HPC

Unit 1

Parallel Computing

- Parallel computing is one of the type of comutation where multiple tasks are execute simulatneously at the same time.
- In parallel computing the larger task is divieded into smaller subtasks and then that subtasks are executed concurrentyly.
- Advantges:
  - saves time and cost
  - solve larger problemes
  - increase resource utilization

Parallel Programming Platforms

- **Pipelining Execution:**
  - In unpiplined system first the instruction is fetched and then its data is fetched from registerd the the instruction is executed the result is written back into the register. In this system architecture most of the parts of processots are in waiting state and they are not utilized propertly.
  - In pipline execution the instruction flows from the processor as if the processor was a pipe. In this architecture the instruction completes its exeuction in different stages. Because of that multiple parts of the processor are busy in handling some instruction and all the hardware units are utilized parallely. Example is two stages pipelined system.
    Two Stage Piplined system:
    - Fetch Unit
    - Execution Unit

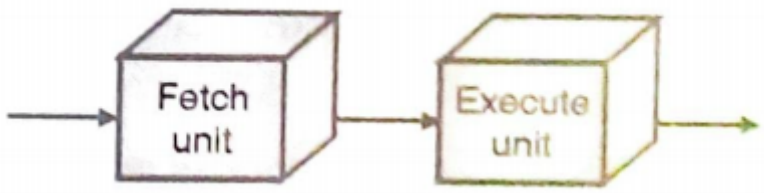


Fig. 1.4.1 : Two stage pipelining

- **Superscaler Execution:**
  - In pipleined system there is only one pipleined unit, one for integer operations and one for floating point operations. This is scaler architecture. But having more pipelined units improves the degree of parallism and this is superscaler architecture. In superscaler architecture multiple instructions are presen in the differnet stages for execution at the same time.
  - In superscaler architecture more than one instrution is executed at a time during single clock cycle.
  - The Superscaler architecture exploits the potential of ILP (Instruction Level Parallalism)
  - Problems: Hardware cost and design complexity

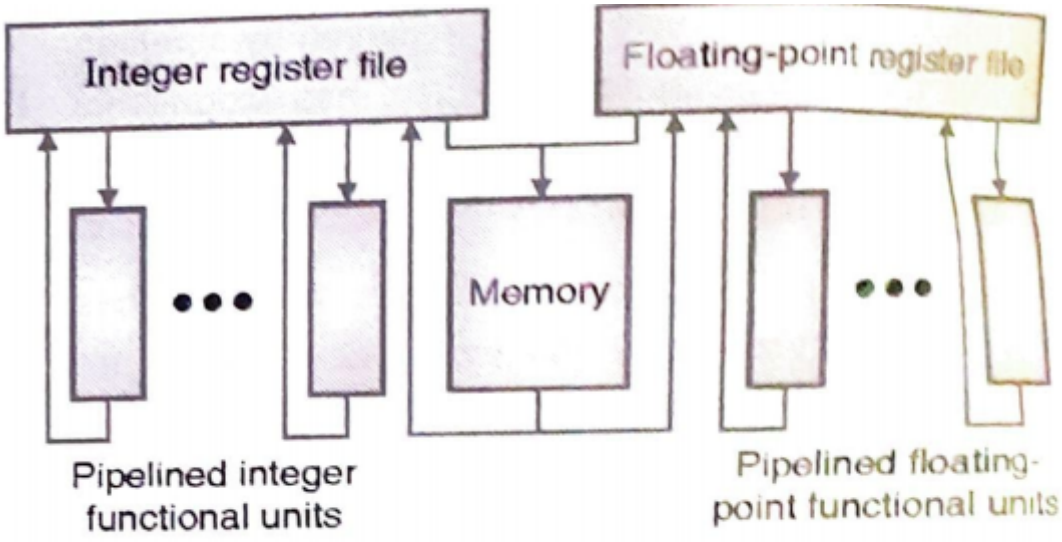


Fig. 1.4.2 : Superscalar Organization

- **VLIW Processors (Very Long Instruction Word Processors)**
  - In VLIW the compiler or preprocessor breaks the instructions into basic instructions which is then grouped together and executed by the processor in parallel.
  - Takes advantage of ILP
  - In VLIW instructions consist of multiple independent operations are grouped together i.e. stored into single word
    - For Example : ADD R1,R2; SUB R1,R5; LOAD R2,data; store R3,data;

      A typical word length considered from 52 bits to 2kbits.
  - There are multiple functional units are present in this architecture and each of the above instruction is given to this indeividual units for exeuction.

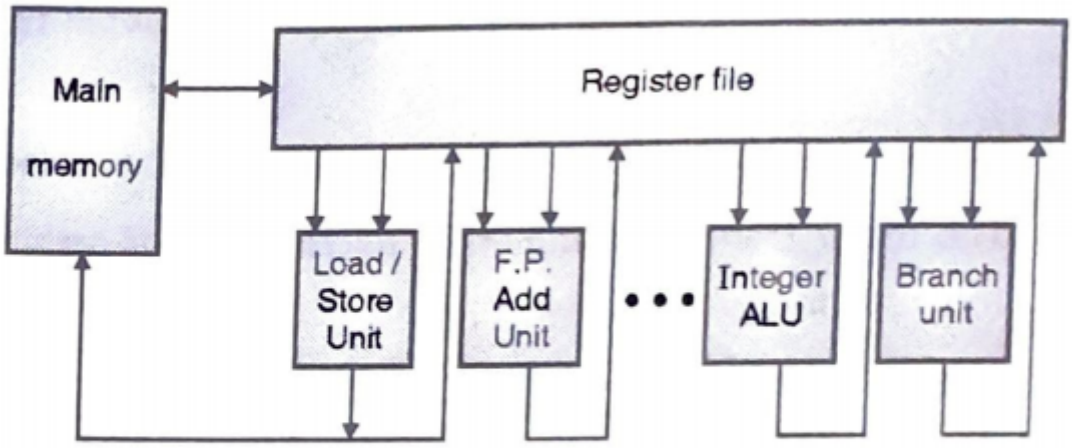


Fig. 1.4.3 : Typical VLIW architecture

-
- Application: Used in Digital Signal Processing

Trends in Micropressor and Architectures

- The major trennd in processors is to exploits the ILP.
- In order to do this there are two ways:
  - Superscaler
  - VLIW
- Superscaler relies on individual hardware to find out the indepedent instruction and then execute on the individual units.
- Whereas VLIW relies on sytsem compiler
- Most of the processors uses superscaler architecture.

Memory Performance

- It is not helpful if we just increase the speed of the processor becaus e if the process doesn't get the data quickly for exeuciton from the memory then there is not use of that processor power.
- So we have to also conside the memory performace.
- Memory performace is mesasure by two parameters latency and bandwidth
  Latency and bandwidth
- Latency: Letency is the time from the issue of memoy rquest by the processor to the time when the processor actually get the data.
- Bandwidth: The rate at which data can be pumped to the processor by the memory. i.e. the rate at which the data flows between memory unit and processor.
- Always make sure that the system should have low latency and high bandwidth

How to improve memory latency?

- By using cache between the CPU and the processor

How to improve memory bandwidth?

- By increasing the size of memory blocks

Hiding Memory Latency Techniques:

- Pre-fetching: We can prefetch the data. So when we actually need the data is is already available.
- Multithreading: In this approach the processor can switch to differne t thread for the execution and comes back when the data for the previous request is ready.

Control Structure of Parallel Platform:

- In parallel computing granularity is a qualitative measure of the ration of computation to communication.
- Granularity is used to describe about the division of task into no of smaller subtassks.
- Two types:
  - Fine grained: In fine grained the system is divided into large networks of small part
  - Course grained: Here the system is divied into smaller networks of large parts.
- Processing units in parallel computers are either operate under single control unit or work independently.
- Models:
  - SIMD (Single instruction stream and multiple data stream)
  - MIMD (Multiple instruction stream and multiple data stream)
- SIMD: we use SIMD when we have to perform same operation on differnt data. Here data is in array. Here we have single control unit.
- MIMD: Differnet operations on differnt data. Multiprocessor system. Differnt control units

Table 1.7.1 : Comparison between SIMD and MIMD

SIMD	MIMD
It is also called as Array processor.	It is also called as multiprocessor.
Here, single stream of instruction is fetched.	Here, multiple streams of instructions are fetched.
The instruction stream is fetched by shared memory.	The instruction streams are fetched by control unit.
Here, instruction is broadcasted to multiple processing elements.	Here, instructions streams are decoded to get multipl decoded instruction streams.
SIMD computers require less hardware than MIMD.	Requires more hardware.

Communication Model of Parallel Platform

- Shared address spaces approach:

- In this approach shared space is available for multiple processors.
- Classified into two types
  - NUMA (Non-Uniform Memory Access): If the time taken by the processor to access any memory is not identical then the platform is NUMA.

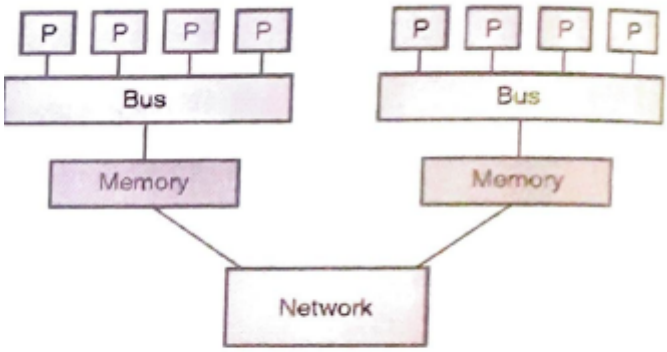


Fig. 1.7.4 : NUMA architecture

    -
    - In NUMA architecture each processor have their local memory and the processor can also access other processor memory but it takes more time as compared to local memory access time.
  - UMA (Uniform Memory Access): If the time taken by the processor to access any memory is identical then the platform is UMA.

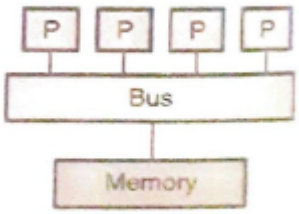


Fig. 1.7.5 : UMA Architecture

    -
    - Here we have central shared memory and each processor can access this memory.
- Message Passing approach
  - In Message passing approach multiple processors can share the data by reading and writing the data into message queue.

Ideal Model for Parallel Computing

PRAM (Parallel Random Access Machine)

- PRAM acts as a framwork for developing the parallel algorithms without considering the physical and implmentation details.
- It is an ideal model for parallell computer and used for analyzing the perfromace of the paralell algorithms.

- It helps to write parallel algorithms without any architecture constriants.
- It is a shared memory multiprocessor with unlimted amount of processors and local memory and access to sharead memory.
- It is suitable for modern architectures i.e. GPU
- It follows SIMD model
- various PRAM models dffer in how they handle read or write conflicts:

  1. ***Exclusive-read, exclusive-write (EREW) PRAM.*** In this class, access to a memory location is exclusive. No concurrent read or write operations are allowed. This is the weakest PRAM model, affording minimum concurrency in memory access.

  2. ***Concurrent-read, exclusive-write (CREW) PRAM.*** In this class, multiple read accesses to a memory location are allowed. However, multiple write accesses to a memory location are serialized.

  3. ***Exclusive-read, concurrent-write (ERCW) PRAM.*** Multiple write accesses are allowed to a memory location, but multiple read accesses are serialized.

  4. ***Concurrent-read, concurrent-write (CRCW) PRAM.*** This class allows multiple read and write accesses to a common memory location. This is the most powerful PRAM model.

  Allowing concurrent read access does not create any semantic discrepancies in the program. However, concurrent write access to a memory location requires arbitration. Several protocols are used to resolve concurrent writes. The most frequently used protocols are as follows:

  - ***Common***, in which the concurrent write is allowed if all the values that the processors are attempting to write are identical.
  - ***Arbitrary***, in which an arbitrary processor is allowed to proceed with the write operation and the rest fail.
  - ***Priority***, in which all processors are organized into a predefined prioritized list, and the processor with the highest priority succeeds and the rest fail.
  - ***Sum***, in which the sum of all the quantities is written (the sum-based write conflict resolution model can be extended to any associative operator defined on the quantities being written).

- Any PRAM model can execute any other pRAM model. i.e. possible to comvert priority PRAM to EREW PRAM.

1.9 Interconnection Network for Parallel Computer

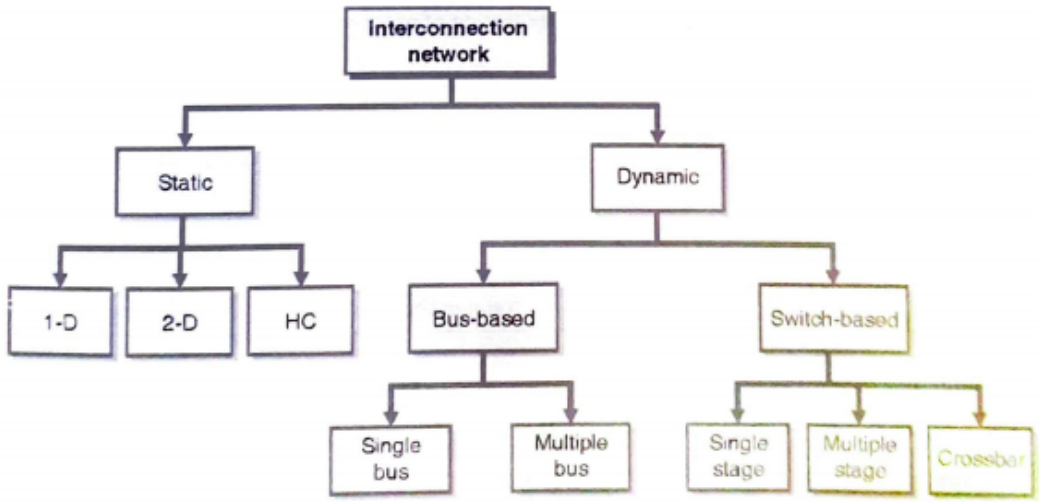


Fig. 1.9.1 : Interconnection Networks

Static Networks:

- Linear topology:
  - n-1 links required to connect n processing elements
  - degree of edge processing elements is 1 while degree of middle processing elements is 2
  - diameter of this topology is n-1
  - not suitable when processing elements are more
- Ring topology:
  - n links
  - degree of all elements is 2
  - diameter is n/2 (one end processor can communicate to other end processor by just traversing half elements)
- Star topology:
  - n-1 links
  - degree is n-1
  - diameter is just 2
- Mesh topology:
  - links 2(n^2-n)
  - degree is 4
  - diamtereis 2(n-1)
  - suitabel for matrix multiplications
- Torus topology:

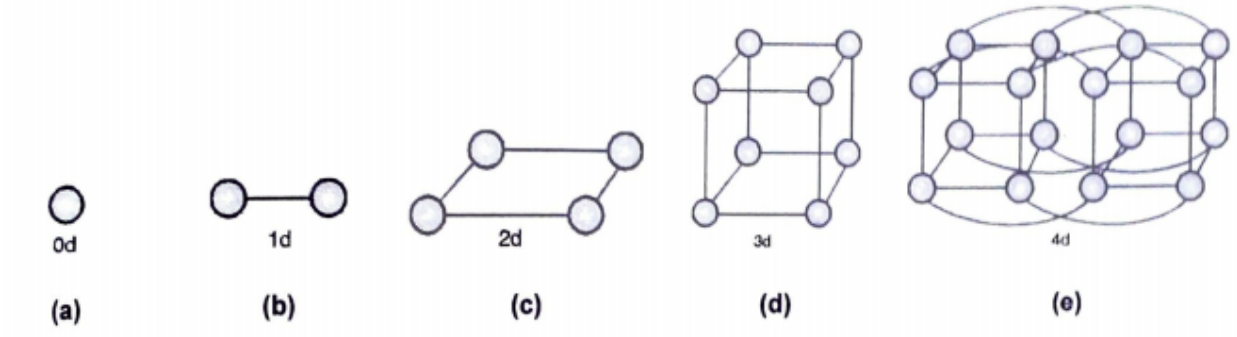
    - links 2n
    - degree 4
    - diamter is n
- Hypercubes:
    - The no of processing elemnets are 2 raised to dimension
    - no of links: (p log p)/2
    - diameter: log p
    - 

