Prof. Ghanu A. N.
Assistant Prof.
PVGCOE, Nasik.
Comput Dept
8087777708

UNIT-II

2. Parallel Programming.

* principle of Parallel Algorithms design :-

- An algo. provides step by step solu^n of the given problem. An algo. accepts from the user and based upon i/p. after performing the defined computations provides the ~~fff~~. o/p.
- The basic steps in the design of ||el algo. are :- partitioning of overall Computation into Smaller Computation and assignment of these small ~~component~~ computations into diff. processors.
- Decomposition, tasks & Dependancy graphs :-

* Decomposition :-

- The overall Computation can be partitioned into ~~small~~ number of small size computation so that these comput^n execute in parallel.
- The decomposition deals with the approaches of partitioning the overall comput^n into sub-parts. when a comput^n is divided into many small tasks., it is referred as fine-grained decomposition
- On other hand, the course-grained decompo^n consists a small number of large tasks.

* Tasks :-

- In prog^n, the basic unit of comput^n is refereed as a task. and is controlled by OS.
- In context of ||el prog^n, the task are unit of computations. based upon that the overall comput^n is decomposed.

* Task-Dependancy graph :-

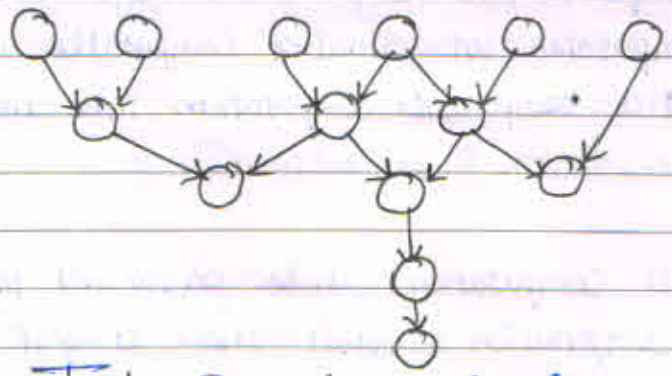
- Is a directed acyclic graph, Typically a graph is a collection of nodes and edges, the task-dependancy graph also contains nodes & edges.
- The node in task-dependancy graph is a task whereas edges bet^n any two nodes represent dependancy bet^n them.

e.g.

There is edge exists bet^n 2 nodes T1 & T2.
If T2 must be executed after T1.

* Consider the task-Dependancy graph and find-out the following :

i) Maximum degree of Concurrency
ii) critical path length.
iii) Total amount of work.
iv) Average degree of concurrency.



Task Dependancy Graph.

→ 1) Max. degree of Concurrency :-
The max^m nos of tasks allowed to execute in 11^{el} is termed as the max^m degree of concurrency.
In the given graph, max^m degree of concurrency is 6.

2) Critical path length :-
- There are 2 types of node included in the task-dependancy graph namely start node and finish node.
- start nodes are nodes with no incoming edges whereas nodes with no outgoing edges are called as finish nodes.
- The critical path in task-dependancy graph is the longest directed path bet^n any pair of start and finish node.
- The quantity referred as critical path length is the sum of weight of nodes on a critical path.
- In the given graph, the critical path length is 5.

3) Total amount of work :-
- Here, the total amount of work is 14 if it is assumed that the each tasks takes one unit of time.

4) Average degree g Concurrency :

- The average degree g concurrency deals with the avg nos g tasks allowed to execute in ||el.

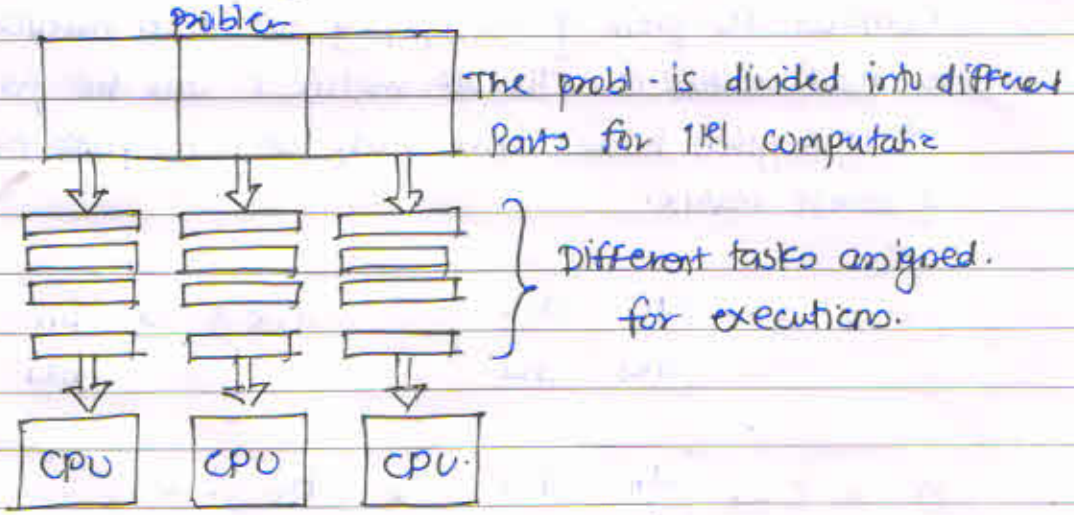
Avg degree g concurrency = Total amount g Work / Critical Path length.

e.g.

Avg degree Concurrency = 14/5 = 2.8.

* processes and Mapping :

- The environment in which a problem is being solved by means g ||el Computation usually has several running proces simultaneously These processes communicate with each other for need g. synch and exchanging g infor.
- The prog. written for such platforms are called ||el progs. & one ||el prog. can have several processes for handling diff. tasks simultaneously.



Parallel Execution of tasks.

* Mapping :

In ||el prog, design, it is required to specify where each task is to execute. As we know proces performs a task. In this approach g context is used for assignment g tasks to proces is called mapping.

* De-composition techniques :

— Data Decomposition :

- The large data sets involved in the problem can be divided into smaller part and process independently by several computer concurrently.
- The data composition is common tech used for Concurrent processing of data.
- There are basically 2 steps in this techniques. In 1st step, the overall I/P data is required for performing the defined computation is divided into multiple part.
- In 2nd step, it is based upon the division of overall Computation into nos of tasks handled on the partitioned data.
- The actual opers implemented on these tasks are similar but data upon which these oper work are different.

Example:

Matrix Multiplication :

Comider the probl. of multiplying two nxn matrices A and B to yield matrix C. The O/P matrix C can be partitioned into 4 task given below. Here each tasks computes one element of result matrix.

$$\text{Matrix } A \rightarrow \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \text{Matrix } B \rightarrow \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$\text{Matrix } C \rightarrow \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$\text{Matrix } C \rightarrow \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{bmatrix}$$

* Recursive Decomposition :-

- As we know, divide & Conquer is one of the strategy of solving the Computational problem. It is based on 2 ideas: 1st approach says to solve prob. directly if it is trivial and in 2nd approach decompose the prob. into smaller prob. if it cannot be solved as it is and solve the smaller prob.
- The recursive decomposition is based on providing Concurrency in problems that can be handled in divide and conquer strategy.
- The prob. to be solved using recursion is divided into multiple sub prob. provided that each of these sub problems is independent each sub prob. can be solved individually by dividing further into other sub problem. & solved using recursion.

