = \frac{T(s)}{F.T(s) + \frac{(1-F)T(s)}{p}} = \frac{T(s)}{T(s)\left[F + \frac{(1-F)}{p}\right]} = \frac{1}{F + \frac{1-F}{p}}$$

Example 1: 70% of seq algo is parallelizable. Remaining must be calculated sequentially. Calc Speedup

1.) 4 Processors

$1 - F = 0.70$

$F = 0.3$

$$\frac{1}{0.3 + \frac{(1-0.7)}{4}}$$

=

2.) Infinite Processors

$$\frac{1}{0.3 + \frac{(1-0.7)}{\infty}}$$

=

Date

Example 2 : 25% of seq algo is parallelizable. Remaining is sequential. Calc max speedup

1.) 5 processors :-

$1-F = 0.25$

$F = 0.75$

$$\frac{1}{0.75+\frac{(1-0.25)}{5}} =$$

2.) Infinite processors

$$\frac{1}{0.75+\frac{(1-025)}{\infty}} = 1.333$$

$\frac{(1-025)}{\infty}$ becomes zero

Q. How many proc needed to achieve max speedup ?

$$\text{speedup} = \frac{1}{F+\frac{(1-F)}{P}} \Rightarrow 1.33 = \frac{1}{0.75+\frac{(0.25)}{P}} \Rightarrow 1.33 = \frac{1}{\frac{0.75P+0.25}{P}}$$

$$1.33 = \frac{P}{0.75P+0.25} = 0.9975P + 0.3325 = P \Rightarrow 0.0025P = 0.3325$$

$P = 0.3325/0.0025 = 133$ processors

Karp-Flatt Metric :-

calculate serial fraction for a given parallel config

$$e = \frac{1/s - 1/p}{1 - 1/p} \quad \therefore s = \text{speedup}$$

Q. in a parallel prog, for 5 processors, you gained speedup of 1.25 determine seq fraction.

$$e = \frac{1/1.25 - 1/5}{1 - 1/5} =$$

Date

① Sometimes the primary reason for the poor speedup is the sequential part of code
② Sometimes when the serial fraction e is steadily increasing with p, parallel overhead contributes to the poor speedup

Types of Parallelism :-

① Data Parallelism : independant tasks applying same operation to diff elements of a data set. eg for(int i=0 ; i<99 ; i++)
All 100 iterations of loop could be exe a[i] = b[i] + c[i]
simultaneously.

② Functional Parallelism :- independant tasks applying diff operations to diff data elements e.g a=2 , b=3
③ ④ can be performed concurrently ① m = (a+b)/2 ② s = (a²+b²)/2

③ Pepelining :- divide whole computation of each instance into multiple stages provided that there are multiple instances of the problem
- output of one stage is input of the other stage
- used for probs where single instance of prob can not be parallelized

engine	doors	wheels	paint
car 1			
car 2	car 1		
car 3	car 2	car 1	
car 4	car 3	car 2	car 1
	car 4	car 3	car 2
		car 4	car 3
			car 4

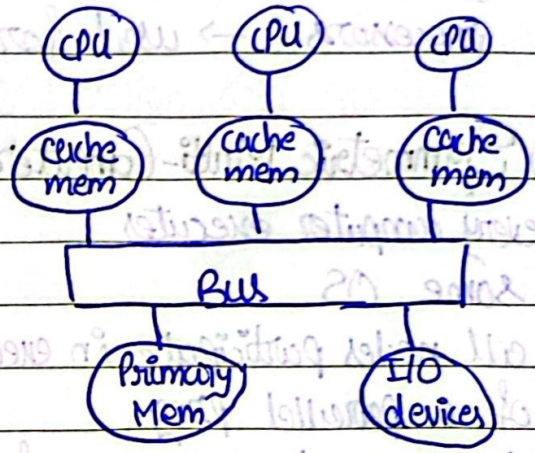
Date

* Multi-Processor :-

→ Multiple CPUs with shared mem
→ same addr on 2 diff CPU's refers to same memory location

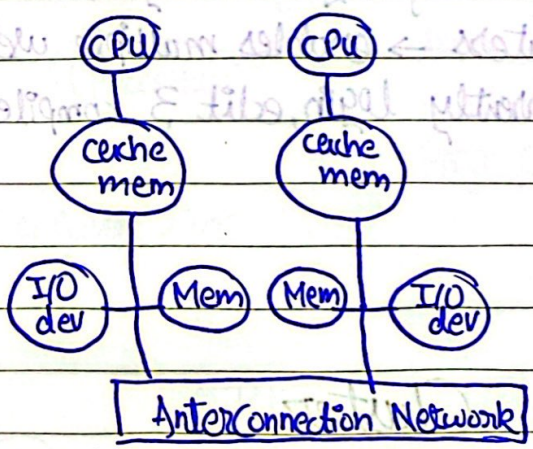
① Centralized MP :-

- all processors share same primary mem
- all memory is at one place
- same access time from every processor
- UMA (uniform memory access)
- SMP (symmetric multi-processor)