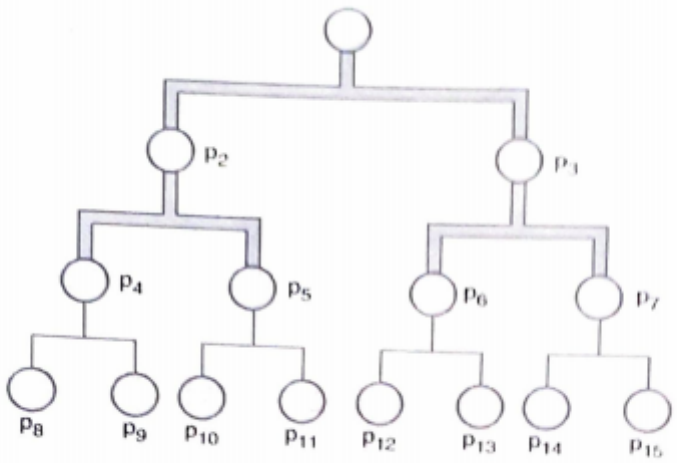
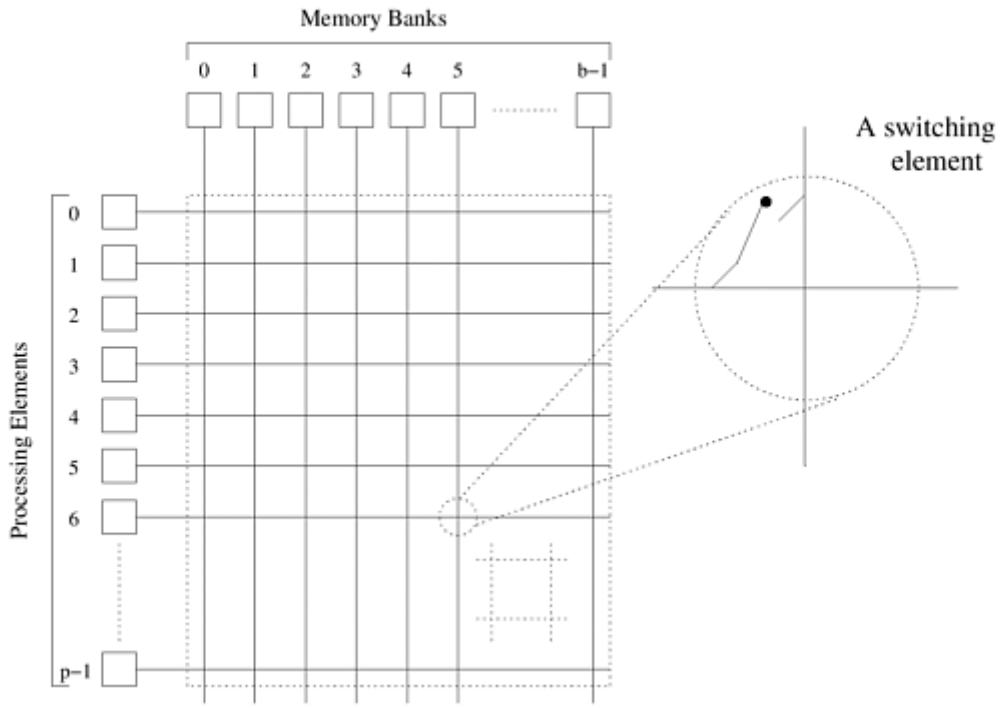
Fig. 1.9.7 : Hypercube of different dimensions i.e. 0, 1, 2, 3 and 4
- Trees:
    - n-1 links
    - degree is 3
- Fully connected network:
    - Each node is conneted to all the other nodes in the netwok via direct link
- Fat tree:
    - thickness of branches nearer root is more as compared to leaf nodes.
    - it means that the bandwidth of edges to root direction is more.
    - 

Fig. 1.9.12 : Fat Tree

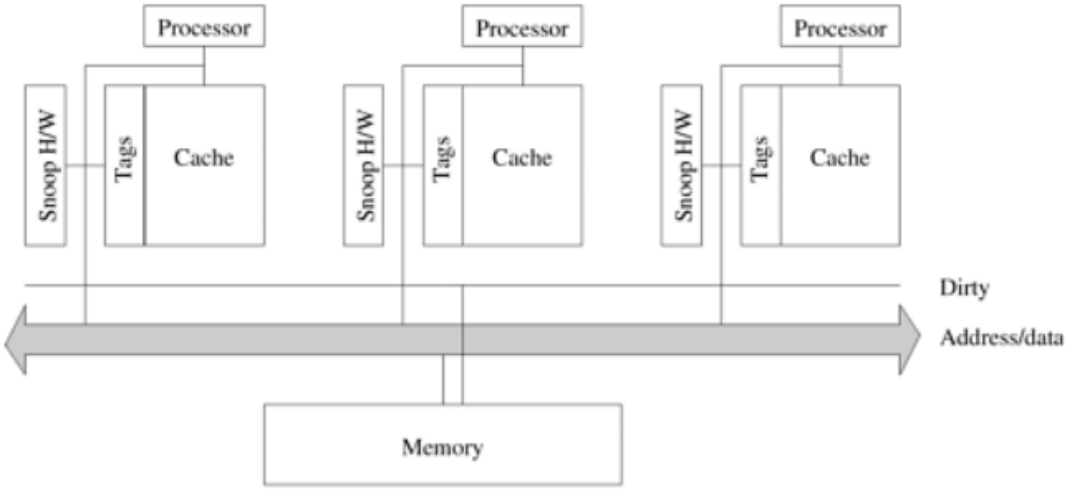
Dynamic Networks:

- Bus network
  - Single Bus
    - Here only one processor can access the mrmoty or I/O at a time.
  - Multi Bus
- Switch Networks
  - Single Stage
  - Multi Stage
  - Crossover Switch:
    - O(p*b) switches are required.
    - Ensure that the system should have at least p memory banks becuase if the memory banks are less then processing ndoes then some processing nodes will be in waiting state.
    - Non blocking architecture.
    - 

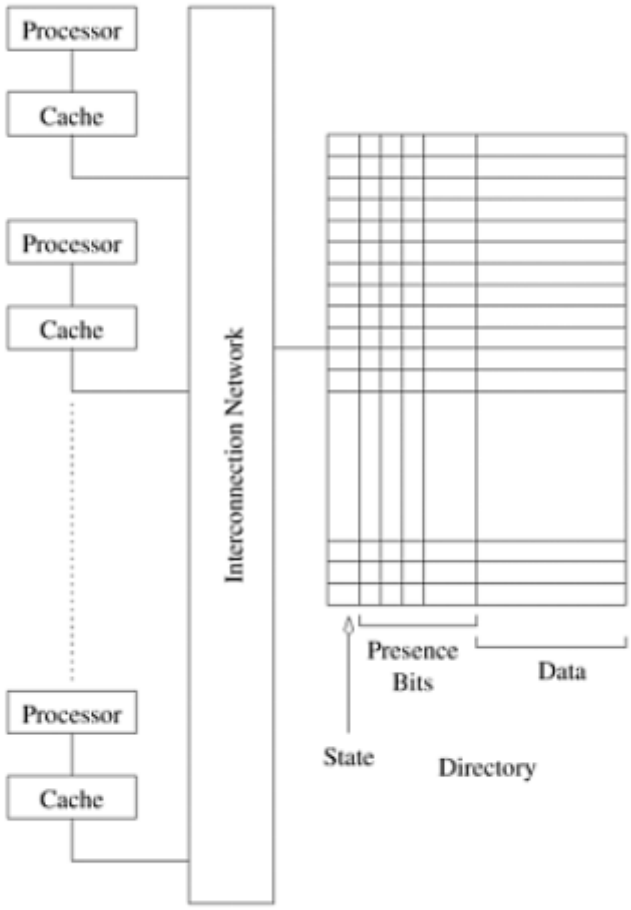
    - as the value of p is increased, the complexity (component count) of the switching network grows as omega(p^2)
    - no of links: p^2
    - diamter: 1

Cache Coherence in Multiprocessor Systems

- Snoopy Cache Systems

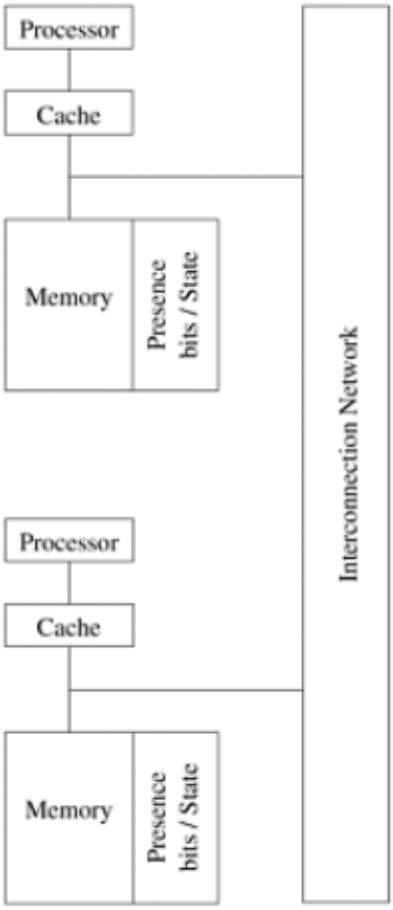


- Directory Based Systems



(a)

  - Here presence bits is a bitmap representing cache blocks and the process at which they are cached
  - state bits is a three state protocol - invalid dirty and shared
- Distributed Directory Systems

- 

(b)

Communication costs in Parallell Machines

The overall efficientcy of parallel algithms is dependent upon the communication cost. Because during execution processing elemennst has to communicate to each other.

N-wide superscaler Architecure

- A wide issue architecture is a computer processor that issues more than one instruction per clock cycle.
- Three broad types:
  - Statically : Scheduled superscaler architecture
  - VLIW
  - Dynamically: Scheduled superscaler architecture

Multicore Processor Architecture

- Multicore means multiple cores on the same processor chips.
- superscaler and multithreading processor are single core
- dual core, trie core and quad core
- multicore considerd in MIMD

Unit 2

Decomposition

- Decomposition is the process of dividing a computation into smaller parts. Some or all parts of this computation may be executed in parallel
- The number and the size of tasks into which the problem is decomposed determined the granularity of the decomposition

Task Dependency Graph

(a) Task T_2 is dependent on T_1

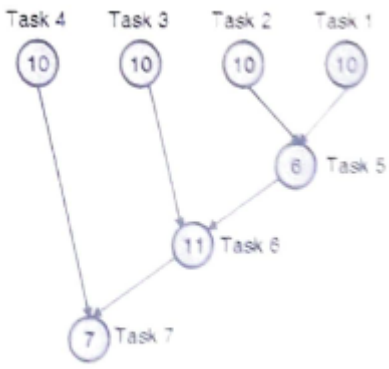


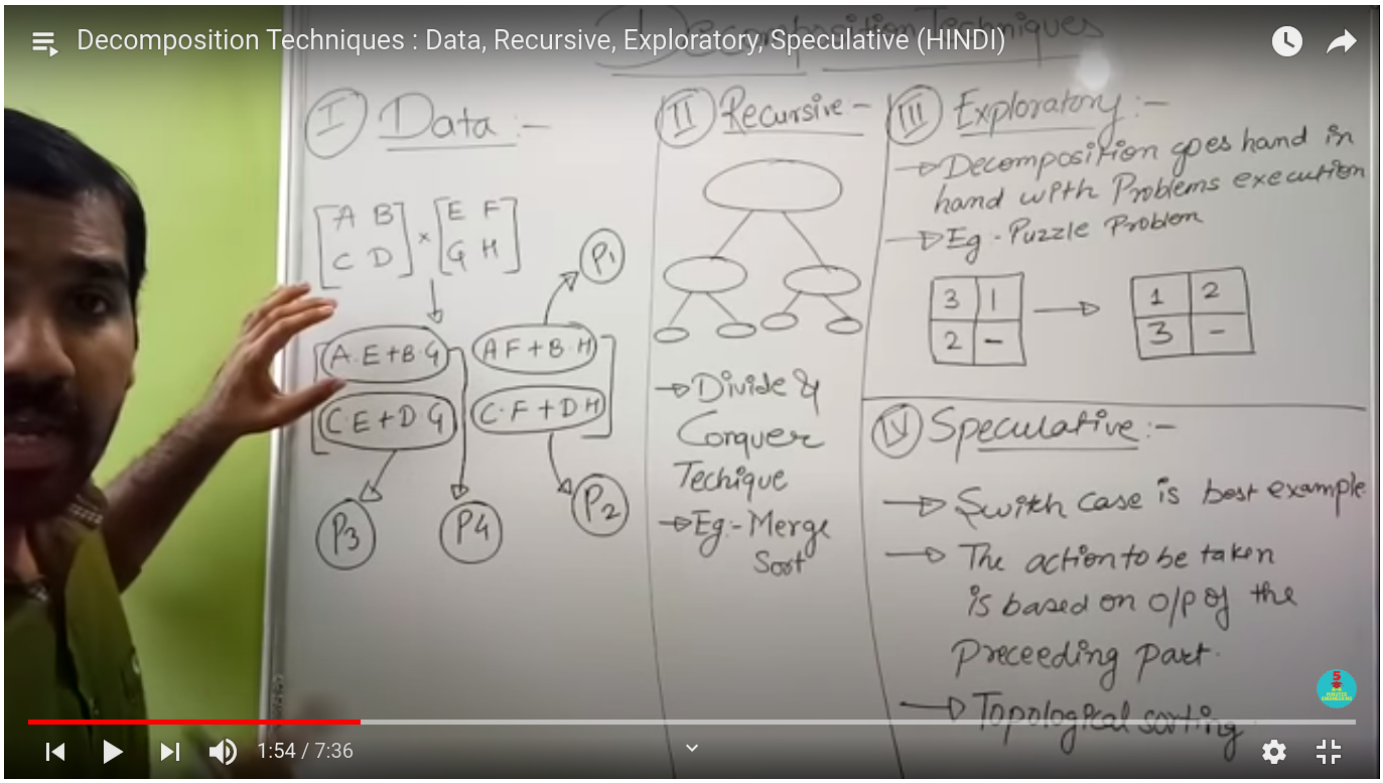
Fig. 2.1.2 : Graph 1

- Critical Path: longest directed path between any pair of start and finish nodes. Above example critical path length: 10+6+11+7 = 34
- Total amount of work: Sum of total work of each task. 10*4+6+11+7=64
- Maximum degree of concurrency: Maximum no of tasks executed simultaneously. maxiumum degree of concurrency is 4
- Avg degree of concurrency: avg no of tasks that can run concurrently over the entire duration of execution of program. 64/34 = 1.88
  Avg degree of concurrency = total amount of work / critical path length
- Degree of concurrency: The no of tasks that can be executed in parallel.

