23/11/24

Page No.
Date : / /

UNIT - 2

Communication Overhead is the time and ~~Depending upon the bandwidth~~ the resources spent on transfering data and messages b/w the parallel components (processing elements or nodes)

→ Depending upon the bandwidth, latency and how summary information is communicated among the processing elements, delays due to communication delay may vary from processor to processor

Sources of overhead

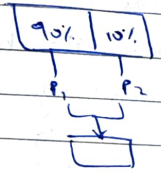
① Interprocess Interaction

Any non-trivial parallel system requires its processing elements to interact and communicate data. The time spent in communicating data b/w processing elements is usually the most significant source of the processing overhead.

eg- a lot of data is getting exchanged but not that much o/p is produced

(2) Idling

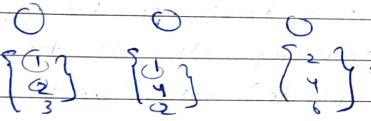
Processing elements in parallel computing may become idle due to many reasons like load imbalance, synchronization, and presence of serial components in the program.



(3) Excess / Essential Computation

The difference in computation performed by parallel program and the best serial program is the excess computation overhead

eg- Fast Fourier Transform



In these tasks, there are redundancy of sub-tasks {1,2} are getting calculated again & again

In serial, this would get calculated once and used again.

Granularity

It is the measure of amount of work or computation performed by a task

$$\text{Granularity} = \frac{\text{Computation time}}{\text{Communication time}}$$

Parallelism based on Granularity —

① Fine-grained parallelism
- program is divided into many smaller tasks
- it facilitates load balancing
- it is best exploited in architectures that support high speed communications

Drawback - If granularity is too fine it is possible that the overhead required for communication and synchronisation b/w the tasks takes longer time than the computation time.

eg- a) Connection machines (CM-2)
b) j-Machines

Page No.
Date : / /

② Coarse-grained parallelism
- relatively large amounts of computational works are done b/w communication or synchronisation events.

- harder to load balance efficiently (bcz kuch task bohot jaldi ho rahe hai some are idle some are doing last)

- message passing architectures are well suited for this parallelism.

eg- Cray Y-MP

③ Medium-grained parallelism
eg- Intel iPSC

Impact of granularity on Performance
① Synchronisation overhead, scheduling strategies can negatively impact the performance of fine-grained tasks.
② To reduce communication overhead, we can ↑ce the granularity.

Definitions
① Degree of concurrency - No. of tasks that can be executed in parallel.

Page No.
Date : / /

② Critical path length - A directed path in the task dependency graph represents a sequence of tasks that must be processed one after the other. The longest path length in task dependency graph will be the critical path length.

Decomposition Techniques
The operation of dividing the computation into smaller parts, some of which may be executed parallely

① Recursive Decomposition
based on divide and conquer algorithms.

② Data Decomposition
it divides the data in program into parts and then assigns them into instructions.
eg Matrix Multiplication Problem.

$$A = \begin{bmatrix} A1 & A2 \\ A3 & A4 \end{bmatrix} \qquad B = \begin{bmatrix} B1 & B2 \\ B3 & B4 \end{bmatrix}$$

Task 1 : $C_1 = [A1 * B1 + A2 \times B3]$
Task 2 : $C_2 = [$
Task 3 : $C_3 =$
Task 4 : $C_4 =$

→ Tasks induced by data decomposition are performed entirely, and each task

Page No.
Date : / /

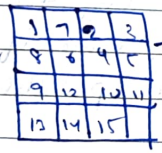
performs useful computation towards the soln of the problem.

30/01/24

(3) Exploratory Decomposition

Decomposition acc to a search of a state space of solutions

eg- 15 puzzle problem



→ This is the state space

→ Tasks induced by exploratory can be terminated before finishing as soon as the desired solution is found.

(4) Speculative Decomposition

When a program may take one of many possible branches depending on the results from computations preceding the choice.

eg- Nested if

```
if( )
{
  if( )
```

In a program if and 'if' statement contains another if and the program is already decomposed, the lines of code are assigned to multiple processors without even checking the conditions.

- different processors execute both ifs.
- The processors will run both the ifs statements and its inner ifs.

Inner if is discard if 1st if becomes false.