② Distributed MP :-

- distributed collection of memories forms one logical addr space
- same addr on diff processors refers to same mem location
- NUMA (Non-Uniform Mem Access)
- Mem access time varies depending on physical location of referenced addr



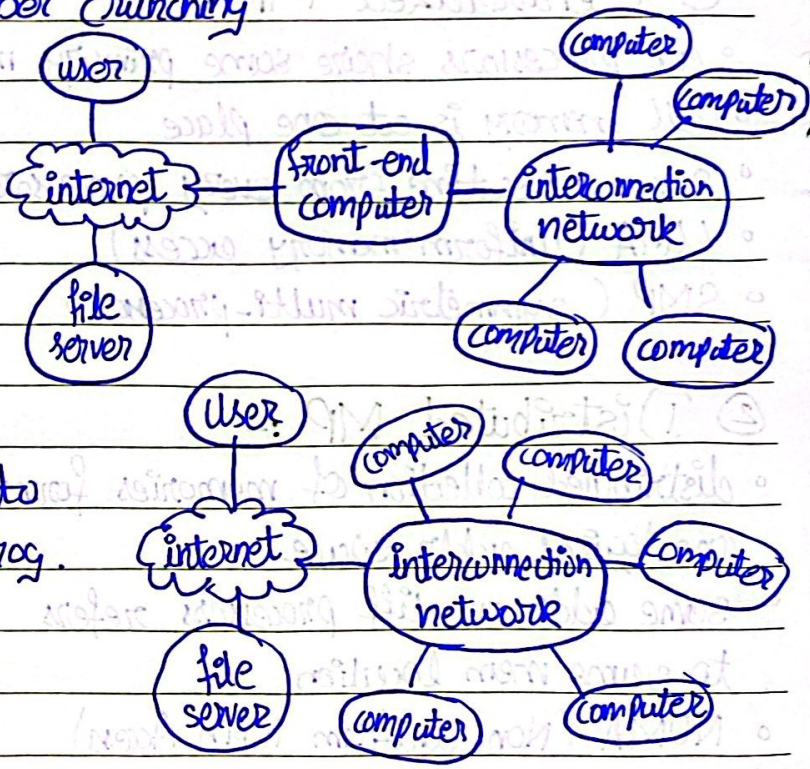
* Multi-Computer :-

→ dist memory
→ disjoint local addr spaces
→ each processor has direct access to their local mem only
→ same addr on diff processors refers to two diff physical mem location
→ processors interact w each other thru passing msgs

Date

① Asymmetric Multi-Computers:

→ front-end computers interact with users & I/O devices
↳ executes a full, multi-programmed OS & provides all functions needed for prog development.
→ Back-ends are reserved for executing parallel prog processors → used for "number crunching"



② Symmetric Multi-Computer:

→ every computer executes same OS
→ all nodes participate in execution of a parallel prog
→ Users may log into any of the computers → enables multiple users to concurrently login, edit & compile prog.

Cluster

Co-located collection of low-cost computers & switches dedicated to running parallel jobs.
Run Same Version of OS
Some computers may not have interfaces for user login
• Use high speed networks for communication

Network of Workstations

• Dispersed collection of computers
• Diff OS & executable programs
• User can login & poweroff their workstations
• Ethernet speed for this network is slow.

Date

1.) Cache Coherence : In a distributed env, each CPU's cache needs to be continuously updated for current values

2.) Snooping : achieve data consistency b/w cache mem & shared mem thru a bus-based system. "Write-Invalidate" & "Write-Update" policies are used for maintaining cache consistency.

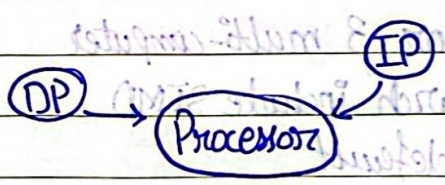
3.) Branch-Prediction : technique used in CPU design that guesses the outcome of a conditional operation & prepare for the most likely result.

Lecture 3 :-

* Flynn's Taxonomy :-

1.) SISD :-

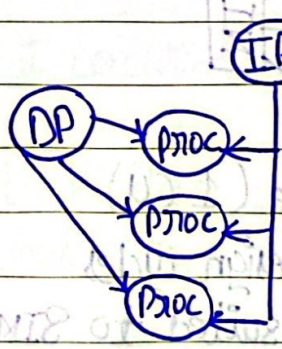
- single instruction, single data
- single core computers
- single instruction stream is in execution at a given time
- only one data stream active at a time



load A
load B
C = A+B
Store C
time

2.) SIMD :-

- parallel architecture with multiple cores
- cores execute same instruction stream, data stream is diff
- Used with Array Operations, img processing & graphics
- Well-Suited for matrix (vector) operations & scientific operations
- Cray Vector Proc Machine



single inst
multiple data

P1	P2
load A(1)	load A(2)
load B(1)	load B(2)
C(1) = A(1) x B(1)	C(2) = A(2) x B(2)
store C(1)	store C(2)