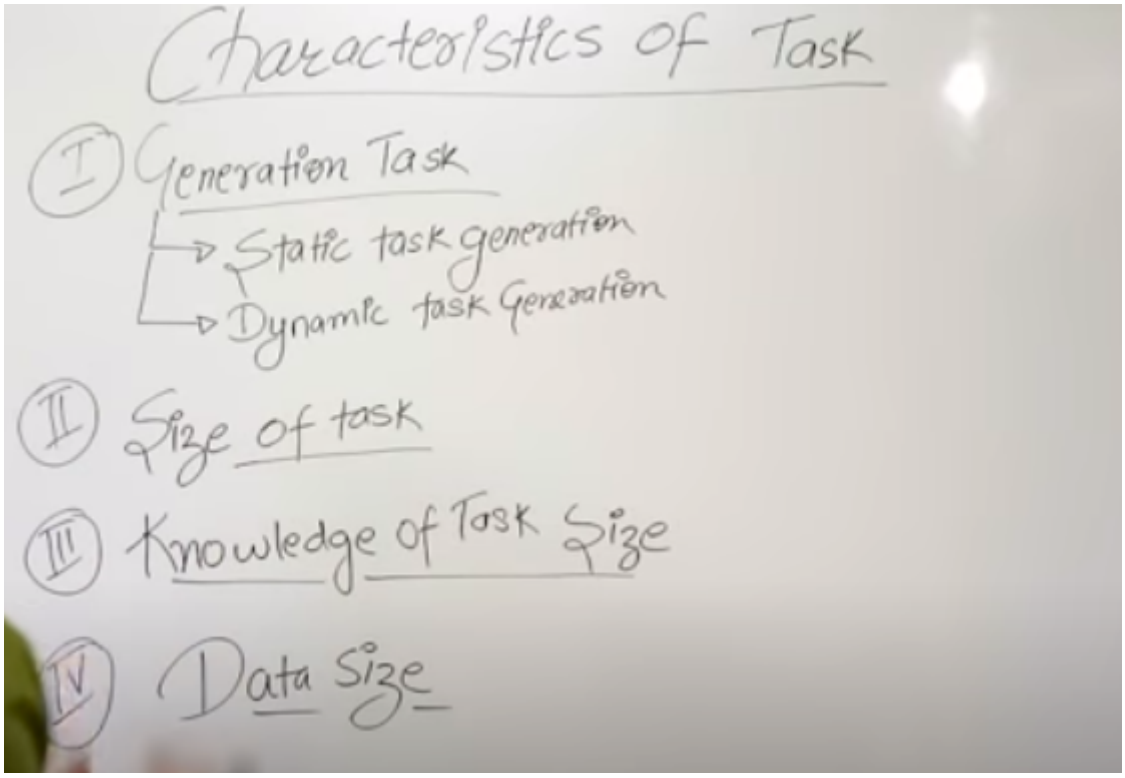
Decomposition Techniques

- Data Decomposition : Here data is different for each process but process method is same for all processes. Like addition and multiplication.

- Recusrive Decomposition
- Exploratory Decomposition
- Speculative Decomposition



Characteristics of Task



Task: Basic unit of entire system or indivisible sequnetial unit of computation.

There are two ways of creating the task:

- static task generation: In this technique we will create the task ahead of time. i.e. we will create task before the algorithm actually needs it.
- Dynamic task generation: In this technique we will create the task at the time of algorithm execution.

Size of the taks: It is nothing but the time required by the task to complete its execution.

Knowledge of the task size: It is important to have knowledge about the task size because for the execution of the task we have to map it to the process.

Data Size: Some amount of data is associated with the task. That data is used at the time of task execution.

Mapping techniques for load balancing

- mapping is the technique used to assign tasks to processes.
- mapping technques are used to solve major source of overheads:
    - load imbalance
    - inter process communication
- two techniques of mapping for load balancing:
    - static mapping:
        - before executing algorihtm tasks are distributed among processes.
        - static mapping is applicable for static tasks
        - if task size is unknown a static mapping can lead to series load imbalances issues.
        - for uniform computational requirements
    - dynamic mapping
        - In dynamic mapping tasks are distrbuted among processes at runtime.
        - also called as dynamic load balancing
        - applicable for tasks which are generaed dynamically
        - for non-uniform computational requirements
        - Two types:
            - centralized dynamic mapping:
                - Here all executable tasks are maintained in a common central data strucure.
                - Master: process manage a group of available tasks
                - slave: depend on master to obtain work
            - Distrbuted dynamic mapping
                - tasks are distributed among processes whcih exchange tasks at runtime to balance work.
                - Each process can send or receive work from other processes.

Method for containing Interaction Overheads

- Maximizing data locality
- Minimizing contention and hot-spots
- Overlapping computation with interaction
- Replicating data or computation
- Using optimized collecive interaction operations.

Parallel Algorithm Models

- An algorithm model is typically a way of structuring a parallel algorithm by selecting decomposition, mapping technique and the stretegy to minimize interaction overheads.
- 6 algorithm models
  - data parallel model
  - task graph model
  - task pool model
  - master-slave model
  - pipeline/producer consumer problem model
  - hybrid model

Data Parallel Model:

- simple algorithmic model
- tasks are statically mapped to processes and each task performs similar operations.
- same identical operations applied to multiple data items, called as data parallalism
- solves large problem effectively
- Ex: dense matrix multiplication

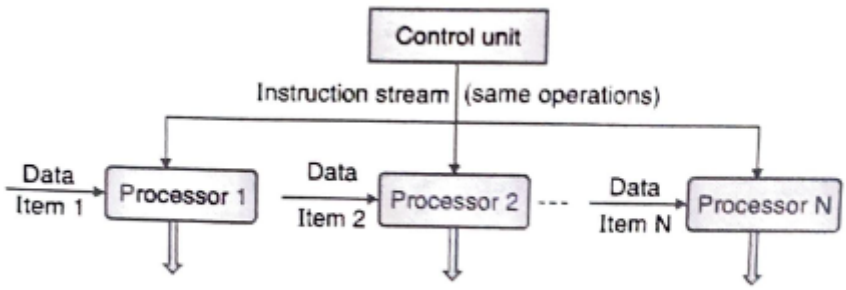


Fig. 2.6.1

Task Graph Model:

- It is used to solve problem in which amount of data associated with tasks is large.
- it uses the task dependcency graph to analyze the interrelationship among the tasks.
- tasks are mapped statically
- Ex: Parallel quick sort, sparse matrix factorization, and parallel algorithms derived via divide-and-conquer approach.

- task parallelism

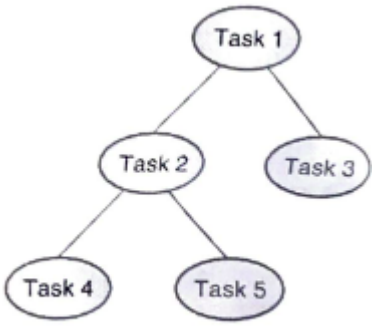


Fig. 2.6.2

Task Pool Model

- also called work pool model
- dynamic mapping of tasks
- centarlized or decentralized
- ex: parallelization of loops by chunk scheduling, Parallel tree search

Master Slave Model

- also called as manager-worker model
- one or more master processes generate work and allocate to worker processes.
- suitable for both shared address space and mesage passing paradigm.

Pipeline/Producer-consumer problem model

- In this mode stream of data is passed on through succession of processes, each of whcih perform same task.
- The simultaneous execution of diff programs on a data stream is called stream parallelism.
- involves static mapping
- a pipeline is a chain of producer and consumer
- Ex: parallel factorization algrithm

Hybrid Model

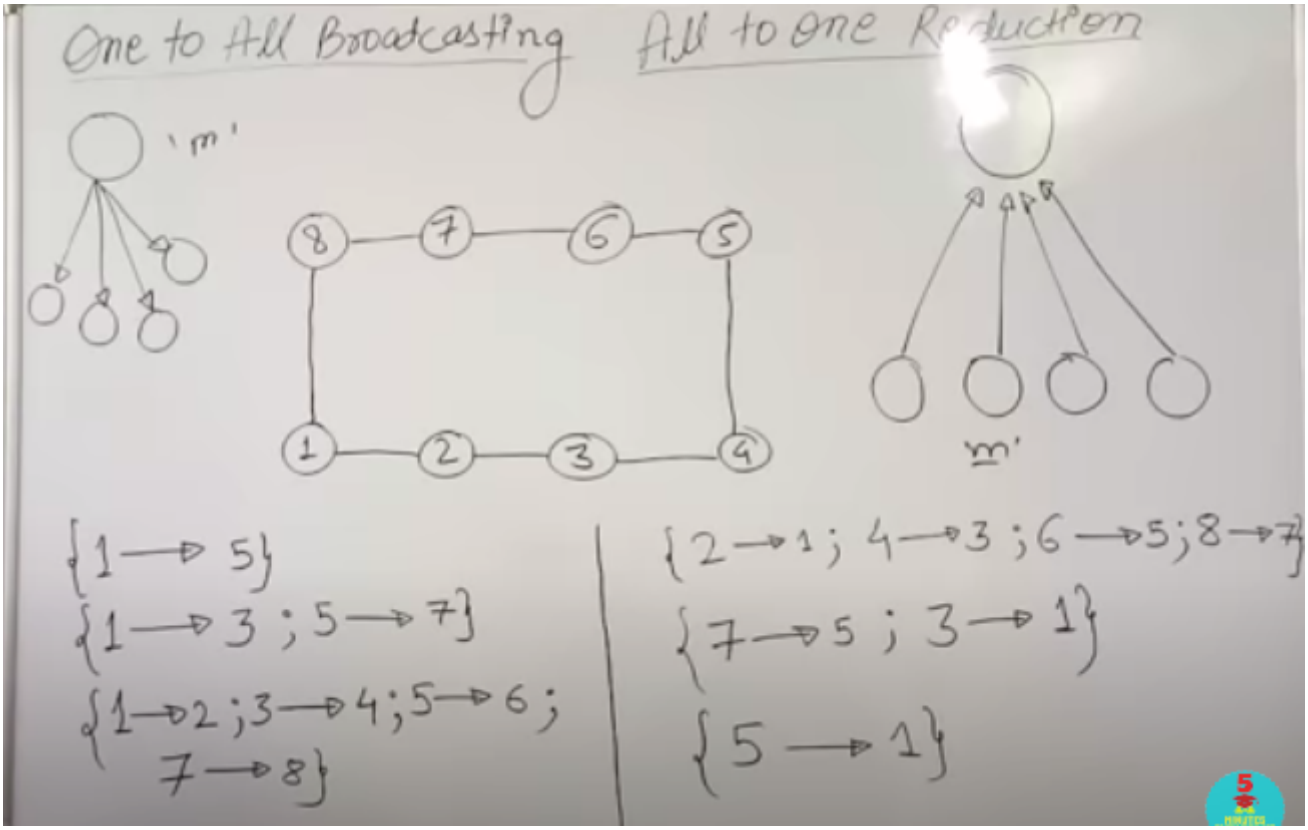
- Combination of multiple models
- Ex: parallel quick sort

GPU (Graphics Processing Unit)

- initially called as video shifters

Unit 3

One to All Broadacast and All to One Reduction



- Ring Implementation: Estabilishing communication between each node is underitlizing that's why we use process called recursive dobling where one send data to other nodes and now there are tota l2 ndoes that 2 ndoes send data to other 2 nodes and become 4 and so on.
- log p steps
- All to one reduction is used for finding sum, product, min and max
- One-to-all broadcast and all-to-one reduction are used in several important parallel algorithmsincluding matrix-vector multiplication, Gaussian elimination, shortest paths, and vector innerproduct.

4.1.6 Cost Analysis

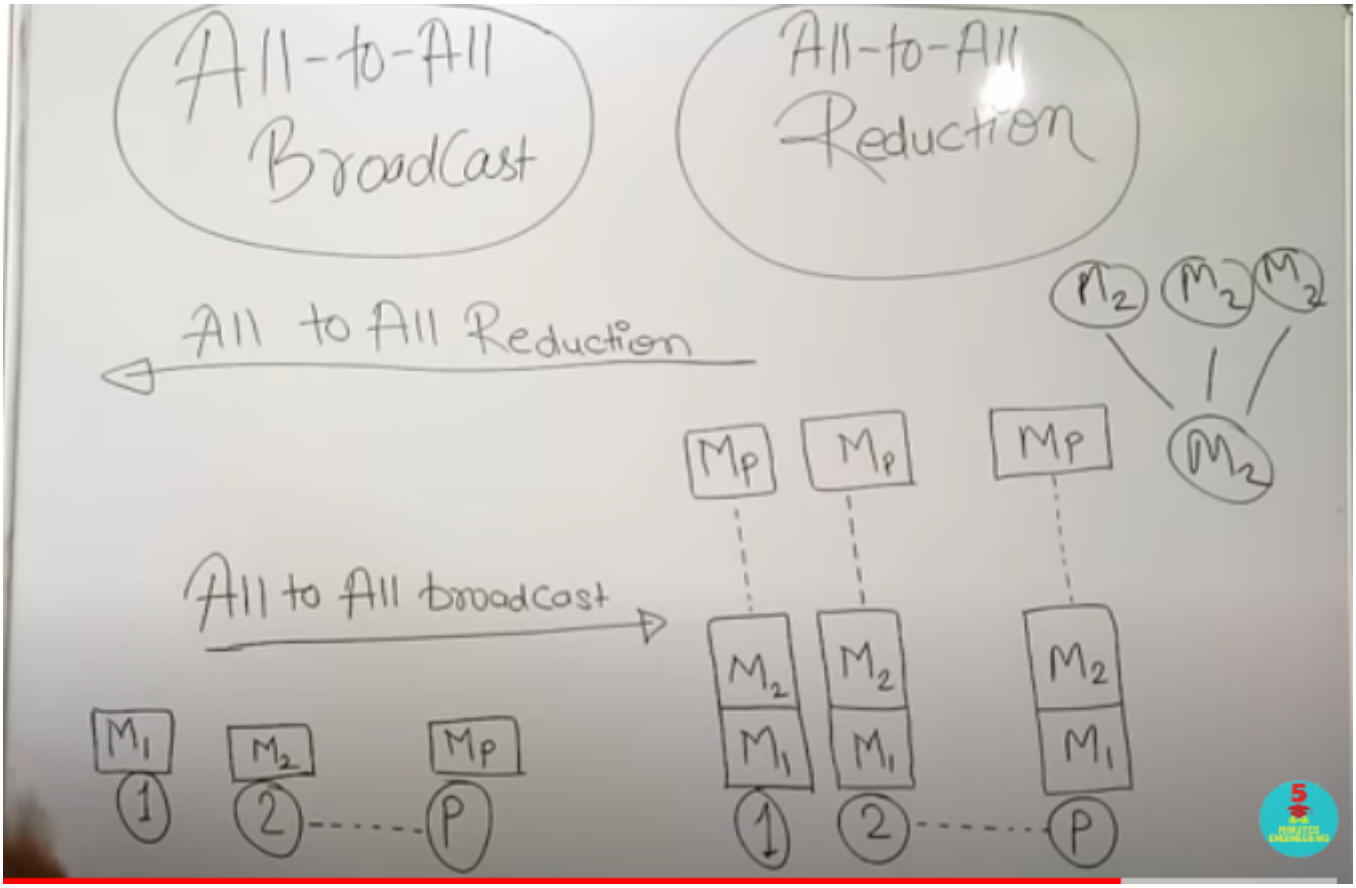
Analyzing the cost of one-to-all broadcast and all-to-one reduction is fairly straightforward. Assume that p processes participate in the operation and the data to be broadcast or reduced contains m words. The broadcast or reduction procedure involves log p point-to-point simple message transfers, each at a time cost of $t_s + t_w m$. Therefore, the total time taken by the procedure is

Equation 4.1

$$T = (t_s + t_w m) \log p.$$

-

All to All BroadCast and Reduction



- **All to All Broadcast**
  In All to All broadcast there are P nodes and P messages one message at one node.
  M1 at node 1
  M2 at node 2
  ...
  MP at node P

Here all nodes send message to all other nodes.
So if node 1 is source node then all other p-1 nodes are destination nodes and 1 is destinated nodes for all other p-1 source nodes.
- Matrix operations such matrix multiplication and matrix vector multiplications are implemented by using all to all broadcast.

- **All to All Reduction**
  In case of All to All reduction all the message are removed and the message related to the node is available to that node.
  For ex at node 1 all other nodes are send to their respective nodes and M1 is present at node1.
  After this process all received messages are group together and combined into one message. This is all to all reduction.