In programming a function or procedure calls it-self is called as recursion.

e.g.

```
main ()
  {
  int i ;
    i = 10;
  demo(i);
  }
demo (int count)
  {
   count --;
  printf ("The value of the count is %d \n" ;(count);
  if (count > 0)
   demo (count);
  printf ("The count is %d \n"; count);
```

* Exploratory Decomposition:

- There are situation where the probl. decompos goes hand in hand with it's exe. such problem typically involve the exploration or search of state space of sol.

The search space probl. is divided into smaller parts & each smaller part is searched concurrently till the point at which the expected sol is found.

Example:

- Consider a state space searching problem, such as finding a solution to puzzle problem.
- The steps are:-

1) The comput required for the decomp can be divided into multiple tasks, where each task is searching for a different portion of the search space.
2) The task of finding the shortest path from initial to final config now translates to finding a path from one of these newly generated nodes to final config.
3) The IS Puzzle probl. is typically solved using tree search tech. starting from initial config, all possible successors are generated.
4) In the 15 puzzle problem, there are 15 tiles numbered 1 through 15 are arranged in 4 x 4 grid shown in fig.
   - one tile is left blank, so that moves can be made.

   The 4 possibles moves here are represented as moves up, down, left & right. The start & final config are specified
5) This problem is characterized by objective determining any seq of moves for a shortest sequesses of moves.

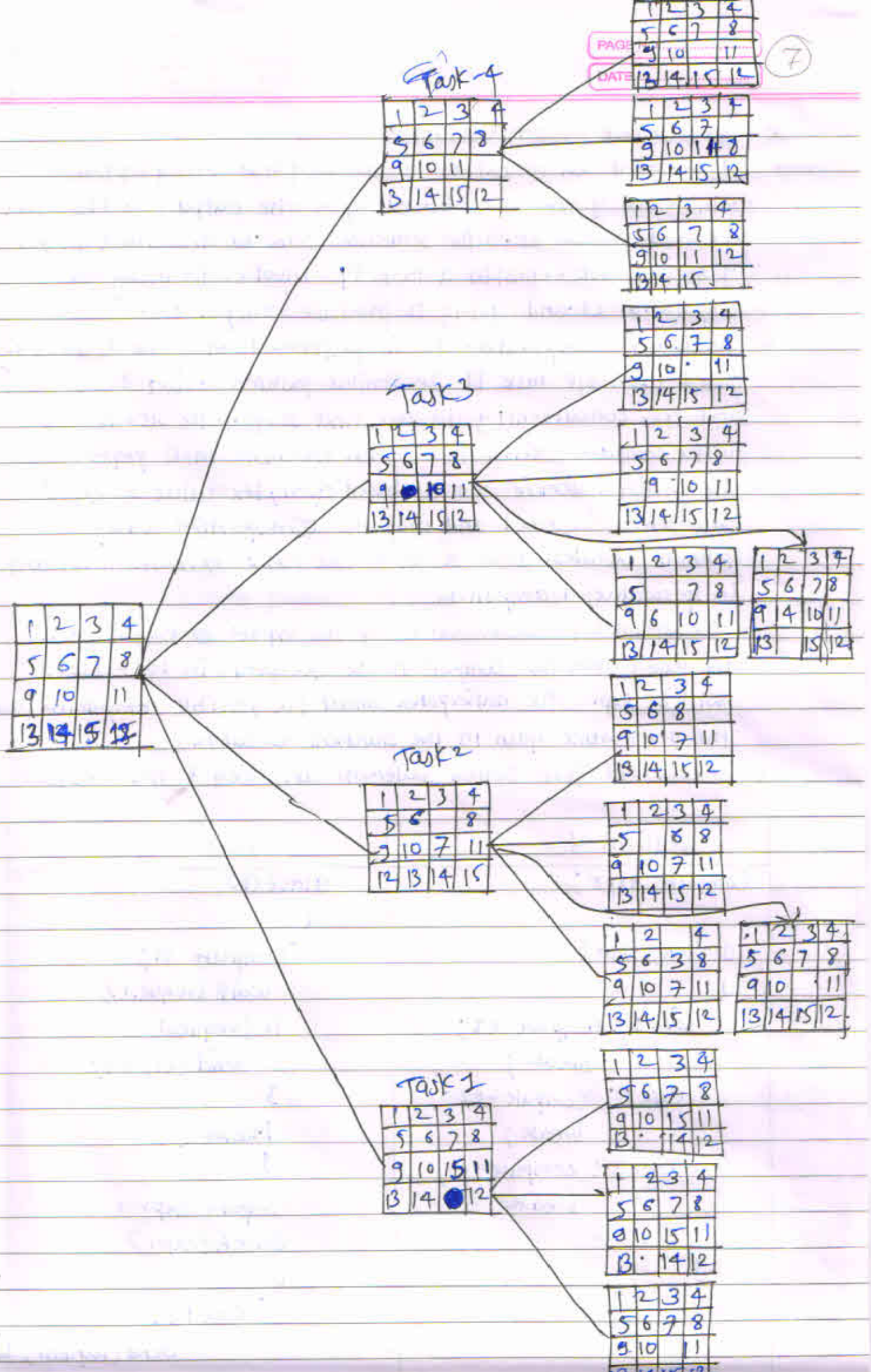
   The sol of the probl. must be searched from an arbitary state

1	2	3	4
5	6	↑	8
9	10	7	11
12	13	14	15

1	2	3	4
5	6	7	8
9	10	←	11
12	13	14	15

1	2	3	4
5	6	7	8
9	10	11	↑
13	14	15	12

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	


Task 4
Task 3
Task 2
Task 1

* Speculative Decomposition :-

— It is used in situation when prog has many options in terms of branches based upon the output of the other part that precedes it. The situation can be described in terms of specific task. Consider a task T1. involves in doing the computation C1 and going to produce output O1.

- The next computation to be performed may be decided by based upon o/p task T1. In similar fashion, other tasks can start exe. concurrently in the next stage. The situation can better compared with the switch statement in C proming.
- The switch statement works based upon, the value of expre. on which it is based and only the corresponding case statement executes. Some or all of the cases executes in advance. in speculative decomposition.
- The situation in which values of the expre is known then the result from the comput to be executed in that case will be kept. The anticipation about the possible computation causes the performance gain in the statement execution.

e.g. Consider foll. Switch statement in seq & ||el version.