(5) Functional Decomposition - Done after numerical Δ

Parallel Processing in Memory

3 types -

1. Shared Memory Architecture
2. Distributed Memory Architecture
3. Hybrid " " (Combi of 2)

① Shared Memory Architecture

a) Multiple processors operate independently but share the common memory as global address space

b) Shared memory systems are tightly coupled systems

c) Only one processor can access the shared mem location at a time.

d) Changes in a memory location affected by one processor are visible to all other processors.

They are able to use the updated values.

→ divided in 2 categories based on the Memory Access Time.

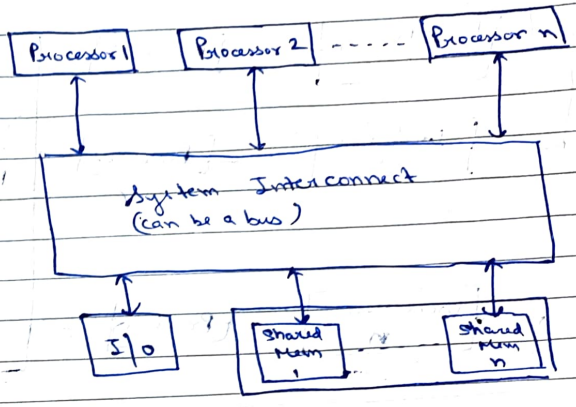
① Uniform Memory Access (UMA)
② Non-Uniform " " (NUMA)
   special category → cache only Mem. Architecture (COMA)

Uniform Mem Access

① The physical mem is shared by all the processors.
② All processors have equal access time to all memory locations.
③ used in symmetric multiprocessor machines (SMP)
④ Each processor may have private ~~access~~ cache is called CC-UMA
   aka CC-UMA
   ↳ cache coherent

cache coherency = ek processor ne update kardia hai toh baki bhi access kar paye update ko

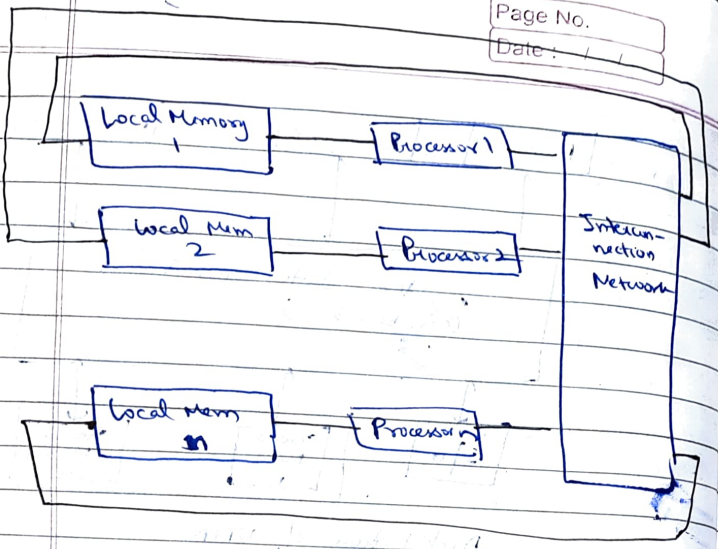
→ if one processor updates a location in shared memory all the other processors know about the update.



Architecture of UMA

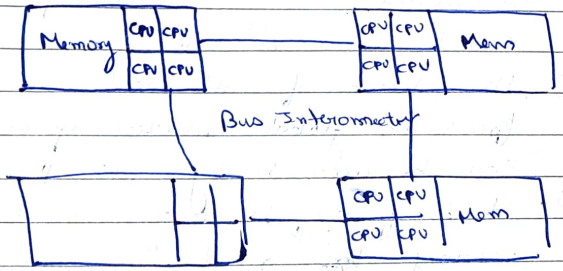
Non-Uniform Memory Access

① access time varies with the location of the memory
② the shared memory is physically distributed among all the processors called local memories.
③ The collection of all local memories forms the global address space which can be accessed by all the processors.
④ It is faster to access a local memory with a local processor.
⑤ The access of remote memory attached to the processors is slower due to the added delay through the interconnection network.