Cost analysis for linear or ring

Equation 4.2

$$T = (t_s + t_w m)(p - 1).$$

-

Cost analysis of mesh and hypercube

Equation 4.3

$$T = 2t_s(\sqrt{p} - 1) + t_w m(p - 1).$$

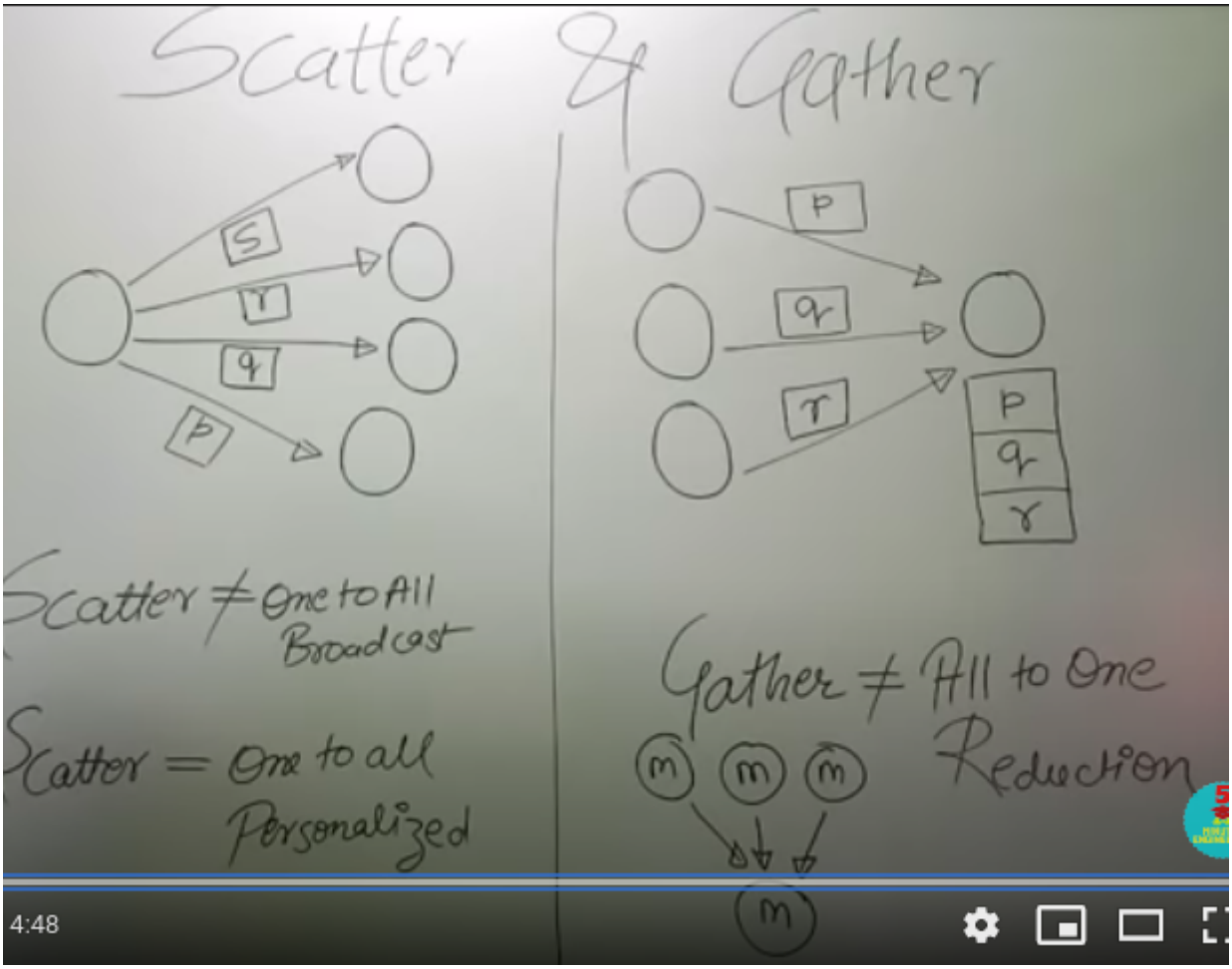
On a p-node hypercube, the size of each message exchanged in the i th of the log p steps is $2^{i-1}m$. It takes a pair of nodes time $t_s + 2^{i-1}t_w m$ to send and receive messages from each other during the i th step. Hence, the time to complete the entire procedure is

Equation 4.4

$$\begin{aligned} T &= \sum_{i=1}^{\log p} (t_s + 2^{i-1} t_w m) \\ &= t_s \log p + t_w m(p - 1). \end{aligned}$$

-
- In Mesh message size at initia phase is m
  and the message size in second phase is msqrt(p)
- In hypercube the message size is doubled

Scatter and Gather



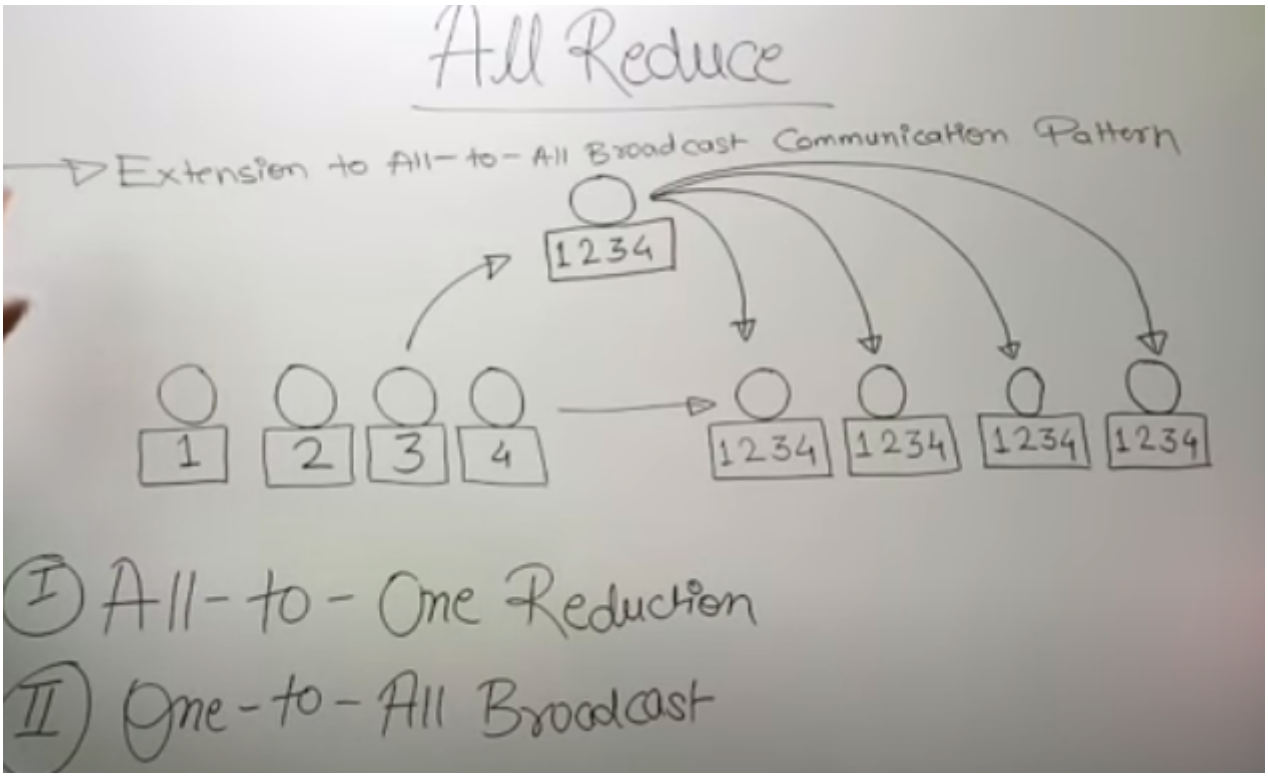
- in scatter operatin size of message is half in each operation on hypercube

- Cost on hypercube

Equation 4.6

$$T = t_s \log p + t_w m(p-1).$$

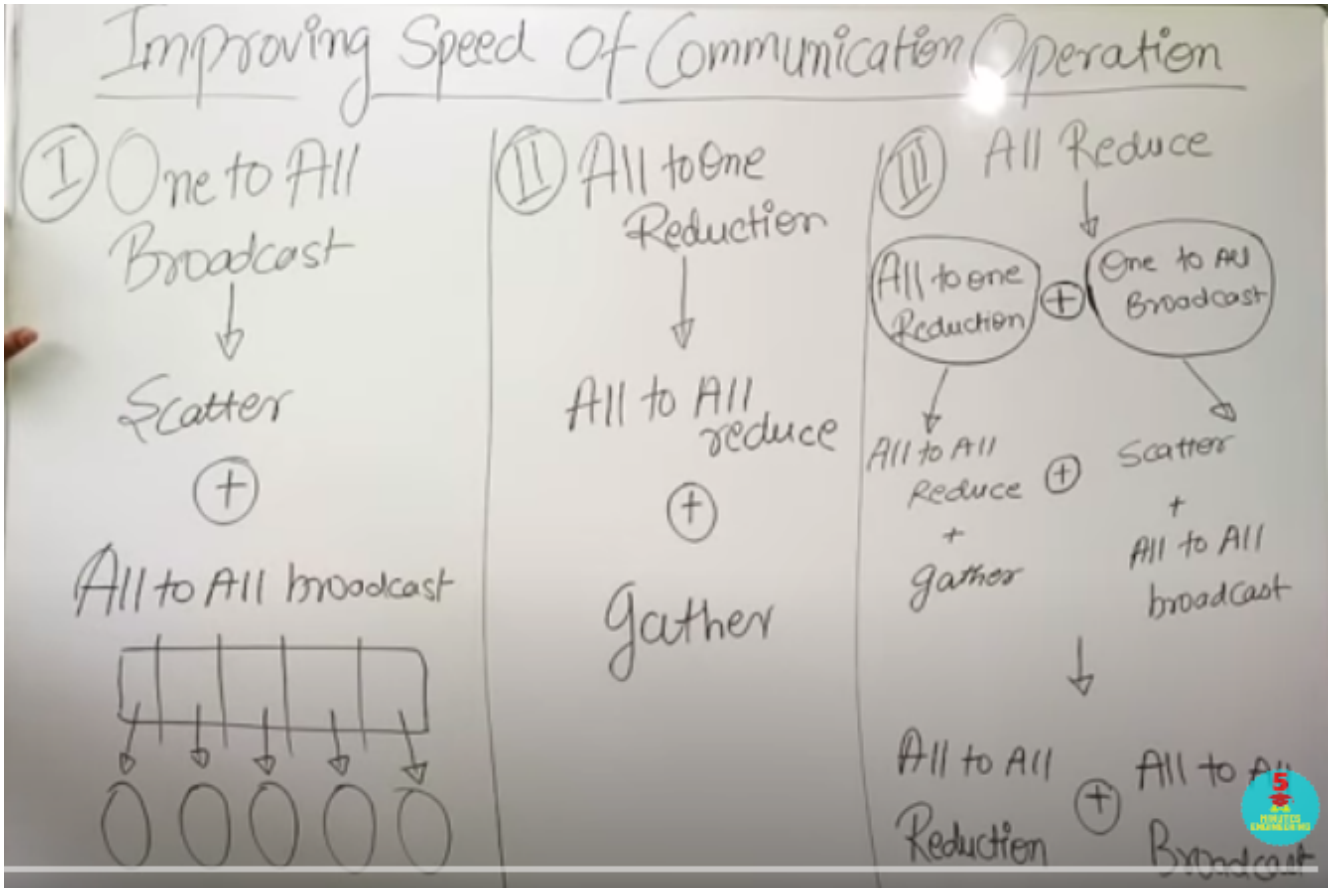
All Reduce



Improving speed of communicaion operation

- Let's say we have very big message which we have to sent to all other p-1 processes so one way is to use one to all brodacast to sent that message but it takes time because the message is big and we are sending serially one by one to each process.
- Optimal technique is to divide the message into multiple small message and use scatter opartion to send those each message chunk to other p-1 processes so now each p-1 processes have a particular chunk of message now we can use all to all broadcast to create whole message at each

p-1 process.



- All to All personalized is also called as total exchange

Unit 4

- A sequential algorithm is usually evaluated in terms of its execution time, expressed as afunction of the size of its input.
- The execution time of a parallel algorithm depends not only oninput size but also on the number of processing elements used, and their relative computationand interprocess communication speeds.
- A parallel system is thecombination of an algorithm and the parallel architecture on which it is implemented.

Sources of overhead in Parallel Programs

- Interprocess Interaction: In parallel algorithm the time required by the processing element to share the inermediate result with other processing element is interprocess interaction.
- Idling: idling can be occured due to load imbalance, synchronization and due to sequential code in the algorithm.
- Execess Computation: It is very hard or impossible to convert the best sequential algortihm to parallel algorithm and that's why we chosse simple or poorer sequnetial algorithm for parallellization.
  The difference in computation performed by the parallel program and the best serial program is the excess computation overhead incurred by the parallel program.

Perfoemance Metrices for Parallel Systems

- Execution Time
  - The serial runtime of a program is the time elapsed between the beginning and the end of itsexecution on a sequential computer. Denoted by T_S
  - The parallel runtime is the time that elapses from themoment a parallel computation starts to the moment the last processing element finishesexecution. Denoted by T_P
- Total Parallel Overhead
  - We define overhead function or total overhead of a parallelsystem as the total time collectively spent by all the processing elements over and above thatrequired by the fastest known sequential algorithm for solving the same problem on a singleprocessing element.
  - Denoted by T_O
  - **Equation 5.1**

    $$T_o = pT_P - T_S.$$
- Speedup
  - how much performance gain is achieved by parallelizing a given application over a sequential implementation.
  - It is defined asthe ratio of the time taken to solve a problem on a single processing element to the timerequired to solve the same problem on a parallel computer with p identical processing elements.
  - Denoted by S
  - $$S = \frac{T_S}{T_P}$$
  - We also express speedup as follows :

    $$S(p) = \frac{T_1}{T_P}$$
  - On Serial we have only one proecessing element and on parallel we have p processing elements.
  - Consider an example of adding n numbers by using n processign elements.
  - $$S = \frac{T_S}{T_P} = \theta\left(\frac{n}{\log n}\right)$$
- Efficiency
  - Efficiency is a measureof the fraction of time for which a processing element is usefully employed

- it is defined as theratio of speedup to the number of processing elements. In an ideal parallel system
- Denoted by E
- In an ideal parallel system, speedup isequal to p and efficiency is equal to one. In practice, speedup is less than p and efficiency isbetween zero and one
- $$E = \frac{S}{p}.$$
- Efficiency of previous example adding n number is

$$E = \frac{\theta\left(\frac{n}{\log n}\right)}{n}$$
$$= \theta\left(\frac{1}{\log n}\right)$$

- Cost
  - Cost is the product of parallel runtime and the number of processing elements used (p x Tp)
  - Cost reflects the sum of the time that each processing element spends solving the problem.
  - A parallel system is said to be cost-optimal if the cost of solving a problem on a parallel computer is asymptotically identical to serial cost.
  - $$\text{Since } E = \frac{(T_S)/(T_P)}{p}$$
    $$= \frac{(T_s)}{p \times T_P},$$
    $$\text{for cost optimal systems } E = \frac{p}{p} = 0(1)$$

    Cost $(p \times T_p)$ is sometimes referred to as work or processor-time product.

The effect of Granularity on Performace

- Always we should have to give large pieces of inputs to the processing elements, this increases the granularity of computation on the processing elements.
- Scaling down is nothing but using less no of processing elements then the no of inputs. So let's say we have the input of size n then instead of using n processign elements we use p processing elements where p < n so this is scaling down.
- We can use parallel algorithm designed for n processign elements by assumign n virtual processign elements but physically use only p processng elemenst. So because of this p processing elements simulate n/p virtual processign elements.
- each processign elements process n/p work

Minimum Execution Time and Minimum Cost-Optimal Execution Time

- As we add more processing elements, either the parallel runtime decreases and after reaching to some minimum value point it starts increasing. So have to detect the minimum execution time of a parallel algorithm without the constaint of processing elements.
- We can find T_P^{min} by derivaing respect to p and equating to 0

$$\frac{d}{dp} T_P = 0$$

-
- Minimum cost optimal execution time is the min time in which a problem can be solved by a cost optimal parallel system.

$$T_P^{cost_opt} = \Omega\left(\frac{W}{f^{-1}(W)}\right).$$

-

Dense Matrix Algorithms
Matrix Vector Multiplication

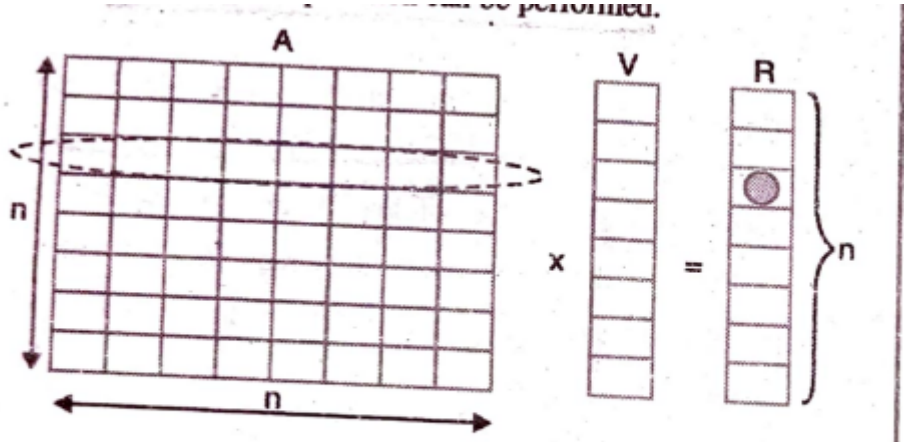


Fig. 4.6.2 : Multiplication of a matrix A and a vector V produces result in R.