Sequential	parallel
Compute expr ;	slave (1)
	{
switch (expr)	Compute ei ;
{	wait (requesr) ;
case 1 : Compute-e1 ;	if (request)
break ;	send (ei, 0) ;
Case 2 : Compute-e2 ;	}
break ;	Master ()
case 3 : compute-e3 ,	{
break ;	Compute expr)
— — —	Switch (expr)
} .	{
	case 1) :
	send (request , 1) ;
	Receive (a1 , i) .
	— —
	}

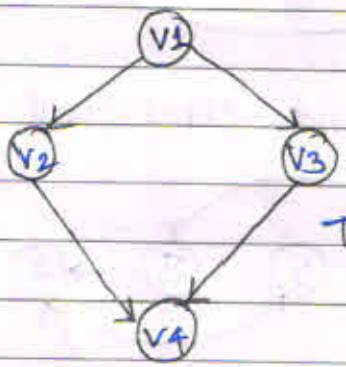
Example ÷ Topological Sorting ÷

- In directed cyclic graph, topological sorting is used to provide an ordering of the vertices.

e.g. two vertices u & v of a graph G. In this graph, when path exists bet' u & v, then v appears after u in the ordering.

- The graph should acyclic property, otherwise for an edge represented as (u,v) there would be a path from u to v. and from v to u, & therefore the ordering cannt be obtained.

- Now, consider there are nos of tasks we need to be performed in which some task depend on the others and it is possible to do only one. The task can be organized in the dependancy graph.



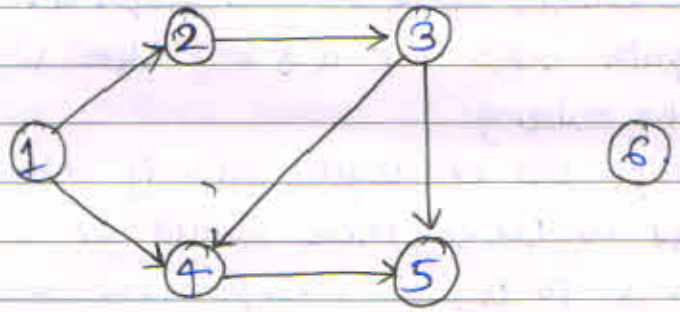
The legal ordering are ÷ V_1, V_2, V_3, V_4 and V_1, V_3, V_2, V_4

The steps of the logical sorting algo. are ÷

1) Initially compute the in-degrees of all the vertices in the graph.
2) Find v as a vertex with degree 0 and store it in the list for ordering.
3) At this stage if no such vertex is detected then there is a cycle found and the algorithms stops.
4) Remove the vertex v and all edges (v,v) it belongs from the graph.
5) At this step, need to update in-degree of the vertices remain
6) Repeat step-2 thro. 4 till the vertices present for procesing.

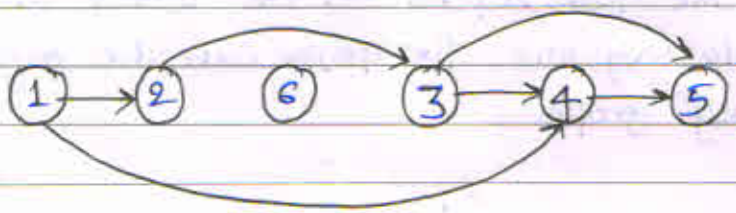
Ex. Topological Sorting :

Consider given directed graph G(V, E) find linear ordering vertices such that : for all edges (v, w) in E, v precedes w in the ordering.

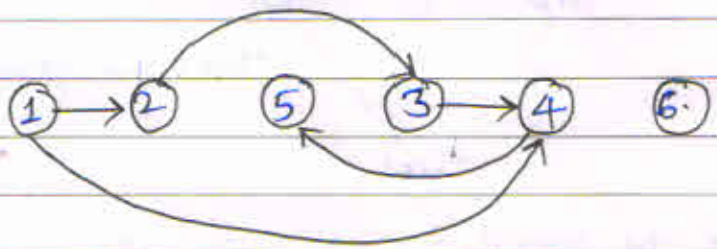


Sol :

→ For topological sorting any linear ordering in which all arrows go to the right · is a valid solution.



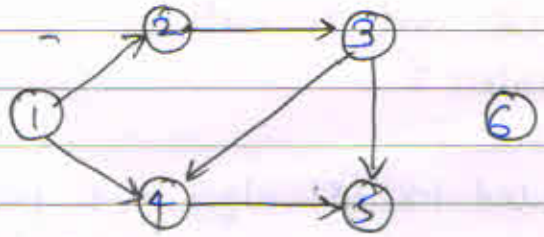
Valid topological sorted graph.



Invalid topological sorted graph

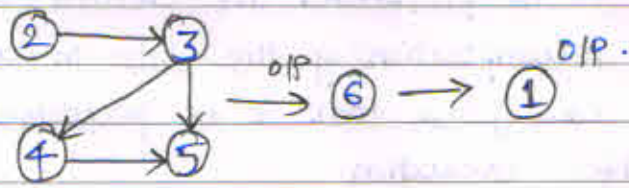
Topological sort Algo :

1) Identify vertices that have no incoming edges (select 1 vertex)
2) Delete the vertex & in-degree 0 and all it's outgoing edges from the graph, place it in the o/p.
3) Repeat step 1 and step-2, until graph is empty.

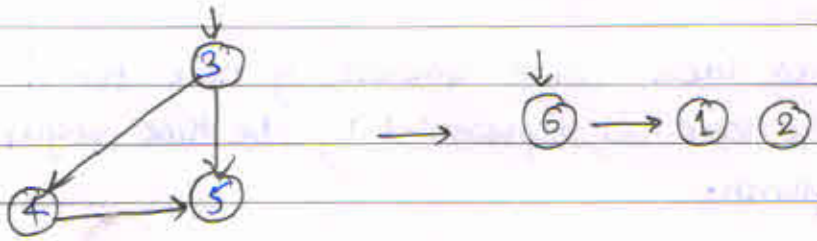


step 1 :- Identify vertices that have no incoming edges

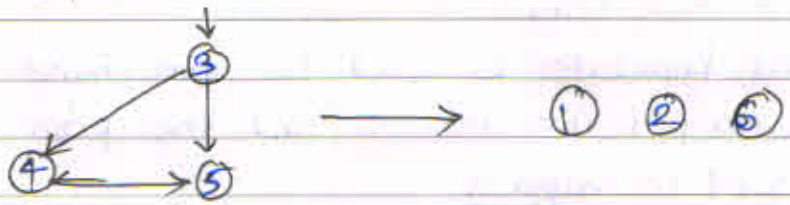
step 2 :- Delete this vertex from graph.



Repeat step 1 & step-2 until graph is empty.



Repeat step 1 & 2 until graph is empty.



Repeat step 1 & 2 until graph is empty.

