16-9-2020

PDC

→ Distributed = largest data
CC = Cache Coherence = Auto update ↑ All processor if one processor updates
Non Unified = seprate memory
Unified Memory = shared memory

MPI = Bewoff Cluster our VMs.
↳ Distributed Memory.

→ Hybrid Distributed-Shared Memory:
Problems:
Word Count 10TB
Data Find → ∞
Draw Back:
Data Duplication

Semaphores: used to handle raise condition.
↳ Dinning Table.

Tasks:
+, −, /, *...

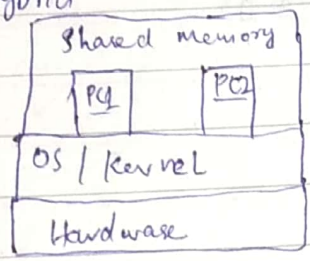
Thread Model:
Har task ko alag alag processor pr divide kr lyg.

GPU:
(blocks, threads)
↓
64 x 64
1 block has 64 threads

CPU → Parallel Processor.
GPU → Massive Parallel Processor.

→ Thread Model Implementations:
Pthread & openMP

→ MPI: Message Passing.
Send C);
Receive ();
Hybrid.



→ Data Parallel Model: (One task i.e) counting)
Any Dataset 100 GB
t1 → 1g * tasks on data.

OpenMP = Hybrid Model.

=> Task Parallelism:
Multiple task on data.

Parallel Programming Platforms.

CH #2

Throughput: data process per unit time.

Scope of parallelism.
- Data Intensive: youtube (video processing).
- Server Applications: video games.
- Scientific Applications: Image processing, Biometric verification.

Types OF Instructions Executions:

1. Sequential Instruction.
2. Parallel " Addition
3. Pine line " Matrix Multiplication
4. Jump (switch case)
5. Branch (If - else)
6. Memory Prediction

functional units = processors.

SUPERSCALER EXECUTION: Fetch decode execute

No functional units utilized = vertical waste

Some _____ = horizontal waste.

VLIW = Very long Instructions word = instructions are packed & dispatched together. = parallel concurrent processors.

↳ limited to 4-way & 8-way parallel execution.

Inorder: $M \times M$ 1st row 1st column

Dynamic: Packets, file sending
chunk order, out of order

$B^+ = 3$ $A^- = 3$
$A = 3.67$
$A^+ = 4.$

Limitations of Memory System Performance:
→ Largely captured by two parameters.

Latecy = delay , Bandwidth = memory access per unit time.

Effective Memory Latecy Time:
→ Using TLB. Cache.

⇒ Multithread
↳ divide program in multiply threads.

⇒ Prefetching for Latency Hiding:
Fetching instructions/data before they are needed.
→ Drawback, wastage of memory.
→ Memory bandwidth

TRADEOFFS OF Multithreading.
→ shared memory
→ Smaller cache residency of each thread.

Interconnection Networks for Parallel Computers:
→ Carry data between processors & to memory.
→ Interconnects are made up of classified as static or dynamic.
(point to point single path) communication

DMA.
DMA → Allocate Memory.

$A = 4.0$ $B^+ = 3.33$ $B^- = 3.33$

$A^- = 3.67$ $A^- = 3.67$

$B^+ = 3.33$ $A^+ = 4.0$ $A^+ = 4.0$

$B^- = 3$ $B^- = 2.67$ $C = 2.7$

Network Topologies: Buses.

Buses. + Direct Connection.

Crossbar:

Many processor communicate each other.

Processor → Memory ⇒ Connection & Communication

Motherboard Interconnectivity (Components of a PC)

↓ Inside a computer systems

→ Not In Parallel

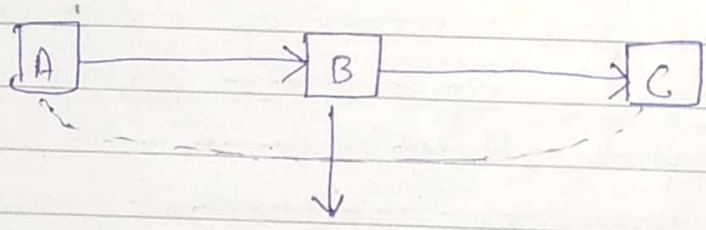
Message Pasing Cost in Parallel Computers:

→ Stertup Time.