NUMA architecture

⑥ It is often made by physically linking 2 or ~~main~~ more SMPs. (Symm. multi Processor)



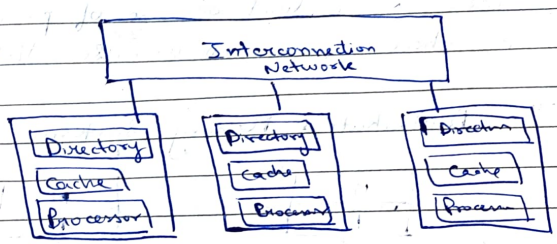
Connections when multiple SMP involved

special case → CCNUMA

① The distributed main mems are converted to caches.

② All caches form a global address space and there is no mem hierarchy at each processor node.

③ Remote cache access is assisted by the distributed cache directors



Directory — it helps in finding which data is located at which position and how to access that position.

Advantages of Shared Mem Architecture

① Global address space provides a user friendly programming perspective to memory

② Data sharing b/w tasks is both fast and uniform due to the proximity of memory to CPUs.

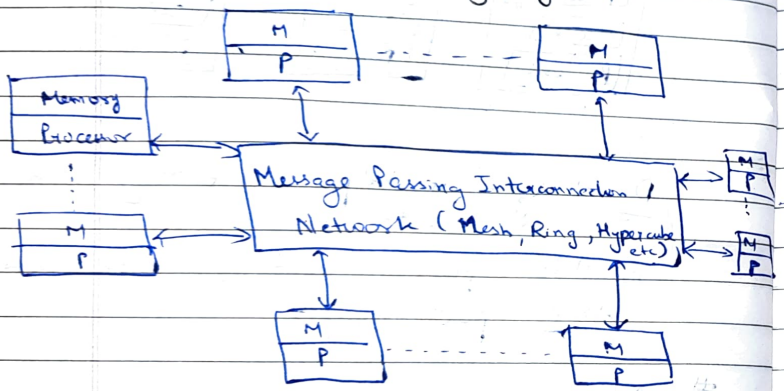
Page No.
Date : / /

Disadv

① Lack of scalability b/w mem and CPU.
② Programmer responsibility for synchronisation constructs that ensure correct access of global memory.
③ Expensive - as more processors are involved

There is a limit on no. of processors (can) in shared mem so we use distributed

31/01/24 Distributed Memory System



① It consists of multiple computers called nodes, interconnected by a message passing network.
② Each node is an autonomous computer consisting of a processor, local memory, and sometimes attached disk or input-output peripherals

Page No.
Date : / /

③ Distributed memory are loosely coupled system (shared mem are tightly coupled)
④ Memory addresses in one processor donot map to another processor. So there is no concept of global address space.
⑤ Bcz each processor has its own local memory, it operates independently.
⑥ Changes made to the local memory have no effect on the memory of other processes. Hence, the concept of cache coherency doesnot imply.
⑦ When a processor needs to access the data from another processor, it is done by passing message b/w the processors. which is usually the task of the programmer.

Advantages

① Memory is scalable with no. of processors
② Increasing the no. of processors, the size of the memory also ↑ses independently.
③ Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
④ Cost effectiveness [because we can use other topologies like mesh ring] can use

commodity, of the shelf processors and networking

Disadvantages

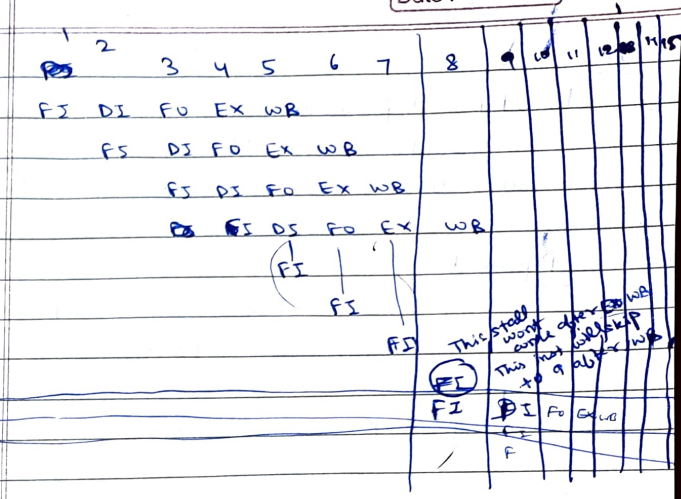
① The programmer is responsible for many of the details associated with data communication b/w the processors.