-
- Here in this problem we have nXn matrix and nX1 vector and the result is also vector.
- Basic sequential algorithm runtime is n^2

- Here is the sequential algorihthm pseudocode

```
procedure MAT_VECT ( A, x, y)
begin
   for i := 0 to n - 1 do
   begin
      y[i]:=0;
      for j := 0 to n - 1 do
          y[i] := y[i] + A[i, j] x x[j];
   endfor;
end MAT_VECT
```

- Now in order to parallelize this matrix-vector multiplication we have two ways: row-wise 1D partioning, 2D partitioning.

Row wise 1D partitioning:

- In row wise 1D partitioning each processor has one row of matrix and 1 element of the vector.

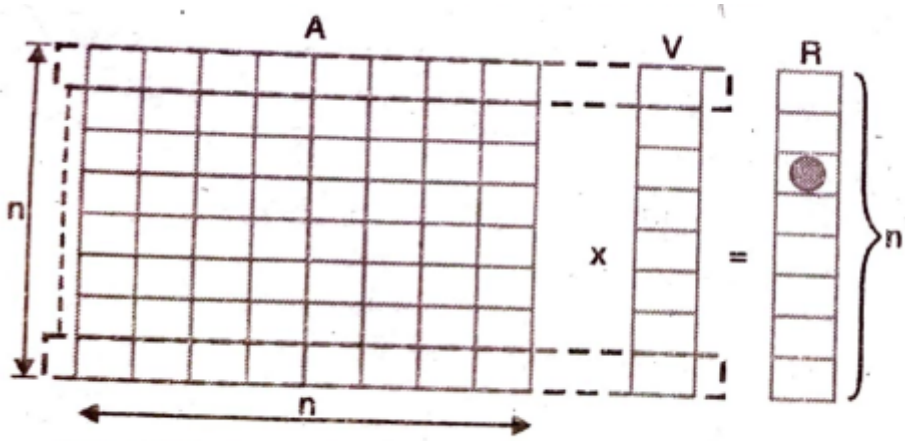


Fig. 4.6.3 : A complete row is stored in each processor

- Before matrix computation we use all to all broadcast so that each processign element contains all the elements of the vector in order to do the matrix vector multiplication.

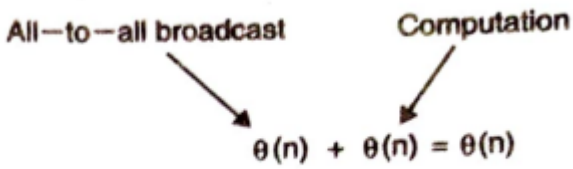


Fig. 4.6.8

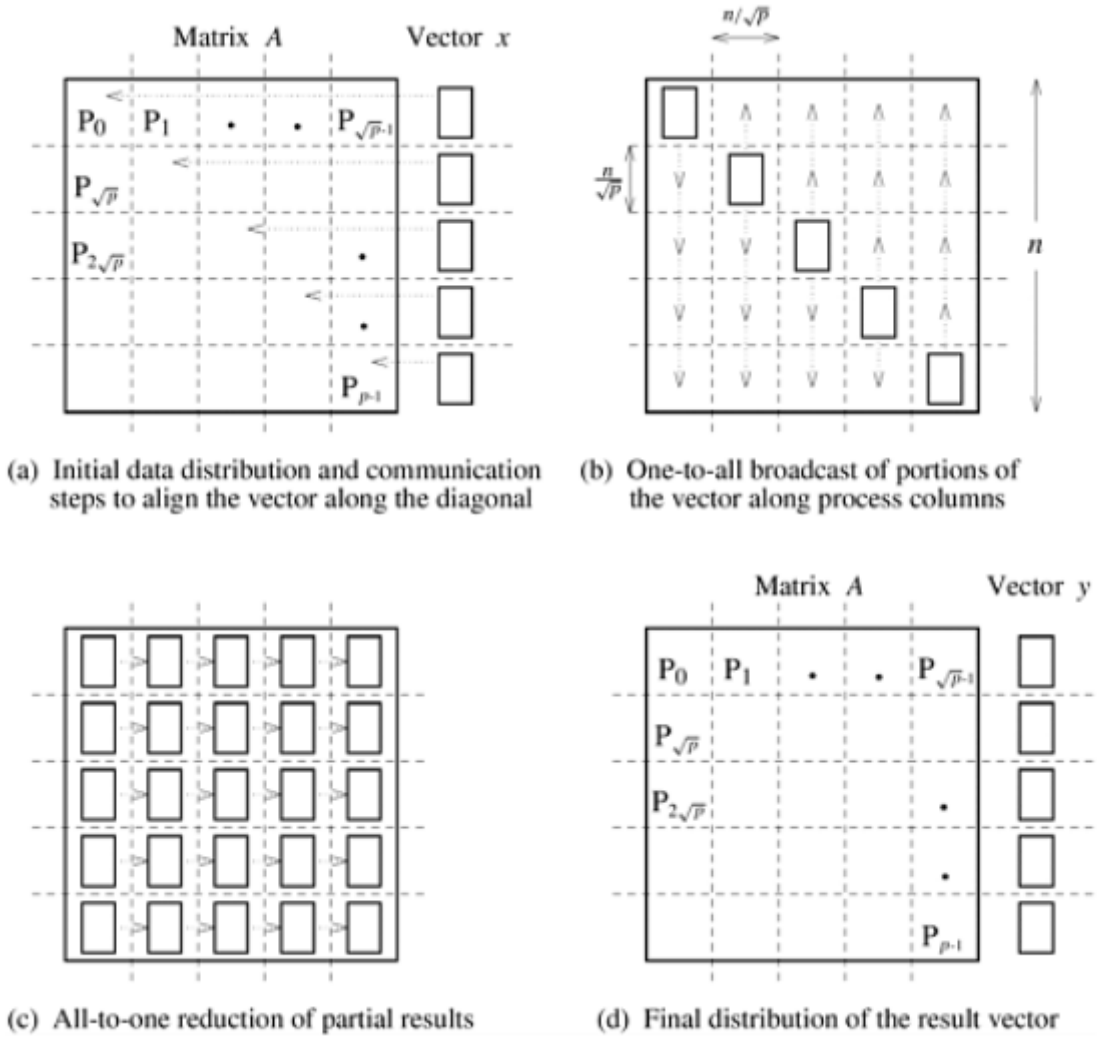
- Here overall parallel runtime is O(n) and if we assume p=n then the computation time is O(n^2)
- and if we use less then n processing elements i.e. if we use p processign elements where p < n then the computation time is n^2/p

Thus, the parallel run time of matrix-vector multiplication based on rowwise 1-D partitioning (p < n) is

$$T_P = \underbrace{\frac{n^2}{p}}_{\text{Computation}} + \underbrace{t_s \log p + t_w n}_{\text{All-to-all broadcast of vector elements}}$$

2D Partitioning

- In 2D partitioning we divide the matrix and assign each elements of the matrix to one procesing elements so for nxn elements n^2 processing elements are used.



(a) Initial data distribution and communication steps to align the vector along the diagonal

(b) One-to-all broadcast of portions of the vector along process columns

(c) All-to-one reduction of partial results

(d) Final distribution of the result vector

- Here we use the one to all broadcast and the all to one reduction. Time of braodacast and reduction is O(logn)
- Here the computation time is O(1) because each elements just have to multiply single element with another.
- Overall parallel runtime is O(logn) + O(1) + O(logn) = O(logn)
- So for p=n^2 the overall cost is O(n^2 log n) is not cost optimal

- Cost for p < n^2

Equation 8.7

$$T_P = \overbrace{n^2/p}^{\text{computation}} + \overbrace{t_s + t_w n/\sqrt{p}}^{\text{aligning the vector}} + \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{columnwise one-to-all broadcast}} + \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{all-to-one reduction}}$$

$$\approx \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p$$

Matrix Matrix Multiplication

- Sequantial algorithm executio time of matrix matrix multiplicatio is O(n^3).
- For parallel we can use the concept of paritioning into submatrices. Here the matris divided into submetrices or blocks and then algebric operations are applied to these blocks. The algebric operations applied on the submatrices are called as block matrix operations.
- Thus total time complexity is given by $\frac{n^3}{p} + t_s \log p + 2t_w \frac{n^3}{\sqrt{p}}$

  Algorithm

  A=

2	3	4	5
9	8	7	6
5	4	2	3
8	7	3	4

  B=

3	5	7	6
2	7	6	3
7	5	3	2
4	3	2	5

  Here we are considering p=4 so total no of steps are sqrt(p) i.e. 2 steps.
  Step1:

Find A1 from A by following process:

A=

$$\begin{array}{cc|cc} 2 & 3 & 4 & 5 \\ 9 & 8 & 7 & 6 \\ \hline 5 & 4 & 2 & 3 \\ 8 & 7 & 3 & 4 \end{array} \quad \leftarrow \text{Left Shift}$$

A1=

$$\begin{array}{cc|cc} 2 & 3 & 4 & 5 \\ 9 & 8 & 7 & 6 \\ \hline 2 & 3 & 5 & 4 \\ 3 & 4 & 8 & 7 \end{array}$$

Now find B1 matrix from B matrix by doing up shift

Find B1 from B by following process:

B=

$$\begin{array}{cc|cc} 3 & 5 & 7 & 6 \\ 2 & 7 & 6 & 3 \\ \hline 7 & 5 & 3 & 2 \\ 4 & 3 & 2 & 5 \end{array}$$

$\uparrow$

Up Shift

B1=

$$\begin{array}{cc|cc} 3 & 5 & 3 & 2 \\ 2 & 7 & 2 & 5 \\ \hline 7 & 5 & 7 & 6 \\ 4 & 3 & 6 & 3 \end{array}$$

Now perform mutliplication of A1 and B1 blocks in each processes.

So process p0 perform multiplication of first block from A1 and first block from B1. Same for other processes.

P0 Process:

$$\begin{matrix} 2 & 3 \\ 9 & 8 \end{matrix} * \begin{matrix} 3 & 5 \\ 2 & 7 \end{matrix} = \begin{matrix} 12 & 31 \\ 43 & 101 \end{matrix}$$

In this way finally we get the C1 matrix

C1=

$$\begin{matrix} 12 & 31 & 22 & 33 \\ 43 & 101 & 33 & 44 \\ 26 & 19 & 59 & 42 \\ 37 & 27 & 98 & 69 \end{matrix}$$

Step2:

Now find A2 and B2 by peforming 2 left shift on A1 and 2 up shift on B1

Step 2:

Find A2 from A1 by following process:

A1=

$$\begin{matrix} 2 & 3 & 4 & 5 \\ 9 & 8 & 7 & 6 \\ \hline 2 & 3 & 5 & 4 \\ 3 & 4 & 8 & 7 \end{matrix} \quad \begin{matrix} \\ \leftarrow \text{Left Shift} \\ \\ \leftarrow \text{Left Shift} \end{matrix}$$

and now again do the same thign and in this way we get C2 matrix

C2=

$$\begin{matrix} 48 & 35 & 32 & 21 \\ 73 & 53 & 111 & 78 \\ 23 & 53 & 12 & 19 \\ 38 & 89 & 17 & 26 \end{matrix}$$

Final Result

C= C1 + C2

60	66	54	54	=	12	31	22	33	+	48	35	32	21
116	154	144	122		43	101	33	44		73	53	111	78
49	72	71	61		26	19	59	42		23	53	12	19
75	116	115	95		37	27	98	69		38	89	17	26

Canon's algorithm

- Canon algorithm is used for efficient matrix-matrix mutliplication.
- It is same as simple parallel matrix-matrix multiplication but it is more memory efficient.

$$T_P = \frac{n^3}{p} + 2\sqrt{p}t_s + 2t_w\frac{n^2}{\sqrt{p}}.$$

Algorithm

- Here there is separte process for each element in the matrix so if we have 16 elements in the matrix then we have 16 processes and sqrt(16) = 4 stages are required here.

A=

2	3	4	5
9	8	7	6
5	4	2	3
8	7	3	4

B=

3	5	7	6
2	7	6	3
7	5	3	2
4	3	2	5

Step 1:
We get A1 matrix from A matrix by performing following operations

A=

2	3	4	5	← 0 Left Shift
9	8	7	6	← 1 Left Shift
5	4	2	3	← 2 Left Shift
8	7	3	4	← 3 Left Shift

We get B1 matrix from B matrix by performing following operations

B=

3	5	7	6
2	7	6	3
7	5	3	2
4	3	2	5
↑	↑	↑	↑
0 Up Shift	1 Up Shift	2 Up Shift	3 Up Shift

Now we get C1 by multiplying individual terms of A1 and B1

C1[i][j]=A1[i][j] * B1[i][j]

C1=

6	21	12	25
16	35	12	54
14	9	35	12
16	40	42	6

Step 2:

Apply 1 left operation on every row A1 matrix to get A2 matrix

A1=

2	3	4	5	← 1 Left Shift
8	7	6	9	← 1 Left Shift
2	3	5	4	← 1 Left Shift
4	8	7	3	← 1 Left Shift

Apply same up shift operation on B1 to get B2 matrix and perform the same individual multiplication to get C2

Step 3: Same as Step 2
This will give C3

Step 4: Same as Step 2
This will give C4

Final Result C = C1 + C2 + C3 + C4

Unit 5

Sorting

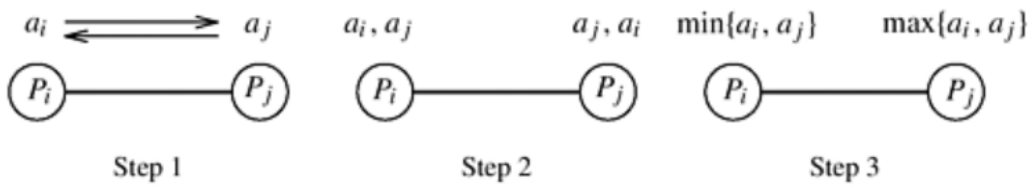
- Sorting is the process of arranging the unordered collcection of data into montonically increasing or decreasing order.
- Internal sorting: The no of elements to be sorted are small enough to fit into the processor memory.
- External sorting: Here the no of elements to be sorted are not enought to fit into the processor memory and that's why we use the seconday storage such as tape or harddisk.
- Twp types of sorting techniques: comparision based and noncomparision based.
- In comparision based sorting techniques the sortign carried out by comparing the pair of elements and then exchange them if they are out of order. The lower bound of any comparision based sequential soring technique is omega(nlogn)
- In noncomparision based sorting we used some property of the elements such as their binary representation, etc to sort the elements. The lower bound of any comparision based sequential soring technique is omega(n)

Issues in sorting on parallel computers

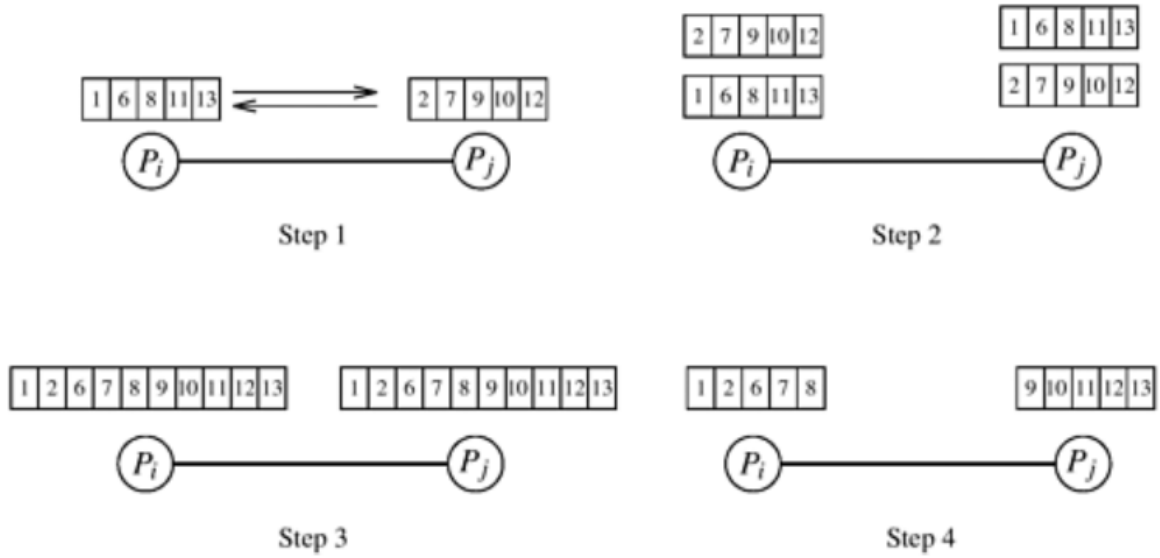
1. Where the input and output sequences are stored
    - In the sequential algorithm there is only one process and that one process only contains all the inut elements and sorted elements so there is not any problem in case of sequential sorting.
    - But in parallel sortign algorithm we divide the elements acrosss the multiple processing elements so there are two ways in the parallel sorting algorthm. One is only one process holds the output sequences and the another way is output sequences are distribued among multiple

processing elements. Here in the 2nd case we consider the enumaration of the processes for ex if we have to processes pi and pj so pi comes before pj in the enumeration sequence then pi contains all the elemenys which are less then the pj process.