→ Per-hop Time. (Network intolution)

→ Per-Word Transfer. (Length of message, error & correction checking)

→ B/w two system. (message passing)



1 · hop delay.

Ab Kalam

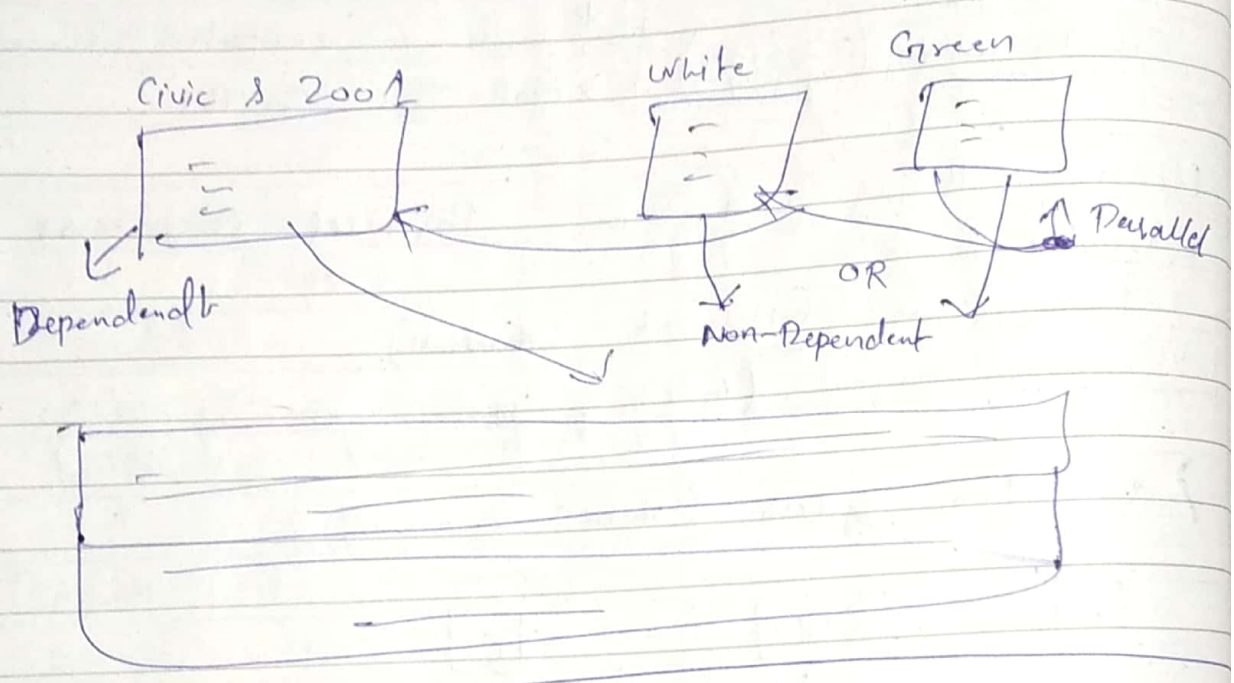
CH #03

Principles of Parallel Algorithm Design.

⇒ Decomposition Tasks & Dependency Graphs:

Task dependent ⇒ Can't be parallel.
↳ Pipeline concept.

Problem discompose to tasks.
(May or may not be of same size.
↳ ex: Vector Dot Product. (No Dependency).
ex: Database searching some result.
↳ Parallel & No Dependendcy.



Granularity:

Fine ← Communication Intensive.
Coarse ← Computation Intensive.

Fine-Grained Decomposition → More communication.
Coarse-Grained Decomposition → More Computation
↳ Matrix multiplication.

⇒ Degree of Concurrency:

No. of tasks executing (going) parallel.

e.g: 4, 5.

AND = Dependent / Not Parallel

OR = Parallel.