② NUMA takes a longer time

Hybrid – combo of all adv & disadv

Numerical (Instruction Pipeline)

Consider an instruction pipeline with 5 stages (FI, DI, FO, EX, WB) without any branch prediction. The stage delays are 5ns, 7ns, 10ns, 8ns & 6ns. There are intermediate storage buffers after each stage and delay of each buffer is 1ns. A program consisting of 12 inst is executed in this pipelined processor. I4 is the only branch instructions and its branch target is I9. If the branch is taken during exe of the program then what is the time needed to complete the program.



Max delay = 10

Cycle time = 11 ns

Execution time = $(k+n-1) \times T_p$

$= (5+9-1) \times 11$

$= 13 \times 11$ ns

Total cycles = 15

3 stall

No. of cycles x Cycle time

Total time = 15 × 11 ns

= 165 ns

Q2- There is inst pipeline with 4 stages. Stage delays 5, 6, 11ns, 8ns. Delay of an interstage register is 1ns. Find speedup of pipline in steady state under ideal conditions as compared to corresponding non pipeline implementation

$\frac{(3+n) \times 12}{n \times 12}$

Page No.
Date : / /

Ans -

Non-Pipelined processor

Time to execute N instructions = ~~cycle time~~ = (5+6+11+8) × N = 30N

Pipelined

Cycle Time = max (5, 6, 11, 8) + 1 = 12ns

For N inst = 12 N

Speedup = $\frac{30}{12}$ = 2.5

Page No.
Date : / /

06/02/24

(5) Functional Decomposition

In this approach, the focus is on the computation that is to be performed rather than the data manipulated by the computation

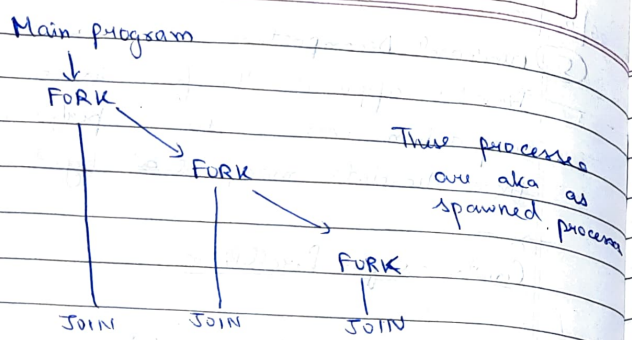
Constructs for Parallelism

(1) Creating concurrent processes
- uses FORK-JOIN

in the original construct, a FORK statement generates one new path for a concurrent process and the concurrent process uses the JOIN statement at their ends.

(2) UNIX Heavy weight Processes

The UNIX system call fork() creates a new process. The new process (child process) is an exact copy of the calling process except that it has an unique process id.
It has its own copy of the parents variable.
They are assigned to the same values as the original variables initially.
On successfully executing, fork() returns zero to the child process and returns the process Id of the child process to the parent process.
Processes are joined with system calls wait() & exit().



Definitions

→ Heavy weight process - It completely separates program with its own variables, stack and memory location.

→ light-weight processes - Shares the same memory space and global variables b/w the routines.

→ Posix Thread (pthread) -

↳ Portable Operating System Interface

pthread is an execution model that exists independently from a language as well as a parallel execution model.

- It allows a program to control multiple different flows of work that overlap in time.

- header used <pthread.h>

- There are around 100 thread procedures all prefixed 'pthread_' and they can be characterised into 4 groups -

① Thread management (creating threads)

② Mutexes

③ Condition Variables

④ Synchronisation b/w threads using read or write locks and barriers.

a) Thread management - Thread operations include thread creation, termination, synchronisation (joins, blocking), scheduling data management, and process interaction

Syntax:

```
int pthread_create (pthread_t* thread,
    const pthread_attr_t * attr,
    void * (* start_routine) (void*),
    void *arg);
```

① thread - gives pointer to an unsigned integer value that returns the thread id of the thread created.

② attribute - pointer to a structure that is used to define thread attributes like detached state, scheduling policy, stack address, etc

Page No.
Date : / /

③ set to null for default thread attributes.
Start routine - pointer to the subroutine that is executed by the thread.
return type is void.