Finally the topological sorting is Shaun hex

* Characteristics of tasks and Interaction :-

1) Task characteristics :-

→ • Generation of Task :-

The task included in the algo. are possible to generate either on prior basis termed as static / dynamic.

* Static task generation :-

The task are defined before starting the exe. of the algo. and specified order is to be followed by the algo. in exe.

e.g. matrix multiplication

* Dynamic task generation :-

- The task to be performed are created dynamically based upon the decomposition of the data in certain situation. In these case, the task to be performed are not available before algo. execution.

e.g. recursive & Exploratory algo.

* Size of task :-

- Each tasks takes some amount of time for its completion. The size of task is represented by the time required for it's completion.

* Knowledge of task sizes :-

- The task size knowledge is used for the choice of mapping. The task are mapped into process and the prior knowledge of the task used in mapping.

e.g. when dynamic decision are required during processing.

* Data sizes :-

- The required data for performing a task should be made available. when mapping it into processes.

- The overhead associated with the data movement can be reduced when the size of data as well as it's memory locations are available.

* Mapping Techniques :

- The mapping tech is used to map tasks into processes so that ||el exe. can be performed.
- The overall computation associated in problem to be solved. divided into nos g task.
- These task are mapped onto processes with the goal g. completing exe. in the minimum amount g time.

There are 2 techniques :

1) Static mapping 2) Dynamic mapping.

1) Static Mapping -

In this tech, the mapping g task onto procen is performed before exe. g algo. This indicates that tasks are distributed among available processes prior to exe. g algo.

- scheme for static mapping :

There are 3 diff. scheme 1) Mapping Based on Data partitioning,
2) Task graph partitioning.
3) Hybrid strategies.

2) Dynamic mapping :

- In this tech, the mapping g task onto the processes. is performed during the exe. g the algorithm. This indicate, the tasks are distributed among available processes when algo. actualy executes.
- There are basic situation where dynamic mapping is applied, The 1st case is if tasks are generated dynamically then this mapping techniques is used.
- Dynamic tech is applied in situation where large amount g data is associated with tasks. In this situation, dynamic mapping moves data among available processes.

There are 2 diff. classes g dynamic mapping.

- 1) Centralized dynamic mapping
2) Distributed dynamic mapping.

* parallel Algorithm model :-

" It is used to describe the strategy for partitioning data and how these data are processes.

A model is used to provide appropriate structuring of ||el algo. on the basis of 2 tech.

1) selection of partitioning & mapping tech
2) proper use of strategy for interaction minimization.

- There are basically six diff. types of model.
  1) Data ||el model
  2) The task graph model
  3) The task pool model
  4) Master/slave model.
  5) pipeline / producer-consumer problem
  6) Hybrid model.

1) Data ||el model :-

- In this, once the data have been distributed ||el operations can be performed. In this way, the oper performed by each task are similar but the data on which these oper works are different.
- The overall task to be performed may be divided into diff phases. The data upon which the tasks work in diff phases may be diff.
- The 2 popular paradigm Share-address space and msg-passing can be used for implem of data-||el algo.
- The ||el matrix multiplication is example of data-||el computation

2) The task graph model :-

- As we have already studied task dependancy graph is used to describe computation in any ||el algo
- The task-dependancy graph explicitly used in mapping of some algo.
- The data interaction cost among the task can be reduced by the use of interrelationship among the task in task dependancy graph
- The data movement cost is optimized by the use of mapping of task to procesn statically.
- The ||elism described with task dependancy graph where each task is an independent tasks is called as task ||elism.
- e.g. parallel quick-sort algo.

3] Task pool model :

→ The task pool model is also called as work pool model. It uses dynamic mapping approach for task assignments.

- In this, centralized or decentralized mapping can be adopted.
- The task-pool model can be used in msg-paning paradigm. when data anociated with the tasks is smaller than computation anociated with the tasks.
- e.g. loop Helization using chunk scheduling, Hel tree search etc

4] Master-slave model :

- In this, one processor is designated as master & other as slave. and master is responsible to coordinate the activities among all slave processes.
- In master, it generates work to be performed & assign it to the nos of workers / slaves processes.
  The alloc of task is depend upon size of task.
- In this, if work allocated to slave, the it complet by slave & slave is allocated to other jobs. it performs diff task.
- master is responsible to synchronize the activities of the slaves after each phase.
- The master-slave model is generally suitable for shared addren space & msg paning paradigm.

5] The pipeline or producer-consumer model :

- This model is based on the paning on stream of data thro. processes arranged in a succesion.
  The data flow thro succesive processes & at that each process does some oper on it. This is called stream Helism.
- Whenever, a new data arrives a new task execution initiates by the proces in the pipeline.
  This model is called as producer-consumer model.
- bcz pipeline acts as chain of producers & consumers.

* parallel programming with CUDA.

- It is based on the coverage of diff. activities required to perform while handling ||el computing using CUDA.

Procesor Architecture :

- The tesla architecture is used for construc of the CUDA capable GPUs. The appli written can be executed on any card which support this architecture.

Each GPU has slightly diff. features hence of diff specificate.

- When kernel is invoked, each thread blocks executes on a 'multi-procesor' this multiprocesor contains the resources to support certain no of threads.
- Each multiprocesor consist of
  1) Scalar procesor cores
  2) 2 special fn unit for transcendental
  3) 1 multithreaded instn set
  4) On chip shared memory.
- One/more thread blocks are assigned to multiprocesor during exe of a kernel. The CUDA runtime handles the dynamic schedulling of thread block on a group of multiprocesor.
- The schduler will only assign thread block to multiprocesor when enough resources are available to support the thread block.
- Each block is split into SIMD (single instn multiple data) group of threads called 'Warps'
- The SIMD unit creates, manages, schedules and executes 32 threads simultaneously to create a warp.
- Every warp is synchronous, and therefore care must be taken to ensure that certain threads within warp do not take abnormally longer compared to other threads in that same warp., hence warp will only execute as fast as the slowest thread.

* Interconnect :

The multiple GPU systems are designed for following 2 reasons :

1) problem Domain size :

In the case, where data sets are too large and unable to fit into the memory of a single GPU.

2) Throughput and efficiency :

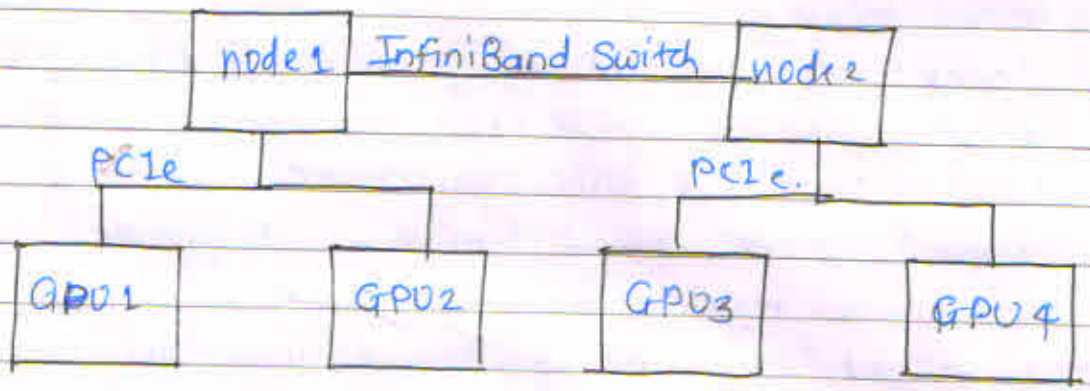
The throughput of an appl? can be increased by processining it in the multiple GPU concurrently.