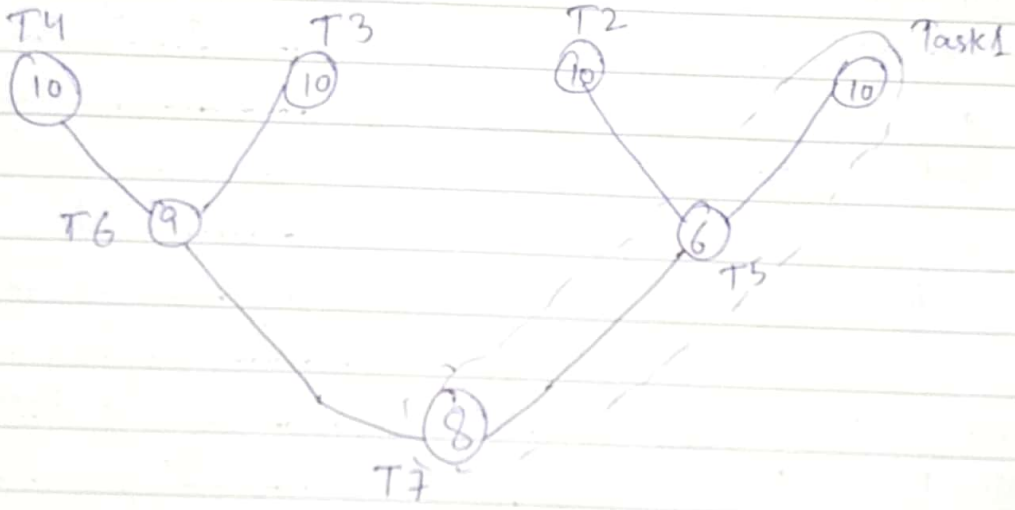
Task Decomposition:

Efficient Parallelism

Critical Path length:

Directed path in the task dependency.

→ Length of longest path in critical task dependency

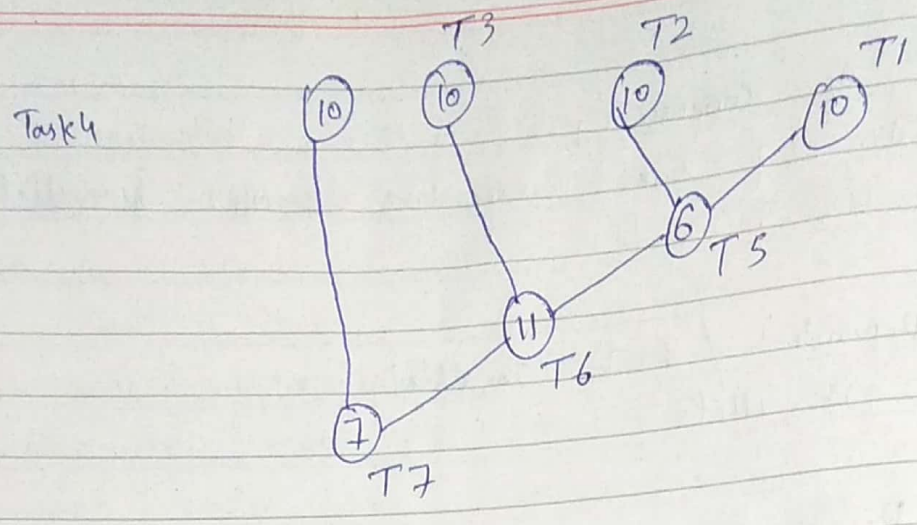


Total amount of work = 10 + 10 + 10 + 10 + 9 + 6 + 8

= 63

critical path length = 27

Avg. degree of concurrency = 63/27 = 2.33



Total amount of work = 64

C.P.L = 34

Avg. degree of concurrency = $64/34 = 1.88$.

Task Interaction GRAPHS: (Data dependence)

→ Subtasks generally exchange data with others in a decomposition.

→ Represent data dependency. to AND in query DBMS is controlled dependency.

Task interaction.

Control Dependency: Pipeline = Branch & Jump

```
ADD  R1, R1, R2
BEQ  R1, R3, Label
```

```
ADD  R2, R3, R4
```

ADD has control dependency on branch

Data / Task Dependency: Parallel = Data Representation.