2. How comparisions are performed
    - One element per process
        - It is called as compare-exchange



        - commnication cost is $t_s + t_w$ where ts is startup time and tw is the w word size transfer time.
        - In this technique both the proess have one element so both the process communicate and share the input elments with each other so now both the processes have two elements so now first process keep min of both the elements and second process keep the max of both the elements.
    - More than one element per process
        - Here we have more than one elements per process.



        - It is called as compare-split
        - communication cost is $t_s + n/p\ t_w$
        - Here we are sending not just single element and each process contains n/p ekemenst so n/p*tw is the transfer time cost here.
        - In this technique both the processes first sort the elements then share the input sequence with the other process. So because of this both the processes have sorted seq of each other input elements. Now the process merge both the sequences and sort them and then

first process keep smaller elements upto middle element and send greater elements from middle to other process.

Bubble Sort

- The sequential bubble sort algorithm compares and exchanges adjacent elements in thesequence to be sorted.
- n-1 iterations and in each iteration the largest elements is sent to the end of the list and we simply ignore the largst element when we go ahead in the iterations.

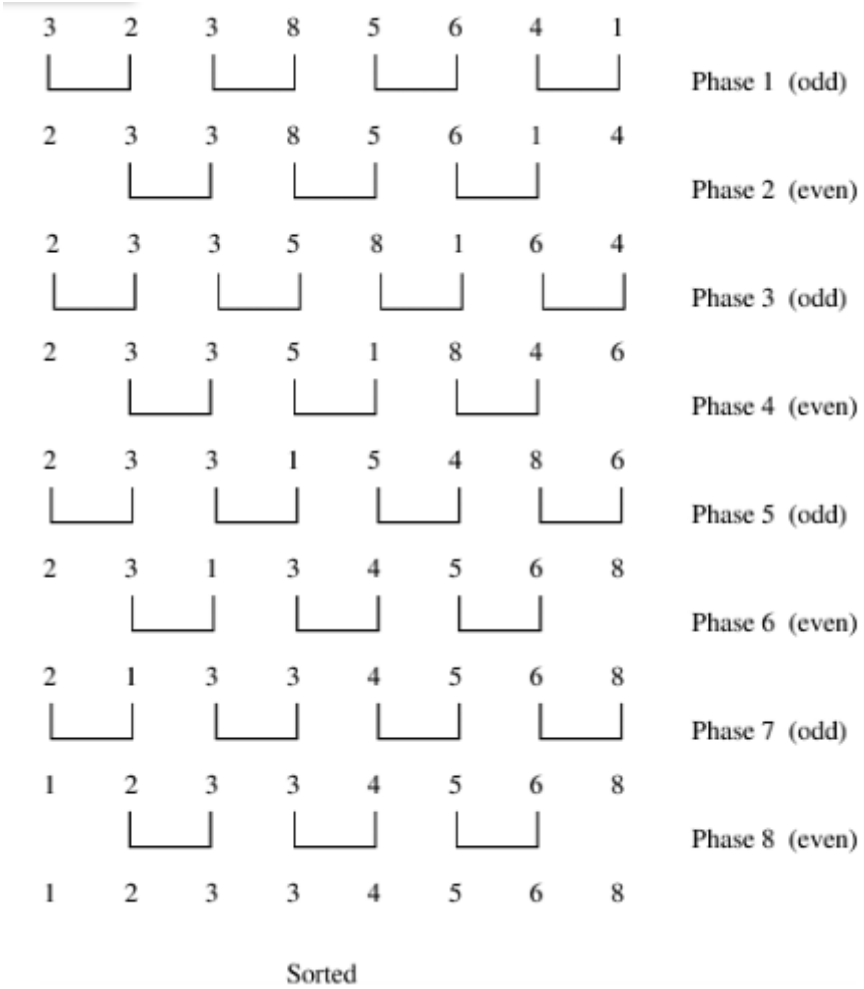
Algorithm 9.2 Sequential bubble sort algorithm.

```
procedure BUBBLE_SORT(n)
begin
   for i := n - 1 downto 1 do
       for j := 1 to i do
           compare-exchange(a_j, a_j + 1);
end BUBBLE_SORT
```

- time complexity is O(n^2)

Odd Even Transposition Sort

- In odd even transposition sort sorting is done in two phases odd phase and even phase and there



are total n such phases.

- sorts n elements in n phases (n is even), each of phase requires n/2 compare-exchang operations.

```
procedure ODD-EVEN(n)
begin
   for i := 1 to n do
   begin
      if i is odd then
           for j := 0 to n/2 - 1 do
               compare-exchange(a_{2j+1}, a_{2j+2});
      if i is even then
           for j := 1 to n/2 - 1 do
               compare-exchange(a_{2j}, a_{2j+1});
   end for
end ODD-EVEN
```

- Algorithm
- Here in each phase there are O(n) comparisions and there are total n phases so overall serial time complexity is O(n^2)

Odd Even Transposition Sort Parallel Implementation

There are two versions of implementing odd even transposition sort parallely. One is one element per process i.e. p=n and other is more than one element per process i.e. `p<n`

First Version (One Element per process i.e. p=n):

1. p = n

	P1	P2	P3	P4	P5	P6	P7	P8
Phase 1(odd)	8	3	5	6	4	2	1	7
Phase 2(even)	3	8	5	6	2	4	1	7
Phase 3(odd)	3	5	8	2	6	1	4	7

Complexity:

- Total parallel execution time is O(n) becasue here each processing element require n stages and in each stage comparision is just O(1)
  But this O(n) is for p processing elements and here in our problem p=n so total cost is O(n^2)
- best serial sorting algorithm complexity is O(nlogn) and here O(n^2) is greater then this best serial sortign algorithm complexity that's why this implementation of this algorithm is not cost-optimal.

Second Version (More than one Element per process i.e. p<n):

P1	P2	P3	P4
{8 3}	{5 6}	{4 2}	{1 7}

Here, we require p/2 odd phases and p/2 even phases.

Complexity:

Parallel Execution Time (Tp)

Local Sort+Comparisons+Communications

$$Tp = \theta((n/p)\log(n/p)) + \theta(n) + \theta(n)$$

To make it cost optimal p=O(log n)

- Here each process have n/p elements.
- also at each process we use quick or merge sort to sort the local elements so local sort time complexity is (n/p)log(n/p) because there are only n/p elements in each process.
- So here overall cost i.e. p*tp is O(logn) <= best sequential algorithm complexity and that's why this algorithm is cost optimal.

Shell Sort Serial Algorithm

- oldest sorting algorithm invented by Donald Shell in 1959
- Approach of shell sort
  - The input elements are arranged in the 2d array
  - Sort the columns of the array

Consider a sequence of numbers to be sorted

3 7 9 0 5 1 6 8 4 2 0 6 1 5 7 3 4 9 8 2

Step 1 : The sequence is arranged in an array with 7 columns (left), then the columns are sorted as shown on the right side.

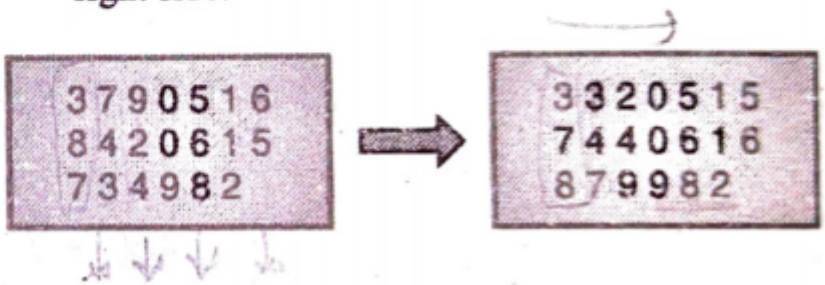


Fig. 5.2.2

- Example

Step-2 : The sequence is now arranged in three columns which are again sorted.

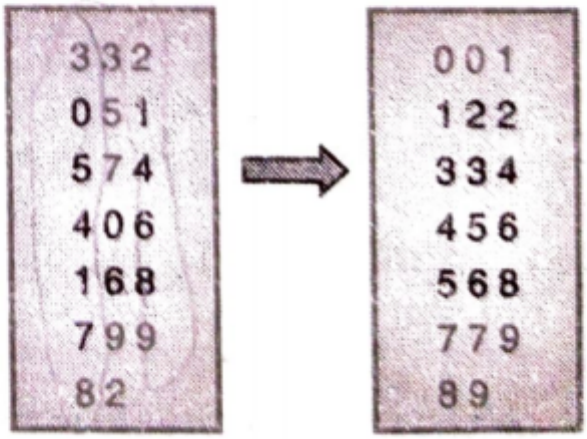


Fig. 5.2.3

– The sequence is almost sorted at this stage, when it will be arranged in one column in the last step. At this stage only elements 6, 8 and 9 have to move a little bit to their correct position.

- Here in this sorting we first start with the more no of columns and then we move towards narrowing the no of columns and finally reach to single column which contains all the sorted elements.
- total no of iterations of the shell algorithm is equal to $\log_2 n$

Shell Sort Parallel Algorithm

There are two phases:

- During the first phase,processes that are far away from each other in the array compare-split their elements.
- During thesecond phase, the algorithm switches to an odd-even transposition sort

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log p\right)}^{\text{first phase}} + \overbrace{\Theta\left(l\frac{n}{p}\right)}^{\text{second phase}}.$$

The performance of shellsort depends on the value of l. If l is small, then the algorithm performs significantly better than odd-even transposition sort; if l is $\Theta(p)$, then both algorithms perform similarly. Problem 9.13 investigates the worst-case value of l.

Quick Sort

- serial quick sort time complexity is O(nlogn)
- Parallel implementation time complexity is O(logn) if we perform the partitioning step in O(1) by using the O(n) processes.

All pair shortest paths

- All pair shortest path means finding the shortest path from every node to every other node.

Dijikstra's Algorithm

- Single source shortest path algorithm
- Greedy stretegy
- time complexity is O(n2)
- does not works with negative weights

As dijistra algorithm is not all pair shortest path algorithm. It is only sinle source shortest path algorihthm, So in order to find the shortest path from each of the node to every other node we can run dijikstra algorithm by considering every node as a source node.

So by default dijikstra algorithm complexity is O(n^2). So serially for all pair shortest path the serial execution time will become O(n^3)

Parallel Formulation:

- Source partitioned formulation: In this approach we have n processes and each process is finding the shortest path by considering the source vertex. In other words we are allocating n diff source vertex to n diff processes and each process will take the serial complexity of O(n2). So here overall complexity is O(n^2) i.e. O(n^3)/n. Here `p=n or p<n`. If p<n then the total vertex are divided among p processes i.e. n/p

- Source parallel formulation: In source parallel formulation instead of single process managing the source vertex here multiple processes are there for every source vertex. and mulitple source vertex are processed at a time paralley by multiple processes. So here p>n and execution time is

Parallel Execution Time:

Tp= θ(n3/p) + θ(nlog p)

Computation= θ(n3/p)

Communication= θ(nlog p)

Floyd's Algorithm

- All pair shortest algorithm
- sequential runtime is O(n^3)
- parallel runtime is

$$T_P = \overbrace{\Theta\left(\frac{n^3}{p}\right)}^{\text{computation}} + \overbrace{\Theta\left(\frac{n^2}{\sqrt{p}}\log p\right)}^{\text{communication}}.$$

Algorithm for sparse graphs

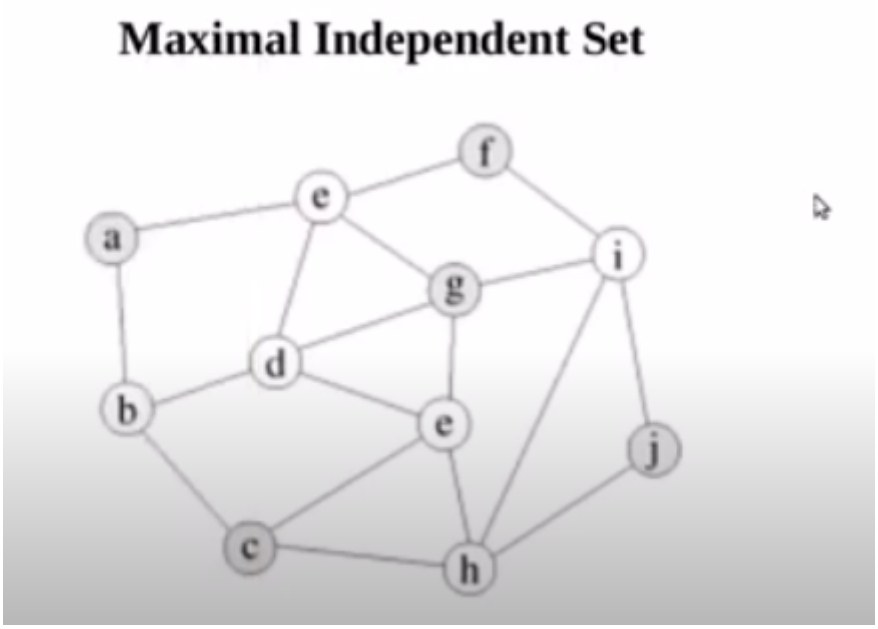
Graph: Represented by G = (V,E)
Representations:

- Adjancency Matrix: complexity is V^2
- Adjancency List: Complexity is V+E

Sparse Graphs: If the graph contains very small no of edges then the possible no of edges then the graph is called as sparse graph otherwise it is dense graph.

Adjacency Matrix is suitable for dense graph and adjacency list is suitable for sparse graphs.