- The inter-GPU comm is required to design properly while converting an appl? into multiple GPU system.
- The efficiency of data transfer among the GPU depends on how GPUs are connected together within node or across a cluster.

There are 2 types of connectivity in Multi-GPU system.

* Multiple GPU connected over the PCIe bus in a single node.
* multiple GPU connected over the n/w switch in a cluster.

Fig shows topology for cluster with two computing nodes.



- GPU1 and GPU2 are connected via PCIe Bus on node1.
- GPU3 and GPU4 are connected via PCIe Bus on node2.
- These two nodes (node1 & node2) are connected to each other through InfiniBand Switch.

* Communication :

- The comm in platform is in term of peer-to-peer comm. The CUDA peer-to-peer API is used to enable direct inter-device comm.
- The CUDA 4.0 or higher version is required for peer to peer comm. The CUDA P2P API supports two modes that allow direct comm bet GPUs :

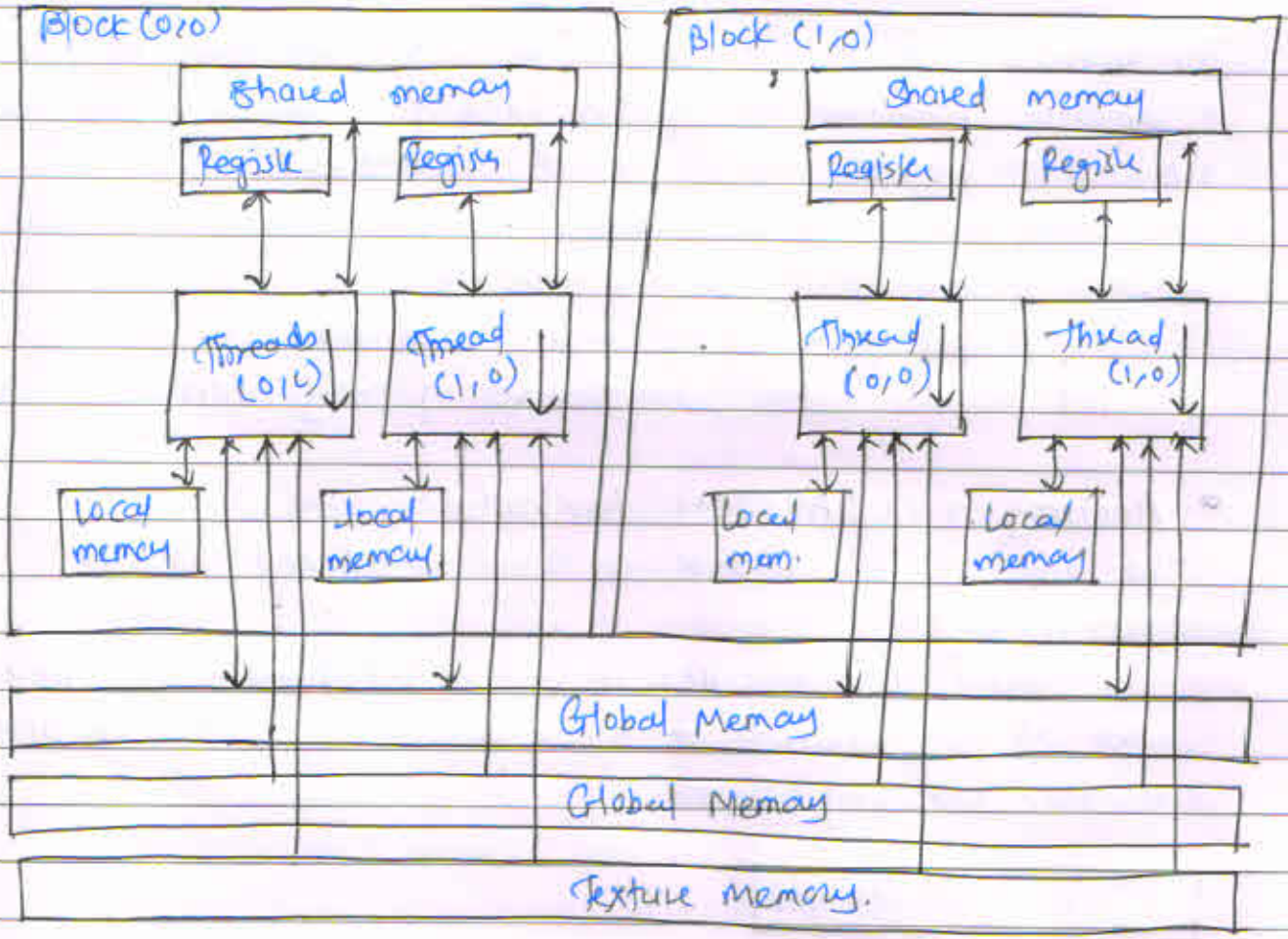
* peer to peer access :
  Directly load & store addresses within a CUDA kernel and across GPUs.

* peer to peer transfer :
  Directly copy data bet GPUs.

- The direct peer to peer access is not supported when two GPUs are connected to diff root nodes within a system.

* Memory Organizations :