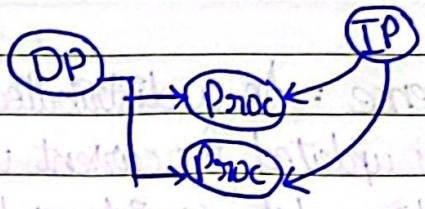
Load A(1)
C(1) = A(1) x 1
Store C(1)

Load A(1)
C(2) = A(1) x 2
Store C(2)

Date

MISD :-

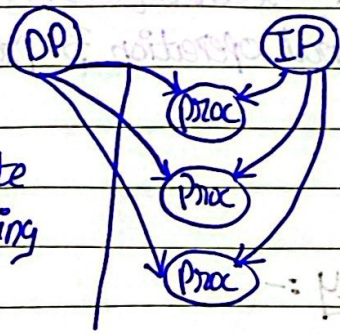
multiple instr stream,
single data stream
pipeline of multiple independantly executing functional units.
same data set



rarely used
e.g Systolic Arrays

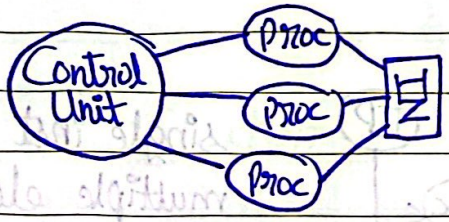
MIMD :-

multiple instr, multiple data
diff CU's can simultaneously execute diff instruction streams manipulating diff data
e.g Multi-processor & multi-computer
Many MIMD arch include SIMD executions by default



load A(1)	call func(0)
load B(1)	u = y x z
store C(1)	sum = u x 2
P1	P2

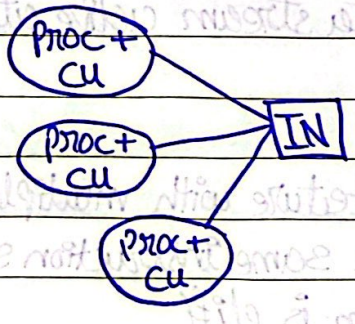
SIMD :



- req less hardware (1 CU)
- expensive & long design cycles
- not all apps are suited to SIMD

MIMD :

IN = interconnection network



- SPMD (Same Prog, Multiple Data) paradigm built from inexpensive common comp with lil effort & short time

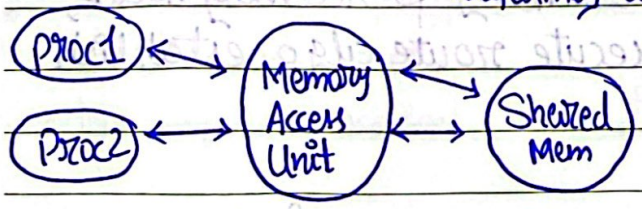
★ SPMD is close variant of MIMD

Date

* PRAM :-

- parallel random access machine
- consists of p processors
- global memory → unbounded size
  ↳ uniformly available to all proc with same addr space

proc share common clock cycle but execute diff inst in each cycle



① EREW :-

→ exclusive read, exclusive write
→ No two proc can perform read/write concurrently
→ Weakest PRAM, min mem access concurrency

② CREW :-

→ read concurrently but cant write at same time
→ multiple write accesses to mem location are serialized

③ ERCW :-

→ Cant Read Concurrently but can write

④ CRCW :-

→ Most powerful PRAM model

* Concurrent Reads cause no probs, but concurrent write ...

Arbitration Protocols for Concurrent Write :-

1.) Common : write only if all values that proc are attempting to write are same.
2.) Arbitrary : write data from random proc & ignore rest
3.) Priority : follow predetermined priority order. Proc with highest priority succeeds, rest fail
4.) Sum : write sum of data items in all write reqs.
   ↳ Can be extended for any associative operator

Date

Communication Costs :-

→ comm is major overhead in parallel prog
→ dist sys suffer from major comm overheads

① Startup Time (t_s) :- Time spent at sending & receiving nodes, prepare msg, add headers, trailers, execute route algo, establish interface b/w node & router

② Per-hop time (t_h) :- num of hops & includes factors such as switch latencies, network delays etc.

③ Per-Word transfer time (t_w) :- includes all overheads determined by length of msg. includes link bandwidths & buffering overheads.

★ Store & Forward Routing :-

→ Msg traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop.

$$t_{comm} = t_s + (mt_w + t_h)l$$

∴ m words
∴ l comm links

$\Rightarrow t_h$ small so $\Rightarrow t_{comm} = t_s + mlt_w$

Packet Routing breaks msg into packets, each packet carries routing info, error checking & other header info so

$$t_{comm} = t_s + t_h l + t_w m$$

∴ t_w = overheads in packet headers

Date

Cut-Through Routing:-

→ div msg into basic units called "flits", they are v small so header info minimized.

→ Trecer Msg programs call intermediate routers → flits then take the same route

→ No seq num needed

→ t_w is much smaller wrt store & forward

$$t_{comm} = t_s + t_h l + t_w m$$

$$t_{comm} = t_s + t_w m$$

∴ t_h v small, ign