Indepedent sets and Maximal indepent sets



Indepedent set {a,d,i,h}

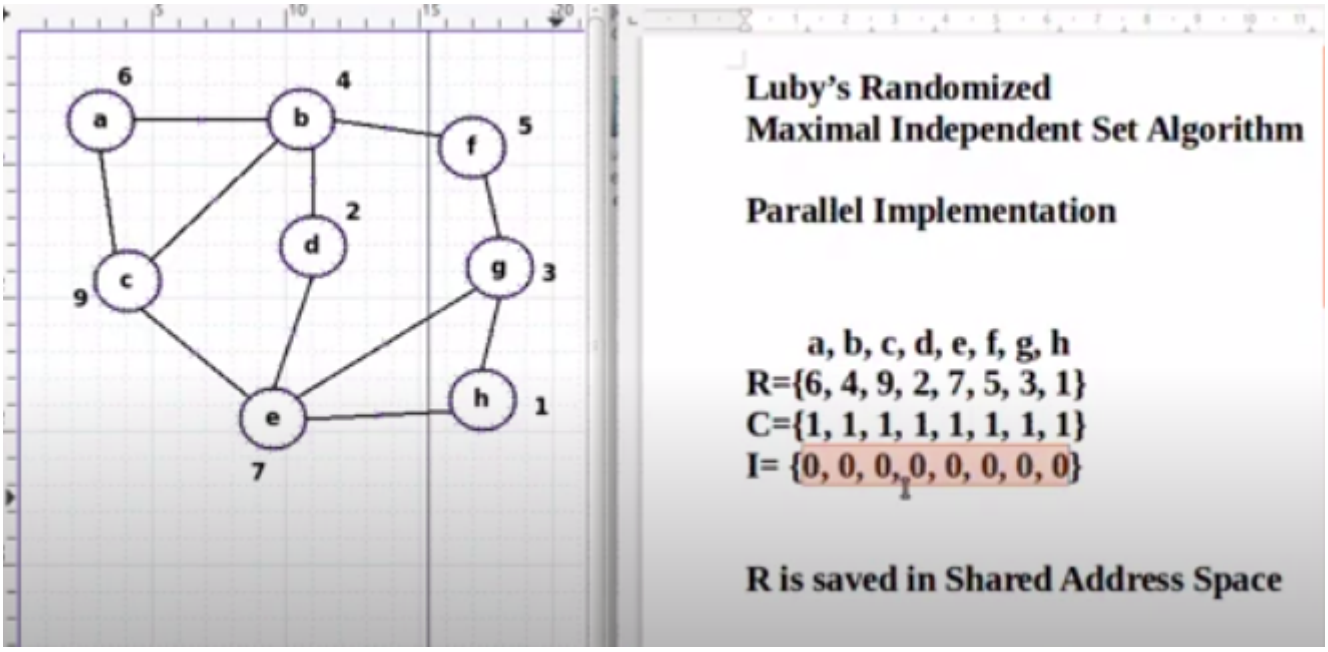
What is indepedent set?
Indepedent set is a set where there is no direct edge between any of the nodes which are present on the set.

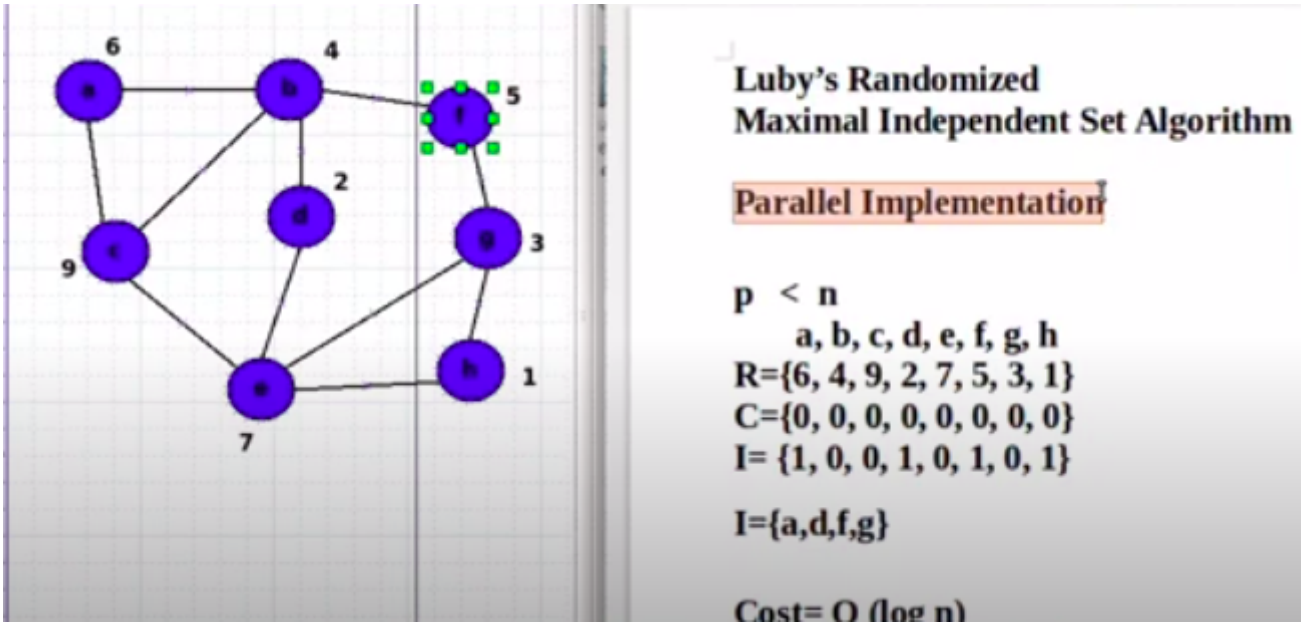
No direct should be present between two nodes is the property of indepedence.

Maximal indepedent set
It is a indepedent set with maximum cardinality

Algorithm to find Maximal indepedent set?
Simple start with the set (say set1) which contains all the vertices of the graph and then select the first vertex from set1 and add it to the set2 and then delete that vertex and all the adjancent vertices of that vertex from set1 and repeat this process until set1 is not empty. The vertices in the set2 are the indepdent set vertices.

Luby's Algorithm

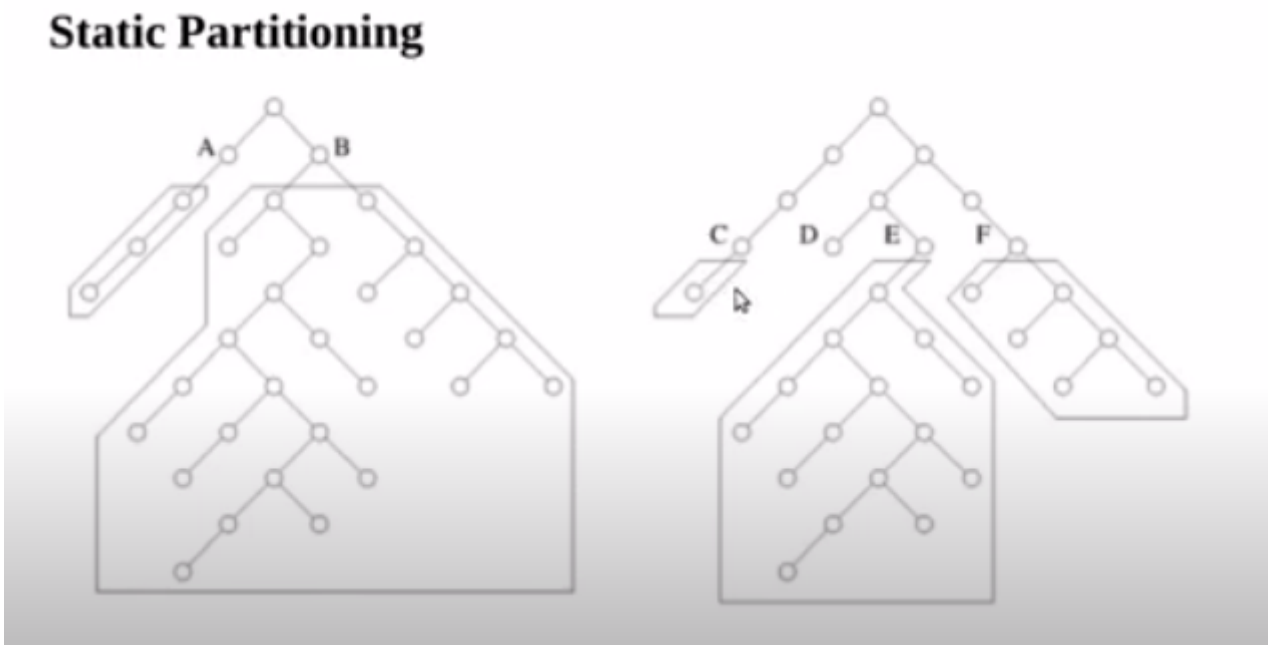




- In Luby's algorithm we have three sets randomized set, candidate set and the indepdenet set.
- The randomized set assign random value for each vertex
- the candidate set values are initially 1 and indepedent set values are initially 0
- Here in the parallel implemnetation we can consider either p=n or p<n if `p<n` then one proess handles more then one vertex.
- Each process checks whether the random value of that process node is less then its neighbours, it it less then we can make corrosponding candidate set entry as 0 and indepedent set values also 0. Also we change value of its neighbours to 0 in the candidate set. And if the values is not less then its neightbours then we don't do anything.
- when all the candidate set values become zero then we simply take the vertices which are marked as 1 in the indepdent sets. That vertices are the maximal indepdent set vertices.
- Here R set is in shared space so that it is accessible to all the processes.

Parallel Depth First Search

- serial DFS time complexity is O(|V| + |E|)
- Parallel DFS can be implements by using static partitioning and dynamic load balancing approach.
- In static partitioning each branch of the tree is allocated to diff process with the separte stack.

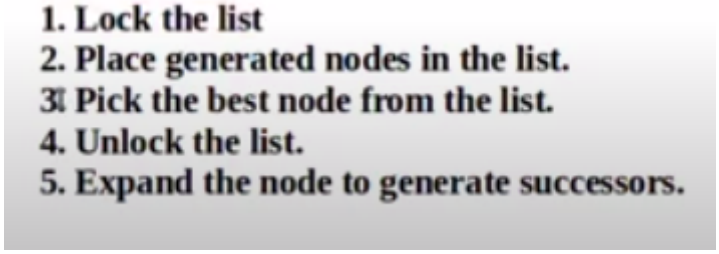


- But sometimes in the static partitioning some proecss gets lot of work and some get very small amount work so the process which get small amount work becomes idle after finishing its work. In order to avoid this issue we can use dynamic load balancing.
- There are two approaches in dynamic load balancing: Message passing approach and shared address space.
- In message passing there is a concept of donot and recipent process where receipent process request for the work to donor process.

Parallel Best First Serach

- In best first search algorithm we have two lists: open list and close list where open list is a priority queue.
- best first serach is informed, greedy stretegy, heuristic function based algorithm.
- Here we pushed the start nodes into the open list then we remote it and move to the closed list then we move all its neighbors into open list and becasue of priority queue we will get the node with the minimum heuritic value first.
- Space and time complexity of serial algorithm is O(b^m) where b is branching factor and m is the maximum depth.
- In parallel version instead of just one processes there are mutliple processes which performs the operation simultaneously on the open list. Here open list is called as global open list.

- And here because of global open list there may be some incosistancies happen that's why every process have to perform this 5 steps



Unit 6

What is CUDA

- CUDA (Compute Unified Device Architecture)
- CUDA introduced by NVIDIA in 2007 as a programming paradigm or interaface to create general purpose program that run on GPU
- The language of the CUDA is very similar to C. There are some extra qualifiers and the APIs are added for parallel programming
- CUDA contains some specific functions called kernals
- Actually in CUDA a kernal can be full program or a function which is invoked by CPU.
- CUDA provided highly parallel execution facilities in terms of threads.
- CUDA also provided shared memory and synchronization among threds.
- CPU is called as host
- GPU is called as device
- CUDA programming model is a heterogeneous model.
- cuda program is saved with the extension `.cu`

CUDA Architecture

- CUDA paradigm is used for SPMD (Single Program Mulitple Data) computing device.

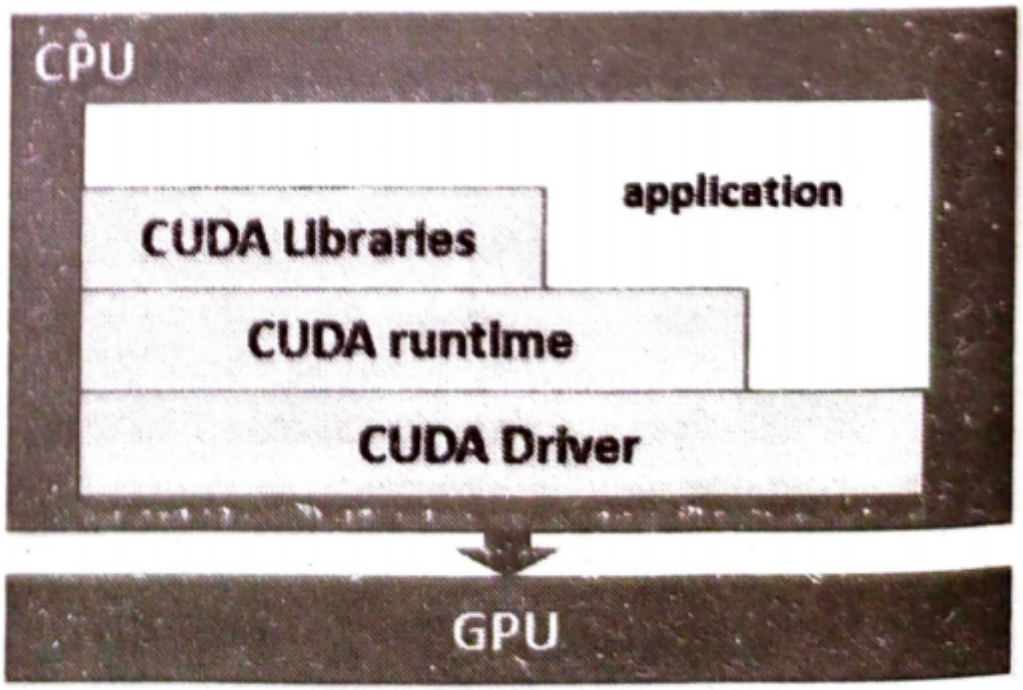
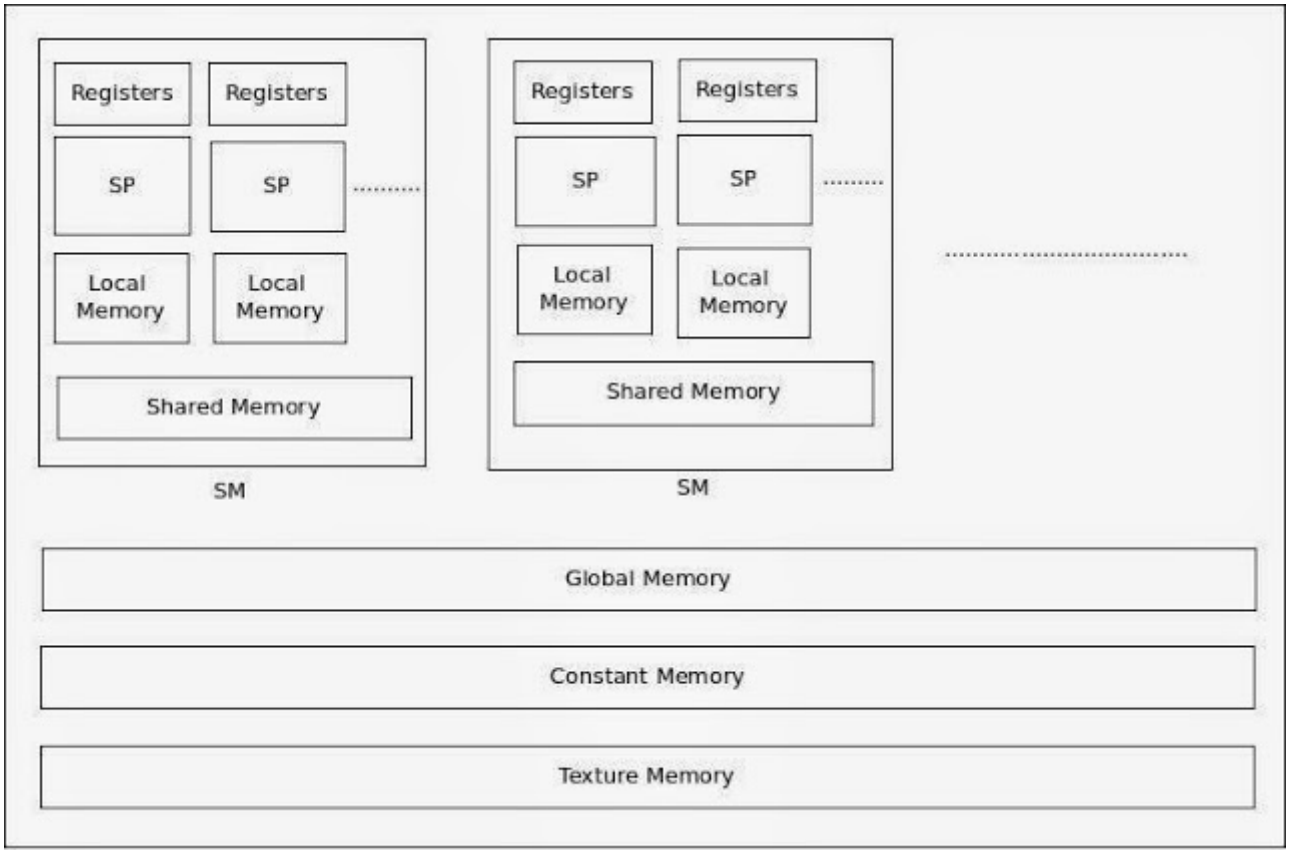


Fig. 6.1.1 : CUDA architecture

- Stream Multiprocessor(SM) and Stream Processor(SP)
  GPU consists of smaller components called as Stream Multiprocesssors(SM). Each SM consists of many Stream Processors(SP) on which actual computation is done. Each SP is also called a Cuda core.
- Thread, Block, Grid
  Thread is a single instance of execution. On one SP, one or more threads can be executed. A group of threads is called a Block. On one SM, one or more blocks can be executed. A group of blocks is called a Grid. One Grid is generated for one Kernel and on one GPU. Also, only one kernel can be executed at one time instance.
- Number of threads in a block running simultaneously on a SM is called a Warp.