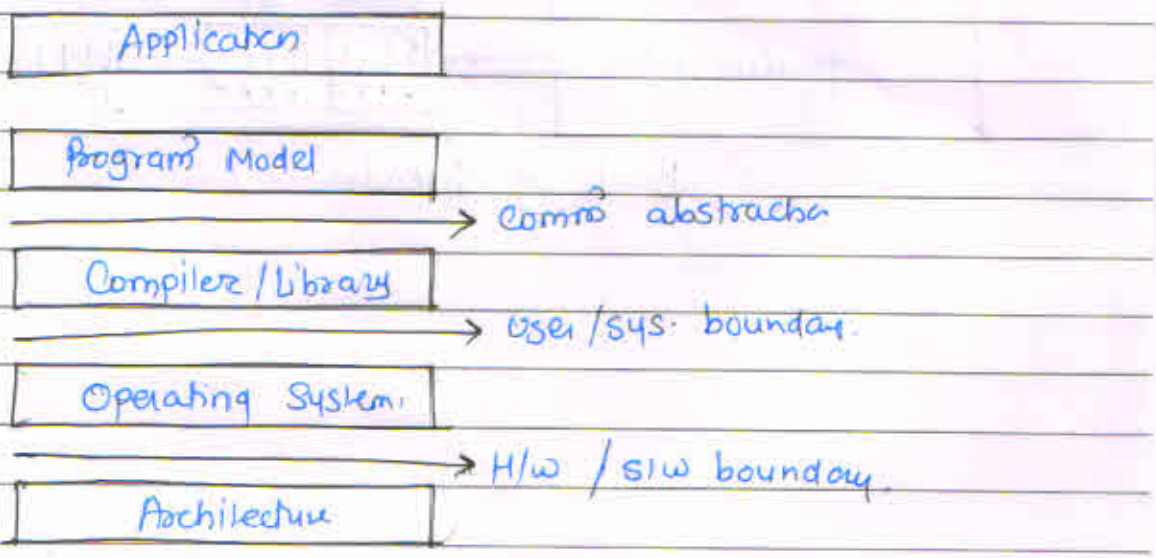
- CUDA uses segmented memory architecture that allow appli to access data in global, local, shared, constant and texture memory.
- There are several level of memory & memory org are arranged in the hierarchical fashion and each level has distinct Read and write characteristics.
- Every primitive thread has access to private "local" memory as well as registers. Every thread in thread block has access
- to unified 'shared memory', shared among all threads for the life of that thread block. Finally all thread have read/write access to global memory.
- There also exist a read-only 'constant' & texture memory in the same loc at global memory.
- Global memory is not cached, thro mem transactions may be 'coalesced' to hide the high memory access latency.
- The read only constant memory resides in the same location as global memory. this mem may be cached.
- The read only texture mem. resides in same loc of global mem. and is also cached.
-



* CUDA programming models for High performance Computing architecture :

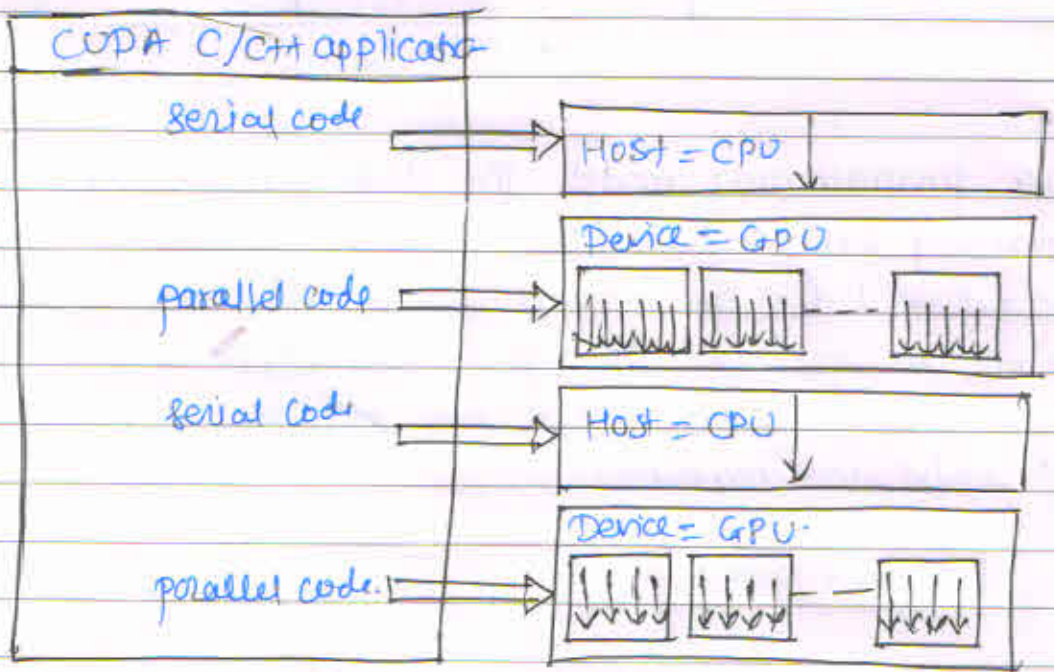
→ The model used for prg in particular archite. acts as a bridge bet an appli and its implem on available h/w.

- Fig. shows layer of abstraction lies bet the prog. and prg model implementation.



- The boundary bet prog & prog model impln is comm abstraction. A compiler / libraries using sys. calls to access hw services. are used to provide such a comm abstraction.
- The components of the prog share info and coordinates their activities by the instn provided thro prog.
- The CUDA prg model share many abstraction with other. ||el prg models. it has some features of fully utilization of GPU.

- Anatomy of CUDA C/C++ Application:
- The appln for the archite contains serial and ||el code shown in fig.
- The serial code executes in a host (CPU) threads whereas the ||el code executes in many devices (GPU) threads across multiple procesing elements.



Block of threads.