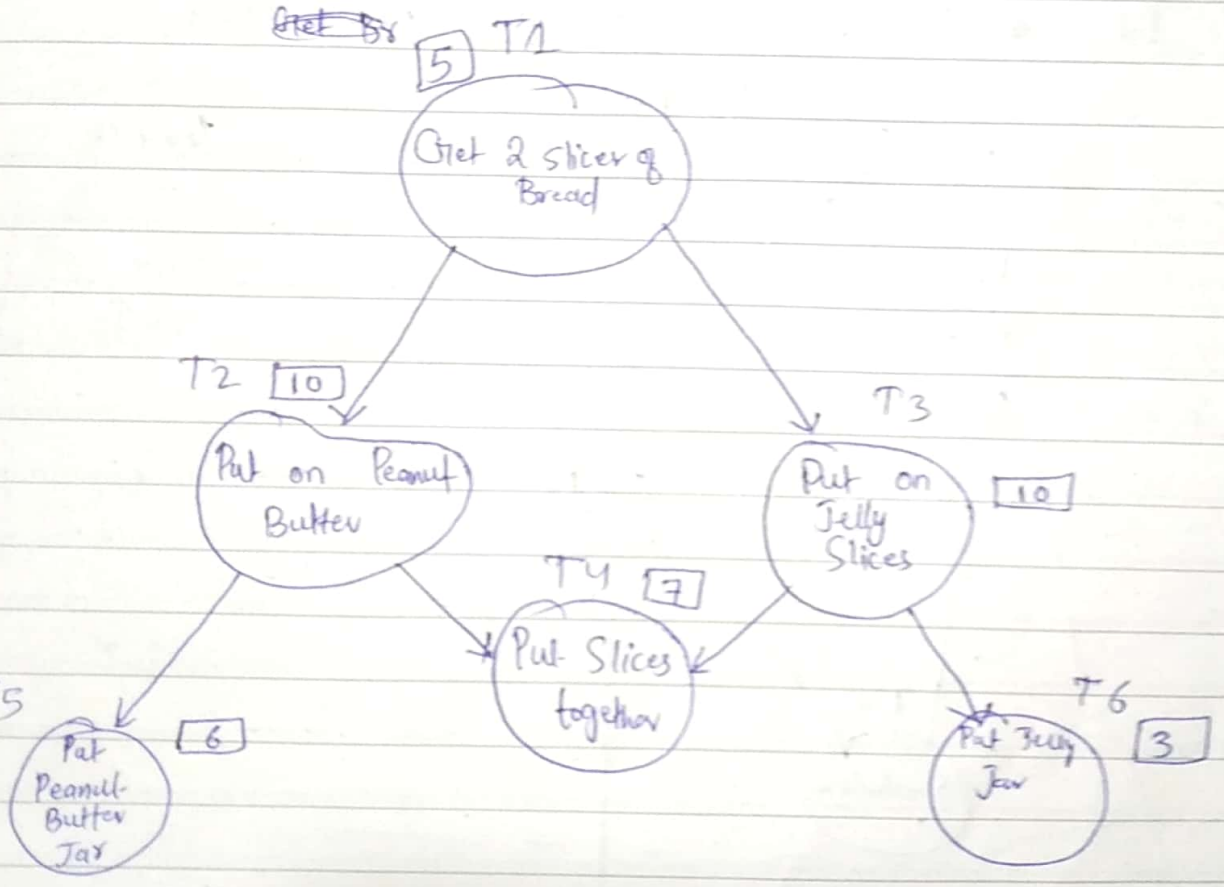
↳ Value doesn't exist until operation is finished. ↳ Data dependency.

```
ADD  R1, R2, R3
SUB  R2, R1, R8
```

Task Dependency Graph

Steps for making butter and jelly sandwitch.
You must first get the bread before spreading the peanut butter Jelly on the bread.
You must ~~first get~~ spread peanut butter before you put away peanut butter and likewise Spread on Jelly before you put away jelly Jar.
And you need to spread or both peanut butter jelly before putting two slices of bread together.
It doesn't matter if you spread peanut butter first or Jelly first.