Memory Organization



1. Local Memory
   Each SP uses Local memory. All variables declared in a kernel(a function to be executed on GPU) are saved into Local memory.
2. Registers
   Kernel may consist of several expressions. During execution of an expression, values are saved into the Registers of SP.
3. Global Memory
   It is the main memory of GPU. Whenever a memory from GPU is allocated for variables by using cudaMalloc() function, by default, it uses global memory.
4. Shared Memory
   Multiple threads in the block can access this shared memory
5. Constant Memory
   Constant Memory is used to store constant values.
   `__constant__` specifier
6. Texture Memory
   Texture memory is again used to reduce the latency. Texture memory is used in a special case. Consider an image. When we access a particular pixel, there are more chances that we will access surrounding pixels. Such a group of values which are accessed together are saved in texture memory.

Applications of CUDA

- Medical Imaging: Magnetic resonance imaging (MRI)

- Video Transcoding
- Computational sciences: MATLAB
- Oil and Natural Resource Exploration
- Fluid Dynaimcs

CUDA C/C++ Application

- The application architecture contains both serial and parallel code.
- Serial code executes in a HOST (CPU) thread whereas parallel code executes in a DEVICE (GPU) mutliple threads.

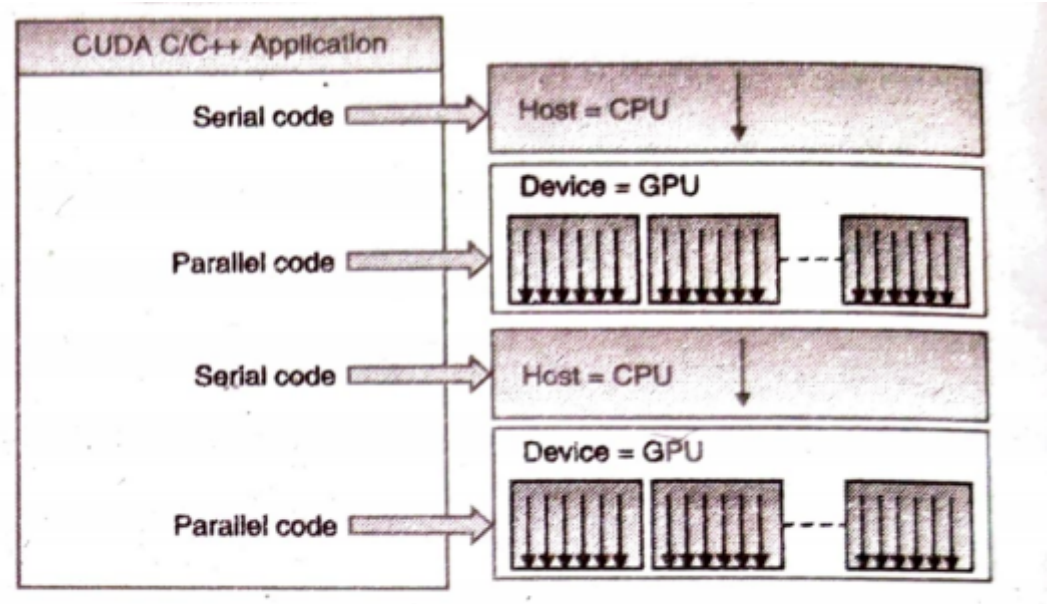


Fig. 6.6.2 : Block of threads

-
- The code that executes on GPU device is considered as a kernal
- The executable code for both host and device are generated by NVCC (NVIDIA C Compiler)

CUDA kernal

- Any function which executes the code on GPU is called as kernal in CUDA world.
- kernal function is declared by using the `__global__` specifier.
- every thread which executes the kernal code assign with unique id called as thread id. Built-in variables **threadidx** is used to access thread ID within the kernal.
- return datatype of kernal function is always void

Basic flow of hetergeneous program:

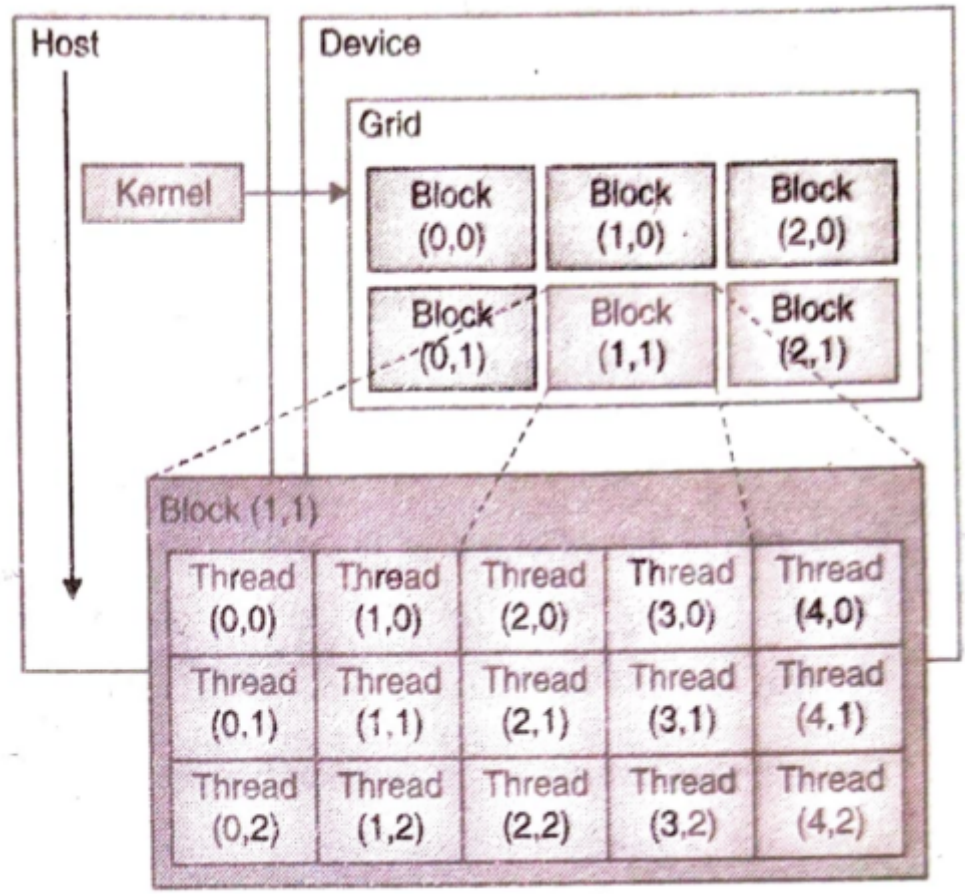
1. Host and device memory declaration and allocation
2. Host memory initialization
3. copy host memory data into device memory
4. execute kernal functions
5. copy data back from device memory to host memory

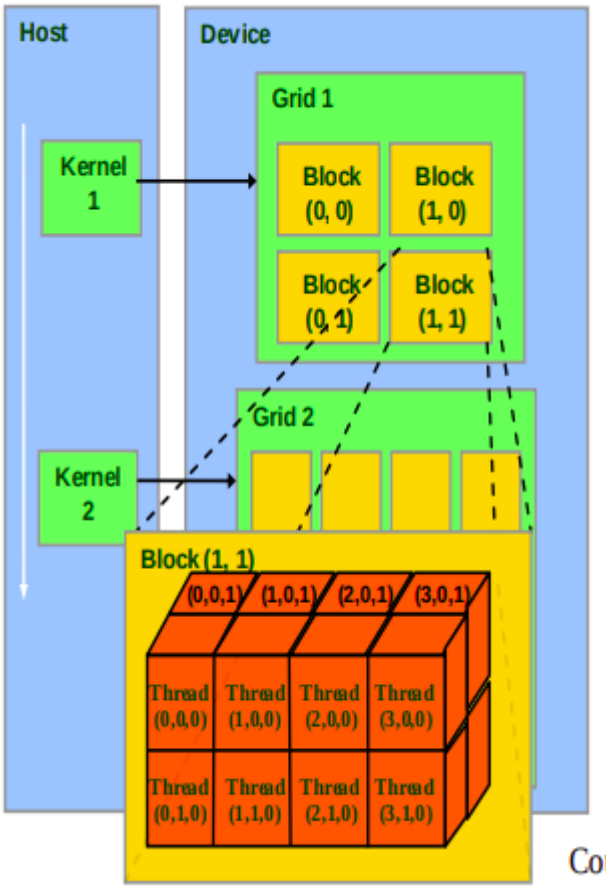
Organizing Threads

Hierarchy of threads is classified into two levels: blocks and grids

block: collections of threads

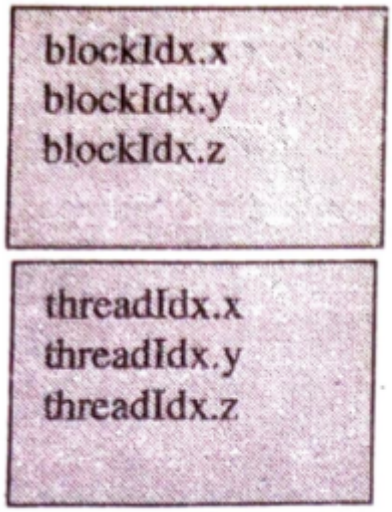
grid: collection of blocks





Cour

- threads in the same block cooperate or communicate with each other. For communication there are two techniques
  - block-local synchronization
  - block-local shared memory
- coordinate variables: blockIdx and threadIdx
- The type of these coordinate variables is unit 3 which is CUDA built in vector type derived from basic integer type.
- In CUDA grids and blocks are organizaed into three dimensions



-
- threads belonging to same block can communicate to each other very easily.
- whereas threads belongs to other block cannot cooperate and communication is very time consuming.

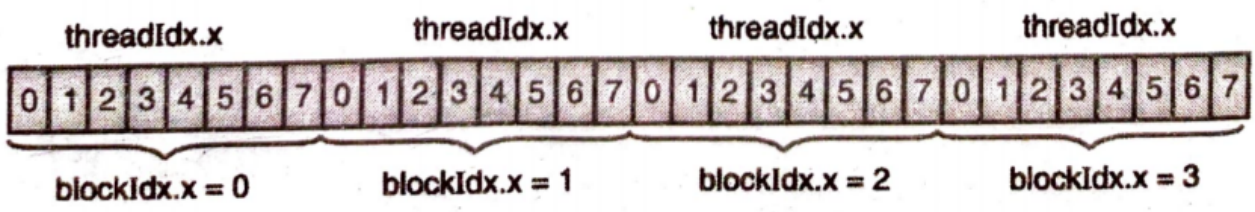
Launching a CUDA kernal

```
kernel_name<<<grid,block>>>(argument list)
```

first value is grid dimension i.e. no of blocks to launch
second value is block dimension i.e. no of threads to launch

Example:

```
kernal_name<<<4,8>>>(argument_list)
```



Qualifiers	Execution	Callable	Notes
__global__	On the device	From the host	Return type must be void
__host__	On the host	From the host only	It is possible to omit return type
__device__	On the device	From the device only	

Manage Communication and Synchronization

- CUDA API provides a hardware thread function named as syncthreads() for synchronization among threads.
- communcation among threads is handled by using the shared memory.

GPU Memory

- A large global memory is available and accessible to all multiprocessors but as it is not a cache memory that's why it is very slow.
- Sharead memory is very fast for read and write operaitons and it is local to each block so other blocks cannot access shared memory of any another block.
- Speed of reading and writing to shared memory is same as register read and write speed.
- shared memory is also called PDC (Parallel Data Cache)
- shared memory size is realtively very small upto 16KB
- Qualifiers `__shared__` and `__device__` are used to declare the variables in shared and global memory.

CUDA Functions

- cudaMalloc

```
__host__ __device__ cudaError_t cudaMalloc ( void** devPtr, size_t size )
```

Allocate memory on the device.

Parameters

`devPtr`
- Pointer to allocated device memory

`size`
- Requested allocation size in bytes

Returns

cudaSuccess, cudaErrorInvalidValue, cudaErrorMemoryAllocation

Ex: `cudaMalloc((void **)&d,6*sizeof(int));`

- cudaMemcpy

  __host__cudaError_t cudaMemcpy (void* dst, const void* src, size_t count, cudaMemcpyKind kind)

  Copies data between host and device.

  Parameters

  `dst`
  - Destination memory address

  `src`
  - Source memory address

  `count`
  - Size in bytes to copy

  `kind`
  - Type of transfer

  Returns

  cudaSuccess, cudaErrorInvalidValue, cudaErrorInvalidMemcpyDirection

  type of transfer:

  - cudaMemcpyHostToHost
  - daMemcpyHostToDevice
  - cudaMemcpyDeviceToHost
  - cudaMemcpyDeviceToDevice
  - cudaMemcpyDefault

    Ex: `cudaMemcpy(d,a,6*sizeof(int),cudaMemcpyHostToDevice);`

- cudaFree

  __host__ __device__cudaError_t cudaFree (void* devPtr)

  Frees memory on the device.

  Parameters

  `devPtr`
  - Device pointer to memory to free

  Returns

  cudaSuccess, cudaErrorInvalidValue

  Ex: `cudaFree(d);`

Other Important Points

- SM (Symmetric Multiprocessing)
- Each block contains 256 or 512 threads
- Each SM can execute 16 blocks and there are total 16 or 32 SM on each GPU
- Each SM contains 8 SPs and Each GPU contains 16 or 32 SMs

- dim3 in cuda
  - dim3 is data type used to define multi dimensional block or grid.
  - It accepts 3 arguments. e.g. dim3 var(z,y,x). We can also provide 1 or 2 arguments like dim3 var(y,x) or dim3 var(x). If we are providing less than 3 arguments, then others are considered to be 1.
  - Variable defined by dim3 is used in kernel invocation.
  - `dim3 grid(3,2) kernelname<<<grid,1>>>(arg1,arg2,arg3);`
- block size: blockDim.x;
- grid size: gridDim.x;
- We can find out id of current GPU by calling function cudaGetDevice().
- We can find out number of CUDA enabled gpus in our system by calling cudaGetDeviceCount() function.