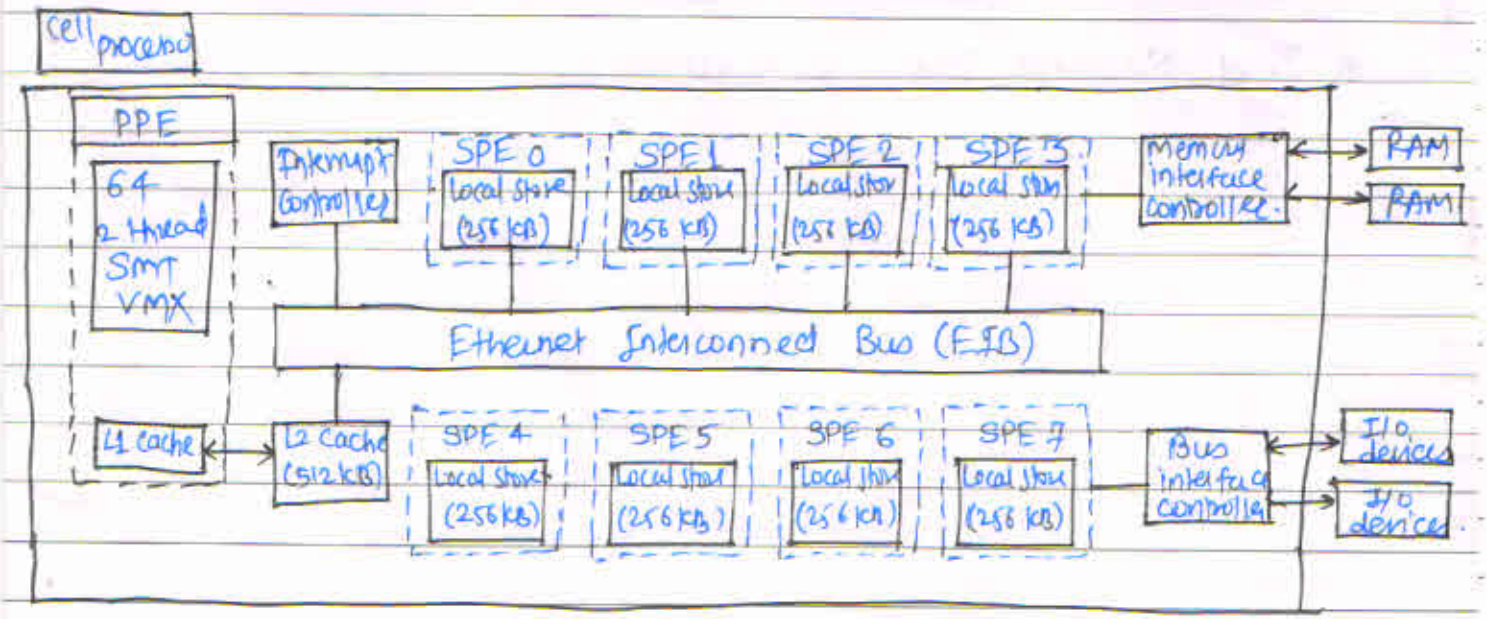
DATE ___/___/___

* IBM Cell Broadband Engine (CBE) :-

→ - The new procesor created by IBM with the collaboration of sony & Toshiba is called as cell.

It is combination of 64-bit dual threaded IBM PowerPC. procesor and eight nos of 'Synergistic Procesing Elements' (SPEs)

- The IBM cell broadband engine (CBE) is basically heterogeneous multicore chip. It is comb of 2 types of procesors.
- The procesor PPE (PowerPC Procesing Element) and SPE are two diff procesor combined in CBE.
- There are 8 nos of SPEs and one PPE included in CBE system.
- The other component of CBE are :-
  one high speed memory controller, one high-bandwidth element interconnect bus, high speed memory & I/O interfaces.
- All components along with 2 procesor are integrated on a chip. In this way, it is considered as hybrid nine-core procesor.
- The procesor included in CME have 2 diff instruction sets as compare to same instruction-sets supported by most of the CMPs.
- The procesor element in the CBE are optimized for certain types of operation but these element can be used for general purpose computing also.
- The IBM CBE is commonly used as a procesor for sony's play station 3 system.



* Nvidia Tesla GPU :-

→ - GPU (graphics processing Unit) is basically used for 3-D applicahcn. A GPU is single chip procesor used to provide lighting effects. and transforms objects every time a 3-D ~~scene~~ scene is redrawn..

- The GPU is used to assist CPU on mathematically-intesive tasks. require in performing graphics related activities. The CPU is freed from the overheads occurred from graphics related mathemahc-intensive tasks and the free cycles of CPU can be used for other tasks.
- The GPU is used along with the CPU to accelerate scientific, analytics, engineering etc. appli's. The overall computing is accelerated by the GPU and therefore this filed of computing is called GPU accelerated Computing.
- The basic difference bet' a CPU & GPU is the use of nos of cores included. The CPU contains few nos of cores for sequential serial procesing whereas a GPU contains more than thousands of smaller and efficients cores for handling multiple tasks simultaneously.
- Nvidia's brand name for the product, they develop for stream processing and general purpose GPUs is called as Nvidia Tesla.
- The GPUs from G80 series onwards are utilized in the Nvidia Tesla Products.

* Intel Nehalem Micro-architecture :-

→

DATE ...../...../.....

* Intel Nehalem Micro-architecture :

→ - It is one of the successors of the intel's core architecture. Intel core architecture uses multiple cores in single die for improvement of performance over single core architecture.

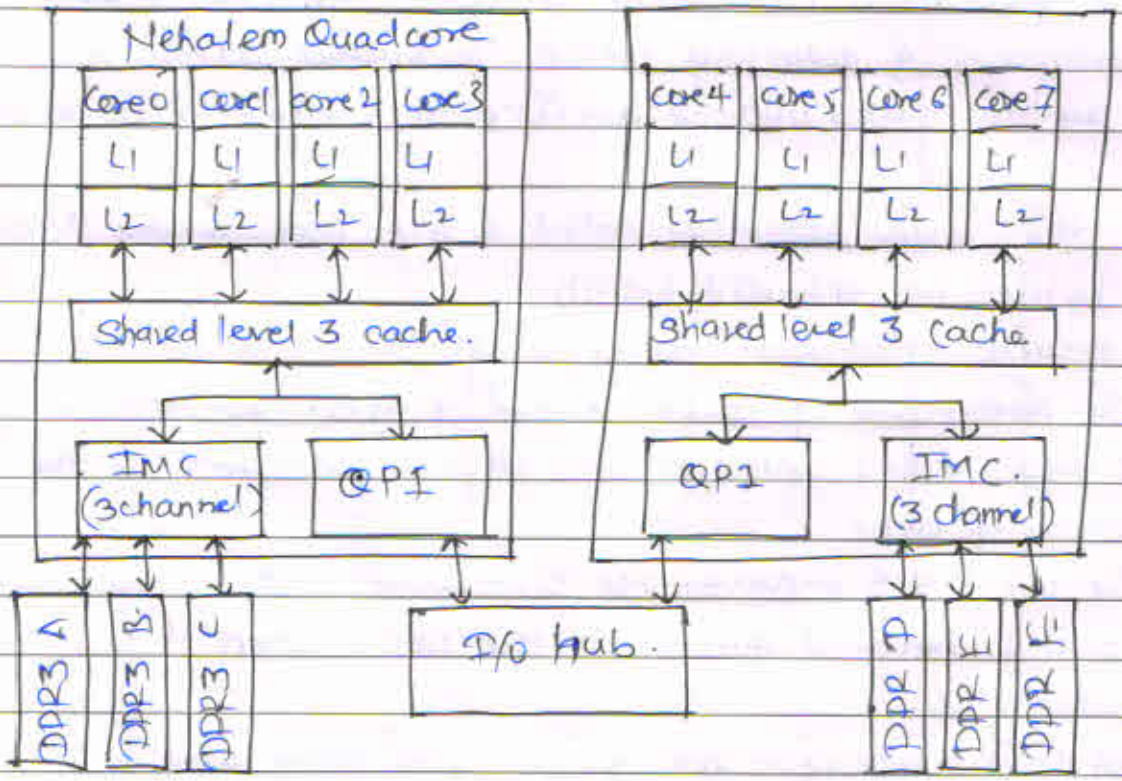
- The intel's Nehalem archite. provides improvements in core-to-core comm^n by the use of point to point topology.

The use of point to point topology provides direct comm^n ~~bet~~ among the microprocessor cores and the sys. mem. can be accessed in more diff. ways.

1) Architecture Approach in Nehalem -

- The nehalem archi. approach is much more modular as compare to the core archit. The modular archit. in nehalem makes it much more flexible. & customizable to the appli^n.
- The main blocks in Nehalem includes microprocessor core. (contain own L2 cache), a shared L3 cache, a bus controller, a graphics core and IMC (integrated mem. controller)

QPI - quick path interconnect



- fig shows e.g. of eight-core Nehalem processor in which two QPI bus controller included.