~~Control~~



Task Dependency = Data dep.
Control Dependency =

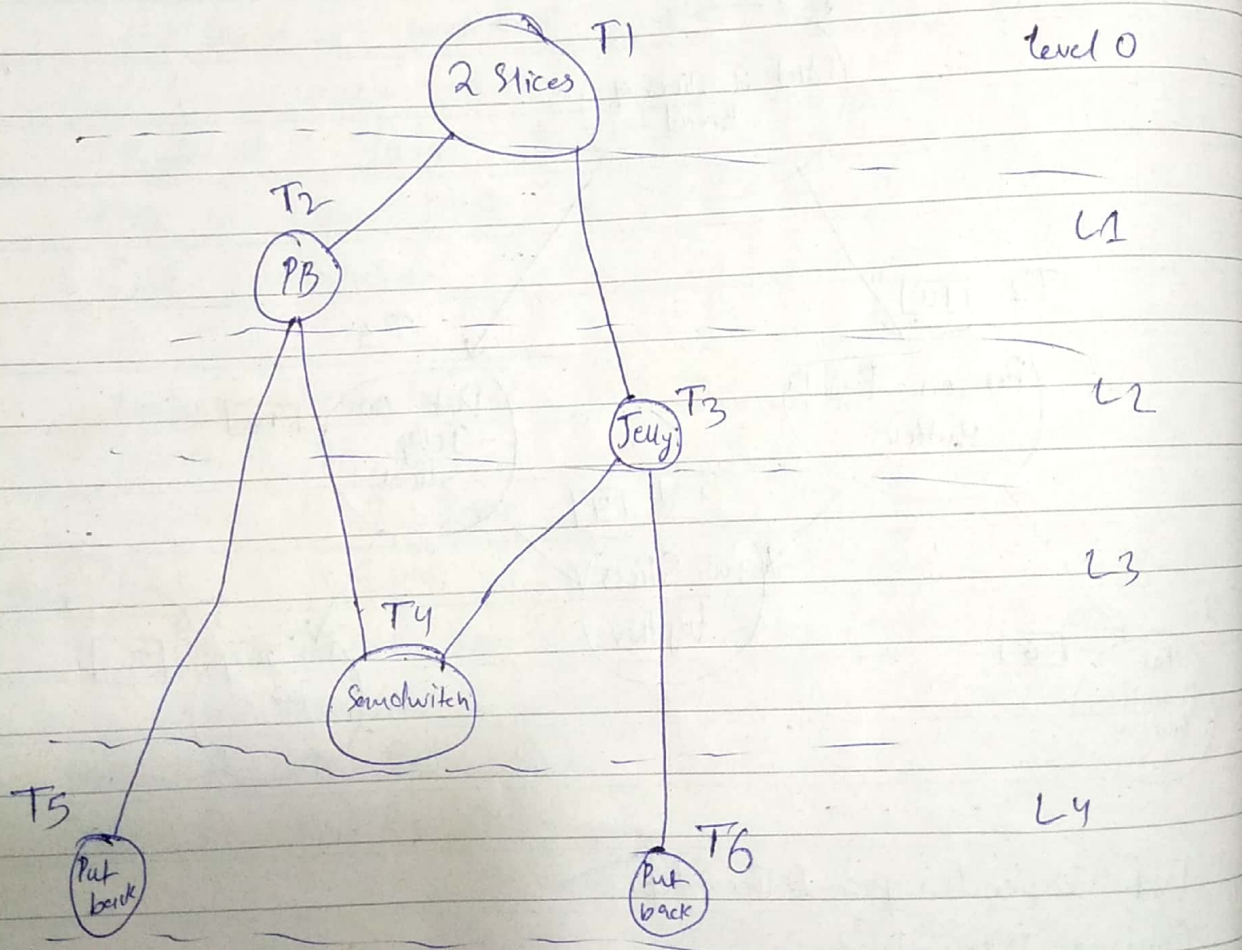
Task Interaction = Data dependency.
Task Dependency = Control dependency / dependencies.

Interactive task = 1, 3, 5
Task Dependency = 2, 3, 4, 5, 6

Total Amount of time = 41
CPL = 22
Degree of Concurrency = $\frac{41}{22}$ = 1.86.

---

ii) Put peanut butter first and then Jelly.



Process & Mapping:

→ Maping are determined by both the task dependency
→ Minimum communication (Course grand → computation)
→ Critical Path to process as soon as they become available.

Minimizing interaction = Coarse grand.
Dense Interaction = Fine grand

Mapping:

① Mapping independant task to different processes.
② Assign C.P to process as soon as they become available.
③ Minimizing interaction b/w processes by mapping with

Decomposition Techniques:

① Recursive Decomposition (Divide & Conquer)
② Data Decomposition (partition data across various tasks)
↳ Matrix Dot Product (MM), DB Transactions.
↳ E.g: Web searching.

↓

Frequency item set Mining
↳ Apriori Algorithm.

CH #07

Programming

Open MP = shared memory space parallel programming
threads = shared
multithreading = shared memory

#pragma omp parallel [clause list]
pragma omp directive [clause list]

Degree of Concurrency:

num-threads (int)
↳ no. of threads created.

Barier = Synchornization = pthread_join.

default (private)

nowait =

default:
first private, private, none, last private.
↓
initilize
to initial
value.