→ The function has a single attribute but if multiple values need to be passed to the fn, a struct must be used.

④ argument - pointer to void that contains the arguments to the function defined in the earlier argument

eg- Main program
:
pthread_create (&thread1, NULL, proc1, &arg);
:
pthread_join (thread1, *status);

thread1
proc1 (&arg)
{
:
return (*status)
}

void pthread_exit (void * retval);
Pointer to an integer that stores the

return status of the thread terminated. The scope of this variable must be global so that any thread waiting to join this thread may read the return status.

int pthread_join (pthread_t th, void** thread_return);

① th → thread id of the thread for which current thread waits.
② thread_return → pointer to the location where the exit status of the thread mentioned in th is stored.

pthread_t pthread_self (void);
self- used to get the thread id of the current thread.

int pthread_equal (pthread_t t1, pthread_t2);
if two threads are equal then it returns a non zero value (here p1, p2 are 2 threadids.

int pthread_cancel (pthread_t thread);

Page No.
Date : / /

b) Mutexes

- used to prevent data inconsistencies due to race conditions.

Race condition often occurs when 2 or more threads need to perform operations on the same memory area but the results of computations depends on the order in which these operations are performed.

c) Condition variables

a variable of time pthread_cond_t and is used with the appropriate fns for waiting and later process continuation.

A condition variable must always be associated with the mutex to avoid the race conditions.

Page No.
Date : / /

13/02/24

2 Types of Ilism

① Hardware Parallelism → Ilism defined by the machine architecture and H/w multiplicity. It displays the resource utilisation patterns of simultaneously executable operations.

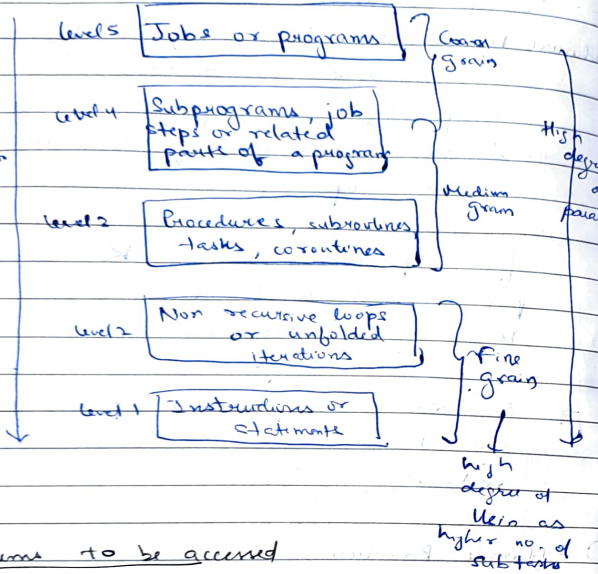
If a processor issues k instructions per machine-cycle then it is called k-issued processor.

→ A conventional pipelined processor takes one M/c cycle to issue a single instruction, cycle

② Software Parallelism → This type of Ilism is revealed in the program profile or in the program flow graphs. (flow chart)

* threads (chhudse dikhna) and processes

Grain Packing



~~Cons~~ Problems to be accessed

Q1- How can we partition a program into ~~lbl~~ branches, program modules, micro-tasks or grains to yield the shortest possible execution time.

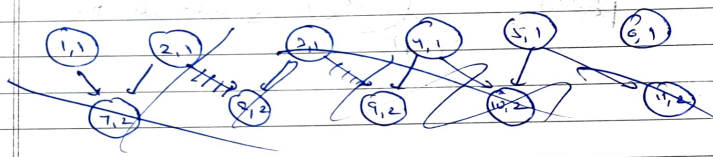
Q2- What is the optimal size of the concurrent grains in a computation.

Solution- Grain Packing - The idea of grain packing solves the above ~~at~~ challenges.

→ The idea of GP is to apply fine grain first in order to achieve higher degree of parallelism. Then one combines multiple fine grain nodes into a coarse grained node if it can eliminate unnecessary communication delays or reduce the overall scheduling overhead.