This config^n is used in the processor attached with intel Nehalem Multiprocessor system.

DATE ..../..../....

2) Branch prediction :-
- The loop in code exe are detected by loop stream detector designed specifically for the core architecture.
- The loop included in code is detected during its exe. & instru are stored in the special buffer so that it is not required to fetch it continuously from the cache. In this way the branch prediction success rate is increased for loops and overall performance improvement is achieved.

3) Out of order exe :-
- The purpose of out of order exe is to minimize pipeline efficiency by filling up the pipeline stalls with the useful instru.
- It was supported by core archite. also but in the Nehalem imple approach used to allow more instru to be ready for immediate exe.

4) Instru set :-
- It has seven new instru added by the intel to take advantages of data level ||ism. It support SIMD. facility.
- Due to SIMD, intensive appli (i.e. Multimedia) can be performed efficiently.
- Newly added instru is called as ATA (Appli targeted Acceleration) by intel bcz enhanced facility.

5) Transacha lookaside buffer (TLB) :-
- The performance of cache depend of TLB. The TLB is used to map virtual addresses into physical addresses in the memory or cache.
- The use of TLB improves the performance. in the sense that a page is accessed quicky in the cache when it is mapped in the TLB.

6) Cache & coherency :- (Note : you can write theory about it)

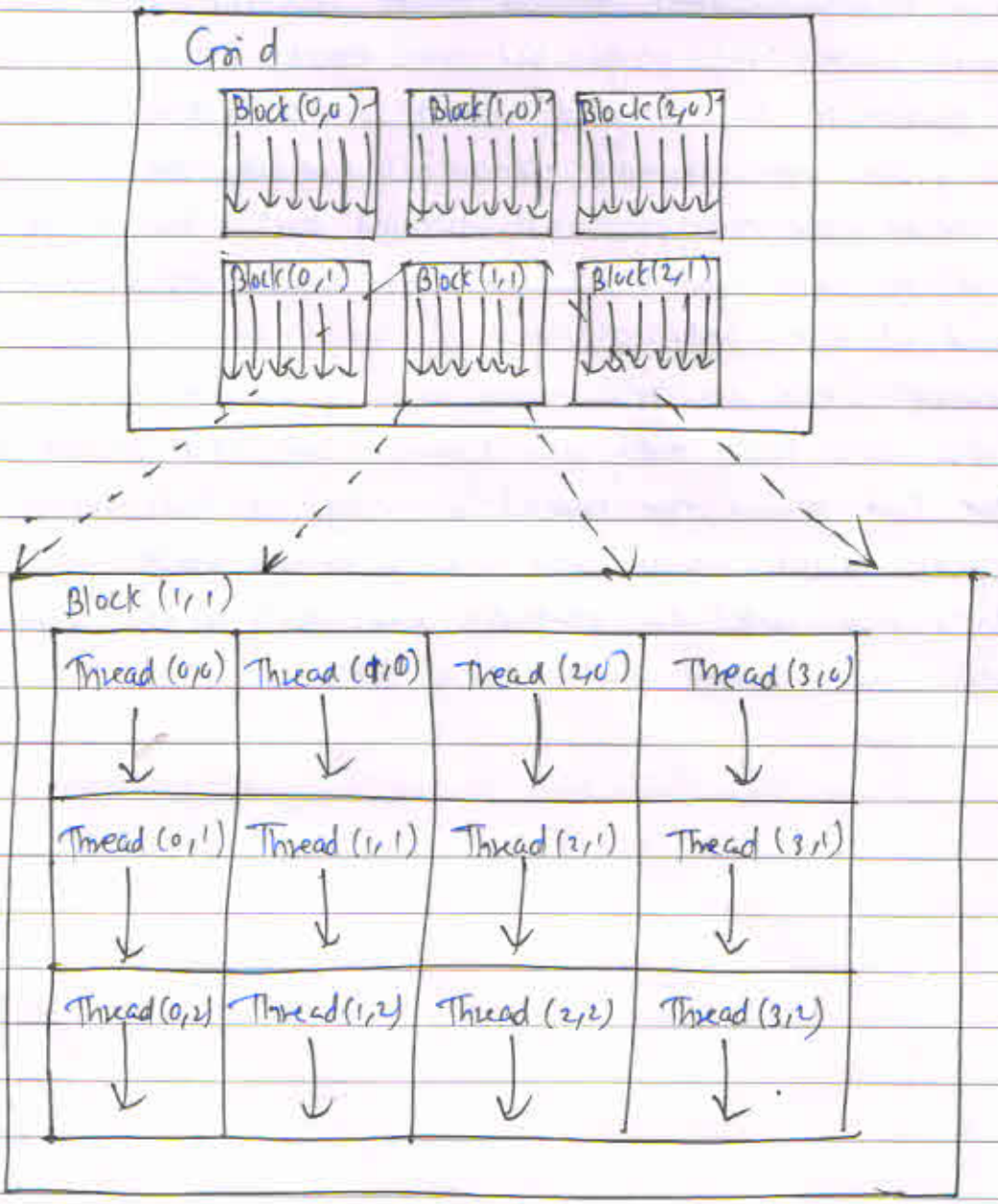
7) Memory Controller :-
- It is located at diff loc as compare to the core prosena. Nehalem archi. integrates mem. controller to the prosena or in the die instead of the off-chip on motherboard.
- due to off-chip mem. controller, it becomes totally free & independent of h/w.
- It provides improvement by running as fast on h/w platform.

DATE ...../...../.....

* Thread Organization :

- The CODA processing paradigm uses an approach is called kernel. A kernel is actually a sub-routine or mini-prog.
- The device like NVIDIA graphics card used inside the host sys. runs kernel.

The kernel prog. is executed simultaneously by nos g primitives thread.



- There are no g batches g primitives threads organized into thread blocks. A thread block contains a specific nos g primitives threads, chosen based on the amount g available shared memory as well as mem. latency hiding tech characteristics desired.

- The nos of thread in thread block is also limited by archi. to total of 512 thread per block. Each thread within thread block can communicate efficiently using shared memory scoped to each thread block.
  Using this shared mem, all thread can also synch. within thread block.
- Every thread within thread block has its own thread ID. Thread blocks are organized into 1D, 2D or 3D array.
- As shown in fig. a grid which is collection of thread block of the same thread which all executes the same kernel. The thread block are physically limited to 512 threads per block. & bcz of that grids are used for computing a large nos of thread block in parallel.
- Thread block in grid may not synchronize with one another bcz they may not communicate via shared memory.
- The diag shows the thread hierarchy. In this represent a given kernel contain 3×2 grid of thread block.
- There are total 72 threads executing in the said kernel where each thread block is a 4×3.