=> Synchronization:
Critical
↳ race condition
↳ Data shared among all processors
↳ Critical section me ek processor
ek waqt me data access karega.

Shared Memory, Distributed Memory, Granularity

ordered:
The structure block is executed in the order = sequential.

Barier:
Each thread waits until all of the other threads of a team have reached this point.

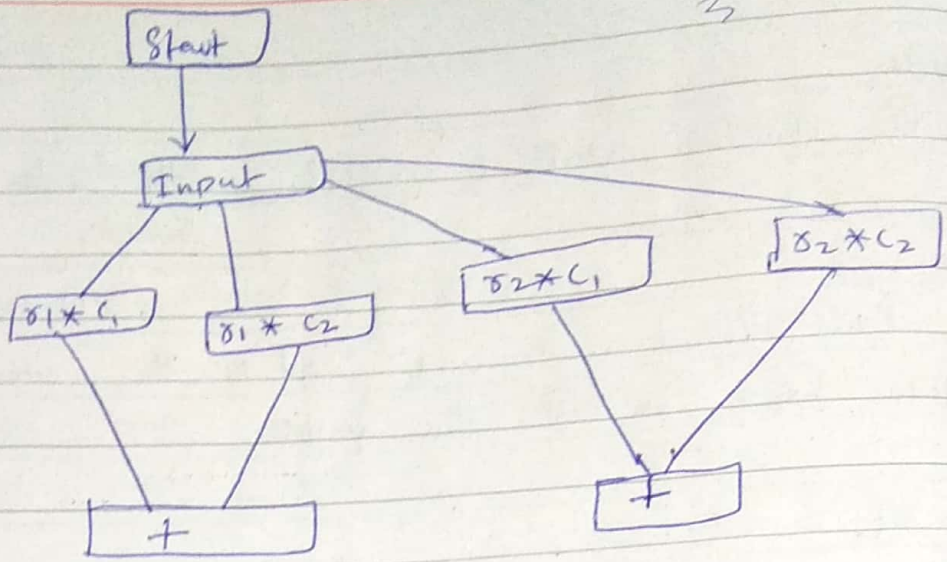
Nowait:
Wait nhi karega jo baqi ~~work~~ threads execute ho rhe hain. Baqi directives a wait nhi karega.

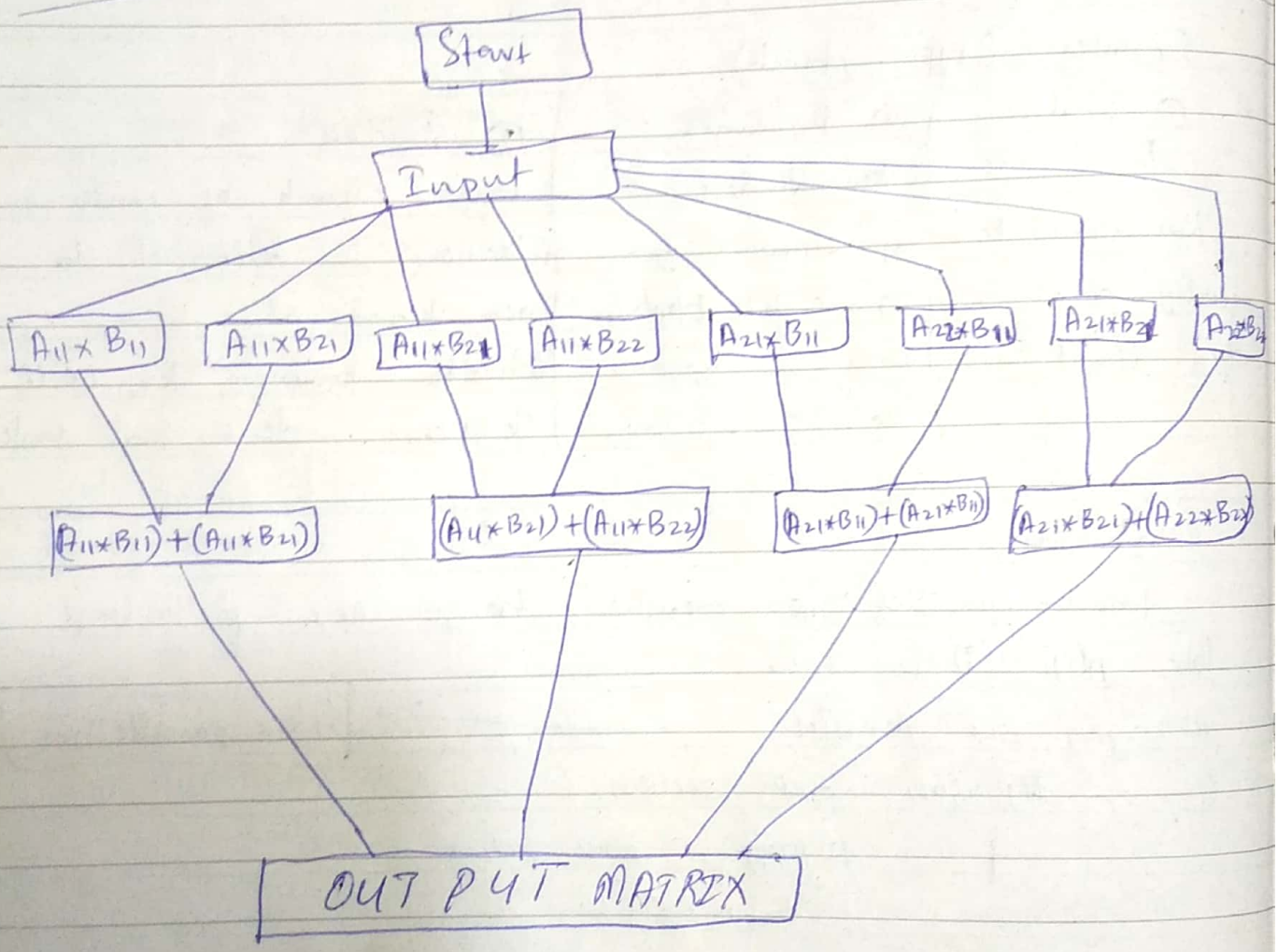
Schedule: (type, chunk)