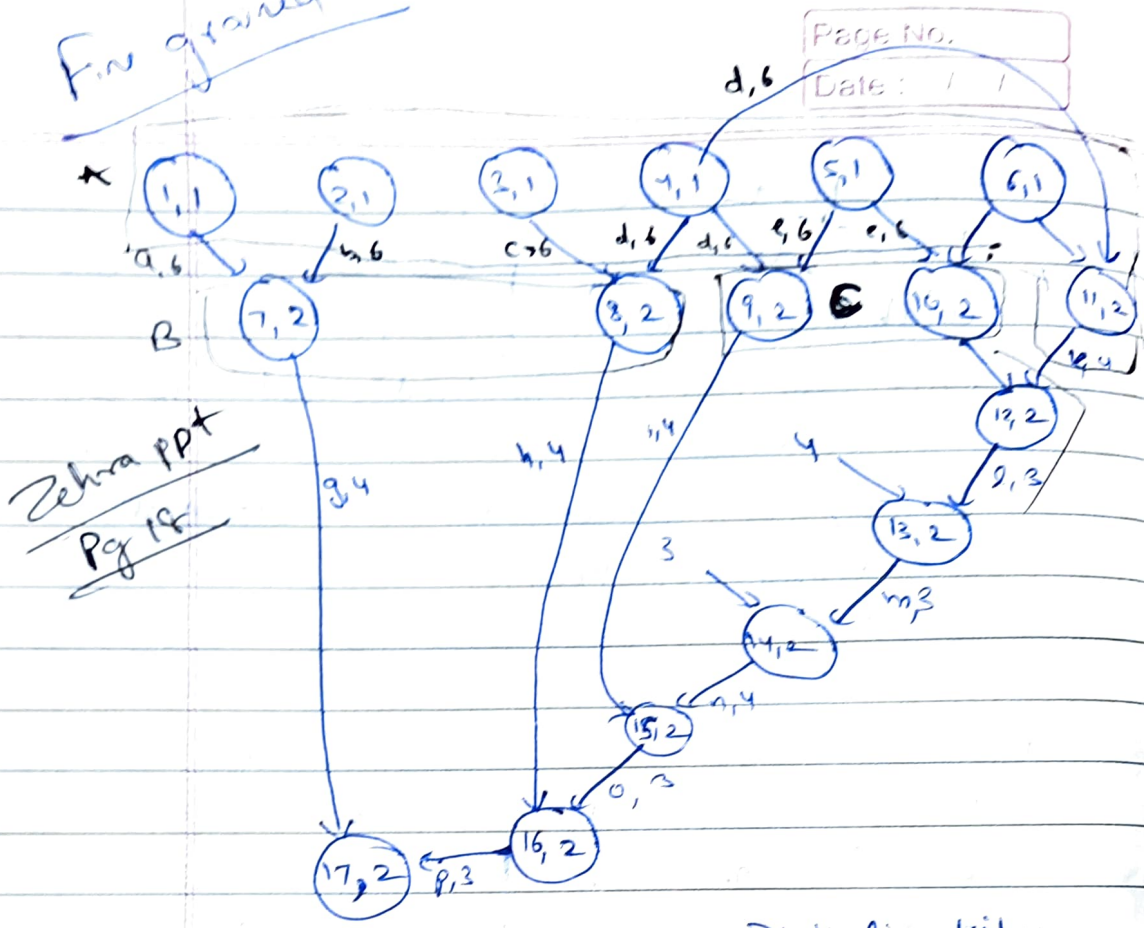
1. a = 1
2. b = 2
3. c = 3
4. d = 4
5. e = 5
6. f = 6
7. g = a x b
8. h = c x d
9. i = d x e
10. j = e x f
11. k = d x f
12. l = j x k
13. n = 4 x l
14. m = 3 x m
15. o = n x i
16. p = o x h
17. q = p x q

Grain cycle: (1,1) → 1 cycle Regra hai (1st inst ko)



P.T.O

Fine grained



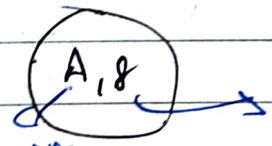
Zebra ppt
Pg 18

(n, s) = (node, grain size) → uske liye kitne cycle lagenge

(x, i) = (input, delay)

(u, k) = (output, delay)

After Packing



Group A → add all the grain