① Static	② Dynamic	③ Guided
↓	Kuch threads	Large chunk of contigous
Har thread ke	apne create kiye.	iterations are allocated to
equally divide de	Dgla chunks ke dain	each thread at a time.
dya chunks me	taky me asse	Khud he compute krg GPU
	kam kar sakon	k pase dena hai task.

Sections:
Task parallel me execute honge aor for loop lohe nhi (iterations).

#pragma omp parallel ——→ Task parallelism
{ #pragma omp sections
{ #pragma omp sections
{ task A(); }
#Pragma omp sections
{ task B(); }
#pragma omp sections
{ task C(); }
}
}





D[0][0]
0 1

```
main() B[2][2]={0};
{   A[2][2]={0};
    C[2][2]={0};
int mult(x,y)          //Function to multiply
 { return **y }
int sum(x,y)           // Function to sum.
 {return x+y}

 # pragma omp section
# Pragma omp parallel num-thread(8)
      for(i=0; i<4; i++)
          {    # pragma omp for nested_loop
               for(j=0; j<2; j++)
          { C[i][j]= multiplying( A[i][j]B[i][j]) }
          } D[i][j] = sum(C[i][j], C[i+1], [j+1])
     # pragma omp parallel for
     for( k=0 ; k<8; k++)
D[k]  Sum(C[k], C[k+1])
     for
```

Multi programming.

Symmetric

SMP → Shared Memory = Coarse grand
ASMP → Distributed Grand = Fine grand

M x M = [128 x 128]

UMA:
shared assay unified.
↳ sabhko eksath mil jayega.

Hybrid Distributed-Shared Memory:

Word count.
↳ Distributed → ~~Parallel~~ Shared
↳ chunk
↓
results to master.

Coarse & Fine grand.

CH #

Instructions Types
1. Serial
2. Pipeline
3. Parallel
4. Jump
5. Branch
6. Iterative

SUPERSCALAR EXECUTION:

Empty
Partial
Full

1. Horizontal wastage. — Not fully parallel utilized
2. Vertical wastage. — No CPU utilize

Data Dependency.

DECOMPOSITIONS:

① Recursive Decomposition { Divide & Conquer }

② Data Decomposition { Matrix Multiplications }
- Transitions
- Item set example.

③ Exploratory:

Game next stage/step prediction.

e.g: Tik Tak Toe game.

④ Speculative Decomposition: { In order

get up	Roll Back
Get ready	
drive to work	switch dependent instructions
work	execute some or
eat lunch	all cases ~~before~~ in advance

Theory

Scenario → Graph → Code → Open MP

Task Dependency Graph.

Interaction Graph.

Symmetric Vs. Asymmetric Multiprogessing:

Symmetric
↓ Shared memory
↳ bus based access to common physical memory.
↳ In parallel computing it describes a model where parallel tasks have all same picture of memory.
↳ race condition
↳ semaphore

Asymmetric
Not shared memory.
Master → P_1
Slave → P_3
Slave → P_2

Distributed Memory:
No CC.
Communication network
Each processor has it's own memory.
NUMA times → Words count
↳ matrix Multiplication
Downloading a File

Distributed Memory:

In hardware:

Refers to a network based memory access for physical memory that is not common.

In Parallel Programming:

Tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

Synchronization:

In simple words synchronization is waiting for one or more tasks in an applicat to reach the same or logically equivalent point.

✗ Is a overhead / Delays.

Granularity:

Qualitative measure of the ratio of computation to communication.

UMA = Unified Memory Access.

→ Coarse Graun:

Computational extensive.

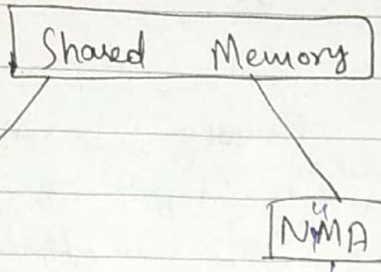
→ Fine Graun:

Communication extensive.

Scalability:

Parallel systems hardware or software ability to demonstrate proportionate increase in parallel speedup with the addition of more processors.

→ Application algorithm.
→ Parallel overhead.
→ Synchronization.

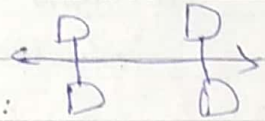


↳ SMP
↳ Identical processors
↳ CC-UMA

CC = Cache Coherence

Equal Access Time

↳ Two or More SMP
↳ Can access another SMP's memory.
↳ Memory access across link is slower
Not equal access time
Slower



Hybrid Distributed-Shared Memory:

→ Cache Coherent Shared Memory Processing.
→ Network communications to move data from one SMP to another.
→ Downloading a file in chunks & combining them.
→ Words Count
→ Matrix Multiplication.

Threads:

Model of parallel programming, a single process can have multiple, concurrent execution paths.

→ Subroutines in main program can be executed
→ Threads communicate with eachother through global memory.

OpenMP

Latency = Dely

Bandwidth = Memory access time / sec

Cache = Low latency high-bandwidth.

Fetch = latency = 200 cls.

Approaches for Hiding Memory Latency:

① Prefetching → fetching data, results before they are actually needed.
② Multithreading → dividing task into multiple subtasks.
③ cache to load faster.

#Pragma omp parallel default (private) shared(a, b, sum) reduction (+: sum)

join all threads each copy of sum to parent. Master thread.

~~ord~~

pragma omp for ____

#pragma omp for nowait

① ordered = executed in order, sequential loop

② Barrier = Synchronization
↳ All threads wait until other
Each thread waits unfill all threads reach at common or logically equavalent point.

③ nowait = threads completing assigned work can proceed without waiting for all threads in the team to finish.

Matrix Multiplication:

```
main()
{
    #pragma omp parallel num_thread('8/2')
    #pragma omp parallel default(private)
    shared (a, b, c, dim)   num_threads(4)

    #pragma omp for schedule(static)
        for(i=0; i<dim; i++)
        { for(j=0; j<dim; j++)
          { c[i][j]=0;
              for(k=0; k<dim; k++)
                c[i][j]+=a[i][k] * b[k][j] }
        }
}
```

↳ assign iteration to threads.
↳ used with loops only.

a[2][2] = $\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$ b[2][2] = $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

i=0

j=0

c[0][0]=0

$\begin{matrix} 0+2 & 2+1 \\ 0+1 & 0+1 \end{matrix} = \begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix}$

k=0 i k k j

c[0][0] += a[0][0] * b[0][0] // 0 += 1*0 = 0

c[0][0] += a[0][1] * b[1][0] // 0 += 2*1 = 2

c[0][1] += a[0][0] * b[0][1] // 0 + 2*1 = 1

c[0][1] += a[0][1] * b[1][1] // 1 + 1*1 = 2

c[1][0]

c[1][1]